Ranking Relevance in Yahoo Search

Dawei Yin[†], Yuening Hu[†], Jiliang Tang[‡], Tim Daly Jr., Mianwei Zhou, Hua Ouyang, Jianhui Chen,
Changsung Kang, Hongbo Deng, Chikashi Nobata, Jean-Marc Langlois, Yi Chang[†]
[†]Relevance Science, Yahoo! Inc.; [‡] Computer Science and Engineering, Michigan State University
[†]{daweiy,ynhu,yichang}@yahoo-inc.com; [‡]jiliang.tang@cse.msu.edu

ABSTRACT

Search engines play a crucial role in our daily lives. Relevance is the core problem of a commercial search engine. It has attracted thousands of researchers from both academia and industry and has been studied for decades. Relevance in a modern search engine has gone far beyond text matching, and now involves tremendous challenges. The semantic gap between queries and URLs is the main barrier for improving base relevance. Clicks help provide hints to improve relevance, but unfortunately for most tail queries, the click information is too sparse, noisy, or missing entirely. For comprehensive relevance, the recency and location sensitivity of results is also critical.

In this paper, we give an overview of the solutions for relevance in the Yahoo search engine. We introduce three key techniques for base relevance – ranking functions, semantic matching features and query rewriting. We also describe solutions for recency sensitive and location sensitive relevance. This work builds upon 20 years of existing efforts on Yahoo search, summarizes the most recent advances and provides a series of practical relevance solutions. The reported performance is based on Yahoo's commercial search engine, where billions of URLs are indexed and served by the ranking system.

Keywords

learning to rank; query rewriting; semantic matching; deep learning

1. INTRODUCTION

The continuing growth of the web consistently expands the information pool available to us and we must increasingly rely on search engines to find useful web documents. Indeed, search engines play an ever more crucial role in information consumption in our daily lives. Meanwhile, search engines are one of the most successful businesses in industry, where the sponsored search model is a predominant form of online advertising.

Ranking relevance has been the most critical problem since the birth of web search. As web content was explosively generated, modern search engines have been required to efficiently and effectively retrieve the relevant URLs from a prohibitively large corpus. This raises tremendous challenges for both industrial and academic researchers. Early works on search relevance concentrated on text matching between queries and web documents such as BM25 [29], probabilistic retrieval model [22], and vector space model [22]. Subsequently, user behavior demonstrated great potential for relevance improvement in the industrial setting, and user behavior modeling has been extensively explored for improving search relevance, e.g., click modeling [9, 20, 19, 32, 27].

The advancing state of the art for search engines presents new relevance challenges, which drives us to think beyond direct text matching and click modeling. First, the semantic gap between queries and web documents is a major barrier for improving base relevance [24]. Query language is usually different from document language. For instance, for the query "how much tesla", relevant documents may contain the term "price" rather than "how much", so direct text matching methods do not work well. Second, search queries follow a long tail distribution and the majority of queries are tail queries that have very low frequency or are completely new to the search engine. As a result, user behavioral information (e.g., clicks) is often not available so click modeling is not applicable to these queries. Third, users tend to treat search engines as general Q&A systems where they can locate useful information directly; thus an increasing number of queries are in the style of natural language, which presents obstacles to many aspects of search technology such as query understanding and semantic matching.

In addition to base relevance, comprehensive relevance includes both temporal and spatial dimensions. First, many queries are best served with up-to-date information, requiring search engines to take freshness into account. For example, for the query "safest cars", users prefer information about the latest tests for recent car models. It would be a poor user experience if a search engine returned a page stating "the safest car is the 1998 toyota celica". Second, with the increasing use of mobile search, more queries are location sensitive, such as "walmart" and "restaurant", requiring search engines to consider location information in relevance.

This paper focuses on sharing our experiences in tackling these relevance issues and introduces a series of practical techniques that have been successfully implemented and deployed to power the Yahoo search engine. We first revisit the state of the art ranking functions on both offline static evaluation data and the Yahoo search engine. We reveal the key factor leading to the performance difference of ranking functions in these two scenarios, which motivates us to design an effective ranking function for a real web-scale search engine. Second, to bridge the semantic gap between queries and web documents, we take full advantage of users' behavior data in various ways, and develop three types of semantic matching features to transfer click information from head queries to tail queries. Finally, we present a rewriting system that remarkably boosts rel-

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939677

evance performance, especially for tail queries. The three components together significantly enhance the Yahoo search engine. In addition, we provide solutions for recency sensitive relevance and location sensitive relevance. The key contributions include,

- Designing a novel learning to rank algorithm for core ranking and a framework of contextual reranking algorithms.
- Developing semantic matching features including click similarity, deep semantic matching, and translated text matching.
- Building an innovative framework to understand user queries with query rewriting and its ranking strategy.
- Proposing solutions to recency sensitive ranking and location sensitive ranking.

2. BACKGROUND

2.1 Overview of Architecture

The Yahoo search engine can retrieve the most relevant documents from a corpus of billions within a fraction of a second. It achieves this by (1) parallelizing the work for one query across many servers; and (2) repeatedly taking the best candidate documents from a cheaper ranking function and reranking them using a better one. The corpus is segmented into equal-sized shards that are served by index servers. Documents are assigned to a shard based upon the MD5 of their URL. Each query is processed by one complete set of shards, each of which returns its best candidate documents. These are then merged and reranked to produce the final result set.

Within a shard, the first step is to find all documents that match the query. This is called *recall*. Typically, document sets for each query term are intersected to obtain documents containing all terms. These documents are then sorted by the *first round*, a lightweight function that scores each document. Here *uniquing* is then applied, enforcing simple diversity constraints such as a limit on the number of documents from a single host or domain. In addition, query-dependent features are extracted from the top candidates, which are then ranked using a much more expensive *second round* function, also named *Core Ranking Function* [34].

To build a core ranking function, the model performance strongly depends on the distribution and the number of labeled samples in the training set. Obtaining labeled samples is very expensive and time-consuming, and active learning plays an important role in selecting informative and representative samples with limited budget as suggested by previous works [21].

2.2 Ranking Features

There are several major groups of features in traditional search engines, which, when taken together, comprise thousands of features [6]. The ranking functions are built on top of these features. **Web graph**: This type of feature tries to determine the quality or the popularity of a document based on its connectivity in the web graph. A famous example is PageRank [26]. Other features include distance or propagation of a score from known good or bad documents [12, 18]. **Document statistics**: These features compute some basic statistics of the document such as the number of words in various fields. **Document classifier**: Various classifiers are applied to the document, such as spam, adult, language, main topic, quality, type of page (e.g., navigational destination vs informational). **Query Features**: which help characterize the query types: number of terms, frequency of the query and of its terms, click-through rate of the query. **Text match**: Basic text matching features are computed from different sections of the document (title, body, abstract, keywords) as well as from the anchor text and the URL. These features are then aggregated to form new composite features. The match score can be as simple as a count or can be more complex such as BM25 [29]. There are also proximity features which try to quantify how far in the document are the query terms (the closer the better) [23]. **Topical matching**: This type of features tries to go beyond similarity at the word level and compute similarity at the topic level. In the context of contextual advertising, details can be found in [2]. **Click**: These features aim to incorporate user feedback, most importantly the clicked results [1]. They are derived either from the search or the toolbar logs. For a given query and document, different click probabilities can be computed: probability of click, first click, last click, long dwell time click or only click. **Time** For time-sensitive queries, the freshness of a page is important. There are several features which measure the age of a document as well as the one of its inlinks and outlinks. More information on such features can be found in [7].

2.3 Evaluation of Search Relevance

There are several ways to evaluate search results, including human labeling (e.g., professional editor's judgment) and user behavioral metrics (e.g., click-through rate, query reformulation rate, dwell time). Usually, user behavioral metrics are complicated, since they may be affected by other factors, such as presentation (e.g., fonts, bold, size, abstract summary etc.) and result diversity. In this paper, we focus on base relevance, which is the most important aspect of search results. Good, relevant URLs are a necessary foundation for impacting behavioral metrics.

To assess base relevance, we leverage professional editors' judgment. Editors manually judge each query-URL pair, assigning one of five grades: Perfect, Excellent, Good, Fair or Bad. We then use Discounted Cumulative Gain (DCG) as the metric to evaluate the search relevance performance. DCG has been widely used to assess relevance in the context of search engines [15]. For a ranked list of N documents, we use the following variation of DCG,

$$\mathrm{DCG}_N = \sum_{i=1}^{N} \frac{G_i}{\log_2(i+1)}$$

where G_i represents the weight assigned to the label of the document at position i. Higher degrees of relevance correspond to higher values of the weight. We use the symbol DCG to indicate the average of this value over a set of test queries in our experiments. DCG will be reported only when absolute relevance judgments are available. In the following sections, we will report DCG1, DCG3, DCG5 with N in $\{1,3,5\}$, respectively. For significance, we employ the Wilcoxon T-test to report p-value.

We sample 2,000 queries from a one year query log as our test queries and evaluate their search results in the Yahoo search engine by capturing the results (query-URL pairs) real time and asking editors to judge them. According to the popularity of the query, the evaluation query set is split into three parts: top, torso and tail. Top queries have very high impressions and, in terms of relevance, they are relatively easy, since lots of click information provides hints for ranking. For torso queries, the click information is very limited and sparse because torso queries occur only a few times in a year. Tail queries generally recur less than once a year, so click information is not available. Tail and torso queries are relatively hard queries for search relevance [8]. Lots of efforts on tail and torso queries have been made [33, 35, 14]. In this paper, we mainly focus on the performance of torso and tail queries and introduce the practical solutions in Yahoo's web search engine, where we have taken implementation and efficiency into consideration.

3. MACHINE LEARNED RANKING

In a realistic scenario such as a commercial search engine, given a query, there is a seeming infinity of irrelevant documents that may dominate the relevant results. However, it is infeasible to use only negatives to train the model, and negative results in training data usually cannot cover all aspects of irrelevancy. As a result, at run time, the percentage of bad results is often higher than we expect. The percentage of bad results is also very important – when users see embarrassing results at top positions, they may abandon the current search engine and switch to its competitors.

Search can be treated as a binary problem. In our experiments, we observe that gradient boosting trees (GBDT) with logistic loss usually are able to reduce the bad URLs at top positions. It finds the decision boundary between relevant and irrelevant URLs for a given query. However, this binary classification cannot rank URLs perfectly. We here introduce a unified method for web search that is based on logistic loss and incorporates the Perfect, Excellent and Good information into the model though scaling the gradient for GBDT. Online evaluation of the Yahoo search engine shows that this framework decreases the percentage of bad URLs by 40% and increases relevance by 5% in terms of DCG5, compared to the leading ranking algorithms [34, 4].

3.1 Core Ranking

We adopt GBDT (Gradient Boosting Decision Tree) [10] into the framework. Based on the gradient boosting framework, we first introduce logistic loss at the core ranking where we aim to reduce bad/fair URLs in top results. We first flat the labels "Perfect", "Excellent" and "Good" to "Positive" (+1) and "Fair", "Bad" to "Negative" (-1). The log likelihood (loss) is

$$L(y, F) = \log(1 + \exp(-yF)), \quad y \in \{1, -1\} \tag{1}$$

Then the pseudo-response of stage m is

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{2}$$

$$= y_i/(1 + \exp(y_i F_{m-1}(\mathbf{x}_i)) \tag{3}$$

Logistic loss targets on binary classification, which can decrease the number of bad URLs for top retrieved results. Logistic loss is not limited to classification, and it can provide reasonable ranking, comparing to other classification loss such as hinge loss since it always places the force on positive/negative towards positive/negative infinite. For the samples far from the decision boundary (e.g., 'Perfect" samples are usually far apart from decision boundary), it is highly likely that the predicted values have larger absolute values (very positive/negative). However, only simple binary classification is insufficient for ranking. To bring the orders of "Perfect", "Excellent" and "Good" back to the model, we introduce gradient scale for GBDT. According to the gradient of logistic loss, given a Perfect/Excellent/Good sample, we know that the gradient is always positive, while for Fair/Bad URLs, the gradient of logistic loss is always negative. To distinguish Perfect/Excellent/Good, we scale the gradient, (which is also known as the pseudo-response of stage i), in different levels (e.g., 3 for Perfect, 2 for Excellent, and 1 for Good). As a result, Perfect data samples get relatively higher forces to positive infinite than Excellent ones, which are higher than the Good ones. The pseudo-response of stage m will become

$$\text{pseudo\_response}(x) = -g_m(\mathbf{x}_i) \times \text{scale}(label)$$

where $\text{scale}(label)$ can be empirically set $\text{scale}(Perfect) = 3$, $\text{scale}(Excellent) = 2$ and $\text{scale}(Good/Fair/Bad) = 1$ to distinguish Perfect/Excellent/Good. This loss function takes advantage of both gradient boosting and logistic loss. It not only is a binary classification but also has ability to distinguish the different positive samples. For Fair and Bad samples, since their gradients are always negative, we needn't make any modification/scale for each pseudo-response. This method is actually equivalent to adding different weights on loss for different labels, but it is different from adding sample weights, which actually affect the tree growth of each stage m but do not change pseudo-response.

query	methods	DCG1	DCG3	DCG5
all	LogisticRank	4.31	7.74	9.78
	GBRank	4.26 (-1.19%)	7.52 (-2.81%)*	9.51 (-2.81%)*
	LambdaMart	4.24 (-1.60%)	7.36 (-4.84%)*	9.21 (-5.83%)*
top	LogisticRank	5.69	9.67	12.08
	GBRank	5.56 (-2.22%)	9.25 (-4.29%)*	11.51 (-4.67%)*
	LambdaMart	5.59 (-1.72%)	9.08 (-6.04%)*	11.02 (-8.75%)*
torso	LogisticRank	3.88	7.23	9.26
	GBRank	3.88 (-1.77%)	7.065 (-2.30%)*	9.08 (-2.03%)*
	LambdaMart	3.81 (-1.88%)	6.97 (-3.64%)†	8.92 (-3.64%)*
tail	LogisticRank	2.91	5.65	7.16
	GBRank	2.99 (3.06%)	5.65 (0.01%)	7.19 (0.37%)
	LambdaMart	2.88 (-0.71%)	5.42 (-4.15%)†	6.91 (-2.78%)†

Table 1: Performance comparison of models using different learning algorithms. * denotes p-value<=0.01; † denotes p-value<=0.05.

We name our learning algorithm *LogisticRank* and compare with the leading ranking algorithms: *GBRank* [34] and *LambdaMart* [4] in the Yahoo search engine. All three methods share the same training data, which is collected through active learning and editor labels and includes about 2 million query-URL pairs. The parameters are carefully tuned on a validation query set. The results are shown in Table 1. Interestingly, the overall performance of LogisticRank is the best and significantly better than GBRank and LambdaMart. Especially, on top queries, we obtain the largest improvement through LogisticRank. GBRank, which mixes pairwise loss and pointwise loss, performs consistently better than LambdaMart, which is a listwise loss function and only learns relative orders. In tail queries, GBRank is able to generate similar performance (no significance) as LogisticRank. On the other aspect, the percentage of bad/embarrassing results is reduced by 40% through LogisticRank. We observe that the improvement of LogisticRank is mainly from bad results removal.

We tried to further understand the above results by performing experiments on an offline dataset—the learning to rank challenge data set (LTR) [6]. The results of LambdaMart and GBRank are consistent with [6], where LambdaMart is around 2% better than GBRank in terms of DCG5. However, we find that both LambdaMart and GBRank are better than LogisticRank on this task: LambdaMart is 2.6% better than LogisticRank while GBRank is 0.6% better than LogisticRank in terms of DCG5. The main difference between LTR data and a real search engine is the data distribution. In static LTR data, there are 27% "Bad" query-URL pairs, compared to >99% "Bad" query-URL pairs in the real search engine, since for a given query, there are usually only tens of relevant URLs among tens of billions of indexed URLs. The performance on the static LTR data set is not matched by real search engine performance due to this big difference in data distribution. Unfortunately, it is prohibitively expensive to generate a static data set with editorial labels that has the same distribution as the real search engine.

In the remaining sections of this paper, we use LogisticRank with the existing features described in Section 2 as our base model, which is referred to as 'base'.

3.2 Contextual Reranking

query	methods	DCG1	DCG3	DCG5
all	base	4.31	7.74	9.78
	base+reranking	4.34 (0.79%)	7.78 (0.52%)	9.91 (1.30%)*
top	base	5.69	9.67	12.08
	base+reranking	5.72 (0.60%)	9.75 (0.88%)	12.24 (1.33%)†
torso	base	3.88	7.23	9.26
	base+reranking	3.88 (-0.18%)	7.27 (0.49%)	9.38 (1.23%)
tail	base	2.91	5.65	7.16
	base+reranking	3.00 (3.40%)	5.63 (-0.34%)	7.26 (1.38%)

Table 2: Performance of systems with and without contextual reranking. * denotes p-value<=0.01; † denotes p-value<=0.05.

The core ranking only considers query-URL pair features, while ignoring contextual information from other candidate results for the same query. There are practical reasons – the result set at this point is very large, and it is distributed over hundreds of machines, hence calculating even a very simple feature like the average popularity of all candidate documents would be prohibitively slow.

The reranking phase is applied after the results from the core-ranking phase have been sorted, aggregated on one machine, and trimmed down to the best tens of results, e.g., 30. That allows us to extract features capturing contextual information about the entire result set, which turn out to be the most important features in the model.

Based on the top tens of results (e.g., 30) of a given query, we extract following contextual features for specific existing features – (1) *Rank*: sorting URLs by the feature value in ascending order to get the ranks of specific URLs. (2) *Mean*: calculating the mean of the feature values of top 30 URLs. (3) *Variance*: calculating the variance of the feature values of top 30 URLs. (4) *Normalized feature*: normalizing the feature by using mean and standard deviation. (5) *Topic model feature*: aggregating the topical distributions of 30 URLs to create a query topic model vector, and calculating similarity with each individual result.

Beside contextual features, another advantage of reranking is that the training data for this phase is less prone to sample selection bias, since it is possible to label a complete set of candidate documents for a given query. By contrast, for the core ranking phase, this is infeasible since the candidate set could consist of hundreds of thousands of documents. In practice, we found that it is more robust to use the ranks of the results as a feature instead of directly using the core-ranking function's scores. Though the core-ranking phase score is more informative than the rank, its value can drift as the index refreshes, degrading third phase performance. Since the reranking function is running on top 30 results, its main purpose is to further distinguish relevant results ("Perfect", "Excellent", "Good") rather than to identify and remove bad results. The impact is shown in Table 2. We see that the overall performance is significantly and consistently improved by reranking.

3.3 Implementation and deployment

The core ranking function is run on the index serving nodes. This allows it to run in parallel on billions of documents, and to have local access to posting lists and document properties for quick feature generation. By contrast, the third phase ranking function must be run on a single *blending node*, after merging and sorting the top results from each index serving node, in order to calculate the contextual features. To support this, we have added the ability to *export* features from the index serving nodes. When processing the query, we augment it with the list of primitive features that are used to calculate contextual features. Each index serving node includes the requested features of its top-k documents in its response. This approach is feasible because the reranking phase also has access to the rank and ranking score of each document in core-ranking phase, so it typically requires only between 30 and 50 primitive features, keeping communication overhead to a minimum.

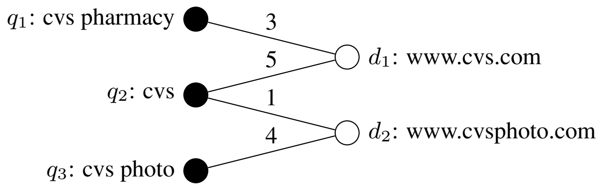

Figure 1: An example of click-through bipartite graph.

4. SEMANTIC MATCHING FEATURES

The features described in Section 2 have some failings. For tail queries, the user behavior features do not work well due to sparsity and noise. The documents relevant to tail queries are often lacking anchor text. Text matching features suffer from the vocabulary gap between queries and documents [24]. To overcome these issues, we introduce three novel features: click similarity, translated text matching and deep semantic matching, which are calculated from click logs in different ways and complement each other.

4.1 Click Similarity

From the click log, we can extract a bipartite click graph to connect queries and documents by gross judgments from users. A vector propagation framework is developed on this graph to learn vector representations for both queries and documents in a shared space. Based on these vectors, relevance scores can be calculated. Traditional content-based models such as VSM [31] represent queries and documents in a vector space by merging their vocabularies. One weakness of such models is the mismatch between query vocabulary and document vocabulary [24]. To bridge this gap, the proposed model represents both queries and documents using terms in query vocabulary.

Edges in the bipartite click graph link queries and their correspondingly clicked documents. They are weighted by the number of clicks since more clicks could imply more confidence in potential relevance. We extract terms from co-clicked queries to represent documents. Terms that are from more co-clicked queries are likely to be more representative of documents. In such a space, the more queries two documents share, the "closer" they are. Similarly, queries that share many co-clicked documents are likely to share similar intent that should also be reflected by the query vectors. Thus we in turn propagate terms in document vectors to their co-clicked queries so that related queries can share high similarity in this vector space as desired.

With aforementioned assumptions, a vector propagation algorithm is proposed in this section. We first construct the bipartite click graph $\mathcal{G}$ with nodes $\mathcal{V} = \mathcal{D} \cup \mathcal{Q}$, where $\mathcal{D}$ and $\mathcal{Q}$ are the sets of documents and queries, respectively. For a document $d \in \mathcal{D}$ and a query $q \in \mathcal{Q}$, an edge is formed between them if there are co-clicks between them. The weight of the edge is indicated by the number of co-clicks. We use $\mathcal{E}$ to denote the set of edges and C to represent the adjacency matrix of $\mathcal{G}$. An example of the bipartite click graph is shown in Figure 1. QV^n is a $|\mathcal{Q}| \times V$ matrix where the i-th row QV_i^n is the query vector of q_i at iteration n and V is the vocabulary size. Likewise, DV is a $|\mathcal{D}| \times V$ matrix where each row is a document vector.

We initialize the representation of each query by its terms with weights proportional to their frequencies in the query. Each vector is normalized to 1. The initial matrix for the queries is QV^0. In the n-th iteration, we compute document vectors DV^n by aggregating their co-clicked queries. Formally, DV_j^n of d_j is calculated as:

$$DV_j^n = \frac{1}{||\sum_{i=1}^{|\mathcal{Q}|} C_{ij} \cdot QV_i^n||_2} \sum_{i=1}^{|\mathcal{Q}|} C_{i,j} \cdot QV_i^n \qquad (4)$$

where DV_j^n is normalized by ℓ_2 norm. Then the query vectors in QV^{n+1} are updated with their co-clicked document vectors:

$$QV_i^{n+1} = \frac{1}{||\sum_{j}^{|\mathcal{D}|} C_{i,j} \cdot DV_j^n||_2} \sum_{j=1}^{|\mathcal{D}|} C_{i,j} \cdot DV_j^n \qquad (5)$$

During the propagation process, queries are enriched with new and relevant terms. The propagation also re-learns the weights to indicate term importance that helps distinguish the informative terms, and is especially helpful for short queries. Document vectors are enriched with terms in queries and adjusted in a similar manner. In this way, query and document vectors mutually enhance each other towards a better representation that reveals more reliable query-document relevance. More details about the proposed framework can be found in [16].

In practice, we use a sparse representation by only considering terms with non-zero weights. The set of non-zero terms could be still quite large. If we use their weights to rank the terms in a vector, we can observe – the first few terms have large weights and the rest are very small. We find that we can keep the top K terms with the highest weight for both query and document vectors in each iteration without losing much accuracy. This optimization is vital for very large click logs. In addition, we have noted that this simple trimming speeds up the convergence of the vectors. Both query and document vectors stay stable after a few iterations.

Evaluation Given the learned query and document vectors in the same space, we can simply compute the inner product as a click similarity feature between queries and documents, which is referred as **CS** and evaluated in a real search engine. As shown in Table 3, through **CS** feature, the overall performance obtains 2.87% relative DCG5 gain, comparing with the base model (LogisticRank with the existing features described in Sec 2). For all query bands (top, torso and tail), CS significantly and consistently improves the performance, especially on the torso and tail queries where 4.18% and 5.14% DCG5 gain respectively. This improvement shows that this vector propagation algorithm successfully reduces the vocabulary gap between queries and documents, and also introduces related terms to learn better representations for both queries and documents. We also study the importance of features and the proposed **CS** feature is ranked as the *top one* feature among thousands.

Implementation and deployment We have created a pipeline that refreshes the query and document vectors monthly to include the latest click log data. This pipeline automatically constructs the click graph from the click log and generates the query and document vectors, which are further quantized to bytes to save space. The quantized document vectors are stored in the forward index on the index serving nodes, and the quantized query vectors are maintained in a table on a blending node. At run time, the similarity between the query and the document is computed as a ranking feature, which is used in the ranking models.

query	methods	DCG1	DCG3	DCG5
all	base	4.31	7.74	9.78
	base+CS	4.39 (1.97%)†	7.91 (2.18%)*	10.06 (2.87%)*
	base+TTM	4.36 (1.31%)†	7.84 (1.36%)*	9.96 (1.79%)*
	base+DSM	4.37 (1.46%)†	7.87 (1.77%)*	9.95 (1.69%)*
top	base	5.69	9.67	12.08
	base+CS	5.66 (-0.40%)	9.75 (0.84%)†	12.20 (0.98%)†
	base+TTM	5.68 (-0.01%)	9.86 (1.99%)*	12.34 (2.14%)*
	base+DSM	5.70 (0.31%)	9.86 (2.01%)*	12.31 (1.94%)*
torso	base	3.88	7.23	9.26
	base+CS	4.02 (3.54%)†	7.47 (3.33%)*	9.65 (4.18%)*
	base+TTM	3.98 (2.34%)†	7.35 (1.57%)†	9.41 (1.55%)*
	base+DSM	3.94 (1.33%)†	7.36 (1.84%)†	9.37 (1.18%)†
tail	base	2.91	5.65	7.16
	base+CS	3.07 (5.75%)	5.84 (3.40%)†	7.53 (5.14%)*
	base+TTM	3.00 (3.10%)	5.61 (-0.73%)	7.26 (1.40%)
	base+DSM	3.05 (5.15%)†	5.71 (1.01%)*	7.31 (2.15%)*

Table 3: Performance of models with and without new features. * denotes p-value<=0.01; † denotes p-value<=0.05.

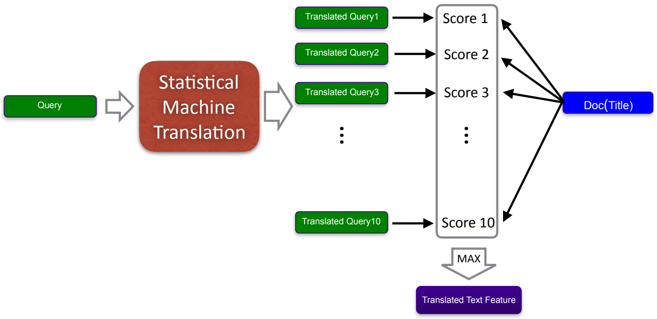

Figure 2: Translated Text Matching (TTM) feature.

4.2 Translated Text Matching

The click similarity feature, although very effective, cannot be directly calculated for queries and documents that have never been observed in the click log. Statistical machine translation—translating queries to documents or reverse—provides an intuitive method to reduce the vocabulary gap, and it can be applied to all queries and documents. We harness the power of machine translation by introducing a set of new features, named "Translated Text Matching" (TTM).

The TTM features are constructed as shown in Figure 2. We first create a translation model, using clicked (query, document title) pairs as the bitext. [1] We use this model to translate each query q into k alternate queries $\{q_w^1, \dots, q_w^k\}$, which are in document title space. Given a document d, we then compute a cosine similarity s_i between the i-th alternate query q_w^i and the document title as the similarity between q_w^i and d. The final similarity score $S(q, d)$ between the original query q and the document d is computed as: $S(q, d) = F(\{s_1, s_2, \dots, s_k\})$ where F denotes a function to aggregate scores in $\{s_1, s_2, \dots, s_k\}$ e.g., max, average, median, etc.

In our experiments, we consider various strategies to construct the translation features and we choose $k = 10$ written candidates. Due to space limits, we omit the details and only report the successful features: EXT_Q_TLM$_1$ is the maximum cosine similarity scores $\{\text{t\_qlm}_\text{i}\}_{i=1}^{10}$ between the rewritten queries and document titles, where the rewritten queries are obtained using a translation model and a language model based on document titles; AGG_Q_TLM is obtained by computing the cosine similarity between the rewritten queries (of the original query) using a title language model and the set of the 10 rewritten titles (of the document) using a query language model.

Evaluation We combine the proposed features EXT_Q_TLM$_1$ and AGG_Q_TLM with existing features described in Section 2 to train a ranking function and evaluate the test data by DCG scores. The results are reported in Table 3, which show that the proposed

[1] The details of translation model are presented in Section 5.

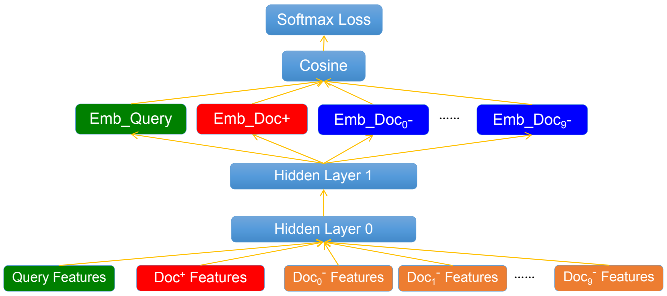

Figure 3: Network structure of the proposed model.

TTM features significantly and consistently improve the relevance performance, compared with the baseline (LogisticRank with the existing features described in Sec 2). We also evaluate the feature importance, and the proposed two features EXT_Q_TLM$_1$ and AGG_Q_TLM are ranked 7th and 10th respectively. These results show that the proposed translation features contain discriminative information for the core relevance task.

Implementation and deployment The implementation of the TTM features is based on a caching strategy which is described in more depth in Section 5. For cache misses, we use a pruned version of the translation model to keep latency low.

4.3 Deep Semantic Matching

Top queries are relatively easier in machine learning based ranking, since many text or click-through features can be used. It may face challenges for torso and tail queries since they occurred only few times in users' historic log. The aforementioned CS and TTM features to some extent mitigate the problem: the CS feature smooths the click information and removes the noise of clicks, while the TTM features are the soft version of query rewriting and record the equivalence of n-grams. However, both methods stay on the word level. To further improve the ranking results, we need to deeply understand the semantic information and users' intentions behind them. Deep learning techniques can help extract semantic and contextual information from top queries, and then we can generalize such information for torso and tail queries.

The rareness of torso and tail queries is mainly caused by misspelling, synonyms, abbreviations, long or descriptive queries. Generally speaking, tail queries tend to be substantially longer than top ones. For example, tail queries on average have four to five words. Longer queries are harder to be matched with the title or content of a web page. However, it might contain more semantic information, compared against shorter queries with one or two words.

We adopted a feed-forward neural network following the work of DSSM [13] as our ranking model. The structure of our network is illustrated in Figure 3. The queries Q and their candidate documents $D_{\pm}$ are fed as inputs, and the embedded vectors V_Q and $V_{D_{\pm}}$ are used to calculate the likelihood function of a clicked document D_+, given by the following softmax:

$$L(D_+|Q) = \frac{exp(-cos(V_Q, V_{D_+}))}{\sum_{D_{\pm}} exp(-cos(V_Q, V_{D_{\pm}}))} \tag{6}$$

For the training purpose, we collected one year of users' query logs of Yahoo's desktop search (US market), from May 1st 2014 to May 1st 2015. We removed spamming queries using simple rules, e.g., whether the number of displayed results per page is larger than 100. A query session is generated by aggregating logs that have the same key of "browser_cookie+time_stamp+view_id". We then aggregated all query sessions based on the query text. We discarded all abandoned sessions, where users did not click on any of the displayed documents. After aggregation, we obtained more than 20 billion unique query-document pairs.

To generate a list of positive and negative documents for training, we use a 10-slot window and slide it down from the top ranked document. The document in the first slot of this window is a positive D_+, and the following ones are regarded as negative D_-. As a result, we generate a large training set of 3 billion query-10-documents samples. The input feature representation is crucial in this model. For documents, we include both document titles and domain names as inputs. As discussed in section 4.2, title is the most informative and representative content about the document, but in a deep understanding model, domain provides another important signal for specific queries, e.g., wiki, weather, imdb, etc. To reduce the vocabulary size, we use 3-letter shingling of words as proposed in DSSM [13], and normalized bag-of-words vectors of dimension $30,000$ are used as input features. The neural network is then trained on a GPU cluster. After training the deep neural network, at runtime, the top level neurons will generate the embeddings of queries and URLs. Then the deep semantic matching (DSM) value is calculated by the inner product of the embeddings of queries and URLs.

Evaluation We evaluate the proposed **DSM** in our ranking framework on the Yahoo search engine. As shown in Table 3, **DSM** obtains 1.69% relative DCG5 gain. It shows significant DCG improvement on all three query bands (top, torso and tail). We also study the importance of features and the proposed **DSM** ranked as the eighth feature among thousands of features.

Implementation and deployment Deploying the deep neural network model is an engineering challenge. Latent vector embeddings of document titles and domain names are pre-computed via forward propagation and stored on index servers. In order to reduce memory footprint, we have to utilize vector quantization to further compress the features represented as vectors of single precision floats. At runtime, we only need to obtain the embedded vector V_Q for the query on the fly. The dot product between V_Q and the embedded document vector V_D is used as the deep learning feature, which is then fed to the GBDT model to calculate the final relevance scores.

5. QUERY REWRITING

Online users play two roles in commercial search engines. Users are information creators that generate web documents and they are also information consumers that retrieve documents for their information needs. Hence web documents and queries often use different language styles and vocabularies; consequently search engines could be unable to retrieve documents matching queries even when queries can perfectly describe users' information needs. For example, a user forms a query " how much tesla", but web documents in search engines use the expression "price tesla". In previous sections, semantic matching features try to solve this issue in a soft way that is mainly focused on improving precision. However, in a commercial web-scale search engine, to boost efficiency, before feeding the query-URL pairs into the ranking function, some steps, e.g., filtering, preselection, are performed. For instance, typically only URLs that contain all query terms are allowed to pass the preselection phase. This results in a recall problem. Query rewriting (QRW) is the task of altering a given query so that it will get better results and, more importantly, to help solve the recall problem. Query rewriting has been proven to be effective in improving search performance [28, 11, 25]. In this section, we will describe the techniques and deployment of QRW in the Yahoo search engine.

5.1 Methodology

We treat query rewriting as a machine translation (MT) problem which translates from a source language of user queries S into a target language of web documents T. In other words, we use QRW to bridge the language gap between queries and documents for search engines. The proposed framework consists of two phases – (1) the learning phase that learns phrase-level translations from queries to documents; and (2) the decoding phase that generates candidates for a given query. Next we detail each phase.

The Learning Phase: The immediate challenge for the learning phase is to obtain a large parallel training data with queries and the corresponding rewritten queries that improve relevance. It is difficult, if not impossible, to use human labeling for the training data because (a) a good translation model requires a prohibitively large bitext; and (b) editors are not very effective at choosing which query will improve relevance. To tackle this challenge, we make use of click graphs. A click graph is a bipartite weighted graph where queries and documents are nodes, edges indicate the co-click between queries and documents and weights are co-click numbers. Typically titles of documents are very informative and representative while queries are usually very short. Titles are more similar to queries than body texts, which suggests that we consider titles of documents as rewritten queries. These observations motivate us to extract query-title pairs from click graphs. Since it is highly possible for us to collect a click graph with millions of queries and titles, we can prepare a very large-scale parallel training data for the learning phase. With query-title pairs, we follow the common steps for a typical phrase-based machine translation framework to learn phrase-level translations – word alignment, phrase extraction and phrase scoring. Note that titles are usually longer than queries; hence we need to adjust the null-word alignment probability to relative large values (say 0.9) to filter out noisy alignments.

The Decoding Phase: Given a query q, there could be many ways to segment it into phrases, and each phrase could then have many translations, leading to hundreds of candidates. The decoding phase should generate the most confident candidate q_w. Typically each candidate q_c is scored via a linear combination of m feature functions and the decoding is formally defined as:

$$q_w = \arg\max_{q_c} \sum\nolimits_{i=1}^{m} \lambda_i h_i(q_c, q)$$

where $h_i(q_c, q)$ is the i-th feature function and λ_i controls the contribution of $h_i(q_c, q)$. λ_i can be either manually tuned or learned through a defined loss function. We here adopt widely used feature functions in traditional statistical machine translation systems including translation scores provided by the learning phase, language model of original queries, word penalty, phrase penalty and distortion. We also develop feature functions specific to the QRW problem that aim to improve search relevance performance. For each pair (q_c, q), we can build three groups of feature functions:

Query feature functions: h_1-number of words in q, h_2-number of stop words in q, h_3-language model score of the query q, h_4-query frequency of q, h_5-average length of words in q;

Rewrite query feature functions: h_6-number of words in q_c, h_7-number of stop words in q_c, h_8-language model score of the query rewrite q_c, h_9-query frequencies of the query rewrite q_c, h_{10}-average length of words in q_c and

Pair feature functions: h_{11}-Jaccard similarity of URLs shared by q and q_c in the query-URL graph, h_{12}-difference between the frequencies of the original query q and the rewrite candidate q_c, h_{13}-word-level cosine similarity between q and q_c, h_{14}-difference between the number of words between q and q_c, h_{15}-number of common words in q and q_c, h_{16}-difference of language model scores between q and q_c, h_{17}-difference between the number of stop words between q and q_c, h_{18}-difference between the average length of words in q and q_c.

Query	DCG1		DCG3		DCG5	
	QRW	QRW-RE	QRW	QRW-RE	QRW	QRW-RE
Top	+0.82%	-0.51%	+2.22%	+0.77%	+1.58%	+0.07%
Torso	+2.82%	+2.55%	+2.79%	+1.42%	+2.12%	+0.88%
Tail	+10.42%	+9.15%	+7.03%	+4.58%	+5.56%	+3.13%

Table 4: The relative DCG performance improvement for QRW.

From our empirical study, we found that h_{11}, h_{12}, h_{13} are the top three most influential feature functions.

5.2 Ranking Strategy

As mentioned earlier, it is difficult to prepare training data for QRW by human labeling. It is also challenging to assess the performance of QRW by comparing original queries and rewritten queries directly via editors. For example, it is hard for editors to reach an agreement about the quality of "how much tesla" vs. "price tesla" as queries. Intuitively a good QRW should lead to an improvement in search performance. Given the original query and rewritten query, we have two options for ranking strategies. Intuitively, we can replace the original query with the rewritten query. However, our empirical finding is that direct replacement is risky, since sometimes some poor rewrites will hurt relevance significantly. Hence we evaluate the quality of QRW in a blending mode, which is able to overcome most poor rewrites: (1) Suppose that q and q_w are the original and rewritten queries, respectively. We use q and q_w to retrieve top-N documents from a search engine separately. Assume that $\mathcal{O} = \{D_i, S_i\}_{i=1}^{N}$ is the set of documents and their corresponding scores (which is from the ranking function described in Section 3) for q and $\mathcal{R} = \{D_j^w, S_j^w\}_{j=1}^{N}$ for q_w; (2) We join $\mathcal{O}$ and $\mathcal{R}$ – if a document D appears in both, i.e., (D, S_i) and (D, S_j^w), $\max(S_i, S_j^w)$ is the ranking score of D after joining; and (3) We rank documents in the union according to their ranking scores in a descending order and choose the Top-N documents as final results for the original query q.

The DCG improvement over the set of queries is demonstrated in Table 4. Note that QRW is the blending mode while QRW-RE is using the rewritten queries to replace original queries for ranking. Both QRW and QRW-RE are able to significantly improve relevance especially for tail queries, and over 5% DCG5 gain is obtained through QRW. Since rewritten queries may change the intent of original queries, the performance of QRW-RE is worse than QRW.

Implementation and deployment The main issue that needs to be handled before deploying statistical machine translation-based QRW is the latency impact of running the decoder. To minimize latency, we cache the best translation candidate for each query after decoding it. Since the rewrites generated by a given model are constant, the cache entries only expire when a new model is deployed. In addition, we preload the cache with rewrites of the most frequent 100 million queries. For cache misses, there is still some latency impact, however, we are able to keep it within our limit by pruning the phrase table before deployment, limiting beam width in the decoder, and simply abandoning the rewrite in the few cases where it is too slow.

6. COMPREHENSIVE EXPERIMENTS

We here report on the performance of all of the proposed techniques when used together. Performance numbers are shown in Table 5. "GBRank" is the GBRank loss function with the baseline

query	methods	DCG1	DCG3	DCG5
all	base	4.31	7.74	9.78
	GBRank	4.26 (-1.19%)	7.52 (-2.81%)*	9.51 (-2.81%)*
	base+feat	4.41 (2.34%)†	7.95 (2.74%)*	10.06 (2.89%)*
	base+all	4.44 (%)*	8.05 (4.09%)*	10.20 (4.25%)*
top	base	5.69	9.67	12.08
	GBRank	5.56 (-2.22%)	9.25 (-4.29%)*	11.51 (-4.67%)*
	base+feat	5.72 (0.65%)	9.82 (1.58%)†	12.22 (1.16%)
	base+all	5.73 (0.83%)	9.89 (2.32%)*	12.27 (1.62%)
torso	base	3.88	7.23	9.26
	GBRank	3.88 (-1.77%)	7.065 (-2.30%)*	9.08 (-2.03%)*
	base+feat	3.96 (1.98%)	7.47 (3.33%)*	9.59 (3.53%)*
	base+all	4.02 (3.42%)	7.58 (4.87%)*	9.75 (5.21%)*
tail	base	2.91	5.65	7.16
	GBRank	2.99 (3.06%)	5.65 (0.01%)	7.19 (0.37%)
	base+feat	3.14 (8.14%)†	5.91 (4.59%)*	7.60 (6.09%)*
	base+all	3.18 (9.48%)*	6.06 (7.20%)*	7.81 (9.09%)*

Table 5: Performance comparison of models with and without the proposed components. "feat" includes CS, TTM and DSM; "all" includes feat, query rewriting and logistic rank. * denotes p-value<=0.01; † denotes p-value<=0.05.

features, essentially a snapshot of Yahoo Search in 2010 [6]; "base" is the LogisticRank with the baseline features; "base+feat" is the LogisticRank with both baseline features and the three types of semantic matching features (CS, TTM and DSM); and "base+all" combines all techniques, including proposed ranking functions, semantic matching features and query rewriting. All relative performance numbers are calculated with respect to "base".

The combination of all new features "base+feat" significantly and consistently outperforms the baseline "base" on all the three DCG scores and all three query groups. The semantic matching features together also perform better than each individual semantic matching feature. This confirms that the three semantic matching features are complementary to each other. When we combine QRW models described in Section 5 with the new features in "base+all", the results are further improved. Comparing with "GBRank", the proposed framework improves the performance by 7% in terms of DCG5. For tail queries, it achieves 9% improvement in DCG5. As Table 6 shows, with the help of the proposed ranking function, new semantic features and the query writing, our search engine significantly improves the ranking result for query "how much tesla".

So far, we have discussed only base relevance issues, the most fundamental aspect of a commercial search engine. In the following sections, we discuss the solutions for recency sensitive ranking and location sensitive ranking in Yahoo Search.

7. RECENCY-SENSITIVE RANKING

In this section, we discuss how to improve ranking quality for recency-sensitive queries, which covers a large portion of users' daily search activities. For such queries, users need the search results to be not only relevant, but also fresh. For example, for the query "safest cars", users prefer information about latest tests for recent car models; for "superbowl", it is likely that users search for the last or next superbowl. If a search engine were to return a page that said "the next superbowl will be in 1989" or "the safest car is the 1988 toyota celica", it could lead to poor user experience.

In order to jointly optimize the relevance and freshness of ranking results for recency-sensitive queries, we define a metric called *recency-demoted relevance* to integrate both relevance and recency scores. Specifically, for each document, we characterize its recency with a five-level label:"very fresh (VF)", "fresh (F)", "slightly outdated (SO)", "stale (S)" and "non-time-sensitive (NT)", and adjust its relevance as follows,

methods	DCG5	DCR5	RDDCG5
Base Ranker	9.175	11.549	7.875
Recency Ranker	9.334 (+1.74%)	12.355 (+6.98%)	8.264 (4.94%)

Table 7: Performance comparison: recency ranker v.s. base ranker.

	VF	F	SO	S	NT
Perfect	Perfect	Perfect	Excellent	Good	Perfect
Excellent	Perfect	Excellent	Good	Fair	Excellent
Good	Good	Good	Fair	Bad	Good
Fair	Fair	Fair	Bad	Bad	Fair
Bad	Bad	Bad	Bad	Bad	Bad

As the table shows, we keep or promote a document's relevance score when it is a very fresh, fresh, or non-time-sensitive document, and demote its score when its slightly outdated or stale. Our goal is to optimize the DCG metric defined based on the relevance score after adjustment, i.e., *recency-demoted DCG*.

To build such a new ranker with distinct goal, the challenge lies in the cost of collecting training data. It is not possible to have as many recency labels as relevance labels, because the recency labels quickly go stale. We must somehow leverage both a large relevance dataset without recency labels and a small recency dataset for building the recency ranker.

We create the recency-sensitive ranker upon the base relevance ranker by learning an additive freshness component to adjust document score based on its freshness. The freshness component is trained based on the recency dataset. We use a time-sensitive classifier to decide whether the component should be added, which prevents changing the score of non-recency queries and non-time-sensitive documents. The new recency ranker $f(x)$ for query-URL feature vector x is formally defined as,

$$f(x) = \begin{cases} f_{\text{rel}}(x) + r_{\text{fresh}}(x) & \text{if } c_{\text{ts}}(x) > 0 \\ f_{\text{rel}}(x) & \text{elsewise.} \end{cases}$$

where $f_{\text{rel}}(x)$ represents the base ranker trained from general relevance labels, $r_{\text{fresh}}(x)$ denotes the freshness component and c_{ts} denotes the time-sensitivity classifier . $r_{\text{fresh}}(x)$ is added only when c_{ts} shows that x is time-sensitive query-URL pair.

Based on the equation, the problem boils down to training time-sensitivity classifier $c_{\text{ts}}(x)$ and freshness component $r_{\text{fresh}}(x)$. To train $c_{\text{ts}}(x)$, we use the recency dataset and transform the freshness labels into binary labels, i.e., "non-time-sensitive (NT)" to negative and other labels to positive and train a binary classifier. To build $r_{\text{fresh}}(x)$, we use $f_{\text{rel}}(x)$ as the base ranker, and add more trees to optimize the goal of *recency-demoted relevance* – the newly added trees become $c_{\text{ts}}(x)$.

We compare the performance of recency ranker $f(x)$ with base ranker $f_{\text{rel}}(x)$ on 500 recency-sensitive queries, which are also sampled and selected from the whole year query log. We compute three different metrics: 1) DCG to measure the result relevance; 2) DCR, which uses the same formula with DCG but replaces the relevance label with a freshness label, to measure result freshness; 3) RDDCG (Recency-demoted DCG) to measure overall quality considering both relevance and freshness.

The result in Table 7 shows consistent improvement by the recency ranker on relevance and freshness metrics for recency-sensitive queries, verifying the effectiveness of our proposed method. Table 8 compares the recency ranking example of a newsy query "holly bobo", which shows that our algorithm helps boost fresher results to the top. The deployment of recency ranking is relatively easy, since it shares the same framework with the core ranking function. Note that this recency ranker is only triggered when the query is classified as a recency-sensitive query by a run time recency clas-

ranking		GBRank	base+all
1	Title	TSLA News: How Much Does a Tesla P85D Cost? ǀ InvestorPlace	Tesla Cars: 2016 Tesla Prices, Reviews, Specs
	URL	http://investorplace.com/2014/1...	http://www.autoguide.com/new-cars/tesla/
2	Title	Here's How Much Tesla Is Worth - NASDAQ.com	New 2015 Tesla Prices w/ MSRP & Invoice
	URL	http://www.nasdaq.com/article/...	https://www.truecar.com/prices-new/tesla/
3	Title	How much Mercedes is in a Tesla? ǀ Forums ǀ Tesla Motors	2013 Model S Price Increase ǀ Tesla Motors
	URL	http://www.teslamotors.com/foru...	https://www.teslamotors.com/blog/2013-mo...

Table 6: Compare the ranking results of query "how much tesla".

ranking		baseline	recency ranking
1	Title	Murder of Holly Bobo - Wikipedia, the free encyclopedia	Murder of Holly Bobo - Wikipedia, the free encyclopedia
	URL	https://en.wikipedia.org/wiki/Murder_of_...	https://en.wikipedia.org/wiki/Murder_of_...
2	Title	Holly Bobo News, Photos and Videos - ABC News	Status hearing in Holly Bobo case... (published on 11/18/2015)
	URL	http://abcnews.go.com/topics/news/holly-...	http://wkrn.com/2015/11/18/status-hearing-...
3	Title	Man granted immunity in Holly ... (published on 02/24/2015)	Defense attorney in Holly Bobo cas... (published on 02/03/2016)
	URL	http://www.cnn.com/2015/02/24/us/holly-...	http://www.jacksonsun.com/story/news/crim...

Table 8: Compare the ranking results of query "holly bobo" with and without recency ranking.

sifier.

8. LOCATION-SENSITIVE RANKING

Web search results typically are identical for all users. However, for some queries, *contextualized* search results are more useful for users. In this section, we will discuss location sensitive ranking. For example, given the query "restaurants", users want to find the pages of restaurants that are near to their current locations. We refer to such queries that are closely related with locations as *location-sensitive queries*. Queries with specific location names (e.g., "restaurants Boston") are referred to as *explicit local queries*, and queries without locations but with location-sensitive intention (e.g., "restaurants") are referred to as *implicit local queries*.

To improve the ranking for location-sensitive queries, a straightforward way is to treat the distance $d(\text{QUERY}, \text{URL})$ between the query and the URL as an extra feature in the learning-to-rank framework [3, 5]. However, typically the text-matching features [30] and the click-based features [17] dominate learning-to-rank models, thus this distance feature is likely to be buried due to its coverage. To overcome this problem, we propose a novel location boosting ranking model to improve the ranking for location-sensitive queries.

To compute the distance between queries and web pages, we first need to extract the locations from both sides. We parse the location from the explicit local queries and use users' locations as the query locations for implicit local queries. The locations of web pages are extracted based on the query-URL click graph from search logs. Given a URL, we parse the locations from the queries that this URL is connected to, and use these locations to describe the URL. We keep all the locations as a location vector weighted by the clicks. In addition, we also parse the locations from URLs directly.

Once we have the locations for both queries and URLs, we can easily compute the distance $d(\text{QUERY}, \text{URL})$, which is further normalized to the range of $[0, 1]$, denoted as $\hat{d}(\text{QUERY}, \text{URL})$. Note that when $d(\text{QUERY}, \text{URL}) = 0$, $\hat{d}(\text{QUERY}, \text{URL}) = 1$. The query-URL relevance is measured by the existing *base ranking function* denoted as $f_b(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$ is the input feature vector. We assume $f_b(\mathbf{x})$ is already available (trained by learning-to-rank methods [3, 5]). We then introduce the logistic function to combine the two conditions and define the ranking function $f(\mathbf{x})$ for location-sensitive queries as follows:

$$f(\mathbf{x}) = f_b(\mathbf{x}) + w\frac{1}{1+e^{\alpha f_b(\mathbf{x})+\beta}}\hat{d}(\text{QUERY}, \text{URL}) \qquad (7)$$

The idea is similar to recency ranking. Here, the logistic function controls the distance boosting based on the base relevance score $f_b(\mathbf{x})$ – if the URL is close to user and the content well matches the query, the boosting term (second term in Equation 7) is more positive, thus the total ranking score $f(\mathbf{x})$ for this URL is larger and the ranking for this URL is boosted; however, if the URL is very close to the user but is not relevant to the query, or if the URL matches the query well but it is too far from the user, the boosting term is close to 0, thus the ranking score is only decided by the base ranking function and the ranking for this URL is not boosted.

The parameters w, α and β are trained using pair-wise data by minimizing:

$$\sum\nolimits_{(p_i,p_j)\in P} \max(0, 1 - f(\mathbf{x}_i) + f(\mathbf{x}_j))^2 \qquad (8)$$

where $P = \{(p_i, p_j) \mid p_i \succ p_j\}$ is a set of preference URL pairs to the same query, obtained by human experts; $p_i \succ p_j$ denotes that p_i is more preferred than p_j. We solve the above optimization problem by a standard gradient descent approach.

Evaluation We compare the search relevance of our new ranking function $f(\mathbf{x})$ and the base ranking function $f_b(\mathbf{x})$ on 500 location-sensitive queries, which are sampled from query logs. The URLs are judged by human experts. The new ranking function improves DCG5 compared to the base ranking function by **+6.92%**. We also conduct online experiments to observe how users interact with the new ranking function. We perform "bucket tests" in a certain period to compare the base ranking function and the new ranking function in the Yahoo search engine. We use click-through rate (CTR) as our user experience metric to compare the two functions. Higher CTR implies a better user experience. The bucket test result shows that our new ranking function improves CTR by **+4.78%** compared to the base ranking function. This result is consistent with the offline experimental results (DCG) and shows that the new ranking function outperforms the base ranking function due to the effective location features. Table 9 compares the search results for a query "cvs" from a user in "Sunnyvale, CA". While the baseline ranks URLs with no specific location context on top (Note that cvs is a chain business that has many locations), the proposed distance boosting function ranks local results relatively higher.

Implementation and deployment To implement this location boosting model at run time, we extract the URL locations offline and store them in the forward index. The model parameters w, α and β are learned offline and stored in a dictionary. At run time, the implementation is similar to the core ranking function intro-

ranking		baseline	with location boosting
1	Title	CVS pharmacy - Online Pharmacy - Shop for Wellness and Beauty	CVS pharmacy - Online Pharmacy - Shop for Wellness and Beauty
	URL	http://www.cvs.com	http://www.cvs.com
2	Title	CVS Pharmacy - Wikipedia, the free encyclopedia	CVS | Facebook
	URL	http://en.wikipedia.org/wiki/CVS_Pharmacy	https://www.facebook.com/CVS
3	Title	CVS/pharmacy (CVS_Extra) | Twitter	CVS/Pharmacy - Sunnyvale, CA | Yelp
	URL	https://twitter.com/cvs_extra	http://www.yelp.com/biz/cvs-pharmacy-sunnyvale-3

Table 9: Compare the ranking results of query "cvs" with and without location boosting.

duced in Sec 3. The distance between queries and URLs and the location boosting scores are computed in parallel on distributed index serving nodes. Then the boosting scores are added to the base ranking function to get the final ranking score. For URLs that are not labeled with a location, the distance is set to be zero, $\hat{d}(\text{QUERY}, \text{URL}) = 0$, thus only the base ranking function is used for ranking. Note that this location boosting module is only triggered when the query is classified as a location-sensitive query by a run time classifier.

9. CONCLUSION

In this paper, we introduce the comprehensive relevance solutions of Yahoo search. The proposed solutions are effective, practical, and have been deployed and tested at scale in Yahoo's commercial search engine.

The proposed solutions are not limited to web search relevance, but also can be leveraged for vertical search engines, e.g., shopping, news, local, etc. We hope this work is able to provide useful insights to the whole research community for both industry and academia.

10. REFERENCES

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

[2] A. Broder, M. Fontoura, V. Josifovski, and L. Riedel. A semantic approach to contextual advertising. In *SIGIR '07*.

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

[4] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07*.

[6] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *JMLR: Workshop and Conference Proceedings*, pages 1–24, 2011.

[7] A. Dong, Y. Chang, Z. Zheng, G. Mishne, J. Bai, R. Zhang, K. Buchner, C. Liao, and F. Diaz. Towards recency ranking in web search. In *WSDM '10*.

[8] D. Downey, S. Dumais, and E. Horvitz. Heads and tails: Studies of web search with common and rare queries. In *SIGIR '07*.

[9] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR '08*.

[10] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

[11] J. Gao, X. He, S. Xie, and A. Ali. Learning lexicon models from search logs for query expansion. In *EMNLP-CoNLL '12*.

[12] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB '04*.

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

[14] S. Huo, M. Zhang, Y. Liu, and S. Ma. Improving tail query performance by fusion model. In *CIKM '14*.

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, Oct. 2002.

[16] S. Jiang, Y. Hu, C. Kang, T. Daly Jr., D. Yin, Y. Chang and C. Zhai. Learning Query and Document Relevance from a Web-scale Click Graph. In *SIGIR '16*.

[17] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*.

[18] A. Joshi, R. Kumar, B. Reed, and A. Tomkins. Anchor-based proximity measures. In *WWW '07*.

[19] C. Liu, F. Guo, and C. Faloutsos. Bbm: bayesian browsing model from petabyte-scale data. In *KDD '09*.

[20] C. Liu, F. Guo, and C. Faloutsos. Bayesian browsing model: Exact inference of document relevancfe from petabyte-scale data. *ACM TKDD*, 4(4):19:1–19:26, Oct. 2010.

[21] B. Long, O. Chapelle, Y. Zhang, Y. Chang, Z. Zheng, and B. Tseng. Active learning for ranking through expected loss optimization. In *SIGIR '10*.

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

[23] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *SIGIR '05*.

[24] C. Müller and I. Gurevych. A study on the semantic relatedness of query and document terms in information retrieval. In *EMNLP '09*.

[25] M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati. Context-and content-aware embeddings for query rewriting in sponsored search. In *SGIRI '15*.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[27] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *WWW '07*.

[28] S. Riezler and Y. Liu. Query rewriting using monolingual statistical machine translation. *Computational Linguistics*, 2010.

[29] S. Robertson and H. Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, Apr. 2009.

[30] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. *Trec*, 1994.

[31] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, Nov. 1975.

[32] R. Srikant, S. Basu, N. Wang, and D. Pregibon. User browsing models: relevance versus examination. In *KDD '10*.

[33] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *WWW '11*.

[34] Z. Zheng, K. Chen, G. Sun, and H. Zha. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR '07*.

[35] K. Zhou, X. Li, and H. Zha. Collaborative ranking: Improving the relevance for tail queries. In *CIKM '12*.