Learning Query and Document Relevance from a Web-scale Click Graph

Shan Jiang[†], Yuening Hu[‡], Changsung Kang[‡], Tim Daly Jr.[‡], Dawei Yin[‡], Yi Chang[‡], Chengxiang Zhai[†]

[†]Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{sjiang18,czhai}@illinois.edu

[‡]Yahoo Research
Sunnyvale, CA, USA
{ynhu,ckang,timjr,daweiy,yichang}@yahoo-inc.com

ABSTRACT

Click-through logs over query-document pairs provide rich and valuable information for multiple tasks in information retrieval. This paper proposes a vector propagation algorithm on the click graph to learn vector representations for both queries and documents in the same semantic space. The proposed approach incorporates both click and content information, and the produced vector representations can directly improve ranking performance for queries and documents that have been observed in the click log. For new queries and documents that are not in the click log, we propose a two-step framework to generate the vector representation, which significantly improves the coverage of our vectors while maintaining the high quality. Experiments on Web-scale search logs from a major commercial search engine demonstrate the effectiveness and scalability of the proposed method. Evaluation results show that NDCG scores are significantly improved against multiple baselines by using the proposed method both as a ranking model and as a feature in a learning-to-rank framework.

Categories and Subject Descriptors

[**Information retrieval**]: Information retrieval query processing—*Query log analysis*; [**Information retrieval**]: Retrieval models and ranking—*Similarity measures*

Keywords

Click-through bipartite graph, vector propagation, vector generation, Web search, query-document relevance

1. INTRODUCTION

Incorporating user feedback is one of the most effective ways to improve a search engine. For a commercial search engine with a large audience, this can be done by logging user actions – the queries they issued, the results they saw, the clicks they made, and so on. This behavioral data can be used to create a training target for a machine learned ranking function, or turned into features to improve the performance of the ranking function. In fact, click-based features are used as one of the primary sources to improve the ranking quality for popular queries.

However, the click information is sometimes *noisy*, and the coverage of clicks is quite limited compared to all possible relevant query-document pairs, which produces *sparsity* for the click-based features [9]. The sparsity and noise problems impact the overall quality of click-based features, especially for less popular queries.

To deal with these problems, an effective way is to learn a vector representation for both queries and documents in the same space [12, 25, 27]. Traditional methods typically learn low-rank vectors in a latent space. Though the low rank embedding of queries and documents has its own advantages, it also has some weaknesses. For example, it hurts interpretability and debuggability of the ranking function because individual dimensions in the latent space are hard to interpret. In comparison, using word features gives a more interpretable representation. However, direct text matching methods, e.g. BM25 [22] and the language models [28], suffer from the lexical gap between queries and documents which is shown to exist by previous study [20].[1] Thus, how to represent both queries and documents in the same semantic space and explore their relevance based on the click logs, remains a challenge.

Moreover, we need an approach that can be generalized to represent the queries and documents that have never been observed in the search logs. This is especially important in real applications because new queries and documents are emerging every day. However, purely click-based features are limited by the range of previous logs, and improving their coverage is a challenge.

Finally, though a Web-scale click graph provides us with unprecedentedly rich information, it also brings in great challenge of how to consume the entire click log efficiently rather than focus on a small sample. As a result, we need

SIGIR '16, July 17-21, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2911451.2911531

[1]Lexical gap means that the query terms are different from the corresponding terms in relevant documents in terms of string matching.

an efficient and scalable approach that can be easily applied to large scale click logs.

To solve all these challenges in a unified framework, this paper proposes a propagation approach to learn vector representation based on both content and click information (Section 2). The vector representation learned by our algorithm can directly improve the ranking performance for queries and documents that have been observed in the click log. For queries and documents that are not in the click graph, which are referred as *click-absent* queries and documents, we further propose a two-step vector estimation algorithm which generates a representation based on their association with the vectors that are already created by propagation on the click graph (Section 3). This generalization significantly improves the coverage of our vectors, which is particularly important for long-tail queries in web search.

This whole framework is very efficient and scalable, especially compared to matrix factorization-based methods [12, 25, 27]. Since a search engine click log can grow by many millions of new entries in a single day, this scalability is essential. In the experiments (Section 4), we collect click-through log data from a major commercial search engine, which contains 25 billion query-document pairs in total, and demonstrate the effectiveness of our proposed approach used as an individual ranking model and a feature in the learn-to-rank framework. More analysis and case study about why and how the proposed approach improves ranking are discussed in Section 5.

To summarize, our main contributions include:

- A unified approach, which generates the semantic representations for both queries and documents in an explicit and interpretable way.
- A generalized method, which estimates vector representations for any unseen queries and documents.
- A scalable and efficient framework, which is easy to implement and works effectively in a real commercial search engine.

2. VECTOR PROPAGATION ON THE CLICK GRAPH

Click-through data provides a soft indicator of relevance between queries and documents, and click-based features are one type of the most important and effective features in ranking task. However, purely click-based features highly rely on the users' behavior, and it sometimes suffers from sparsity and noise in the click data, especially for less popular queries. On the other hand, content-based features capture the intrinsic relevance embedded in the text content, and they are independent of other resources such as click logs. However, the lexical gap between queries and documents and the fact of missing users' feedback make the content-based features less effective in real ranking problems.

This paper bridges the two information resources and proposes a vector propagation algorithm which makes use of both click and content information to learn the vector presentation of queries and documents. This algorithm not only introduces new words from the relevant queries and documents through the propagation, but also smoothes the term weights based on the click information. Besides, the sparsity and noise are also reduced during this process thus better quality of the vector representations is guaranteed. Once we obtain the vectors for queries and documents, similarity functions such as cosine similarity can be applied to compute the query-document relevance.

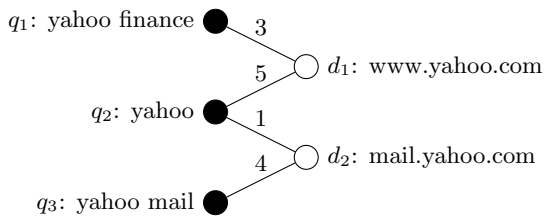

Figure 1: An example of click-through bipartite graph.

2.1 Preliminaries and Notations

Click-through data consists of queries and documents with their co-click information[2]. Let $\mathcal{D}oc$ be the set of documents and $\mathcal{Q}uery$ be the set of queries. We construct the click graph $\mathcal{G}$ with the node set $\mathcal{V} = \mathcal{D}oc \cup \mathcal{Q}uery$. For a document $d \in \mathcal{D}oc$ and a query $q \in \mathcal{Q}uery$, if there is at least one co-click between them, i.e., at least one user clicked on d when he or she searched for query q, an edge is built between them. The weight of the edge is set to be the number of co-clicks. The set of edges in the graph is denoted as $\mathcal{E}$. An example of the click graph is shown in Figure 1. Documents are denoted as open circles and labeled by the corresponding URL, while queries are represented as solid circles.

The adjacency matrix of the graph is denoted as C where the entry in the i-th row and j-th column ($C_{i,j}$) equals to the weight of the edge between query q_i and document d_j. If there is no edge between them, $C_{i,j}$ is 0.

We denote Q as the matrix where i-th row Q_i is the vector representation of query q_i. $Q^{(n)}$ is the query vector matrix at the n-th iteration. If the vocabulary size is V, then the size of Q is $|\mathcal{Q}uery| \times V$. Similarly we denote D as the document vector matrix and its size is $|\mathcal{D}oc| \times V$.

2.2 Vector Propagation Algorithm

The goal of this vector propagation algorithm is to learn the representation of queries and documents in the same semantic space (either the query space or the document space). It starts with the content information initialized as vectors on one side of the click graph, and propagates the vectors to the connected nodes on the other side of the click graph. Co-clicks between queries and documents are used to weight the vectors so that the important terms stand out while less informative terms are gradually filtered out. More details are introduced as follows.

We first choose one side of the click graph (either side) and initialize the vectors based on the content information, e.g., the query words on the query side, or the titles and abstracts on the document side. This initialization keeps the word-level information in the vectors. Words can be considered as human-designed features and past studies have repeatedly shown that keeping the word-level representation is very helpful. In traditional content-based models such as VSM [23], vocabulary from query and document is merged together. Though it makes the vector representation easier and more natural, it may also bring in the lexical gap between queries and documents [20]. Our method starts with

[2]If a document is clicked by a user for a proposed query, it is referred as a co-clicked query-document pair in this paper.

one vocabulary space, and learns the vectors for queries and documents based on one consistent vocabulary space (either query vocabulary or the document vocabulary). Thus the lexical gap is reduced.

In the next step, vectors are propagated through the click graph. Our basic idea is to represent queries based on their relevant documents and vice versa. Queries that are co-clicked with a lot of common documents should have similar representations, and documents sharing many co-clicked queries should also be close in the vector space. Moreover, the click information is a soft indicator of relevance and the confidence of relevance is positively correlated to the number of clicks. Thus, co-clicked pairs that have a higher frequency should dominate the vectors. The propagation starts from the side where vectors are initialized, then the initialized vectors are weighted by the clicks and propagated to the connected nodes on the other side of the click graph. For example, if we start from the query side, the document vectors are calculated by aggregating all query vectors that are co-clicked with them. In the next iteration, vectors are propagated in the opposite direction, and this process is conducted repeatedly until convergence.

We take starting from the query side as an example. In the initialization, each query is represented by its own words. Words are weighted in proportion to their frequency and normalized by the $\mathcal{L}2$ norm. The initialized vector matrix is denote as $Q^{(0)}$. In the n-th iteration ($n \geq 1$), we first compute document vectors $D^{(n)}$ by a weighted summation of their co-clicked queries where the weights depend on the number of clicks. Formally, given the j-th document d_j, its vector representation $D_j^{(n)}$ is calculated as:

$$D_j^{(n)} = \frac{1}{||\sum_{i=1}^{|\mathcal{Q}uery|} C_{i,j} \cdot Q_i^{(n-1)}||_2} \sum_{i=1}^{|\mathcal{Q}uery|} C_{i,j} \cdot Q_i^{(n-1)} \quad (1)$$

where $Q_i^{(n-1)}$ is the vector representation for the i-th query in the (n-1)-th iteration. $D_j^{(n)}$ is also normalized by $\mathcal{L}2$ norm. Then the query vectors in $Q^{(n)}$ are updated based on their co-clicked document vectors:

$$Q_i^{(n)} = \frac{1}{||\sum_{j}^{|\mathcal{D}oc|} C_{i,j} \cdot D_j^{(n)}||_2} \sum_{j=1}^{|\mathcal{D}oc|} C_{i,j} \cdot D_j^{(n)} \quad (2)$$

Through this propagation process, we learn the vectors for both queries and documents using the query vocabulary. Without the lexical gap, better query-document relevance can be achieved. Similarly, if we start from the document side, document vectors (denoted as $D^{(0)}$) are initialized by using their content words, and the vector propagation starts from the document side to the query side in each iteration:

$$Q_i^{(n)} = \frac{1}{||\sum_{j}^{|\mathcal{D}oc|} C_{i,j} \cdot D_j^{(n-1)}||_2} \sum_{j=1}^{|\mathcal{D}oc|} C_{i,j} \cdot D_j^{(n-1)} \quad (3)$$

$$D_j^{(n)} = \frac{1}{||\sum_{i=1}^{|\mathcal{Q}uery|} C_{i,j} \cdot Q_i^{(n)}||_2} \sum_{i=1}^{|\mathcal{Q}uery|} C_{i,j} \cdot Q_i^{(n)} \quad (4)$$

Take the click graph in Figure 1 as an example. If we start from the query side, Q_1, Q_2 and Q_3 are initialized as {yahoo:$\frac{1}{\sqrt{2}}$, finance:$\frac{1}{\sqrt{2}}$, mail:0}, {yahoo:1, finance:0, mail:0} and {yahoo:$\frac{1}{\sqrt{2}}$, finance:0, mail: $\frac{1}{\sqrt{2}}$} respectively. Then document vectors are derived from the co-clicked queries. For instance, the vector representation for d_1 (D_1) is calculated as $(\frac{3}{8}Q_1 + \frac{5}{8}Q_2)/||\frac{3}{8}Q_1 + \frac{5}{8}Q_2||_2$. If we start from the document side, we first initialize the document vectors by their titles: "Yahoo Finance - Business Finance, Stock Market, Quotes, News" and "Yahoo" for d_1 and d_2 respectively.[3] Then the document vectors are propagated to the query side and the query vectors are generated in a similar way. Because of the lexical gap between queries and documents, the learned vectors starting from different sides are indeed different, and in fact they complement each other in the real ranking system. More analysis is in Section 5.

The proposed vector propagation is similar to the importance propagation in HITS algorithm [14]. In HITS algorithm, mutual reinforcement bonds the authority and hub scores through the Web link structure, while our propagation algorithm aims at mutual improvement of query and document vector representations. Instead of propagating importance scores, we propagate the feature vectors which are derived from the text content. In this way, both content and click information are encoded in the learned vectors.

In practice, we represent each vector in a sparse way (only keeping the terms with non-zero weights), which helps to increase the computational speed. However, a potential problem in this propagation algorithm is the rapid growth of the non-zero entries, which leads to a considerable increase in time and space consumption in a Web-scale scenario. However, if we rank the terms in a vector according to their weights, we can clearly see that the vectors have long tails: the weights drop significantly after the first few terms, and the remaining terms in the vector have very small weights (examples can be found in Section 5.1). Thus we can simply keep the top K terms with the highest weights for both query and document vectors in each iteration to further reduce both time and space consumption.

3. VECTOR GENERATION FOR CLICK-ABSENT QUERIES AND DOCUMENTS

Though the vector propagation algorithm proposed in Section 2 enables us to get the relevance feature for any query-document pair in the click graph, it still cannot handle queries or documents that have no click information. As a matter of fact, the ability to deal with click-absent queries and documents is of great importance in real applications, since we can always see new queries and documents that are not included in previous logs.

An intuitive way is to use bag-of-words model to generate their vector representations, e.g., use the query words to represent the queries, and use the title words to represent the documents. However, it introduces the lexical gap between queries and documents that we try to avoid in this paper, and it is limited to the content words instead of expanding the vectors with additional related terms. Besides, it also breaks the indirect connections between the click-absent queries/documents and the documents/queries in the click graph, thus reweighting cannot be learned from the clicks.

To better estimate the vectors for click-absent queries and documents, this section proposes a novel and efficient two-step framework. We first break down queries or document titles (depending on from which side we start from in the vector propagation algorithm) into different units (e.g.,

[3]Titles are extracted from the HTML sources.

Figure 2: An example of unit vector generation. (The black thin lines are the edge of the click graph, while the edges represented by the gray thick lines indicate the pseudo clicks between units and documents.)

ngrams), and learn a vector representation for each unit based on the vectors we have already learned from the click graph. We then learn a weight for each unit by a regression model, and finally estimate the vectors for the click-absent queries and documents by a linear combination of the unit vectors. This framework connects the click-absent queries and documents to the click graph through the unit vectors and gets high-quality representations for them, thus enlarges the coverage of the vectors.

3.1 Extracting Unit Vectors

Vectors learned by the propagation algorithm keep word-level features, which provides a potential bridge to connect the click-absent queries and documents with those in the click graph. Even though we cannot exactly match a new query or document with an existing one in the search logs, the key words or phrases of these new queries and documents may be already contained in the observed queries and documents. These key terms (such as unigrams, bigrams and trigrams) can be treated as basic message units in both click-absent and existing queries and documents, and we can get these ngram units from the text content. More specifically, if the propagation starts from the query side, all the queries are broken into unigrams, bigrams and trigrams. Similarly, document contents such as titles or abstracts are decomposed to ngrams if we use document vocabulary. The set of units is denoted as $\mathcal{U}$.

The next step is to learn a vector representation for each unit. In the vector propagation algorithm (Section 2), query vectors are learned by aggregating document vectors according to their co-clicking relationship, and we can learn the unit vectors in a similar way. Let's assume the vector propagation starts from the query side and first build up the unit vectors in the query vocabulary space. For each unit u_i in $\mathcal{U}$, we find all the queries containing the unit u_i, and denote the set as O_{u_i}. We also find all the documents that are connected to any of these queries, and denote the document vector set as K_{u_i}. Next, we connect u_i with vectors in K_{u_i} based on the click information: for the k-th vector in O_{u_i} and the j-th vector in K_{u_i} (denoted as $O_{u_i}(k)$ and $K_{u_i}(j)$ respectively), if there is a click between their corresponding query and document, we say that there is a pseudo click between u_i and $K_{u_i}(j)$ through $O_{u_i}(k)$, denoted as $P_{i,k,j}$. The value $P_{i,k,j}$ equals to the query-document co-click number $C_{k,j}$. If we aggregate all the pseudo clicks between u_i and $K_{u_i}(j)$ over all queries in O_{u_i}, we get the pseudo click between u_i and $K_{u_i}(j)$ and denote it as $P_{i,j}$:

$$P_{i,j} = \sum_{k=1}^{|O_{u_i}|} P_{i,k,j} \tag{5}$$

As the example shown in Figure 2, if u_i is "yahoo" and the propagation starts from the query side, O_{u_i} is the set of queries that contains the unit "yahoo". In this example, we have three queries in O_{u_i}, which are q_1, q_2 and q_3 respectively. d_1 and d_2 are found to be co-clicked documents for these three queries, so their vectors constitute K_{u_i}. The pseudo clicks are built between them and u_i with aggregated weights. If the unit is a bigram or trigram, we only consider the queries that contain the unit as a bigram or trigram. For example, if the unit is "yahoo finance", then queries containing "yahoo" and "finance" as separate unigrams (e.g., "yahoo news in finance") are not included.

Finally, the vector representation of u_i (denoted as U_i) is calculated by a weighted sum of vectors in K_{u_i}, normalized by $\mathcal{L}2$ norm:

$$U_i = \frac{1}{||\sum_{j=1}^{|K_{u_i}|} P_{i,j} \cdot K_{u_i}(j)||_2} \sum_{j=1}^{|K_{u_i}|} P_{i,j} \cdot K_{u_i}(j) \tag{6}$$

Similarly, if the vector propagation starts from the document side, we extract all the units from the document content. Then for each unit u_i, we find all the documents containing the unit u_i and denote the set as O_{u_i}, then find all the queries that are connected to any of these documents and denote their vectors as K_{u_i}. We then can follow the same process to build up the connections between units and the query vectors to estimate the unit vectors in the document vocabulary space.

As we can see from this process, these unit vectors are generated from the related query or document vectors that are learned by the propagation algorithm in the click graph, so click information is also embedded in them.

3.2 Learning Unit Weights

In the next step, we learn a weight per unit to capture the importance for each unit globally, which is later used for generation of vectors for click-absent queries and documents. Here we learn the weights through a linear regression model, and the vectors where the units originate from are used as training examples (named as target vectors for convenience). The basic idea is to use the linear combination of unit vectors to approximate the target vectors, and then minimize the difference between the target vector and the

approximated vector which is measured by the square of Euclidean distance:

$$\min_{W} \sum_{i=1}^{|T|} ||T_i - \sum_{u_j \in \mathcal{U}_{T_i}^{all}} W_j \cdot U_j||_2^2 \quad (7)$$

where T is the set of target vectors and T_i is the i-th vector in T. If the units originates from queries, T consists of all the query vectors learned by the propagation algorithm; otherwise it contains all the document vectors. $\mathcal{U}_{T_i}^{all}$ is the set of all the units (unigrams, bigrams and trigrams, excluding itself) contained in the corresponding query or document content of T_i. W is the weight vector with the j-th entry W_j representing the weight for u_j in the unit set $\mathcal{U}$.

3.3 Estimating Vectors

In Section 3.1 and Section 3.2, we introduce how to generate the unit vectors and weights respectively. Given a new query or document without any click information, we can represent it by the weighted combination of unit vectors.

For a click-absent query q, we first decompose it into all possible units. If a unigram is contained in a bigram, or a bigram is contained a trigram, it is removed to avoid overlapping information. For example, given a new query, "walmart credit card", assume the set of unigrams, bigrams and trigrams contained in unit vocabulary includes {"walmart", "credit", "card", "credit card"}, then we only keep "walmart" and "credit card" in the unit set. The final unit set which is decomposed from q by the above rule is denoted as $\mathcal{U}_q$. Then the query vector q_v is calculated as a linear combination of the unit vectors in $\mathcal{U}_q$:

$$q_v = \sum\nolimits_{u_i \in \mathcal{U}_q} W_i U_i \quad (8)$$

For a click-absent document d, we can estimate its vector representation in a similar way. Any possible content of documents can be used, and here we use document titles, which are concise yet provide key information. We decompose the title of d to a unit set $\mathcal{U}_d$ according to the same rule as mentioned above, and the vector d_v is calculated as:

$$d_v = \sum\nolimits_{u_i \in \mathcal{U}_d} W_i U_i \quad (9)$$

Note that this generation algorithm estimates the vectors either in query vocabulary space or in the document vocabulary space, depending on how the unit vectors are built. Based on the vector propagation algorithm and the above vector generation algorithm, we now represent all queries and documents in the same semantic space. Relevance between any query-document pair can be measured by similarity functions such as cosine similarity, which is further used as a feature for ranking.

4. EXPERIMENTS

We apply the proposed method to a Web-scale click graph from a major commercial search engine. The learned relevance score can be either used directly to rank documents for a given query or employed as one of the features in a learning-to-rank framework. We show that the proposed method helps to improve ranking results in both cases.

4.1 Dataset

We construct the click-through bipartite graph from a major commercial search engine's search log. There are about 25 billion co-clicked query-document pairs, containing about 8 billion unique queries and 3 billion unique documents.

To investigate whether the relevance score learned by our algorithm can help to improve ranking in a learning-to-rank framework, we use another dataset that includes 63k queries and 775k query-document pairs as training examples and 16k queries with 243k query-document pairs as test set. The relevance score of each pair ("perfect", "excellent", "good", "fair" or "bad") is annotated by human annotators. 92.5% of these testing queries can be found in the search log and the others do not have click information. 78.9% of the testing documents are in the search log. For the 21.1% documents missing in the search log (about 51k documents), 91.7% of them have titles.

4.2 As an Individual Ranking Model

To give more insight into how well the proposed method captures the relevance between queries and documents, we use the relevance scores to rank documents and calculate NDCG scores to evaluate the performance. The trimming parameter K in vector propagation algorithm is set as 20.[4] We compare our methods against multiple baseline methods:

- **VPCG** is our vector propagation algorithm proposed in Section 2, and the generalized version **VPCG&VG** also predicts the vectors for the click-absent queries and documents as in Section 3. **_QUERY** and **_DOC** denote that the propagation starts from the query or document side, respectively.
- **BM25SD**, a better variation of traditional BM25 [22], uses individual words, ordered and unordered word sequences from page content to compute the score [6].
- **BM25SD_MULT**, a linear combination of BM25SD across multiple fields including page content, anchor text, and query words for which the document was clicked.
- **CTR**, a behavioral feature which measures the ratio between the impressions and the nevigational clicks in a sessions for each query-document pair, where a navigational click is defined as the only click in a session, and a session is defined as some timeout (e.g. 30 minutes) between queries for the same user. The lower bound of wilson confidence interval for this ratio in multiple sessions is used as the final feature value.
- **WMD**, a word embedding-based framework using the Word Mover's Distance [15] to measure the query-document relevance, based on a word embedding vector set trained from Google News [19].[5]

As shown in Table 1, our methods VPCG and VPCG&VG in general achieve the best performance. VPCG_QUERY performs even better than VPCG_DOC, most likely because titles sometimes contain noisy words or are not descriptive of page content. However, VPCG_QUERY and VPCG_DOC capture the query-document similarity from different viewpoints and complement each other, which is further discussed in Section 5. The generalized version VPCG&VG_QUERY

[4]Multiple experiments are done to evaluate the sensitivity of the selection of K, but it turns to be not very sensitive, thus we use $K = 20$ in this paper.

[5]The Google News word vectors are available in https://code.google.com/p/word2vec/

Table 1: Performance as an individual ranking model.

Feature	NDCG@1	NDCG@3	NDCG@5	NDCG@10
BM25SD	0.4373	0.4937	0.5460	0.6542
BM25SD_MULT	0.6132	0.6346	0.6668	0.7464
CTR	0.5769	0.5941	0.6238	0.7064
WMD	0.4585	0.5145	0.5641	0.6674
VPCG_QUERY	0.6268	0.6498	0.6797	0.7509
VPCG&VG_QUERY	**0.6344***	**0.6618***	**0.6948***	**0.7687***
VPCG_DOC	0.5648	0.6209	0.6623	0.7382
VPCG&VG_DOC	0.5668	0.6268	0.6717	0.7509

Two-tailed t-test is done for paired data where each pair is VPCG&VG_QUERY and any of the other methods, and * indicates *p-value* < 0.01 for all tests.

and VPCG&VG_DOC further improve the results of VPCG_QUERY and VPCG_DOC respectively, because the feature coverage is improved. The CTR feature used here is a very accurate predictor of relevance, but it is only available for query-document pairs that have been observed in the log, so it does not perform well on the click-absent ones.

4.3 As a Feature in a Learning-to-rank Model

A typical industrial learning-to-rank setting builds powerful nonlinear models based on a large set of high quality features [7]. Indeed, these models can often combine many weak features to outperform a single strong feature [26]. It is therefore challenging to create a new feature that significantly improves the overall model performance. Here we use a variant of the gradient boosting machine proposed in [11] to implement a learning-to-rank framework with over 2,400 features.[6]

We denote all the 2,400+ basic features as "Base". This basic feature pool contains an immense variety of features, including CTR, BM25SD and their variations, etc. We then extend the feature pool by adding new features, and the performances of the learning-to-rank model trained based on different feature pools are shown in Table 2. "Base0" is a subset of the standard feature pool which excludes CTR and BM25SD_MULT. "Base + WMD" means the standard feature pool plus WMD feature. "Base + VPCG" extends the feature pool by adding VPCG_QUERY and VPCG_DOC, and "Base + VPCG&VG" means adding VPCG&VG_QUERY and VPCG&VG_DOC to the feature pool.

As shown in Table 2, removing the powerful features CTR and BM25SD_MULT does not change the model performance much, since the remaining features in the feature pool have already contributed similar inputs to the ranking system. The result is just slightly increased by adding WMD. This is likely because WMD introduces a notion of semantic similarity between different words that is not present in most text matching features. In general, the differences between the performance of "Base0", "Base" and "Base + WMD" are all very marginal.

By contrast, we do observe a significant improvement in overall model performance after adding our features to the pool. Our features make good use of both click and content information, and bring in new information which is not presented in the existing feature set. It is interesting that the performance difference between VPCG&VG and VPCG is narrowed in the learning-to-rank framework compared to when using them as individual ranking models (Table 1). This is because the learning-to-rank framework is able to rely on other features when the "VPCG" feature is missing. Another reason is that only a small portion (about 7.5%) of our test queries are click-absent, limiting the impact of the VG method on this data set. However, we do find click-absent queries where the ranking is significantly improved by using the VG method. More detailed discussion is provided in Section 5.

[6]For both space and confidentially reasons, we cannot here define the complete set of features.

Table 2: Performance as a feature in a learning-to-ranking framework.

Feature pool	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Base0	0.6872	0.7024	0.7288	0.7925
Base	0.6862	0.7014	0.7281	0.7923
Base + WMD	0.6881	0.7023	0.7291	0.7934
Base + VPCG	0.7068	0.7207	0.7470	0.8080
Base + VPCG&VG	**0.7096**	**0.7256***	**0.7506***	**0.8108***

Two-tailed t-test is done for paired data where each pair is "Base + VPCG&VG" and any of the other methods, and * indicates *p-value* < 0.01 for all tests.

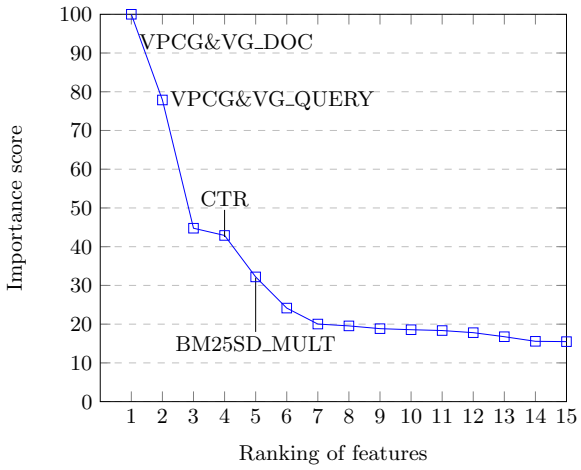

Figure 3: Importance score of the top 10 features

To further investigate the impact of different features, we calculate the relative importance score for all the features [11], where the most influential feature is assigned a score of 100 and the estimated scores for others are scaled accordingly. The distribution of the relative importance scores for the top 15 features are shown in Figure 3. We are not able to provide details about the features for confidentiality reasons, but we can still conclude that our features indeed play an important role in the retrieval system, since VPCG&VG_DOC and VPCG&VG_QUERY rank as the top two best features. CTR and BM25SD_MULT rank as the fourth and fifth respectively. The importance scores drop dramatically between the second and the third one, while the difference between other pairs are much smaller, indicating the superiority of our features. This is further evidence that our features significantly improve the performance both as a single feature and in the learning-to-rank models.

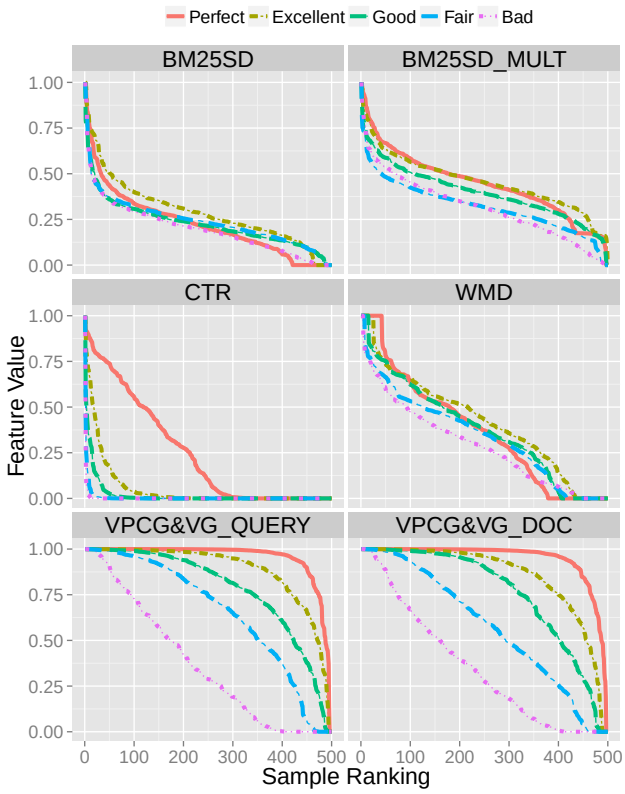

Figure 4: Separability of different features.

4.4 Separability of Features

In Section 4.2 and Section 4.3, we show significant improvement on NDCG scores by our methods in all experiment settings. In this section, we look into more details about how our features work to distinguish different documents according to their relevance to a given query.

As mentioned in Section 4.1, each document is labeled with one of the five degrees of relevance, namely "perfect", "excellent", "good", "fair" and "bad". To better evaluate the features, we measure the "separability" of the features at these five labels: we randomly sample 500 tuples (⟨query, url, feature value⟩) for each of the five labels from the test dataset, sort the tuples by their feature values, and plot them in Figure 4. Each point (x,y) in this figure corresponds to a tuple where X-coordinate is its ranking and Y-coordinate is its feature value. For the convenience of illustration, BM25SD and BM25SD_MULT are normalized by its maximal value so that it is scaled between 0 and 1. Note that the normalization does not impact the ranking.

We can see that CTR separates "perfect" from others, and WMD distinguishes "bad" from others. BM25SD is good at identifying "excellent", but its ability of distinguishing others are not as good as the enhanced version BM25SD_MULT. Compared to the baseline methods, both VPCG&VG_QUERY and VPCG&VG_DOC have a much stronger separability over all the five labels. Especially, VPCG&VG_QUERY is better at distinguishing "perfect", "excellent", "good" and "fair" from "bad", while VPCG&VG_DOC is better at seperating "perfect", "excellent" and "good" from "fair" and "bad"; and VPCG&VG_QUERY has a slightly better seperability among "perfect", "excellent" and "good", which explains why VPCG&VG_QUERY performs better as a single feature while the learning-to-rank framework benefits from the combination of them.

5. DISCUSSION

To get a better understanding about the vectors generated by our method, we study some query vectors in details and analyze how they influence the ranking result. Besides, we also further evaluate the vector generation algorithm for click-absent queries and documents, and illustrate how the generated vectors help to improve the ranking quality.

5.1 Latent Effect of Vector Propagation

The main change of query and document vectors made by our propagation method lies in two aspects: the introduction of latently related words and the re-weighting of vector terms. Introducing related words helps to explain the query or document better and re-weighting makes the key terms stand out, which facilitates tasks like disambiguation and query intent understanding.

Table 3 shows several examples of the learned query vectors (vector propagation starts from query side). We can see that keywords such as "eagles", "album" and "lyrics" can help us to distinguish the query relevant to the song named "hotel california" from an alike query "hotel california palm springs hotel" which is about a real hotel. In the third example, the meaning of "dtd amc" is hard to figure out even for a human user who is not familiar with the background knowledge. Actually "dtd" means "Downtown Disney" and the query is asking about AMC theaters in Downtown Disney. In this example, informative keywords such as "disney", "downtown", "theater" and "movie" are introduced in the learned vector. As shown in the table, ranking model trained without feature generated by the vector propagation algorithm (Base) performs very badly on this query (two of the top three are "bad"). When adding our feature, the result gets much better: both the top two retrieved documents are "good" and the third one is "fair".

5.2 Propagation Starting from Different Side

Our proposed vector propagation algorithm is very flexible to start from either query or document side, denoted as VPCG_QUERY and VPCG_DOC respectively. To study the difference between the two features and how they complement each other, we take the query "choise fm" as an example (Table 4). "choise fm" is an urban music radio station owned by Global Radio and anchored by an FM operation in London, and it is now changed to "capital xtra". Because this query is the old name of the music radio station, if we start from the query side, our algorithm propagates the old information "choise fm" to the document side and learns larger weights for "choice" and "fm". Though the relationship between "choise fm" and "capital xtra" is captured in the vector, it has much smaller weights for "capital" and "xtra". Thus the top three ranking results are all about "choise fm", and it fails to rank the perfect URL "http://www.capitalxtra.com/" in top 3. However, if we start from the document side, the document title includes the new name "capital xtra" and our algorithm propagates such information to the query side "choise fm". Thus the query vector from VPCG_DOC not only captures the relationship

Table 3: Case Study: Introduction of new words and reweighting helps ranking

Example of learned vectors		
Query	Vector	
hotel california eagles	hotel:0.675, california:0.667, eagles:0.300, **album**:0.057, **song**:0.053, **lyrics**:0.041, meaning:0.030, what:0.022, cover:0.022, youtube:0.017, about:0.016, band:0.010, video:0.010, to:0.005, songs:0.004, tyga:0.003, wiki:0.003,list:0.002, covers:0.002, art:0.002	
hotel california palm springs hotel	palm:0.596, springs:0.582, hotel:0.358, california:0.329, hotels:0.223, **ca**:0.138, resorts:0.021, spring:0.016, best:0.007, **tripadvisor**:0.006, eagles:0.005, **trip**:0.005, viceroy:0.005, resort:0.005, palms:0.004, advisor:0.003, marriott:0.003, lyrics:0.002, riviera:0.002, meaning:0.002	
dtd amc	amc:0.707, **disney**:0.475, **downtown**:0.414, island:0.158, **theater**:0.146, **movie**:0.121, **theaters**:0.121, pleasant:0.089, anaheim:0.072, pleasure:0.067, 24:0.066, orlando:0.059, movies:0.027, 12:0.024, walk:0.023, theatres:0.018, yelp:0.016, ca:0.016, dine:0.014, district:0.007	
Ranking Results for Query "dtd amc"		
Ranking	Base	Base + VPCG
1	http://acronyms.thefreedictionary.com/DTDS	http://www.yelp.com/biz/amc-downtown-disney-12-ca-anaheim
2	http://www.mydreamsofdisney.com/2014/01/saving-mr-banks-amc-fork-screen/	http://www.dadsguidetowdw.com/downtown-disney.html
3	http://www.thefreedictionary.com/DTDS	http://www.mydreamsofdisney.com/2014/01/saving-mr-banks-amc-fork-screen/

Table 4: Case Study: The vector representations from VPCG_QUERY and VPCG_DOC complement each other.

Query Vectors for Query "choise fm"		
VPCG_QUERY	**choice**:0.714, **fm**:0.666, radio:0.148, choicefm:0.110, london:0.085, nevis:0.039, **capital**:0.033, **xtra:0.027**, 3:0.024, choicefm1053:0.019, station:0.019, uk:0.016, 105:0.014, music:0.012, choicefm105:0.012, playlist:0.008, choiceradio:0.006, 3fm:0.0045, extra:0.004, 1:0.004	
VPCG_DOC	**xtra**:0.446, **capital**:0.446, urban:0.439, dance:0.438, uk:0.426, **choice**:0.112, skn:0.079, times:0.079, ltd:0.079, **fm**:0.041, radio:0.028, music:0.025, listen:0.024, online:0.020, 3:0.020, com:0.016, choiceradio:0.013, 107:0.011, 105:0.011, your:0.009	
Ranking Results for Query "choice fm"		
Ranking	Base + VPCG_QUERY	Base + VPCG_QUERY + VPCG_DOC
1	http://en.wikipedia.org/wiki/Choice_FM	http://www.capitalxtra.com/
2	http://www.choicefmnz.com/	http://en.wikipedia.org/wiki/Choice_FM
3	http://tunein.com/radio/Choice-FM-1053-s89470/	https://www.facebook.com/choicefm

Table 5: Performance of vector generation algorithm

Method	Similarity
BOW	0.4833
Unigram vector + equal weight	0.5368
Unit vector + equal weight	0.5927
VG	**0.6057**

between "choise fm" and "captial xtra", and learns more accurate weights for "capital" and "xtra". As a result, with the help from both VPCG_QUERY and VPCG_DOC, our model is able to rank the perfect URL "http://www.capitalxtra.com/" as the top 1 result. Similarly, VPCG_QUERY may capture more accurate query intent than VPCG_DOC for some queries, and due to space limit, we do not list more examples here. In general, VPCG_QUERY and VPCG_DOC complement each other and together improve the ranking results as shown in Section 4.

5.3 Evaluation of Vector Generation Algorithm

The goal of the vector generation method proposed in Section 3 is to get a good vector estimation for queries and documents that have no click information. In Section 4.2 and Section 4.3, we have already shown that those generated vectors are indeed helpful for ranking. Here we further look into these generated vector representations to see how well they approximate the target vectors.

We use the query vectors generated by the vector propagation algorithm starting from the query side as an example for the evaluation. The set is divided into two subsets, namely training set and test set. The training set is used to learn the unit vectors and their corresponding weights, based on which vectors for the test set are generated by the vector generation algorithm. The vectors learned by the vector propagation algorithm are used as ground truth. The evaluation metrics is the average of cosine similarity between the groundtruth vector and the vector generated by methods that estimate the vectors in other alternative ways.

We compare our method "VG" against several baselines, and the results are shown in Table 5. Here "BOW" is the bag-of-words model. "Unigram vector + equal weight" uses a linear combination of the vectors for each unigram with equal weights. For example, the query vector for "yahoo finance" is calculated as Q("yahoo") + Q("finance") where Q("yahoo") and Q("finance") are vectors for query "yahoo" and "finance" which are learned by the vector propagation algorithm. "Unit vector + equal weight" uses the unit vectors we introduce in Section 3. It decomposes queries into a set of units in the same way as "VG" does, but assigns equal weight to each unit instead.

From Table 5, we can see that methods making use of existing query vector information ("Unigram vector + equal weight", "Unit vector + equal weight" and "VG") outperform bag-of-words model. Using unit vectors works better than using original vectors for unigram. "VG" brings in further

Table 6: Case Study: Unit vector helps to improve ranking for click-absent queries

Unit vectors for Query "how long is into the storm"		
Unit	Weight	Vector
how long is	0.363	how:0.375, long:0.320, contagious:0.167, 5k:0.044, heat:0.039, pregnancy:0.037, eye:0.037, pink:0.036, flu:0.036, dog:0.035, dogs:0.034, does:0.030, cold:0.028, good:0.023, light:0.022, you:0.022, strep:0.022, to:0.022, many:0.020, throat:0.020
is into the	0.562	into:0.570, woods:0.303, wild:0.161, mccandless:0.118, movie:0.085, christopher:0.061, mystic:0.055, chris:0.055, storm:0.040, rncm:0.032, musical:0.028, appropriate:0.025, children:0.025, kids:0.024, van:0.020, morrison:0.020, how:0.019, rated:0.015, r:0.015, why:0.015
into the storm	1.038	storm:0.665, into:0.641, **movie**:0.159, **2014**:0.050, **watch**:0.041, online:0.040, **reviews**:0.039, free:0.034, **trailer**:0.030, torrent:0.024, black:0.017, sky:0.017, cast:0.012, **imdb**:0.011, **review**:0.010, full:0.010, tornado:0.009, rotten:0.008, tomatoes:0.008, dvd:0.007

Ranking Results for Query "how long is into the storm"		
Ranking	Base + VPCG	Base + VPCG&VG
1	http://www.rottentomatoes.com/m/into_the_storm_2014/	http://www.rottentomatoes.com/m/into_the_storm_2014/
2	http://www.ign.com/articles/2014/08/07/into-the-storm-review	https://en.wikipedia.org/wiki/Into_the_Storm_(2014_film)
3	http://science.nasa.gov/science-news/science-at-nasa/2001/ast16aug_1/	http://www.ign.com/articles/2014/08/07/into-the-storm-review

improvement compared to assigning equal weights to units. In general, our proposed generalization algorithm provides the most accurate approximation of the ground truth vector.

5.4 Query Vector Generation in Ranking

In previous sections, we have shown the quantitative analysis of the vector generation algorithm, including its performance in ranking and the quality of the estimated vectors. In this section, we continue our discussion about how it helps to improve ranking by illustrating a query example "how long is into the storm", which is about the length of a movie "Into the Storm". This query is not in the search log so we can only use the vector generation method.

To generate the query vector, "how long is into the storm" is decomposed into three units, namely "how long is", "is into the" and "into the storm", whose vectors are given in Table 6. The keyword "movie" is added to the unit vector for "into the storm" with a high weight, which increases the relevance scores of movie-related documents. Besides, our regression model successfully captures that "into the storm" is the most influential unit in this query and gives the highest weight to it. In this way, the generated query vector is more likely to be connected with documents about the corresponding movie. In our experiment, the learning-to-rank framework trained without VG (Base + VPCG) ranks a science news from NASA which is not relevant at all as a top result (number three). With the help of VG (Base + VPCG&VG), we get more reasonable results, where all the top three documents are about the movie (Table 6).

6. RELATED WORK

The most straightforward way to exploit the click log from a search engine is to aggregate statistics such as click probability and dwell time by $(query, document)$ pairs and use them as features in a learning-to-rank framework, as described by Agichtein et al. in [1].

To deal with the noise and sparsity problem in these features, there are two basic approaches. One solution is to cast it as a smoothing problem, as in [9], [13], and [29]. An alternative way is to learn a semantic representation of queries and documents, either in vocabulary space or in a latent space, as in [12], [25], and [27]. One advantage of the semantic representation approach is its flexibility. It is easier to incorporate other useful information in the learning of the semantic representation. Besides, the relevance between queries and documents only depends on the representation, so similar documents always get close relevance scores, which reduces the bias caused by the raw click-through data. The semantic approach is the one adopted in this paper.

It is common to represent a click log as a graph with edges representing clicks, an early example being Baeza-Yates and Tiberi's work [3]. Our approach is most similar to the label propagation technique of [16]. However, instead of requiring extrinsic labels or features, we propagate parts of the content information (i.e., query or document title) itself.

The semantic click models that we are aware of either make use of modeling techniques that can be difficult to scale to billions of samples, such as statistical machine translation [12], and SVD [27], or else have very many hyperparameters that are difficult to tune [12, 25]. By contrast, our method is strikingly simple and exceedingly scalable. It also has the advantage of producing a representation – a small set of weighted terms from queries or documents – that is amenable to human inspection.

Besides ranking, click-through data can be utilized for various tasks in information retrieval. Click-through history records user behavior from which user's information need can be discovered [24]. It also provides clues to discover potential relations between queries, and can be used to facilitate tasks such as query clustering [4] and query suggestion [2, 10, 17, 18]. Click-through data can also be explored for document representation and organization. In [21], the authors represent documents in the query vocabulary space, where TFIDF is used for weighting. Compared to their method, ours improves both the query and document vector representation by involving more iteractions between them.

7. CONCLUSION

This paper presents a unified approach to learn query-document relevance by utilizing both click and content information. For queries and documents in the click graph, we propagate the content information from either side on the click-through bipartite graph and represent both queries

and documents as vectors in the same semantic space. For queries and documents without click information, their vector representations are generated by connecting them with the vectors learned from the click graph based on the content information. This whole process expands the vector representations with relevant new words through the propagation, and smoothes the term weights based on the click information. Besides, the sparsity and noise are also reduced during this process thus better quality and larger coverage of the vectors is guaranteed. The proposed approach works efficiently on a Web-scale click graph. The experimental results show the effectiveness of the proposed method in a ranking task. It stands out from thousands of features in a learning-to-rank framework, which is a nontrivial achievement in real practice.

Besides ranking, one potential application of our method is query and document expansion. The learned vectors usually consist of related terms propagated from their neighbors in the click graph, which can be further applied to improve automatic expansion of queries and documents in the search task.

8. REFERENCES

[1] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *Proceedings of SIGIR*, pages 3–10, 2006.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Proceedings of Workshop at EDBT*, pages 588–596, 2005.

[3] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proceedings of SIGKDD*, 2007.

[4] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of SIGKDD*, pages 407–416, 2000.

[5] M. Bendersky, D. Metzler, and W. B. Croft. Effective query formulation with multiple information sources. In *Proceedings of WSDM*, pages 443–452, 2012.

[6] A. Broder, E. Gabrilovich, V. Josifovski, G. Mavromatis, D. Metzler, and J. Wang. Exploiting site-level information to improve web search. In *Proceedings of CIKM*, pages 1393–1396, 2010.

[7] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning*, 2011.

[8] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In *Proceedings of CIKM*, pages 704–711, 2005.

[9] N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of SIGIR*, pages 239–246, 2007.

[10] H. Deng, M. R. Lyu, and I. King. A generalized co-hits algorithm and its application to bipartite graphs. In *Proceedings of SIGKDD*, pages 239–248, 2009.

[11] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[12] J. Gao, X. He, and J. Nie. Clickthrough-based translation models for web search: from word models to phrase models. In *Proceedings of CIKM*, 2010.

[13] J. Gao, W. Yuan, X. Li, K. Deng, and J. Nie. Smoothing clickthrough data for web search ranking. In *Proceedings of SIGIR*, 2009.

[14] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[15] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *Proceedings ICML*, 2015.

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

[17] H. Ma, H. Yang, I. King, and M. R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proceedings of CIKM*, pages 709–718, 2008.

[18] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *Proceedings of CIKM*, pages 469–478, 2008.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Processings of Workshop at ICLR*, 2013.

[20] C. Müller and I. Gurevych. A study on the semantic relatedness of query and document terms in information retrieval. In *Proceedings of EMNLP*, pages 1338–1347, 2009.

[21] B. Poblete and R. Baeza-Yates. Query-sets: using implicit feedback and query patterns to organize web documents. In *Proceedings of WWW*, pages 41–50, 2008.

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

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

[24] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of SIGIR*, pages 43–50, 2005.

[25] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceeding of WWW*, 2014.

[26] K. M. Svore and C. J. Burges. A machine learning approach for improved bm25 retrieval. *Proceedings of CIKM*, 2009.

[27] W. Wu, H. Li, and J. Xu. Learning query and document similarities from click-through bipartite graph with metadata. In *Proceedings of WSDM*, pages 687–696, 2013.

[28] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of SIGIR*, pages 334–342, 2001.

[29] M. Zhukovskiy and T. Khatkevich. An optimization framework for propagation of query-document features by query similarity functions. In *Proceedings of CIKM*, 2015.