Classification-Enhanced Ranking

Paul N. Bennett
Microsoft Research
One Microsoft Way, Redmond
WA USA
paul.n.bennett@microsoft.com

Krysta Svore
Microsoft Research
One Microsoft Way, Redmond
WA USA
ksvore@microsoft.com

Susan T. Dumais
Microsoft Research
One Microsoft Way, Redmond
WA USA
sdumais@microsoft.com

ABSTRACT

Many have speculated that classifying web pages can improve a search engine's ranking of results. Intuitively results should be more relevant when they match the class of a query. We present a simple framework for classification-enhanced ranking that uses clicks in combination with the classification of web pages to derive a class distribution for the query. We then go on to define a variety of features that capture the match between the class distributions of a web page and a query, the ambiguity of a query, and the coverage of a retrieved result relative to a query's set of classes. Experimental results demonstrate that a ranker learned with these features significantly improves ranking over a competitive baseline. Furthermore, our methodology is agnostic with respect to the classification space and can be used to derive query classes for a variety of different taxonomies.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*

General Terms

Algorithms, Experimentation

Keywords

query classification, learning to rank

1. INTRODUCTION

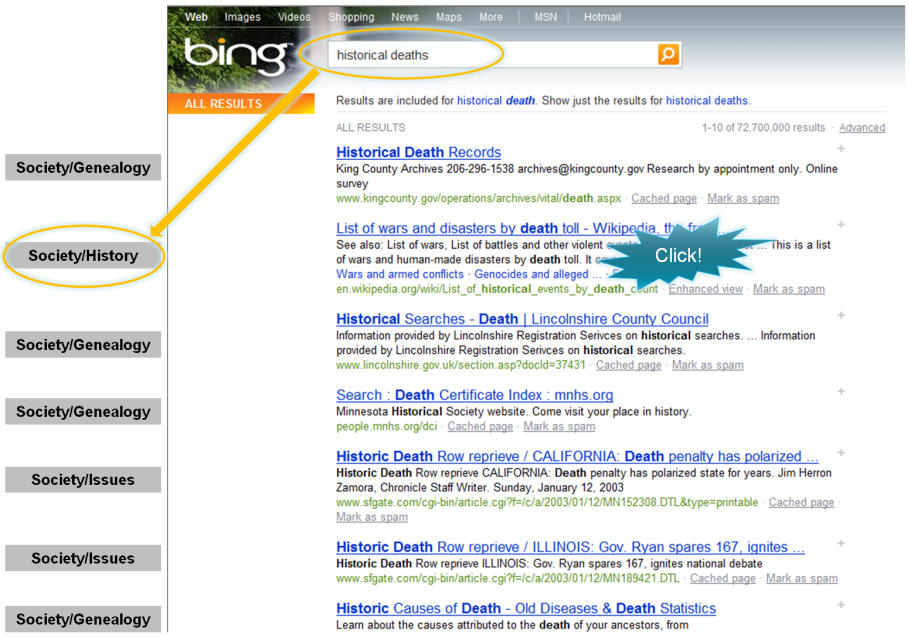
Many have speculated that classifying web pages can improve a search engine's ranking of results relevant to a query [12, 23, 26, 13]. Intuitively results should be more relevant when they match the class of a query as well as less relevant when they do not. While much research has focused on attempting to improve query and document classification under the assumption that relevance gains would follow, we focus on why and how classification can directly improve relevance. In particular, we present a simple framework for classification-enhanced ranking that uses clicks in combination with the classification of web pages to derive a class distribution for the query. At the heart of our approach is the assumption that clicks on a specific URL should not only serve as a relevance signal for that URL but can also be used as a relevance signal for other URLs that are related to the same class.

This situation is depicted in Figure 1a, where we observe a user issuing the query "historical deaths" and clicking on the second result. The gray boxes depict the automatically predicted classes for each result. Note these classes *are not actually displayed to the user*. We interpret a click on the second result to provide evidence that the user desires results from the "Society/History" class in general - including both the clicked URL as well as other URLs belonging to the same class. That is to say, using the query logs we implicitly derive the class of this query is "Society/History". Rather than create a heuristic to balance evidence for the class of the URL versus other features, we introduce this evidence within a machine learning framework and let the model learn the weight to balance these competing features. Using click information solely on a per URL basis would only improve the ranking by moving the clicked result higher. However, in Figure 1b, we see that by using class information to boost results that share the same class as the clicked result, we see improved relevance of results in positions 1-3 (with the clicked result in position 1).

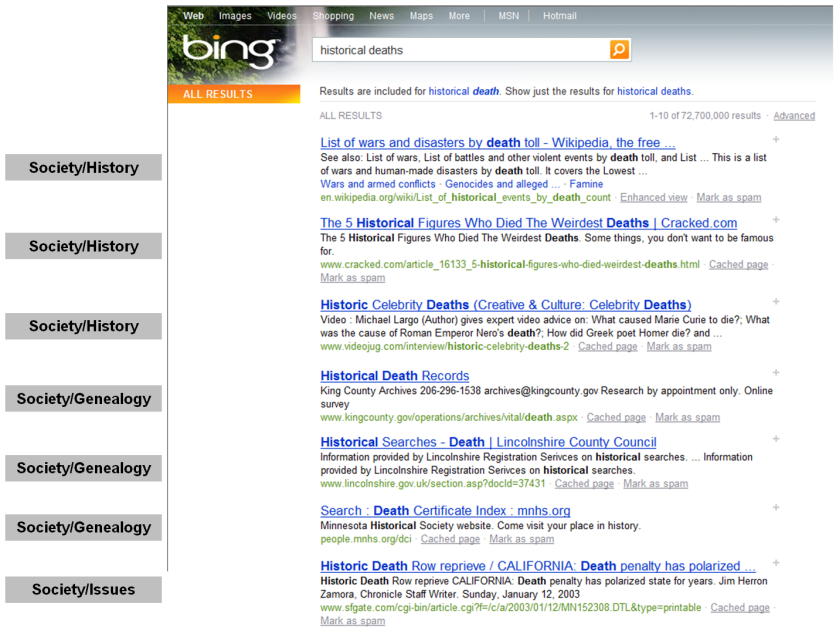
Although previous research [12] has demonstrated categorical interfaces can significantly decrease the time a user spends finding a search result, these interfaces are fraught with their own challenges that has prevented widespread adoption (*e.g.* precision of the classifiers, labels of the classes, ordering of the classes). As a consequence research in this area has often focused on improving these components or the underlying precision of the prediction mechanisms, assuming categories will be directly displayed to the user. In this paper, we do not propose to show the categories to users but rather to use them to improve ranking. By using the classes of clicked results we can boost those results as well as other results that share the same classes.

To achieve this, we define a variety of features that capture the match between the class distributions of a web page and a query, the ambiguity of a query, and the coverage of a retrieved result relative to a query's set of classes. Experimental results demonstrate that a ranker learned with these features significantly improves ranking over a competitive baseline. In an empirical study, we demonstrate significant improvements over a large set of web queries. In a further breakdown of those web queries, the statistical gains continue to hold even on tail and long queries where click data is sparse. Furthermore, our methodology is agnostic with

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.



(a)



(b)

Figure 1: (a) In the query logs, a user issues the query "historical deaths" and clicks on the second result. Using automatically predicted classes for the documents (gray boxes *which are not actually displayed to the user*) and the click information we automatically derive the query class as "Society/History". (b) This provides evidence to a learned ranking function that can boost results related to "Society/History" to provide improved results in the future.

respect to the classification space, and therefore is general enough to be used as a mechanism to derive query classes for a variety of alternative taxonomies.

In the remainder of the paper, we first give a brief overview of related work, placing this work in the broader context of query classification research. Then, we describe our approach in detail. After which, we present the results of an empirical study which highlight the improvements achievable via this method and conduct a small study of feature importance to indicate promising avenues of further study. Finally, we discuss future work and conclude.

2. RELATED WORK

While an exhaustive survey of query classification is not possible, we highlight some of the key related studies. Our study is most similar to work by Broder *et al.* and Gabrilovich *et al.*[4, 13] In their work, they targeted improving the relevance of advertising by improving the match between advertisement classes and query classes. In order to classify the queries, they perform a weighted aggregation of the classes of URLs returned by a search engine. They focus on classifying ads, but we address the end-to-end process of improving web relevance and not just query classification accuracy. In addition, they study only a narrow segment of queries, while we examine a broad range of queries. To be used to improve relevance ranking a method must be able to handle all types of queries or at least detect when it should be applied. Furthermore, while they argue it is possible to perform offline classification and state, "we are currently working on incorporating classification of all indexed pages into the search engine infrastructure", we actually present experiments run using classifications obtained from a classifier run during indexing – using a compact representation that can be used at scale. Finally, their approach assumes the initial output of a ranker or search engine, which introduces a multiple ranking architecture assumption that typically increases computational and communication load requirements at scale. In contrast, we use a smoother weighting function directly derived from the motivation above that enables more efficient computation of the query class distribution without the initial output of a ranker.

Relatedly, Sahami and Heilman [24] also study aggregating the results returned in response to a query. However, in their case, they aggregate the *words* of the returned URLs or their snippets in order to construct an expanded "context" representation of the query to be used in a kernel function that produces query-query similarity estimates. This is then used as the basis for related query suggestions – although it could be used in any kernel-based machine learning algorithm. Our approach differs from theirs both in that we aggregate in the output space instead of the input space (*i.e.* over classes of URLs and not their words) and in that we target search relevance rather than related query suggestion. Additionally, our aggregation is driven by weights derived from user clicks.

Beeferman and Berger [2] present a clustering of queries that are derived from a query log by viewing queries and clicked upon URLs as a bipartite graph. While they claim the content-independent nature of their algorithm is a strength, we point out it can also be a weakness. In particular, it requires click information on related URLs, for at least some queries, to know when related URLs should be displayed. The method we present here instead provides an avenue for independent generalization based on the content alone. We are able to demonstrate our approach works well on all queries, even when click information for any URL is relatively sparse.

Beitzel *et al.* [3] investigate using query logs to classify queries. However, their classification target is more akin to vertical selection where URLs have already been partitioned into backend databases and the goal is to route the query to the correct target (perhaps with target-specific query rewriting). In contrast, our approach is applicable even when URLs have no such structure and instead provides a way to leverage the URL relevance signal from one likely relevant URL to others via the class information.

A variety of other work grew from the KDD Cup-2005 challenge [18] which presented participants with the challenge of classifying 800K queries into one of 67 categories given only 111 queries as "training data". The most successful approaches [25, 17, 27, 26] developed during this challenge make use of external resources such as WordNet and ODP or intermediate taxonomy mappings to improve query classification performance however they typically do so in a manner that does not scale. We not only adopt methods that scale – in particular by augmenting a search index with a small amount of automatically generated class metadata – but we also use methods that work directly with the taxonomy of interest. In addition, they focus on improving query classification over labels that were collected directly on approximately 800 queries (only 800 of the 800K queries were evaluated). Thus, a relationship between improved query classification and performance is only implicit. In contrast, we focus directly on improving relevance and are able to effectively demonstrate our methods over a test set of 12K queries that represents a broad spectrum of web queries. Additionally, we demonstrate gains even over hard queries where click data is sparse.

3. PROBLEM APPROACH

In contrast to other work, we focus on using class information to improve relevance ranking directly, rather than intermediate measures like query classification performance. Our approach consists of two primary components. The first is a method of using click-data to derive query class distributions from class distributions over each result document. Second, rather than try to handcraft the weight given to the match between query and document class distributions, we introduce a set of features that capture important properties of these two distributions. These features can then serve as input to a ranking algorithm that can learn appropriate weights during training and use them for prediction during retrieval.

Furthermore, we demonstrate how to make direct use of click-information to derive the weighting function used to aggregate result classes into a query class distribution. This generalizes the approach to situations where a multistage architecture can be a computational burden. Additionally, because our method can be applied to a large set of URLs (no assumption of top k cutoff), it also can be applied to distributed retrieval where the weighting functions become an approximation of the function that would be used if all click-information were aggregated. By using click-information within a session for a query, we are also implicitly leveraging the user's context [7]. We achieve these gains not only through query and URL classification but by introduc-

ing a novel set of information-theory inspired features which capture the ambiguity of a query, the specificity of a URL, and how well a URL covers a query's set of topics. Furthermore, to our knowledge we conduct the first large scale study that uses classification performed offline during indexing and compressed to a compact enough representation to be used in a large-scale search engine. Finally, to our knowledge this is the first work to demonstrate a fully scalable system where clicks on one URL can improve relevance by boosting topically related URLs.

3.1 Query Classification

We start from a simple assumption. When only one URL is relevant, we define the class of a query to be the class of that URL. When multiple URLs are relevant to the user's information need, then it is natural to broaden this to say the query class *distribution* is an aggregation over all relevant URLs. Using a distribution rather than a single class can be useful when result documents can have multiple aspects. Similarly, when optimizing for a query over a set of users, the distribution can be used to reflect the weighted average of each user's intent.

This definition is both natural and avoids several potential pitfalls of query classification. In particular, model capacity is not wasted on potentially esoteric intents of a query never employed by system users. That is, from the system's point of view, the ranking is optimized for what users actually mean when they type a query rather than what they could mean. Also, while not addressed here, providing "gold-standard" query classification labels for queries for some tasks can be problematic even for human judges primarily because short queries are ambiguous. This definition decomposes the task into two separate problems with better reliability: (1) providing query-URL relevance judgments; (2) providing URL class judgments.

3.1.1 Click-Weighted Query Classification

Given the definition above, we need only identify relevant URL(s) to derive the query class distribution for a query. Previous work has shown that click information can be leveraged to derive a relevance signal [8, 15]. Additionally, the requirements here are actually weaker than if one tries to directly assess relevance.

In particular, note that our model can also benefit from a type of click that could be viewed as erroneous. That is, suppose a promising-looking result entices a user to click on it, but they then find the URL is not relevant. Because the URL is more promising than many other results, the clicked URL is likely to have a similar class to the desired result. Thus by propagating the user's relevance signal along the class dimension, it is possible to increase the likelihood that future searches yield more relevant results even though the click was not on a relevant result.

Likewise, this approach can boost results that have never been clicked but which match the query's predicted class distribution. This flexibility means that it is likely to yield gains for tail queries, which do not have very much click data, as well as head queries.

3.1.2 Model Formalities

More formally, $P(c \mid q)$ is the probability that results from class c are desired given query q was issued. $P(u \mid q)$ is the probability a particular result URL, u, is relevant given query q was issued. $P(c \mid u)$ is the probability that c is the class of a relevant result URL u.[1]

By marginalizing over all URLs we then have:

$$P(c \mid q) = \sum_u P(c \mid u, q) P(u \mid q) \quad (1)$$

Independent by assumption that class is determined by class of relevant URL

$$= \sum_u P(c \mid u) P(u \mid q). \quad (2)$$

We note that this is the same decomposition as that arrived at by [4, 13], however we use a different weighting function $P(u \mid q)$ that is directly derived from click data based on the above motivation. This enables integrating the technique directly into the normal ranking procedure. In contrast, their method requires the output of an original ranker first before determining the weight. This would add a layer of computation that is best avoided if possible.

We can estimate $P(c \mid u)$ given standard classification models and a set of labeled data where URLs have been assigned to classes. The choice of models and features will, of course, depend on the nature of the classes we are modeling. In this work, we examine a set of topical classes and use a standard text classification model based on content extracted from the URLs.

This then leaves us with estimating the term $P(u \mid q)$. This term is essentially the same as what many approaches have tried to model under the assumption that a particular clicked upon URL is relevant. We also incorporate a click-based feature that takes into account last-click behavior in a simple way. In particular, given a click-based function, $\text{evid}(q, u)$, that returns the click-based weight for URL u's relevance to query q with a relevance prior of p, then we perform a simple Bayesian m-estimate [19] smoothed computation:

$$\hat{P}(u \mid q) = \frac{\text{evid}(q, u) + mp}{m + \sum_u \text{evid}(q, u)}. \quad (3)$$

Here p is set to uniform and the URLs in the summation are limited to a candidate set returned by a retrieval algorithm for ranking by the learned model.[2] For any URLs not in the candidate set $\hat{P}(u \mid q)$ is defined to be 0. Obviously richer models that have been developed for estimating $P(u \mid q)$ can also be applied here and should only increase performance with improved predictive accuracy.

Note this model can be used in two cases. In the first case, the classes are mutually exclusive and $P(c \mid u)$ and $P(c \mid q)$ are multinomial variables conditional on u and q, respectively. In the second case, the classes are not mutually exclusive and there is a multivariate Bernoulli vector of variables, $P(c \mid u)$ and $P(c \mid q)$ each representing a binary class property c. In either case, we can imagine a generative

[1] Semantically, one could also interpret this as the proportion of the result covering or discussing class c. In this case, the query classification semantics become the desired percentage of results to cover a class. In either case, the application is the same.

[2] As is common in the learning to rank literature (*e.g.* the well-studied LETOR collection [21] implicitly assumes this), we assume that a basic retrieval function has already reduced the billions of possible URLs that could be retrieved down to a relatively small candidate set – which could still be on the order of thousands – for ranking.

(msn web, 0.668)
(webmessenger, 0.662)
(msn online, 0.640)
(windows web messenger, 0.632)
(talking to friends on msn, 0.613)
(school msn, 0.599)
(msn anywhere, 0.567)
(web message msn com, 0.548)
(msn messager, 0.531)
(hotmail web chat, 0.523)
(messenger web version, 0.501)
(browser based messenger, 0.381)
(im messenger sign in, 0.300)
(msn web browser download, 0.093)
(install msn toolbar, 0.003)
...

Figure 2: Extract of the query click field [14] for the site *http://webmessenger.msn.com*. The first element of each tuple is the query that led to a click on the URL *http://webmessenger.msn.com*, and the second element is the query score for this query as in Equation 4.

model where a user generates a query, q, and conditional on the query, desired URLs are drawn from all possible URLs.

3.1.3 Click-Based Feature

We follow [1, 14] and derive the click-based feature, evid(q, u) in Equation 3, from user session data. A session is defined to be a sequence of interactions with results for the same query within a window of time (here 30 minutes). Thus, clicking on a result and going back to the results page to click on a different result can be taken into account. First a click field for each URL u is constructed. A click field contains queries q_s that resulted in u being shown in the top 10 results along with session-click information. To obtain a query score for query q_s we use the heuristic function introduced in [14]

$$\texttt{Score}(u, q_s) = \frac{\mathrm{C}(u, q_s) + \beta * \mathrm{LastC}(u, q_s)}{\mathrm{Imp}(u, q_s)}, \qquad (4)$$

where $\mathrm{Imp}(u, q_s)$ is the number of impressions where u is shown in the top 10 results for q_s, $\mathrm{C}(u, q_s)$ is the number of times u is clicked for q_s, $\mathrm{LastC}(u, q_s)$ is the number of times u is the temporally last click for q_s, and β is a tuned parameter we set to 0.2 in our experiments. Since the last clicked URL for a query is a good indicator of user satisfaction, the score is increased in proportion to β by the last click count. Other heuristic functions [1] also take into account the dwell time of the click, assuming that reasonably long dwell time (e.g., 10 - 60 sec) is a good indicator of user satisfaction. For clarity, Figure 2 shows an example of the query click field for the site *http://webmessenger.msn.com*, extracted from [14].

Finally, to derive the evidence score for a query q and URL u, we follow one of the formulations given in [14] and consider all queries, q_s in the click-field for a URL, u, which contain all terms in q. That is $q \subseteq q_s$. So we have:

$$\mathrm{evid}(q, u) = \sum_{q_s \in \mathrm{ClickField}(u) | q \subseteq q_s} \texttt{Score}(u, q_s). \qquad (5)$$

In Figure 2, if the figure contained all queries observed for this URL, then evid(*web messenger*, *webmessenger.msn.com*) = 1.134 from the sum of the "windows web messenger" and "messenger web version" lines.[3]

3.1.4 Classification Space

While the approach presented here can be used with any classification space that can be sensibly applied to documents (*e.g.* locality, topic, reading-level), we restrict our current study to topical classes. We chose to use the ODP [20] for classification because of its broad, general purpose topic coverage and availability of reasonably high-quality training data. For the experiments reported in this study we used 219 topical categories from the top two levels of the ODP. Additional details of category selection are presented in Section 4.3.

3.1.5 Efficient Query Classification

Note that if estimates for $P(c \mid u)$ are readily available, then the query classification distribution can be computed quite efficiently. In particular, Equation 3 requires one pass over the candidate set to compute the normalizing constant, and on the second pass, estimates for $\hat{P}(u \mid q)$ can be produced at the same time the query class distribution estimate is updated via Equation 1.

Since classifying the URL, that is estimating $P(c \mid u)$, can be a potential bottleneck, other studies have often resorted to snippet classification[4] because fetching the content of a URL is an expensive operation to perform at retrieval time. However, snippet classification can also be unreliable because snippets themselves are short [13]. Instead, we perform classification on the whole content of the URL. To ensure that this approach can scale, we store the class predictions in the search index. With the efficiency of state-of-the-art linear classifiers, it is possible to perform this type of classification even at web scale.

Keeping search indices compact is desirable for performance reasons. This can be motivated both from the point of view of tradeoffs between what can be kept in memory versus disk and in terms of possible metadata bloat. That is, in the latter case, one can imagine that many types of metadata (both manually generated and automatically generated as here) may be of use during retrieval. However, if this were unchecked, then metadata could quickly outstrip average document size. Thus space in the index of retrieval architectures is typically carefully managed. To demonstrate the suitability of our approach even under tight space constraints, we reduce the class prediction information stored. We store at least one and up to the top three predictions for each URL when they surpass a minimal threshold. We further reduce the confidences to two bits or four levels (low, low-medium, high-medium, and high confidence). For the experiments described in this paper we have fewer than 256 categories, so we need only 8 bits to store the class ID and 2 bits for the confidence. Thus the extra storage requirement for each URL due to class predictions is at most 30 bits and minimal index bloat is incurred. When using the URL's class

[3]Depending on whether the tokenizer chosen tokenizes "webmessenger" as two terms or one then that line will also match or not match, respectively.

[4]Although snippet classification may be an arguable research substitute when forced by necessity by a lack of access to a web-scale crawl.

distribution in computation, any class not stored in the index is defined to be zero. Any class stored in the index for the URL has its confidence category replaced by the threshold value required to fall into that category. Thus this is a truncated discretized version of the class-prediction vector.

From an applications perspective, this means that this method can easily scale. From a research perspective, this reduction in information means results here are likely a pessimistic estimate on the amount of performance gains that can be gained through class information.

Type	Name	All	Lvl 1	Lvl 2
URL	URLClassEntropy	√	–	–
Query	QueryClassEntropy	√	–	–
Query-URL	QueryClassURLMatch	√	–	–
	QUClassNoMatch	√	–	–
	QUClassMatch	√	–	–
	ArgMaxOdds	√	√	√
	MaxOdds	√	√	√
	KLDistance	√	√	√
	CrossEntropy	√	√	√

Table 1: Class Features

3.2 URL and Query Class Distribution-Based Features

After having computed both the URL class distribution and query class distribution, we need to know how well they match. Rather than trying to create ad-hoc rules to be used in the retrieval engine, we follow a common machine learning paradigm and define features that capture our intuitions of what factors might impact ranking and leave it to the learning algorithm to learn the weights and model the interactions with other features. Features for learning to rank are usually broken into URL features (a feature whose value only depends on the URL), Query features (a feature whose value only depends on the query), and Query-URL features (a feature whose value depends on both query and URL). Table 1 summarizes the features used in our experiments.

The only URL feature we introduce based on the class distribution information is *URLClassEntropy*. This is the entropy of the estimated distribution of $P(c \mid u)$. The intuition is that documents on fewer topics, *i.e.* those with lower entropy, may be preferred higher in the ranking because they are more focused documents.

The only Query feature we introduce is *QueryClassEntropy*, which measures the entropy of the class distribution for a query. As demonstrated in [9], this entropy can capture the ambiguity of a query with more ambiguous queries having a higher entropy. As for ranking, the intuition is that the weight given to Query-URL class matches may need to be reduced for queries with high entropy since the ambiguity dilutes the importance of a match.

Next, we introduce a variety of simple Query-URL features that encode match quality of the most likely query class to the most likely document class. *QueryClassURLMatch* captures the specificity of the match by giving higher weight to deeper matches in the hierarchy (outputs level of most specific match). For example, comparing "Business" to "Sports" would yield a value of 0, while "Sports" to "Sports/Football" would give 1, and "Sports/Football" to "Sports/Football" would be 2. *QUClassNoMatch* is a binary variable that indicates the condition that the classes do not match exactly. For the three previous examples, this variable would take the values 1, 1, and 0, respectively. *QUClassMatch* is a binary variable indicating a partial or complete match. Again, for the three previous examples, this variable would take the values 0, 1, and 1, respectively.

The next two Query-URL features try to model the cases where a URL captures some classes of the query better than others. *ArgMaxOdds* measures how well a URL matches the most likely class of a query weighted by how likely that class is for the query. Let c_q^* be the most likely class for the query, that is $c_q^* = \text{argmax}_c \hat{P}(c \mid q)$. Then, *ArgMaxOdds* is:

$$\hat{P}(c_q^* \mid q) \log \frac{\hat{P}(c_q^* \mid u)}{\hat{P}(c_q^*)}, \tag{6}$$

which is the change over the prior (a measure of odds) caused by observing the content of this URL for the query's most likely class weighted by the probability of observing that class. Likewise, we can ask how well a URL matches any aspect of the query. *MaxOdds* does this by loosening the definition of *ArgMaxOdds* to be over all classes and computes:

$$\max_c \hat{P}(c \mid q) \log \frac{\hat{P}(c \mid u)}{\hat{P}(c)}. \tag{7}$$

The last two Query-URL features capture the match between the entire class distribution for both query and URL. Let c_q be the class distribution for a query and c_u for a URL where c_u is smoothed with a uniform prior over classes to handle the truncation for class compression discussed in Section 3.1.5. *KLDistance* measures the KL distance in bits between c_q and c_u. That is,

$$D_{\text{KL}}(c_q || c_u) = \sum_i c_q[i] \log \frac{c_q[i]}{c_u[i]}, \tag{8}$$

where $0 \log 0 \doteq 0$. *CrossEntropy* is related to KL-divergence and here it is motivated as a measure of how much a topic present in the query is not covered by the URL. Taking once again $0 \log 0 \doteq 0$, *CrossEntropy* is

$$-\sum_i c_q[i] \log c_u[i]. \tag{9}$$

Finally, each of the features described can be defined as operating over the whole classification space, over just the top-level (or Level 1) categories in the hierarchy or over just the Level 2 categories in the hierarchy. Except for the final four, we compute only the versions over all classes. We denote features restricted to only Level 1 or Level 2 classes by appending a "1" or "2" to them respectively. The breakdown of which versions are computed are given in Table 1.

4. EXPERIMENT ANALYSIS

4.1 Datasets

We evaluate our method on a real-world Web-scale data collection. The data consists of a set of input-output pairs $(\mathbf{x}, y)$, where $\mathbf{x}$ is a feature vector that represents a query-URL pair (q, u), and y is a human-judged label indicating the relevance of u to q on a 5-level relevance scale from 0 to

4, with 4 meaning URL u is the most relevant to query q and 0 meaning u is not relevant to q. The data contains queries sampled from query log files of a commercial search engine and corresponding URLs; each query is associated with, on average, 150-200 URLs. All queries are English queries and can contain up to 10 query terms. The features consist of several hundred query-URL match features, such as BM25 [22], query-only features, and URL-only features.

Our train/validation/test data contains 54K/15K/12K queries, respectively. We report overall results over the 12K queries in the test set. In addition, we examine two splits of our test set to understand the performance of our methods on different query types. One split separates short queries (< 4 terms in query) from long queries (≥ 4 terms in query). The other split separates head (more popular) queries from tail (less popular) queries. We use the amount of click and anchor information as an indicator of query popularity[5]. The short/long test sets contain 9K/3K queries, respectively. The head/tail test sets contain 6K/6K queries, respectively. We choose these two breakdowns in particular because our query classification method is click-based. One would expect to see much sparser click information in both tail queries and long queries. Therefore these splits enable us to examine the robustness of the class-based features for difficult queries.

4.2 Rankers

4.2.1 LambdaRank

Because of its competitiveness in other published studies and widespread familiarity within the ranking community, we choose to focus on a state-of-the-art ranking algorithm called LambdaRank [6] that optimizes for IR measures. We review it briefly here and refer the reader to [6] for complete details. LambdaRank is both a list-based and a pair-based neural network learning algorithm; it is trained on pairs of URLs per query, where URLs in a pair have different relevance labels. It is an extension of RankNet [5], another pair-based ranking algorithm whose cost function is a sigmoid followed by a pair-based cross-entropy cost.

In most machine learning tasks, a target evaluation measure is used to evaluate the accuracy of the model at test time, and an optimization measure, generally a smooth approximation to the target measure, is used to train the system. Typical IR target costs, such as mean NDCG, are either flat or non-differentiable everywhere. Hence, direct optimization of the target measure is quite challenging. LambdaRank [6] leverages the fact that neural net training only needs the gradients of the measure with respect to the model scores, and not the function itself, thus avoiding the problem of direct optimization. The gradients are defined by specifying rules about how swapping two URLs, after sorting them by score for a given query, changes the measure. Recently, it has been shown that LambdaRank [6] has been shown empirically to find a local optimum [11, 29] for Mean Average Precision, Mean Reciprocal Rank, and NDCG, and likely for other IR measures as well.

4.2.2 LambdaMART

To explore the importance of features for future study, we use the LambdaMART [28] algorithm. While it is a state-of-the-art ranking algorithm in its own right – based on boosted regression trees and LambdaRank [6] – we choose it for feature exploration because of its robustness and interpretability. In particular, it is more robust to a set of features with different ranges of values such as categorical features. Additionally, since LambdaMART produces a tree-based model, it can be employed as a feature selection algorithm or as an algorithm for determining a ranked list of important features in the model. The feature importance is determined by adding the contribution of the feature across all splits and all trees in the model, based on the decrease in mean-squared error when applying the given feature and split value.

[5]We could also split based on the frequency (number of times issued by users) of the query.

4.3 Classifiers

To train topic classifiers, we used a crawl of ODP from early 2008. We first split the data into a 70%/30% train/validation set, then identified the topic categories (some categories like *regional* are not topical and were discarded) that had at least 1K documents as good candidates for models that could be learned well and would be broadly applicable. This resulted in 219 categories at the top two levels of the hierarchy. To simplify the prediction-phase of the classification model, we simply flattened the two levels to a m-of-N (where $N = 219$) prediction task. A logistic regression classifier using an L2 regularizer was trained over each of the ODP topics identified. The resulting models were then applied at indexing time in the manner described in Section 3.1.5.

4.4 Performance Measures

We evaluate the quality of the ranking produced by the learned ranker using NDCG, Normalized Discounted Cumulative Gain (NDCG) [16], a widely used measure for search metrics. NDCG for a given query q is defined as follows:

$$\text{NDCG@}L_q = \frac{100}{Z} \sum_{r=1}^{L} \frac{2^{l(r)} - 1}{\log(1 + r)} \tag{10}$$

where $l(r) \in \{0, \ldots, 4\}$ is the relevance label of the URL at rank position r and L is the truncation level to which NDCG is computed. Z is chosen such that the perfect ranking would result in $\text{NDCG@}L_q = 100$. Mean NDCG@L is the normalized sum over all queries: $\frac{1}{N} \sum_{q=1}^{N} \text{NDCG@}L_q$. NDCG is particularly well-suited for Web search applications since it accounts for multilevel relevance labels and the truncation level can be set to model user behavior. In our work, relevance is measured on a 5-level scale. We evaluate our results using *mean* NDCG@1 to NDCG@5. We focus on improvements on the very top of the ranking since those are typically harder to achieve and have a more significant impact on the user. In particular, we want to be certain that class information will pull relevant information not simply into the top 10 but to a level that will be felt by the user. We also perform significance tests, using a paired t-test with a significance level of 0.05. A significant difference should be read as significant at the 95% level.

4.5 Experimental Methodology

For each experiment, we train 2-layer LambdaRank models on the training set and choose the best epoch, learning rate, and number of hidden nodes based on the validation set. We find consistently that a learning rate of 10^{-5} and 15 hidden nodes performs best. The chosen epoch varies across experiments but is typically between 200 and 300 epochs.

	Baseline	+Class Features	Delta
NDCG@1	50.10	**50.22**	+0.13
NDCG@2	47.52	**48.01**	**+0.49**
NDCG@3	46.63	**47.16**	**+0.53**
NDCG@4	46.27	**46.93**	**+0.66**
NDCG@5	46.26	**46.91**	**+0.65**

Table 2: Overall results. Best performance in each row is in bold. Significant differences are denoted using bold in the delta column.

	Baseline	+Class Features	Delta
Overall	46.26	**46.91**	**+0.65**
Head	57.71	**58.55**	**+0.83**
Tail	34.42	**34.88**	**+0.47**
Short	49.30	**50.08**	**+0.78**
Long	37.67	**37.98**	**+0.31**

Table 3: Overview of NDCG@5 overall and broken down over subsets. Best performance in each row is in bold. Significant differences are denoted using bold in the delta column.

Similarly, we choose the optimal number of leaves and epochs for LambdaMART based on the validation set. We find 10–15 leaves works well, with a shrinkage of 10^{-2}, and roughly 500–1000 epochs.

In Equation 3, m is set to 0.04. We have not found performance to be particularly sensitive to changes in this parameter. The primary function of this parameter is simply to provide some smoothing for queries that have little or no click information.

We focus on the performance of two models – one without category information and one with it. The *Baseline* model uses a large number of query, URL, and query-URL features to learn a very competitive state-of-the art ranker. The *+Class Features* ranking model is learned using those same features plus the Class Features from Table 1.

4.6 Results & Discussion

First, in examining NDCG at the top 5 positions in the overall test set (Table 2), the class-based features yield statistically significant improvements in all cases except NDCG@1. Furthermore, these gains increase steadily as one goes down in the ranking until finally leveling off. This suggests that part of the power of this approach is based on leveraging some information regarding the class information of top-ranked clicked results to pull additional relevant results higher.

The fact that any gains occur at NDCG@1 hints that clicks occurring on relevant classes but not the top ranked URL might indeed be a source of relevance information available in the logs. If so, richer models may be able to yield significant gains in this position.

More surprising is that in Table 3, where the performance is broken down into the head/tail and short/long splits described above, significant improvements are gained in every cell. Since click information is important in our approach, we expected that the head and short queries (both of which have large amounts of click data) would have gains larger than the average, and indeed we find this to be the case. Although the tail and long queries have gains that are less than the average, the improvement seen even for these difficult queries is significantly better than the baseline. Techniques that can further improve the coverage of the query classification model are most promising. This might include more sophisticated methods of smoothing that backoff to something other than a uniform prior. For example, backing off to click-evidence that uses partial query matches may be preferable and give broader coverage.

	Feature	Score
Most Important	QueryClassEntropy	0.1108
↓	KLDistance1	0.0970
	KLDistance2	0.0767
	KLDistance	0.0599
	CrossEntropy2	0.0568
	CrossEntropy	0.0441
	ArgMaxOdds	0.0322
	URLClassEntropy	0.0287
	MaxOdds	0.0286
	CrossEntropy1	0.0226
	ArgMaxOdds2	0.0213
	MaxOdds2	0.0196
	MaxOdds1	0.0185
	ArgMaxOdds1	0.0184
	QueryClassURLMatch	0.0091
	QUClassNoMatch	0.0076
Least Important	QUClassMatch	0.0068

Table 4: Feature Importance

Finally, we note that we were able to achieve these gains using only 2-bits of confidence per class prediction. We feel this is due in part to the robustness of the model. By deriving the query class directly from the classes of the results, our method is tolerant of errors in the underlying text classification. That is, suppose all URLs about "football" are classified as "biology". As long as queries rarely tend to retrieve true "biology" results together with "football", the model can still yield improvements. All that matters is that the classes separate an intent of a query from its other intents.

4.6.1 Feature Importance Analysis

In order to understand the relative importance of the different features to prioritize them for future study, we used LambdaMART to obtain feature importance scores since it tends to be more robust for this type of analysis. As mentioned in Section 4.2.2, feature importance is determined by adding the contribution of the feature across all splits and all trees in the model, based on the decrease in mean-squared error when applying the given feature and split value. We present the results of the feature importance analysis in Table 4.

It is interesting to note that *QueryClassEntropy* has by far the largest importance score. This feature has been shown to be a measure of query ambiguity [9], and its importance here implies that different ranking functions may be called for when a query has many intents versus when it has a few.

This score is followed by the features that capture the match of the entire distributions and coverage of the document's topics to aspects of the query. Finally, we see that in general the features based on information theory dominate the top positions while the simple matching features seem to have less impact.

4.6.2 Oracle Study

We are interested in understanding the extent to which the performance we have observed in our experiments could be increased by improving the accuracy of specific components of the process. Of particular interest is the query classification component. In order to estimate this, we conducted an oracle study designed to discover how much improved *query classification* could improve the relevance of results. We used the same URL classifications, but we derived the query class distribution by reducing the candidate set for query class estimation to only those documents rated in the top three levels of relevance on our 5 point scale. This is a direct implementation of the motivation discussed in Section 3.1, *i.e.*, the ideal query class distribution is the aggregated distribution of relevant results. We then learned a model using this "ideal" query class distribution with the same category-based features as described here. When few results are marked as relevant, this may be an overly optimistic upper-bound. For example, for a single relevant result, this "ideal" distribution can essentially encode the identity of the result. To avoid this, we limited ourselves to cases where there were more than five URLs meeting the relevance criterion for a query. This condition still allows us to use significantly more than half of the queries in our test set in this analysis. Over these queries, the model based on the ideal query class distribution yielded a 3.5 point NDCG@5 gain over the baseline.

This result is important in several ways. First, it indicates that even given this particular taxonomy, significant room is still available for improvement given better query classification. Thus, improving query classification remains a significant research challenge that is likely to yield further improved relevance. Furthermore, we believe that this kind of oracle experiment is more broadly applicable. That is, the relevance judgments that are available in existing test collections can be leveraged to perform similar oracle experiments. These existing judgments can be used in the same manner we have described above to provide ground truth for query classification experiments. Such experiments could be used to evaluate the influence of query classification on search ranking, and in particular, assuming a similar positive outcome, create a dataset where increased query classification accuracy is known to impact ranking relevance.

5. FUTURE WORK

There are a variety of avenues for future work. As mentioned, the method for computing the URL relevance weighting was simple and more sophisticated methods (*e.g.*, cascade models [10] and related ones [8, 15] that can account for positional biases more accurately) might be able to improve upon the estimation. Furthermore, it may not be the case that click information should be used in the same way when incorporated directly into ranking as a feature and when used to derive a weighting for class data. For example, clicking on a result and going back to the results page to click on a different result may carry positive weight on the first result's class for deriving class distributions but not when directly inferring relevance. Thus, it might be possible to loosen some assumptions used when inferring relevance from click data to gain additional inference power for query classification.

Another interesting possibility is to explore the use of negative data. For example, the classes of URLs that are skipped may provide negative information about what a user is *not* seeking and this can be used to downweight certain intents of the query.

Because the framework is agnostic with respect to the classification space, studying the empirical performance using alternate choices (*e.g.* other topic hierarchies) and alternative classification spaces (*e.g.* reading level) is of interest. Furthermore, understanding both the tradeoffs of depth of classification into a topic hierarchy as well as optimal choices of hierarchies as they relate to click behavior and confusable query intents is an open and challenging problem.

6. SUMMARY

In this work we demonstrated that topical class information can be used to improve the relevance of retrieval results by generalizing along the class dimension to identify other relevant results. In order to do this, we introduce a natural definition of query class that stems from the results that are relevant for that query and can be estimated using click-behavior. Our approach is notable for its focus on directly improving ranking relevance rather than indirect measures like query classification. Furthermore, it scales well to large search engines. We use the class distributions we estimate to compute several information-theory inspired measures that capture query ambiguity, URL specificity, and the match between query and URLs classes. These features based on the query and URL class distributions not only yielded significant improvements overall but also yield improvements over tail and long queries where click data tends to be sparse. Future work that broadens the coverage of the click-based weighting by considering related queries as well as improved models of classification and weighting hold great promise.

7. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR '06*, pages 19–26, 2006.

[2] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD '00*, pages 407 – 416, 2000.

[3] S. M. Beitzel, E. C. Jensen, D. D. Lewis, A. Chowdhury, and O. Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Transactions on Information Systems*, 25(2), 2007.

[4] A. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR '07*, pages 231–238, 2007.

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

[6] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS '06*, pages 193–200, 2007. See also MSR Technical Report MSR-TR-2006-60.

[7] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang. Context-aware query classification. In *SIGIR '09*, pages 3–10, 2009.

[8] O. Chapelle and Y. Zhang. A dynamic bayesian network click model for web search ranking. In *WWW '09*, pages 1–10, 2009.

[9] K. Collins-Thompson and P. N. Bennett. Estimating query performance using class predictions. In *SIGIR '09 as a Poster-Paper*, pages 672–673, 2009.

[10] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM '08*, pages 87–94, 2008.

[11] P. Donmez, K. Svore, and C. Burges. On the local optimality of LambdaRank. In *SIGIR '09*, pages 460–467, 2009.

[12] S. T. Dumais, E. Cutrell, and H. Chen. Optimizing search by showing results in context. In *CHI '01*, pages 277–284, 2001.

[13] E. Gabrilovich, A. Broder, M. Fontoura, A. Joshi, V. Josifovski, L. Riedel, and T. Zhang. Classifying search queries using the web as a source of knowledge. *ACM Transactions on the Web*, 3(2), 2009.

[14] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie. Smoothing clickthrough data for web search ranking. In *SIGIR '09*, pages 355–362, 2009.

[15] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW '09*, pages 11–20, 2009.

[16] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR '00*, pages 41–48, 2000.

[17] Z. Kardkovacs, D. Tikk, and Z. Bansaghi. The ferrety algorithm for the KDD Cup 2005 problem. *SIGKDD Explorations*, 7(2):111–116, 2005.

[18] Y. Li, Z. Zheng, and H. Dai. KDD CUP-2005 report: Facing a great challenge. *SIGKDD Explorations*, 7(2):91–99, 2005.

[19] T. M. Mitchell. *Machine Learning*. McGraw-Hill Companies, Inc., 1997.

[20] Netscape Communication Corporation. Open directory project. http://www.dmoz.org.

[21] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval Journal*, 2010. DOI: 10.1007/s10791-009-9123-y.

[22] S. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *SIGIR '94*, pages 232 – 241, 1994.

[23] D. E. Rose and D. Levinson. Understanding user goals in web search. In *WWW '04*, pages 13–19, 2004.

[24] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06*, pages 377–386, 2006.

[25] D. Shen, R. Pan, J. Sun, J. Pan, K. Wu, and J. Yin. Q2c@ust: Our winning solution to query classification in KDDCUP 2005. *SIGKDD Explorations*, 7(2):100–110, 2005.

[26] D. Shen, J. Sun, Q. Yang, and Z. Chen. Building bridges for web query classification. In *SIGIR '06*, pages 131–138, 2006.

[27] D. Vogel, S. Bickel, P. Haider, R. Shimpfky, and P. Siemen. Classifying search engine queries using the web as background knowledge. *SIGKDD Explorations*, 7(2):117–122, 2005.

[28] Q. Wu, C. Burges, K. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Journal of Information Retrieval*, 2009. DOI 10.1007/s10791-009-9112-1.

[29] Y. Yue and C. Burges. On using simultaneous perturbation stochastic approximation for IR measures, and the empirical optimality of LambdaRank. *NIPS '07 Machine Learning for Web Search Workshop*, 2007.