Scalable Semantic Matching of Queries to Ads in Sponsored Search Advertising

Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, Ricardo Baeza-Yates
Yahoo Labs, Sunnyvale, CA, USA
{mihajlo, nemanja, vladan, silvestr, rby}@yahoo-inc.com

Andrew Feng, Erik Ordentlich, Lee Yang, Gavin Owens
Yahoo Hadoop Group, Sunnyvale, CA, USA
{afeng, eord, leewyang, gowens}@yahoo-inc.com

ABSTRACT

Sponsored search represents a major source of revenue for web search engines. This popular advertising model brings a unique possibility for advertisers to target users' immediate intent communicated through a search query, usually by displaying their ads alongside organic search results for queries deemed relevant to their products or services. However, due to a large number of unique queries it is challenging for advertisers to identify all such relevant queries. For this reason search engines often provide a service of advanced matching, which automatically finds additional relevant queries for advertisers to bid on. We present a novel advanced matching approach based on the idea of semantic embeddings of queries and ads. The embeddings were learned using a large data set of user search sessions, consisting of search queries, clicked ads and search links, while utilizing contextual information such as dwell time and skipped ads. To address the large-scale nature of our problem, both in terms of data and vocabulary size, we propose a novel distributed algorithm for training of the embeddings. Finally, we present an approach for overcoming a cold-start problem associated with new ads and queries. We report results of editorial evaluation and online tests on actual search traffic. The results show that our approach significantly outperforms baselines in terms of relevance, coverage, and incremental revenue. Lastly, we open-source learned query embeddings to be used by researchers in computational advertising and related fields.

Keywords

Sponsored search; ad retrieval; word embeddings.

1. INTRODUCTION

The remarkable growth of the Internet in the last decades has brought huge benefits to its users, allowing easy, one-click access to all kinds of information and services. However, along with the increase in size and variety of the web there inevitably came the increase in its complexity, which left many users lost in the immense digital jungle of web content. To mitigate the situation, very early in the web's existence there appeared search engines such as Altavista, AOL, or Lycos, that allowed users to easily browse and discover webpages of interest. The service has evolved and improved significantly since these beginnings, and today search engines represent one of the most popular web services.

Advertisers are interested in making use of the vast business potential that the search engines offer. Via issued search query users communicate a very clear intent that allows for effective ad targeting. This idea is embodied in the sponsored search model [19], where advertisers *sponsor* the top search results in order to redirect user's attention from original (or *organic*) search results to ads that are highly relevant to the entered query. Sponsored search drives significant portions of traffic to websites, and accounted for 46% of overall online advertising spend of astonishing $121 billion in 2014 alone[1]. Moreover, with the advent of handheld devices the sponsored search also made a successful transition to mobile platforms. Here, it equally shares the advertising market with display ad formats, and is projected to reach $12.85 billion spend in 2015[2].

The booking process for ad campaigns in sponsored search is typically self-served. This means that the advertisers create their own ads by providing ad creative to be shown to the users (comprising concise title, description, and display URL), along with ancillary ad parameters that are visible only to the web search publisher. These include the list of bid terms (i.e., queries for which advertisers wish to show their ad) and their bid values (i.e., monetary amounts they are willing to pay if the ad is shown and clicked). In the model currently used by most major search engines, in a case that multiple advertisers are competing for the same query the selected ads enter a generalized second price (GSP) auction [10], where the winner pays the runner-up's bid value when a user clicks on the shown ad.

The principal strategy for ad selection is based on explicit matching of user search queries to all the bid terms in the system, referred to as *exact match*. Provided that the advertiser-defined bid terms are indeed relevant to the ad, exact match ensures that advertisers interact only with the relevant audience. However, as it is nearly impossible for advertisers to explicitly list all commercially relevant bid

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

DOI: http://dx.doi.org/10.1145/2911451.2911538

[1] zenithoptimedia.com, accessed January 2016
[2] emarketer.com, accessed January 2016

Table 1: Examples of variant and broad match queries

Ad title	Host a Fun Murder Mystery Party
Ad description	Huge selection of fun murder mystery games for all ages, groups. #1 site for instant downloads and boxed sets of exciting murder mystery party games!
Bid term	murder mystery party
Variant matches	murder mystery parties murder mistery party
Broad matches	murder mystery dinner murder mystery parties at home murder mystery dinner party how to host a mystery party friends game night detective games murder mystery games for parties

terms, it may also limit user reach and lead to lower traffic volume for the advertiser [14]. This can clearly result in lost opportunities and revenue for both the search engine and the advertisers. To address this problem a service of *advanced matching* is usually offered to the advertisers.

More specifically, with advertiser's permission search engines attempt to match ads to related queries not explicitly provided by the advertisers, by implementing *variant* and *broad* match techniques (illustrated in Table 1). Variant match looks for slight variations of the bid term (e.g., typos, inflections), typically by leveraging Levenshtein distance [24]. Broad match is a more involved process of matching queries to advertiser-provided bid terms that do not necessarily have any words in common, yet have same or similar meaning (e.g., query "running machine" and bid term "elliptical trainer"). Existing techniques include expanding user query to find similar queries that can be matched exactly to existing bid terms, commonly referred to as query rewriting [4, 15], or computing similarity between query and text in ad title and description. Query rewriting approaches are limited to matching only against defined bid terms and may not be suitable in cases when advertisers do not provide relevant bid terms to begin with. On the other hand, matching queries to ads based solely on textual information is problematic in case when no common words can be found.

In this paper we describe a novel broad match technique that was recently deployed in sponsored search system of Yahoo Search, which effectively and efficiently addresses the shortcomings of the existing broad match techniques. Motivated by recent advances in distributed neural language models [2, 8, 25, 28], we aim at learning low-dimensional continuous representations of queries and ads in a common vector space. To do so we rely on the search sessions, defined as sequences of queries, ad clicks and search link clicks performed by a particular user, and leverage surrounding context of queries and ads in a session to learn vectors that best describe them. As a result ads and queries which occurred in similar contexts will have similar representations, and finding relevant ads for a specific query amounts to a simple K nearest neighbor search in the learned embedding space, without ever considering the text in queries and ads.

Compared to our previous work [15], novel contributions of this paper are summarized below:

- **Implicit negatives**. We incorporated implicit negative signal (in terms of skipped, non-clicked ads) into the training procedure, leading to higher quality broad matches compared to training with random negatives;
- **Dwell time weights**. We integrated dwell time (i.e., time spent on the advertiser's page following an ad click) into the training procedure, allowing us to weight short and long clicks differently, according to a strength of the interest signal;
- **Large-scale training**. To address the scalability problems due to training data size and the need for large vocabulary (amounting to hundreds of millions of unique queries and ads), we proposed a novel distributed algorithm for training embeddings using parameter server framework that results in 10x training time speedup and over 5x bigger vocabulary compared to single machine training described in [25];
- **Cold-start ad embeddings**. We solved an important problem of computing embeddings for ads that were not observed in training, such that newly booked ads can be instantly matched to user queries;
- **Cold-start query embeddings**. We proposed a procedure for calculating high-quality embeddings for new and tail queries using embeddings of head queries;
- **Experiments on real search traffic**. We provide insights on how different state-of-the-art algorithms perform when applied to actual search traffic;
- **Open-source vectors**. As part of this work we open-sourced a portion of query embeddings for research purposes.

We trained a search embedding model using more than 9 billion search sessions, resulting in ad and query representations of very high quality. Extensive evaluation on real-world search traffic showed that the proposed approach significantly outperformed the existing state-of-the-art broad match methods on a number of critical performance measures, including relevance, reach, and click-through rate (CTR).

2. RELATED WORK

In the following we introduce related work on advanced query-ad matching in sponsored search, and recent advances in neural language models and distributed embeddings.

2.1 Neural language models

It has been shown that a number of problems in Natural Language Processing (NLP) domain can be effectively solved by defining a probability distribution over word sequences, shown to perform well in chunking, parsing, or sentiment analysis, to name a few. For this reason researchers have proposed a number of methods called language models to mathematically represent generation of language, aimed to capture statistical characteristics of written language [1, 22]. Classic methods for language modeling represented words as high-dimensional, sparse vectors using one-hot representation. Using this approach word vectors have dimensionality equal to the size of the entire vocabulary, with zero values everywhere except for the element corresponding to the given word. However, the approach often gives suboptimal results due to sparsity issues and curse of dimensionality.

In order to mitigate the problems with word representations in high-dimensional spaces, neural language models

have been introduced. Unlike classical approaches, these methods learn distributed, low-dimensional representations of words through the use of neural networks [2, 8, 28]. The networks are trained by directly taking into account the word order and their co-occurrence, based on the assumption that words frequently appearing together in the sentences also share more statistical dependence.

Despite their effectiveness, training a large number of parameters of neural network-based approaches has been a serious obstacle to wider use of the neural models. In particular, a single parameter update requires iteration over the entire vocabulary, which can grow prohibitively large for many practical applications. However, with the recent advances in the NLP domain, and in particular with the development of highly scalable continuous bag-of-words (CBOW) and skip-gram (SG) language models for word representation learning [25], the models have been shown to obtain state-of-the-art performance on many traditional language tasks after training on large-scale textual data sets.

2.2 Broad match in sponsored search

Broad match is a well established, often used approach in sponsored search advertising, responsible for billions of dollars of search engine revenue. In addition to exact and variant match, which typically have the highest click-through rates, many advertisers use broad match to increase the reach to relevant queries that were not explicitly provided, entrusting search engines with this task.

Most of the existing broad match techniques rely on query rewriting to rewrite an unmatched user query to several similar queries, in expectation that one of the rewrites will match an advertiser-provided bid term, resulting in retrieval of ads that bidded on that term. A basic query rewriting approach involves representing queries as bag-of-words using tf-idf weighting, and calculating similarity between the query and all other queries in order to find good rewrite candidates. However, the bag-of-words representation of queries is extremely sparse, which makes it impossible to find related queries that do not share common words. Moreover, it can lead to false positives when queries share common words that have different meanings in different contexts, e.g. "house music" and "house prices". To mitigate these issues some researchers proposed to enrich query representation using text from web search results [7]. Others proposed to extend query rewriting beyond string matching by applying graph-based methods, such as Query Flow Graph (QFG) [4], where similar queries are found by a random walk over a bipartite graph of queries and link clicks [11], or neural networks [15], where query feature representations are learned from web search data. However, none of these methods can overcome an inherent limitation of the query rewriting approach. More specifically, once the query is rewritten it can only be matched to ads that have the resulting rewrites as bid terms, potentially overlooking other relevant ads.

Alternative methods aim to address this issue by directly retrieving ads by calculating similarity between bag-of-words representation of queries and bag-of-words representation of ads obtained from ad title and description. Even though it reduces sparsity, this approach does not completely solve the vocabulary mismatch problem [26] and may even introduce additional issues, especially in the case of *templated* ad creatives where majority of phrases in titles and descriptions have little to do with the advertised product (e.g., "free shipping", "best prices", "buy now").

Finally, to rank the ads by relevance (e.g., in the case too many ads are retrieved via query rewriting), it is common to apply the learning-to-rank approach [16] that learns weights for query and ad features based on editorially-labeled training set. These methods typically require enormous amounts of accurate editorial grades for (query, ad) matches that are often very expensive to obtain.

In this paper we go beyond query rewriting and supervised learning-to-rank, and propose to learn query and ad embeddings from search data in an unsupervised manner, followed by directly matching and ranking ads for queries based on the distances in the learned embedding space.

3. METHODOLOGY

To address the shortcomings of existing broad match algorithms in sponsored search, we propose to take a new approach to this task, motivated by the recent success of distributed language models in NLP applications [25, 28]. In the context of NLP, distributed models are able to learn word representations in a low-dimensional continuous vector space using a surrounding context of the word in a sentence, where in the resulting embedding space semantically similar words are close to each other [25]. Our objective is to take advantage of this property for the task of query-ad matching in sponsored search, and to learn query and ad representations in a low-dimensional space where queries would be close to related ads. This would allow direct matching of queries to ads, instead of taking the longer route of query rewriting. Clearly, such approach reduces the complex broad match task to a trivial K-nearest-neighbor (K-nn) search between queries and ads in the joint embedding space.

Finding distributed query and ad representation, as opposed to finding word representations, brings very unique challenges. For example, while the basic unit for learning word representations is a sentence $s = (w_1, \ldots, w_M)$ consisting on M words $w_m, m = 1, \ldots, M$, in our proposed approach the basic unit for learning query and ad representations are user actions within a search session $s = (a_1, \ldots, a_M)$, where in the simplest case an action can be a search query or an ad click. Moreover, search sessions have a number of additional contexts that can be used to improve quality of the final model. For instance, search sessions typically contain link clicks. Even though we are only interested in query and ad embeddings, link clicks can provide additional context. In essence, the query and ad embeddings will be affected by the co-click behavior between queries and search link clicks, resulting in an improved model. In addition, ad dwell time can be leveraged to distinguish between good ad clicks and unsatisfactory or accidental clicks [21], and can provide useful context for training better quality embeddings. Finally, ads that are skipped in favor of a click on a lower positioned ad can be used as implicit negative signal and serve as negative context during training to improve the resulting model.

To formalize the training procedure, let us assume we are given a set $\mathcal{S}$ of S search sessions obtained from N users, where each session $s = (a_1, \ldots, a_M) \in \mathcal{S}$ is defined as an uninterrupted sequence of M user actions comprising queries, ad clicks, and link clicks. A new session is initiated whenever there is a time gap of more than 30 minutes between two consecutive user actions [12]. Then, the learning ob-

jective is to find D-dimensional real-valued representation $\mathbf{v}_{a_m} \in \mathbb{R}^D$ of each unique user action a_m.

The proposed search embedding model (called *search2vec*) learns user search action representations using the skip-gram model [25] by maximizing the objective function $\mathcal{L}$ over the entire set $\mathcal{S}$ of search sessions, defined as follows,

$$\mathcal{L} = \sum_{s \in \mathcal{S}} \sum_{a_m \in s} \sum_{-b \leq i \leq b, i \neq 0} \log \mathbb{P}(a_{m+i}|a_m). \tag{3.1}$$

Probability $\mathbb{P}(a_{m+i}|a_m)$ of observing a neighboring user action a_{m+i} given the current action a_m is defined using the soft-max function,

$$\mathbb{P}(a_{m+i}|a_m) = \frac{\exp(\mathbf{v}_{a_m}^\top \mathbf{v}'_{a_{m+i}})}{\sum_{a=1}^{|\mathcal{V}|} \exp(\mathbf{v}_{a_m}^\top \mathbf{v}'_a)}, \tag{3.2}$$

where $\mathbf{v}_a$ and $\mathbf{v}'_a$ are the input and output vector representations of user action a, hyperparameter b is defined as a length of the relevant context for action sequences, and $\mathcal{V}$ is a vocabulary defined as a set of unique actions in the data set comprising queries, ads, and links. From (3.1) and (3.2) we see that search2vec models temporal context of action sequences, where actions with similar contexts (i.e., with similar neighboring actions) will have similar representations.

Time required to compute gradient $\nabla\mathcal{L}$ of the objective function in (3.1) is proportional to the vocabulary size, which may be computationally infeasible in practical tasks as $|\mathcal{V}|$ could easily reach hundreds of millions. As an alternative we used negative sampling approach proposed in [25], which significantly reduces computational complexity. Negative sampling can be formulated as follows. We generate a set $\mathcal{D}_p$ of pairs (a, c) of user actions a and their contexts c (i.e., actions within a window of length b that either precede or follow action a), and a set $\mathcal{D}_n$ of negative pairs (a, c) of user actions and n randomly sampled actions from the entire vocabulary. The optimization objective then becomes,

$$\underset{\theta}{\operatorname{argmax}} \sum_{(a,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_a}} + \sum_{(a,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_a}}, \tag{3.3}$$

where parameters θ to be learned are $\mathbf{v}_a$ and $\mathbf{v}'_c$, $a, c \in \mathcal{V}$. The optimization is done via stochastic gradient ascent.

Visual representation of the search2vec model is presented in Figure 1. As discussed previously, and as depicted in Figure 1, we propose to utilize additional context specific to sponsored search domain, namely implicit negatives and dwell-time. In the experimental section we explore this approach and empirically evaluate their benefits in detail.

Leveraging ad dwell time. Let us assume that a user performed a search query q_1, followed by an ad click ad_1. We define *ad dwell time* as the time that the user spent on the ad landing page. When the user stayed longer on the ad landing page, it is usually interpreted as a positive engagement signal, as opposed to the user immediately bouncing back from the advertiser's website. There also exists a notion of accidental ad clicks with dwell time of only a few seconds, which is especially apparent on mobile phone devices. To incorporate this signal into our learning framework we propose to weight query-ad pairs with higher dwell times more, while penalizing shorter dwell times.

More specifically, when optimizing the objective function in (3.3), we introduce an additional weight $\eta_t = \log(1 + t)$ for each element of the first sum, where t denotes the dwell time in minutes. For dwell times above 10 minutes we set $\eta_t = 1$. Moreover, we use this approach to compute weight only when updating the ad vector with the immediately preceding query and when updating the query vector with the immediately succeeding ad, otherwise we set $\eta_t = 1$.

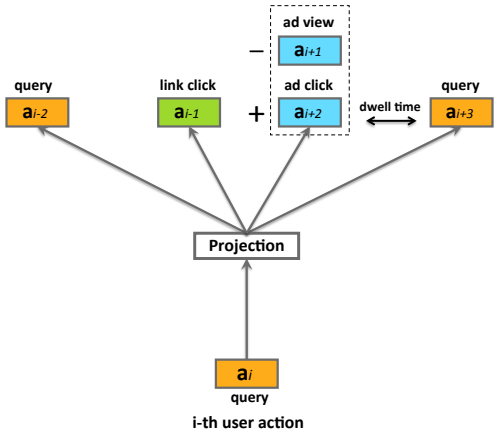

Figure 1: search2vec: dwell-time implicit negative skip-gram

Leveraging implicit negative signals. Unlike in the NLP domain, in the domain of sponsored search there does exist an implicit negative signal in a form of ads that the user decided to skip in favor of a click on a lower positioned ad, which can be utilized during training. In particular, given an issued user query q_1, let us assume that after the GSP auction a total of k ads, $ad_1, ad_2, \ldots, ad_k$, were shown at different positions on the page, with ad_1 being shown at the top position in the page followed by the remaining $k-1$ ads below. Note that the top position has the highest value to the advertisers, as it has been shown to have the highest probability of being clicked, especially on mobile devices due to limited screen sizes. We adopt an assumption that if the user scrolls down the page and clicks on a lower positioned ad, all the ads ranked higher were of no interest to him and can be treated as negative signal for query q_1. We consider implicit negatives only on top $k = 3$ positioned ads, and only in cases where user had a single ad click in the session with dwell time of over 10 seconds.

We formulate the use of implicit negatives in the following way. In addition to sets $\mathcal{D}_p$ and $\mathcal{D}_n$, we generate a set $\mathcal{D}_{in}$ of pairs (q, ad) of user queries q and the ads ad at higher positions which were shown to the user, and skipped over in favor of a click on ad at one of the lower positions. The new optimization objective then becomes as follows,

$$\underset{\theta}{\operatorname{argmax}} \sum_{(a,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_a}} + \sum_{(a,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_a}} + \sum_{(q,ad) \in \mathcal{D}_{in}} \log \frac{1}{1 + e^{\mathbf{v}'_{ad} \mathbf{v}_q}}, \tag{3.4}$$

where parameters θ to be learned remain the same as before.

Inference. Given learned query and ad embeddings, ad retrieval for a given query q is performed by calculating cosine similarity between its vector $\mathbf{v}_q$ and all ad vectors $\mathbf{v}_{ad}$ in the vocabulary. The K ads with the highest similarity are retrieved as broad matches. For large vocabularies, efficient K-nn search can be done via locality-sensitive hashing [13].

3.1 Scalable training algorithm

To realize the full potential of the proposed approach, we found it necessary to train embeddings for several hundred million of actions (i.e., *words* in the "vocabulary"), which include queries, ads, and links comprising the search session data. Existing implementations for training word embeddings were found to fall short for our vocabulary size target, as they require that all of the input and output vectors fit in the memory of a single machine. For example, a vocabulary of 200 million words with 300-dimensional vectors requires 480GB of RAM memory, which is beyond the capacity of typical machines. To address this issue, we developed a novel distributed word embedding training algorithm based on a variation of the parameter server paradigm.

A parameter server (PS) [9] is a high-performance, distributed, in-memory key-value store specialized for machine learning applications, typically used to store latest values of model parameters throughout the course of training. A PS-based training system includes a number of PS shards that store model parameters and a number of clients that read different portions of the training data from a distributed file system (e.g., Hadoop Distributed File System).

In the conventional PS system, the word embedding training can be implemented as follows. Each PS shard stores input and output vectors for a portion of the words from the vocabulary. Each client reads a subset of training data in minibatches (e.g., 200 lines at a time), and after each minibatch performs the vector updates of words found in the minibatch and randomly sampled negative words. The updates are achieved by transmitting the IDs of the words to all the PS shards, which will, as a result, send back the corresponding vectors to the client. The client calculates the gradient descent updates with respect to the minibatch restriction of the objective (3.3) and transmits them to the PS shards where the actual updates happen.

However, due to transmission costs of the actual word vectors over the network, the conventional PS-based implementation of word embedding training requires too much network bandwidth to achieve acceptable training throughput. For this reason we propose a different distributed system architecture that requires significantly less network bandwidth. The proposed algorithm, illustrated in Figure 2, features the following contributions with respect to the conventional PS-based training approach:

- Column-wise partitioning of vectors among PS shards;
- No transmission of word vectors across the network;
- PS shard-side negative sampling and computation of partial vector dot products.

In our algorithm, each PS shard stores distinct portion of vector dimensions for *all* words from the vocabulary $\mathcal{V}$, as opposed to previously described word-wise partitioning where each PS shard stores entire vectors for a subset of words. For example, assuming a vector dimension $d = 300$ and 10 PS shards, shard $s \in \{0, \ldots, 9\}$ would store 30 dimensions of input vectors $\mathbf{v}(w)$ and output vectors $\mathbf{v}'(w)$ with indices i in the range $30s \leq i < 30s + 30$ for all vocabulary words.

Since each PS shard now stores the entire vocabulary (i.e., different vector dimensions of *all* words), we can implement the random sampling step and the dot product calculation step to the PS shards themselves. In this way we avoid transmission costs of the actual vectors. Each client still processes

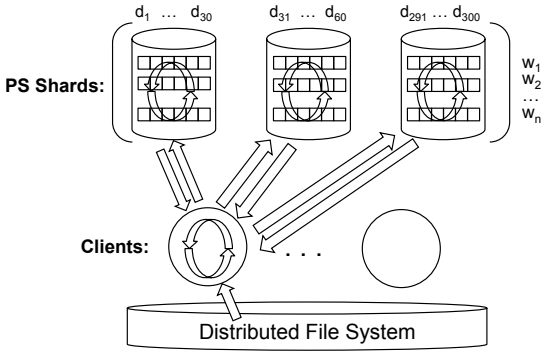

Figure 2: Scalable word2vec on Hadoop

data in minibatches, and sends to the PS shards the indices of (word, context) pairs that need to be updated. In addition, the client broadcasts common random number generator seeds, such that all shards can sample the same negative context words. Each PS shard then calculates the partial dot products with respect to their local dimensions for each of the positive and negative (word, context) pairs and sends the result to a client that made the request. Finally, the client aggregates local dot products it received from all PS shards and calculates the global coefficient needed for gradient updates. The single client's minibatch step ends by sending the global coefficients to all PS shards, where input and output vectors are updated correspondingly, without any locking or synchronization mechanism.

In a typical at-scale run of the algorithm, the above process is carried out by multiple client threads running on each of a few hundred cluster nodes, all interacting with the PS shards in parallel. The dataset is iterated over multiple times, and after each iteration the learning rate α is reduced. The distributed implementation handles implicit negatives in the same manner as positive contexts only with negative multiplier in the gradient update, and can handle dwell-time weights seamlessly. In comparison to [25], our system was found to be up to 10x faster depending on the load on the shared cluster, training set, and vocabulary size.

3.2 Cold start ad embeddings

Contrary to everyday language where new words come into existence occasionally, in ad serving business new ads are introduced to the system every day. Matching service is required to find good query candidates for these new ads even before they are shown to users. In this section we propose a method that addresses this problem by producing cold start ad embeddings of good quality, which are then used to find relevant queries for that ad.

Given a newly booked ad, in absence of ad clicks that could be used to learn a context vector, we must rely on its textual content, including the bid term, ad title, and description. In particular, in the process of learning query context vectors V_q we learned embeddings for a large number of words and phrases, many of which appear in ad titles and descriptions. In addition, the bid term context vector is likely to be available (in 80% of cases by our evaluation).

The bid term context vector itself could be a good choice for the cold start ad embedding. However, even though the bid term should summarize the message being conveyed in the ad, oftentimes advertisers bid on general terms, such as "shoes", "fitness", or "sports", to maximize their traffic po-

Data: ad title, ad description, ad URL, bid term b, threshold τ_c, set of context query vectors V_q
Initialization: $\tau_c = 0.45$, get bid term vector $\mathbf{v}_b$ from V_q, set $\mathbf{v}_{ad} = \mathbf{v}_b$, create ad document $\mathbf{d}_{ad}$ from all n-grams ($n = 1, .., 10$) in ad title, description, URL;
forall the *phrases p in ad document* $\mathbf{d}_{ad}$ **do**
  if $\mathbf{v}_p$ *exists in* V_q **then**
    if *sim*($\mathbf{v}_b$,$\mathbf{v}_p$)$> \tau$ **then**
      $\mathbf{v}_{ad} = \mathbf{v}_{ad} + \mathbf{v}_p$;
    end
  end
end
Result: ad content vector $\mathbf{v}_{ad}$

Algorithm 1: content search2vec

tential. Evidently, one could find additional useful phrases in the ad title and description, whose vectors, in combination with the bid term vector, could produce an embedding that is closer to more specific query matches for that ad.

To compute a content-based ad vector we propose to extract relevant words and phrases from ad content and compute the linear combination of their context vectors,

$$\mathbf{v}_{ad} = \sum_{t=1:T_a} \mathbf{v}_{p_t}, \tag{3.5}$$

where T_a is the number of extracted words and phrases with vectors $\mathbf{v}_{p_t}$ in V_q. The resulting ad vector is of the same dimensionality and in the same feature space as the query context vectors. Consequently, the same retrieval procedure as the one used for context ad vectors can be used to find relevant broad match queries for the cold-start ads.

The most challenging part of this procedure is extracting relevant words and phrases from the textual ad information. For example, let us consider an ad with title "Visit Metropolitan Opera House in New York" and bid term "performing arts". While phrases "opera house" and "metropolitan opera house" are relevant in the context of this ad, other phrases such as "visit", "house", or "house in new york" do not carry any useful information or have completely different meaning from what is communicated in the ad. For example, phrase "visit" is too general and "house in new york" refers to real-estate in New York with no connections to performing arts. In addition, ads often contain generic phrases such as "free shipping", "buy now", "30% off", and "best deals on", which if used in the final sum (3.5) may cause the resulting vector to diverge from relevant queries in the embedding space, leading to poor broad matches.

To decide which n-grams should be formed from words in short text, a typical approach is to use a CRF segmentation model [20]. However, CRF models have their limitations: 1) they require a large training set with human-annotated examples of correct text segmentations, which needs to be constantly updated to account for new entities; 2) they cannot automatically detect phrases that should not be considered and would require creating a blacklist of terms that should be excluded from (3.5). Similar shortcomings are present in the paragraph2vec model [23] that suggests simply using all words in a document to learn its embedding.

To this end we propose an *anchor phrases* model, summarized in Algorithm 1, that uses search context vectors to automatically filter out stopwords and non-relevant phrases and keep only the ones closely related to the *anchor*. The choice of the anchor falls on the advertiser-defined bid term, as that is the one phrase we know is related to the ad. In rare cases when anchor context vector is not available, one can be constructed as described in Section 3.3. The algorithm uses context vectors of queries learned from search sessions. It initializes the content ad vector to the context vector of the bid term. Next, it creates a textual ad document from all possible n-gram phrases contained in the ad title, description, and URL. Finally, it considers all phrases from the ad document for which a context vector exists and has a high enough cosine similarity to the bid term vector (above threshold τ_c). These relevant phrases are then added to the ad vector to further focus its representation. The resulting ad vector lies in the same embedding space as the search queries. Therefore, relevant broad match query candidates can be found via K-nearest neighbor search. We refer to this method as *content search2vec*.

3.3 Cold start query embeddings

Quality of the query embedding is contingent on the number of times it was observed in the training data. Therefore, even though our scalable training procedure allows us to train vectors for several hundred millions of queries, we are still limited by the frequency of queries in search sessions. Following the procedure from [25] we train context vectors for queries that occur at least 10 times. Considering that frequency of queries in search logs follows a power low distribution [27], a large number of queries will not meet this frequency requirement. They are referred to as *tail* queries, as opposed to *head* and *torso* queries that occupy the portion of the power law curve for which a context vector can be learned. While individually rare, *tail* queries make up a significant portion of the query volume. For this reason, these queries have significant potential for advertising revenue.

Most major search engines pre-compute broad matches for a large number of *head* and *torso* queries. This table T_b is then cached for fast access and retrieval. The nonexistence or lack of presence in search sessions makes *tail* queries much harder to match against ads [6], as they lack useful contexts of surrounding web activities to help in matching. Matching them to ads solely based on textual information results in matches of low quality, primarily due to the so-called vocabulary mismatch problem [26]. Finally, short text in queries introduces limitations into producing query embeddings based on their content.

To address this problem we propose to expand the head queries from table T_b with K nearest neighbor queries in the learned embedding space and build an *inverted index* to be used for efficient matching against tail queries. In the online phase, when a new user query arrives that has no matched ads we use the inverted index to map the query into related previously-seen queries that have ads associated with them. Finally, the user query inherits ads of the top matched query. To illustrate the matching procedure, in Table 2 we show examples of tail queries and top matched queries, obtained via inverted index. Bolded words are the ones matched against the user query. We refer to this method as *elastic search2vec*.

Previous attempts at building an inverted index of head queries for dealing with tail queries were based on features extracted directly from words of the head query [6] or from words of queries that co-occurred in the same sessions as the head query [5], such as typos and reformulations. While words from query are good representations of its syntax, they fail to accurately represent its semantics. On the other

Table 2: Demonstration of elasting search2vec

User search query (not present in V_q)	Matched query-document	Query-document features (K-nn queries)
metropolitan opera house that is in new york city	metropolitan opera house	**metropolitan opera house**, ny **opera** **new york opera**, **new york opera house** **metropolitan opera**, nyc **opera house**
best flight deals that travel to paris france	best flight tickets to paris	**best flight** tickets to **paris**, cheap flights to **paris** cheap tickets to **paris**, **best flight** fares to **paris** cheap airfare **paris france**, **travel to paris france**
what is the best stock ticker trading app in appstore	best stock ticker apps	**best stock ticker** apps, **stock** tracker **app** in **appstore** **best stock** apps in **appstore**, **stock ticker** apps **best trading** apps, real time stocks on the **app** store

hand, words from co-occurring queries may not be the best choice as semantically similar queries often have similar contexts (e.g., same link and ad clicks), but do not necessarily appear in the same session. For this reason, we build the inverted index by considering words of semantically similar queries obtained by K-nn search in the embedding space V_q. As our experiments confirm, this leads to more accurate cold-start query embeddings and better quality ad matches.

4. EXPERIMENTS

In this section we describe the training data set and give empirical evaluation of the proposed search2vec method. The performance was evaluated in terms of relevance on editorially judged set of (query, ad) pairs, as well as in terms of click-through rates from a live bucket tests on Yahoo Search traffic. We compared several variations of our method discussed in the previous sections, as well as a number of baseline approaches. Finally, we conducted experiments to evaluate the proposed models for cold-start ad and query embeddings that address the limitations of the context models by allowing us to produce matches for new queries and ads.

Search2vec training. In order to learn query and ad embeddings we organized our search data into sessions, or uninterrupted sequences of user's web search activities: queries, ad clicks, and search link clicks. Each ad impression was uniquely identified by its creative ID and bid term ID. Search sessions that contained only 1 user action were discarded. We discarded from the vocabulary all activities that occurred less than 10 times in the search sessions. The most frequent actions were downsampled with threshold of 10^{-5} as in [25]. The dimensionality of the embedding space was set to $d = 300$, the window size was set to 5 and the number or random negative samples per vector update was set to 5.

We learned embeddings for more than 126.2 million unique queries, 42.9 million unique ads, and 131.7 million unique links, using one of the largest search data set reported so far, comprising over 9.1 billion search sessions collected on Yahoo Search. End-to-end search2vec training via parameter server using 10 iteration over the data set takes about 30 hours to complete. In comparison, multi-threaded single machine implementation [25] on 256GB RAM box with 24 threads and same training parameters takes more than a week to complete and can train only up to 80 million vectors.

4.1 Broad match models

1) word2vec model uses publicly available word vectors[3] trained on Google News data set. Query vectors were generated by summing vectors of word tokens within a query, and ad vectors were generated by summing vectors of word tokens from title, description, bid term, and display URL. English stopwords were not considered in the summation.

2) TF-IDF model generates broad match candidates by computing the cosine similarity between bag-of-words representation of queries and ads (constructed from ad title, description, bid term, and display URL, excluding stopwords).

3) QFG model was trained using a click-flow graph constructed from search sessions data set $\mathcal{S}$. Broad matches were produced by query rewriting of the ad bid term.

4) search2vec model was trained using search sessions data set $\mathcal{S}$ composing of search queries, ads and links. Broad match candidates for a query were generated by calculating cosine similarity between the query vector and all ad vectors.

5) search2vec$_{dwell,in}$ is a variation of search2vec method with dwell time weights, and implicit negatives added to the training data. We combined dwell time and implicit negative enhancements under one framework as both of them aim at reducing the effect of bad clicks.

6) content search2vec is tailored for the cold-start case (i.e. ads that do not appear in search sessions). Broad match candidates are found by calculating cosine similarity between the context query vector the content ad vectors.

7) elastic search2vec aims at finding ads for tail queries by matching against an inverted index of context queries that already have ad matches.

Our evaluation did not include topic models such as LDA [3] or PLSA [17], as a number of researchers [18] found that they obtain suboptimal performance on short texts.

[3]https://code.google.com/p/word2vec, accessed Jan. 2016

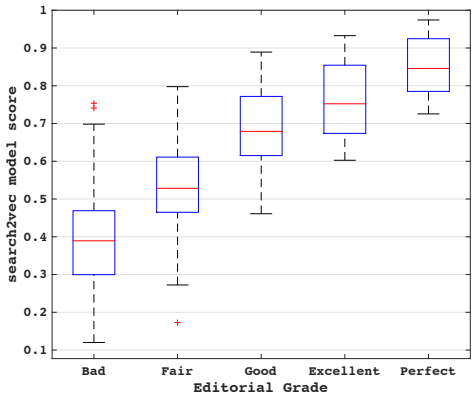

Figure 3: Box plot of editorial grades vs. search2vec scores

4.2 Offline relevance results

In the first set of experiments we used editorial judgments of query-ad pairs to compare broad match methods in terms of relevance. For this purpose we used an in-house

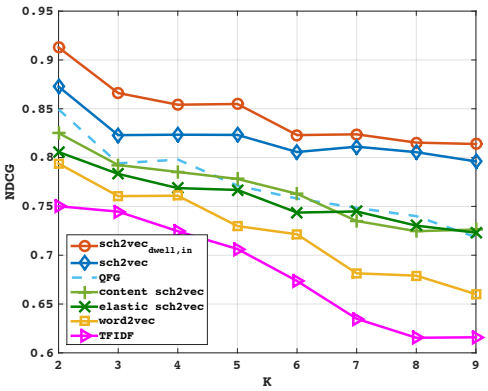

Figure 4: NDCG@K for different models

Table 3: Comparison of different methods on editorial data

Method	oAUC	Macro NDCG
word2vec	0.6573	0.7308
TF-IDF	0.6407	0.6983
QFG	0.6928	0.7848
search2vec	0.7254	0.8303
search2vec$_{dwell,in}$	**0.7392**	**0.8569**
content search2vec	0.6881	0.7758
elastic search2vec	0.6805	0.7612

dataset consisting of a query-ad pairs that were graded editorially. The editors were instructed to grade more than 24,000 query-ad pairs as either *Bad*, *Fair*, *Good*, *Excellent*, or *Perfect* match. For each ad, the editors had access to bid term, ad title, description, and display URL to help them reach their judgment. Note that the candidate query-ad pairs were not generated by any of the tested methods. For each query there were up to 9 judged ads, allowing us to evaluate ranking of ads in addition to relevance.

Using each of the compared methods we generated query features and ad features for each judged pair. Then, to calculate score for a particular pair, we calculated cosine similarity between the query vector and the ad vector. In case of QFG query rewriting model, the query-ad relevance score was calculated between the query and the ad bid term. In case of elastic search2vec model, query vector was obtained by inverse index search that excluded the searched query and calculating the cosine similarity of the top retrieved context query vector and the ad context vector.

There are several questions one could ask regarding algorithm relevance. First, how well can the algorithm score distinguish between good and bad query-ad pairs. Second, how well can the ads be ranked based on the algorithm score, and how much does this ranking deviate from ranking based on editorial grades. For this purpose we concentrated on the following two metrics to measure relevance:

1) oAUC Ordinal AUC measures how well the cosine similarity scores computed by the trained model can discriminate between editorial grades, by computing the average AUC (area under ROC curve) across four classifiers: perfect-and-above vs. below-perfect, excellent-and-above vs. below excellent, good-and-above vs. below good, and lastly fair-and-above vs. below fair.

2) Macro NDCG measures how well the ranked scores align with the ranked editorial grades. Given numerical grades (1 for Bad, 2 for Fair, 3 for Good, 4 for Excellent and 5 for Perfect) we used $(2^{grade} - 1)$ as NDCG labels and position discounting of log(position in ranked list). Finally, we average the metric across all queries. Unlike oAUC, this performance measure incurs a greater penalty for mistakes found at the top of the list.

In Table 3 we report the averaged oAUC and Macro NDCG of editorial query-ad pairs for different methods. In addition, to get a sense of how different models compare at a particular rank we show the NDCG metric at values of rank ranging from 2 to 9 in Figure 4.

Several conclusions can be drawn. First, we can see that the models that did not utilize search sessions (i.e., word2vec and TF-IDF) perform worse than the models that use historical search logs (i.e., QFG and search2vec). Second, search2vec embedding method performs better than QFG graph method. This can be explained by the fact that QFG only makes use of the co-occurrence of user actions in search sessions, while search2vec also accounts for similarity of their contexts. Third, introduction of dwell time and implicit negatives improves relevance over baseline search2vec model.

Interestingly, the increase in oAUC of search2vec$_{dwell,in}$ over search2vec was not as large as the improvement in Macro NDCG. This higher improvement in ranking measure can be explained by the fact that search2vec$_{dwell,in}$ leverages ad views in addition to ad clicks. Therefore, if two ads have the same number of clicks but one of them was viewed an order of magnitude more frequently, search2vec$_{dwell,in}$ will rank it lower than the ad which was shown less times, thus obtaining a better result. Finally, relevance based on cold-start query and ad embeddings produces reasonable results, indicating that they can be successfully used until context vectors can be learned from sessions.

To better characterize the search2vec method, in Figure 3 we show a box plot of cosine similarities between editorially judged queries and ads for search2vec$_{dwell,in}$ model, grouped by different editorial grades. We can see that the cosine similarity scores correlate well with the editorial judgments, indicating a strong potential of the learned embeddings for the task of retrieval of high-quality query-ad pairs.

4.3 Online test results

Following successful offline evaluation of the proposed broad match methods, in the second set of experiments we conducted tests on live Yahoo Search traffic. Five buckets with five different broad match techniques were set up (QFG, search2vec, search2vec$_{dwell,in}$, content search2vec and elastic search2vec), all on 5% of overall search traffic and compared against control bucket, which employed a broad match model currently used in production. Note that, although the buckets used different broad match techniques, all of them still included the same exact match and variant match techniques. The online test was run for 7 days, and was performed on a subset of active sponsored search ads.

Different search2vec methods produced broad matches by finding $K = 30$ nearest ads in the embedding space for each search query from our 126.2M vocabulary, and keeping only ads with cosine similarity above $\tau = 0.65$. The threshold was chosen based on editorial results from Figure 3, where it can be observed that threshold of 0.65 captures most Good, Excellent, and Perfect query-ad pairs while discarding most of

Table 4: Comparison of broad match methods in A/B test

Method	CTR	Coverage
QFG	-	-
elastic search2vec	−4.81%	+43.79%
content search2vec	−1.24%	+35.61%
search2vec	+6.21%	+26.44%
search2vec$_{dwell,in}$	+10.64%	+19.95%

Table 5: Comparison of ad cold-start methods

Method	Average	STD
bid term vector	0.731	0.128
words	0.574	0.059
anchor words	0.688	0.077
phrases	0.665	0.067
CRF phrases	0.604	0.075
anchor phrases	**0.792**	**0.077**

Fair and Bad ones. While search2vec and search2vec$_{dwell,in}$ were restricted to only ads for which we previously observed clicks in historical search logs, content search2vec generated embeddings for all active ads, and it is expected to have higher ad coverage. In the case of QFG method, the retrieval was done via query rewriting. For each query we determined the most relevant bid terms, and retrieved all ad creatives with that bid term, up to $K = 30$ ads per query.

The results of the test in terms of CTR and coverage are summarized in Table 4. For CTR we measured the number of clicks on particular broad match ads, divided by the total number of shown ads by that broad match model. Then, for each broad match model we report a relative improvement of CTR over QFG method, as well as relative change in coverage, defined as a number of ad impressions.

Considering the results, we can conclude that search2vec$_{dwell,in}$ is the best choice among the competing method as it achieves the highest CTR while maintaining a large coverage. When dwell time and implicit negatives were incorporated in search2vec training we observed a large improvement in CTR, with slight coverage drop, which can be explained by the fact that some ad candidates were eliminated due to low dwell time of high number of skips. Content search2vec and elastic search2vec produced a large increase in coverage since they are not restricted to only clicked ads or queries that appeared in search sessions. At the same time they had satisfactory CTR, indicating that they are good alternatives to search2vec$_{dwell,in}$.

One of the highlights of the A/B test with the control bucket was that more than 80% of broad matches produced by search2vec$_{dwell,in}$ were unique to that method (i.e., not found by the production model) while retaining high CTR, which can directly lead to incremental gains in revenue. Our search2vec$_{dwell,in}$ combined with cold start ad embeddings was launched in production and it is currently serving more than 30% of all broad matches.

4.4 Cold start experiments

To evaluate different algorithms for cold start ad embeddings we conducted the following experiment. We used search2vec$_{dwell,in}$ context vectors of queries to construct ad content vectors based on available textual information: title, description, display URL and bid term. To achieve this task, we tested several methods that use different strategies to extract the most relevant keywords from ad text and calculate the embedding as a linear combination of their vectors (3.5). To this end, we evaluated the following techniques: 1) *bid term vector*, which uses the vector of a bid term query as the ad vector 2) *words*, similar to [23] the sum (3.5) uses vectors of all words in the document 3) *phrases*, which uses vectors of all possible n-grams from the ad texr, $n = 1, .., 10$ 4) *CRF phrases*, which uses vectors of phrases obtained by CRF segmentation [20] 5) *anchor phrases*, which is explained in Algorithm 1 and 6) *anchor words*, which is the Algorithm 1 that uses words instead of phrases. In all above-mentioned algorithms, except *anchor* algorithms, a hand curated collection of stopwords (e.g., "free shipping" and "best prices") were dropped to improve performance. The performance was measured in terms of cosine similarity between the ad context vector and its corresponding content vector. High similarity indicates that the cold-start method can produce a vector that is close enough to the one that would be learned from sessions, and could therefore be matched to relevant queries in the embedding space. In Table 5 we show results on a subset of 2M ads. We can observe that *anchor phrases*, which automatically discards uninformative phrases, outperformed the competing baselines by a large margin.

Table 6: Comparison of query cold-start methods

Method	Average	STD
words	0.452	0.101
phrases	0.574	0.120
CRF phrases	0.514	0.119
elastic co-occurred queries, K=10	0.621	0.084
elastic search2vec, K=5	0.685	0.085
elastic search2vec, K=10	**0.717**	**0.091**
elastic search2vec, K=100	0.693	0.089

We repeat a similar procedure for queries. Specifically, we test how well the vectors of last 50M queries in our vocabulary could be generated using vectors of 40M head queries. It should be noted that generating query content vectors is much more challenging as we have less content (i.e., text) to work with and no concept of *anchor* point that could help in extraction. To address this issue we proposed *elastic search2vec* described in Section 3.3. We evaluate 3 different sizes of query expansions when creating the inverted index: K= 5, 10 and 100. In addition to previously mentioned baselines, we also test an inverted index approach similar to [5], where index is generated based on words from K= 10 top co-occurring queries. In Table 6 we summarize the results, where we can observe that the proposed approach with K= 10 achieved the best performance.

4.5 Open-source query embeddings

As part of this research, we open-sourced 8M query vectors trained using search2vec[4]. The vectors may serve as a testbed for query rewriting task as well as to word and sentence similarity task, which are common problems in NLP research. We would like for researchers to be able to produce query rewrites based on these vectors and test them against other state-of-the-art techniques. In addition, we provide an

[4]http://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=73

Table 7: Comparison of query rewriting methods

Method	oAUC	Macro NDCG@5
word2vec	0.817	0.929
search2vec	**0.880**	**0.959**

editorially judged set of 4016 query rewrites, on which we compared search2vec performance against word2vec. The results are summarized in Table 7.

5. CONCLUSIONS AND FUTURE WORK

We proposed a novel broad match method based on neural language models. The method learns low-dimensional representations of search queries and ads based on contextual co-occurrence in user search sessions. To better leverage available web search contexts, we incorporate link clicks, dwell time and implicit negative signals into the training procedure. We evaluated the proposed method using both editorial data and online test on live search traffic. When compared to the baseline approaches, we showed that proposed search2vec model generated the most relevant broad matches between queries and ads, and had the highest CTR. Moreover, we found that in the case of new queries and ads the proposed cold-start embeddings are a good substitute for the learned ones. The results clearly indicate significant advantages of search2vec model over the existing broad match algorithms, and suggest high monetization potential in sponsored search. Finally, to address the scalability issues with regards to data and vocabulary size, we propose a novel distributed algorithm that can achieve 10x training speedup and train 5x larger vocabularies, compared to a single machine algorithm. In our future work we plan to make use of additional search contexts to further improve search2vec ad ranking, including user's age, gender and geo location.

6. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval*, volume 463. ACM press, 1999.

[2] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning pp. 137-186*. 2006.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[4] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. CIKM '08, pages 609–618, 2008.

[5] D. Broccolo, L. Marcon, F. M. Nardini, R. Perego, and F. Silvestri. Generating suggestions for queries in the long tail with an inverted index. *Inf. Process. Manage.*, 48(2):326–339, Mar. 2012.

[6] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. WWW '09, pages 511–520, New York, NY, USA, 2009. ACM.

[7] A. Z. Broder, P. Ciccolo, M. Fontoura, E. Gabrilovich, V. Josifovski, and L. Riedel. Search advertising using web relevance feedback. In *CIKM 2008*.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[9] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[10] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. In *National Bureau of Economic Research*, 2005.

[11] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW 2008*.

[12] D. Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Inf. Sci.*, 179(12):1822–1843, May 2009.

[13] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[14] S. Goel, A. Broder, E. Gabrilovich, and B. Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *WSDM 2010*.

[15] M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati. Context-and content-aware embeddings for query rewriting in sponsored search. In *SIGIR*, pages 383–392. ACM, 2015.

[16] L. Hang. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, 94(10):1854–1862, 2011.

[17] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57. ACM, 1999.

[18] L. Hong and B. D. Davison. Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*. ACM, 2010.

[19] B. J. Jansen and T. Mullen. Sponsored search: An overview of the concept, history, and technology. *International Journal of Electronic Business*, 6(2):114–131, 2008.

[20] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[21] M. Lalmas, J. Lehmann, G. Shaked, F. Silvestri, and G. Tolomei. Promoting positive post-click experience for in-stream yahoo gemini users. In *SIGKDD 2015*.

[22] V. Lavrenko and W. B. Croft. Relevance based language models. In *SIGIR*. ACM, 2001.

[23] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. 2014.

[24] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[26] B. Ribeiro-Neto, M. Cristo, P. B. Golgher, and E. Silva de Moura. Impedance coupling in content-targeted advertising. In *SIGIR 2005*.

[27] A. Spink, D. Wolfram, M. B. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2001.

[28] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *ACL 2010*, pages 384–394.