Improving Cloud Storage Search with User Activity

Rolf Jagerman, Weize Kong, Rama Kumar Pasumarthi, Zhen Qin, Michael Bendersky, Marc Najork
{jagerman,weize,ramakumar,zhenqin,bemike,najork}@google.com
Google Research
Mountain View, CA

ABSTRACT

Cloud-based file storage platforms such as Google Drive are widely used as a means for storing, editing and sharing personal and organizational documents. In this paper, we improve search ranking quality for cloud storage platforms by utilizing user *activity logs*. Different from search logs, activity logs capture general document usage activity beyond search, such as opening, editing and sharing documents. We propose to automatically learn text embeddings that are effective for search ranking from activity logs. We develop a novel *co-access* signal, i.e., whether two documents were accessed by a user around the same time, to train deep semantic matching models that are useful for improving the search ranking quality. We confirm that activity-trained semantic matching models can improve ranking by conducting extensive offline experimentation using Google Drive search and activity logs. To the best of our knowledge, this is the first work to examine the benefits of leveraging document usage activity at large scale for cloud storage search; as such it can shed light on using such activity in scenarios where direct collection of search-specific interactions (e.g., query and click logs) may be expensive or infeasible.

CCS CONCEPTS

• **Information systems → Learning to rank**;

KEYWORDS

User activity logs, Learning to Rank

ACM Reference format:
Rolf Jagerman, Weize Kong, Rama Kumar Pasumarthi, Zhen Qin, Michael Bendersky, Marc Najork. 2021. Improving Cloud Storage Search with User Activity. In *Proceedings of Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8–12, 2021 (WSDM '21),* 9 pages.
https://doi.org/10.1145/3437963.3441780

1 INTRODUCTION

Cloud storage platforms, such as Google Drive or Dropbox, are widely used as a means for storing, editing and sharing personal and organizational documents. They are characterized by storing documents in the cloud and facilitating easy assess across different devices. Unlike traditional Web-based search, *cloud storage search* is challenging because it involves searching through private collections of documents. Due to the private nature of the documents, it is impossible to gather relevance judgments from professional annotators, instead necessitating the use of click logs [40]. As such, existing work has extensively explored the use of click logs [19, 32, 33, 40, 41]. Most of this existing work focuses on extracting features and labels from large-scale search logs and using those to train a ranker.

Semantic matching models are effective at matching queries to documents in situations where traditional lexical matching features fail due to the vocabulary gap [39]. Recent work on semantic matching models use neural networks to improve ranking quality [17]. Semantic matching models can be trained using either (a) self-supervision (e.g. word2vec [25]), or, (b) labeled data (e.g. DSSM [18]). Self-supervised models, while they do not require any search logs or human labels for training, may not work well as they cannot take advantage of any relevance or user behavior data. On the contrary, supervised semantic matching models require large amounts of labeled data and may fail to generalize when sufficiently large search logs are not available. In this paper we investigate *non*-search user interaction data with which we can train semantic matching models: *user activity logs*. User activity logs are typically more abundant than search logs and can provide an adequate substitute for search logs when training semantic matching models.

Our goal is to improve the search ranking quality for cloud storage platforms by utilizing *user activity logs*, which record user's interactions with the cloud storage platform. Examples of such interactions include opening, editing or sharing a document. In contrast to search logs, activity logs contain a richer set of interactions beyond clicks, span more than just the search component, and are available in much more abundant quantity. As demonstrated in several user studies on personal search [14, 16], activity-based signals such as the recency and the context in which the documents are accessed play an important role in determining the relevance of the document to the tasks that the user may be working on.

Despite the fact that activity logs are more abundant than search logs, using them to train semantic matching models is not trivial. In contrast to search logs, activity logs are not necessarily tailored towards ranking and activity-trained semantic matching models may not work well for improving search quality. To overcome this, we propose to use *co-access*, a signal that measures whether two documents are accessed in sequence and within a short time span as a proxy for relevance; we then use this information as a weakly supervised label for training text embeddings and semantic similarity models.

Our experiments using large-scale Google Drive search and activity data show that semantic matching models trained with co-access can improve ranking performance significantly compared to lexical matching and semantic matching baselines that are *not* trained on

WSDM '21, March 8–12, 2021, Virtual Event, Israel

ACM ISBN 978-1-4503-8297-7/21/03.
https://doi.org/10.1145/3437963.3441780

activity logs. Furthermore, we show that incorporating activity-trained semantic matching models with strong hand-crafted features further improves ranking performance, especially in cases where hand-crafted features alone do not provide sufficient signals. Finally, our results show that, by leveraging activity logs, practitioners of cloud storage systems can train effective ranking models without requiring a large amount of search logs. This shows that activity logs can significantly help when search logs are not available in abundant quality.

To the best of our knowledge, this is the first work to examine the benefits of leveraging non-search related document usage activity for cloud storage search in a large-scale production system. We demonstrate that such activity can be used to not only provide features to ranking models, but also to derive semantic similarity models that improve lexical text matching. As such, this work can shed light on employing user activity in scenarios where direct collection of search-specific interactions (e.g., query and click logs) may be expensive or infeasible, or in cloud storage services that have no existing search functionality.

2 RELATED WORK

Cloud storage is widely used by individuals and organizations as a means of storing, organizing and sharing documents. Cloud storage faces several research challenges, for example: scaling to large traffic volume [8, 13], anonymizing data [36], and optimizing search [3]. Cloud storage search is related to email search [3, 6], enterprise domain search [38], desktop search [21], and other personal search problems [14, 16] in that they typically involve searching through private corpora; however, it is also unique in that rich non-search user activities, such as opening, editing and sharing documents, can be logged at large scale in cloud storage systems, which inspires our work.

Learning to Rank (LTR) is a widely used approach to optimize search engine quality. Recent research on LTR focuses on learning from user interactions [19, 24, 40]. State-of-the-art approaches for optimizing search rely on combining large sets of high-quality hand-crafted features into a well performing model. Tree-based methods, specifically LambdaMart [5, 43], excel at combining high-quality features into a single strong performing model. LambdaMart has enjoyed considerable success in the past and is still considered state-of-the-art [42]. In particular, LambdaMart has repeatedly outperformed all other models on public benchmarks, where the number of training examples is limited [7, 20], even with the advancement of neural ranking methods [4]. Therefore, we use it as the ranking model of choice in this paper.

More recently, advances in neural networks enabled significant progress on semantic matching models for information retrieval (see [26] for a recent comprehensive survey on this topic). Most generally, these semantic matching models use either search logs [17, 18, 35, 35, 45] or weak supervision [9, 11] to learn text embeddings that best capture *query-document* similarities. Subsequent work introduced multiple advanced variations on these semantic matching models including (among many others) convolutional networks [10], kernel pooling [44] and transformers [29]. However, note that these advanced method cannot be readily applied to our heavily reducted dataset, which, due to privacy constraints, does not preserve word ordering (see Section 3.3).

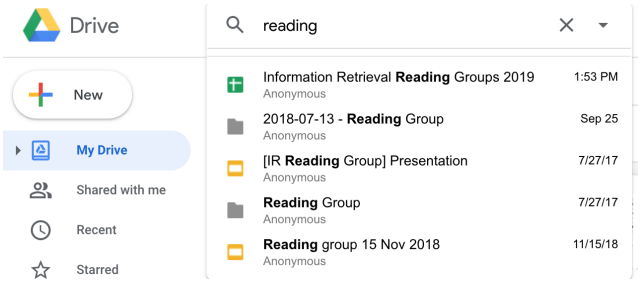

Figure 1: An example of Google Drive's top-5 ranking list.

In contrast to prior work on semantic matching models, text embeddings based on *user activity* do not require learning query-document similarities. This is a highly desirable property for cloud storage systems where either no existing search solutions exist or search logs are sparse due to low search volume and / or private nature of search intents. For instance, a study by Ai et al. [1] shows that email search queries are much shorter than those in web search, and are often based on a particular email metadata (e.g., sender or received / sent date). Such queries are less likely to generate robust generalizable semantic matching models.

User activity has been extensively used for user modeling [2, 28, 31], predicting future user behavior [12, 37], or document retrieval [14, 16] to name just a few. More recently, Kong et al. [22] used activity logs for learning to cluster documents. However, to the best of our knowledge, there is no prior work on learning semantic matching models from activity logs at large scale, especially in the context of cloud storage search.

3 PROBLEM SETTING

3.1 Cloud Storage Search

Cloud storage search can be formulated as a Learning to Rank (LTR) problem. More formally, we wish to learn a ranking function $f(q, d)$, which, for given query q and document d, produces a score such that relevant documents are assigned high scores and less relevant documents are assigned low scores. The learned function can then be applied to a query q and a collection of candidate documents $D = \{d_1, d_2, \ldots, d_n\}$ to rank the documents by relevance, placing highly relevant documents at the top.

3.2 Solution Overview

In this paper we focus on improving the top-5 ranking of the search component of Google Drive (see Figure 1). We provide an overview of our solution in Figure 2. More specifically, we train a LambdaMart [43] model on a click log spanning several weeks. We chose to use a Gradient Boosted Decision Tree (GBDT) approach as these methods have demonstrated very strong ranking performance [43], are computationally easy to scale, are robust against outliers and can naturally handle input of varying distributions [15]. We incorporate the output of activity-trained semantic similarity models as features for training the GBDT.

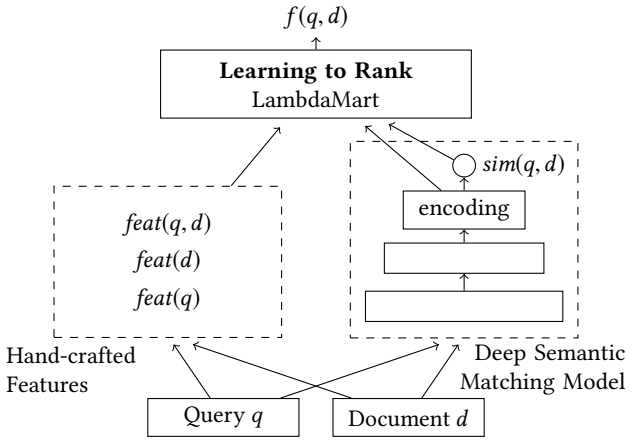

Figure 2: An overview of the proposed solution for cloud storage search. On the left are hand-crafted features for given query q and document d. On the right a deep semantic matching model produces both an encoding (the last hidden layer) and a similarity score $sim(q, d)$. Both the hand-crafted and learned features are combined in an LTR model.

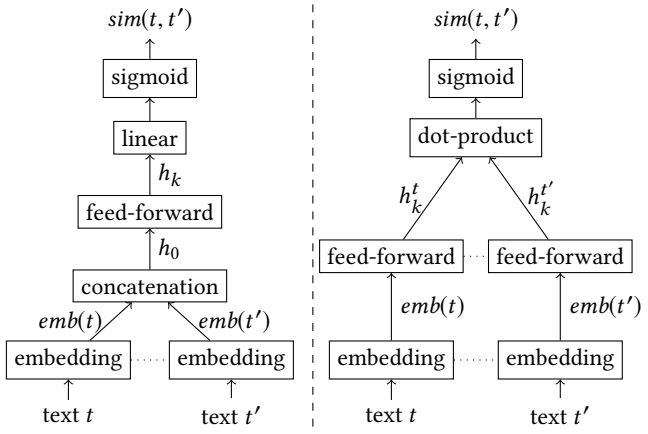

Figure 3: On the left is the Concatenation Semantic Matching Model (Concat) and on the right is the Siamese Semantic Matching Model (Siam). Dotted lines indicate shared weights between components.

3.3 Privacy

Google Drive contains private document corpora and requires special treatment to protect user privacy. For this reason, our data used for the experiments are k-anonymized [36] and are inaccessible to individual engineers. Moreover, for our proposed methods, we limit the use of text content to the document titles, not the full content. The titles are also k-anonymized, i.e., only frequent words used by sufficiently many users in the corpus are retained with no word sequence information preserved. Note that this limits our choices of text embedding models to word-level and character-level embeddings. Some of our baseline models / features are based on document full content, but these models / features are computed on the fly – document content is never materialized for model training.

4 USING ACTIVITY LOGS FOR SEMANTIC MATCHING

Directly applying existing semantic matching models on query logs for cloud storage search is not trivial because click logs for cloud storage search may not be available in sufficient quantity to learn effective semantic matching models, and training them with too few data points may lead to poor generalization. To overcome this limitation we propose to use a different, more abundant, source of data for training semantic matching models: *activity logs*. Our goal is to learn semantic matching models from activity logs and apply them for ranking.

4.1 Semantic Matching Models

A Semantic Matching Model (SMM) learns to compute the semantic similarity of two pieces of text (typically a query and document for search problems) [17, 18]. The model takes a text pair (t, t') as input and produces $sim(t, t')$, a score indicating the similarity between the two texts, as output. This formulation is quite general as the text pair (t, t') could be any two pieces of short text. For example, in ranking we would use a query q and document d to predict the similarity between the query and document: $sim(q, d)$. Similarly, if we were interested in modeling document similarity we could use two documents (d, d') as input and predict their similarity: $sim(d, d')$. As we will see later, this general formulation is beneficial as we will train our semantic matching models on *document pairs*, but apply them to *queries and documents* when ranking.

To train semantic matching models we would need to obtain a dataset containing textual pairs and their respective relevance labels. It is not possible to collect labeled examples from professional annotators due to the privacy considerations when dealing with cloud storage data. Even for cases where collecting labels from human annotators is possible, conducting such an annotation process at large scale could be too expensive and infeasible. Our proposed solution to this is to extract relevance labels automatically from users' activity logs.

We emphasize that the specific architecture of the SMM is not the focus of our paper, nor do we claim that any specific architectures we use are novel. Instead we are interested in understanding how one can learn semantic matching models from *activity data*. We use two popular architectures, a concatenation semantic matching model and a Siamese semantic matching model, shown in Figure 3 and described briefly below.

(1) Concatenation Semantic Matching Model (Concat). The model first computes an embedding for the texts t and t'. Each character n-gram in the text is mapped to an embedding, where only the most frequent n-grams are retained to limit the vocabulary and make the problem computationally feasible. The character n-grams for t and t' are then averaged to obtain $emb(t)$ and $emb(t')$ respectively. The representations are concatenated to obtain:

$$h_0 = \left[emb(t), emb(t') \right].$$

Note that more advanced text encoders such as recurrent neural networks and transformers are not applicable to our problem. This

is because, as mentioned in Section 3.3, to protect user privacy, we are only allowed to use k-anonymized words with no sequence information for model training.

This joint representation is then passed through a series of dense feed-forward layers, where each layer h_i is defined as:

$$h_i = \phi(W_i h_{i-1} + b_i),$$

where ϕ is an activation function such as ReLU [27] or tanh. Finally, the last layer of this feed-forward neural network, h_k, is reduced to a scalar value and mapped to a probability via a sigmoid function:

$$sim(t, t') = \text{sigmoid}(W_{final} h_k + b_{final}).$$

(2) Siamese Semantic Matching Model (SIAM). This model embeds the texts t and t' to their respective representations $emb(t)$ and $emb(t')$. The process of producing these embeddings is exactly the same as in the CONCAT. Next, these representations are passed through a shared feed-forward neural network. More formally:

$$h_0^t = emb(t), \qquad h_0^{t'} = emb(t'),$$
$$h_i^t = \phi(W_i h_{i-1}^t + b_i), \quad h_i^{t'} = \phi(W_i h_{i-1}^{t'} + b_i).$$

Note that the weights W_i and b_i at each layer are shared for both t and t'. The output vectors of the last layer, h_k^t and $h_k^{t'}$, are joined via dot product and then passed through a sigmoid to obtain:

$$sim(t, t') = sigmoid\left(h_k^t \cdot h_k^{t'}\right).$$

To apply the two SMMs for ranking, we feed the given query q and document d into the models, and extract the following output and hidden states as features for a LTR model: (a) the predicted semantic similarity, $sim(q, d)$; (b) the last hidden layer of CONCAT, h_k, which provides a richer representation for the query-document pair; and (c) the last hidden layers in the towers of SIAM, h_k^q and h_k^d, which encode q and d respectively. This is illustrated in Figure 2.

4.2 Weak Supervision Using Co-access Labels

In order to train semantic matching models we would like to obtain relevance labels $y_{t,t'} \in \{0, 1\}$ that indicate whether text t is semantically related to text t'. We take a weakly-supervised learning approach, and propose a *co-access* label, with the assumption that it can serve as a proxy for relevance.

We say that two documents are *co-accessed* iff a user opens the two documents *in sequence* and *within a τ-minute time window*. We illustrate this concept in Figure 4 with τ = *2 minutes*[1]. While there are multiple alternative ways to design the co-access label, we found that the method proposed here is conceptually simple, yet empirically effective. The co-access label is motivated by the fact that users often open multiple related documents in a single session, yet it strives to reduce the number of false positive labels by keeping a narrow time window and discarding non-consecutive co-accesses.

The following procedure is employed to collect training examples with co-access labels: (1) We first sample segments of a user's activity logs, which we call *activity segments*. Each activity segment contains events from the same user in a consecutive time window; (2) For each activity segment, we collect a set of documents the user accessed, $D = \{d_i\}_{i=1}^{|D|}$. From the document set we then collect all the unordered pairs of documents in the document set, $\mathcal{P}_D = \{\{d, d'\} \mid d, d' \in D \wedge d \neq d'\}$; (3) Lastly, we extract co-access labels for all the document pairs, and the co-access label $y_{d,d'}$ is defined as:

$$y_{d,d'} = \begin{cases} 1, & co\_accesses(d, d') > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $co\_accesses(d, d')$ is the number of co-access events between d and d' in the activity segment. To give an example, for the activity segment shown in Figure 4, the collected document set is $D = \{d_1, d_2, d_3, d_4\}$ and the extracted co-access labels for all the document pairs are: $y_{d_1,d_2} = y_{d_1,d_3} = 1$, and, $y_{d_1,d_4} = y_{d_2,d_3} = y_{d_2,d_4} = y_{d_3,d_4} = 0$.

This procedure yields a training dataset

$$\mathcal{D} = \left\{(d, d', y_{d,d'}) \mid \{d, d'\} \in \bigcup_{i=1}^{N} \mathcal{P}_D^{(i)}\right\}$$

from a large number of activity segments, where N is the number of the segments. We then train a semantic matching model by minimizing the weighted cross-entropy loss defined as follows:

$$-\sum_{(d,d',y_{d,d'})\in\mathcal{D}} y_{d,d'} \log(sim(d, d')) + \lambda(1 - y_{d,d'}) \log(1 - sim(d, d')), \tag{2}$$

in which we use document titles as the text representations for each document when scoring $sim(d, d')$, and use $\lambda \in (0, 1]$ as a hyper parameter to down-weight the loss for negative document pairs. We use this weighting to address data imbalance problem in the dataset – in practice only a small percentage of document pairs are co-accessed and the majority of the document pairs are not co-accessed.

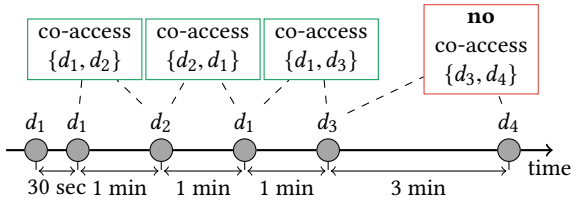

Figure 4: Co-access as a label for document similarity. Document pairs $\{d_1, d_2\}$, $\{d_2, d_1\}$ and $\{d_1, d_3\}$ are co-accessed, but $\{d_3, d_4\}$ is not since their co-access time is > 2 minutes.

4.3 Co-access and Document Relevance

The co-access label as described in the previous section is attractive for modeling relevance for a few reasons. First, it can be extracted automatically from activity logs at large scale with little cost, even for cloud document repositories with no available search data. Second, it produces semantically related document pairs (d, d'), where d can viewed as an information need, and d' as a document satisfying this need. Lastly, the co-access label definition in Equation 1 closely matches the assumptions of the widely adopted probability ranking principle (PRP) [34], where (a) relevance is assumed to

[1] During our initial investigations we found that co-accesses have a long tail distribution, with 70% of co-accesses occurring within the two minute window. Therefore, we fix the co-access time window to two minutes in the remainder of this paper.

be a property of a document and the information need, and no other information, and, (b) relevance can be modeled as a binary probabilistic variable.

5 EXPERIMENTAL SETUP

We conduct experiments using large-scale Google Drive logs to better understand whether semantic matching models trained on activity logs can improve the ranking quality of the search component in Google Drive. As mentioned in Section 3.2 we focus on the top-5 ranking component of Google Drive and train a LambdaMart model that incorporates the output of our semantic matching models (see Figure 2).

5.1 Data collection

We collect two datasets from the search and activity logs of Google Drive: (1) Co-access dataset $\mathcal{D}_c$ used for building our Semantic Matching Models, and (2) Search dataset $\mathcal{D}_s$ used mainly for building learning to rank models. We describe them in more detail below.

5.1.1 Co-access dataset. We collect the co-access dataset $\mathcal{D}_c$ from activity logs as follows: (1) We randomly sample different sets of users for training, validation and testing. (2) For each user, we sample 10 segments of the user's event stream from the activity logs. Each segment contains activity events of 3 consecutive weeks. We sample the start timestamps of the activity segments uniformly within a 2-week time window. We filter out segments that do not contain sufficient activity events, i.e., less than 75 events. (3) From each activity segment, we collect up to n most recently accessed documents from the first two weeks of the segment. For each collected document pair, we extract document titles and co-access labels from the last week of the segment (i.e., whether the two documents are co-accessed in that week). The size of the document pair set could be very large – up to $n \cdot (n-1)/2$ pairs. To improve efficiency as well as reducing noise, we filter the document pairs by requiring them to be co-accessed in the first two weeks of the segment. This reduces the size of document pair set dramatically to around 10 per segment on average.

Table 1 shows some statistics for the collected co-access dataset for training, validation and testing. The table reports the number of activity segments and document pairs. Note that the training set $\mathcal{D}_c^{train}$ is fairly large. This is because we can easily extract the co-access data from a large set of eligible Google Drive users and sample multiple activity segments from their activity logs. The table also reports the percentages of co-accessed document pairs (postive rate), which are quite consistent across the training, validation and testing sets. Note that the positive rates are only around 10%. To address this data imbalance issue, we down-weight the negative document pairs when training our models as described in Section 4.2. Moreover, the dataset is k-anonymized to protect user privacy as described in Section 3.3.

5.1.2 Search dataset. The search dataset $\mathcal{D}_s$ comprises a set of queries and clicks collected from the Google Drive search logs over a period of several weeks. Here we select dates that are after the dates of the co-access dataset $\mathcal{D}_c$ to prevent any potential data peeking issues when training our LTR models on $\mathcal{D}_s$. We then extract all the features (see Section 5.4) and search clicks for the sampled queries from the search logs and activity logs. Each query is associated with about 5 documents on average, which is a direct result of the search user interface. We discard all the queries without clicks. We then split the data into training, validation and testing set by dates, using the earlier dates for training, later dates for validation and the latest dates for testing. This data split prevents accidental leaking of future test queries and documents into the training set. In total, we collected 31,421 queries for training, 25,412 queries for validation, and 24,595 queries for testing.

Table 1: Statistics for co-access dataset $\mathcal{D}_c$. The *train*, *vali*, *test* superscript indicates the dataset is for training, validation or testing respectively. "#segments", "#pairs" and "% co-accessed" report the total number of activity segments, document pairs and percentage co-accessed pairs respectively.

Data split	#segments	#pairs	% co-accessed
$\mathcal{D}_c^{train}$	44.3M	448M	10.5%
$\mathcal{D}_c^{vali}$	20.6K	205K	10.3%
$\mathcal{D}_c^{test}$	20.6K	205K	10.2%

5.2 Evaluation

We evaluate the performance of each ranking model using Mean Reciprocal Rank (MRR) and Negative Average Click Position (NACP), which are defined as,

$$\text{MRR} = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{rank_q}, \quad \text{NACP} = -\frac{1}{|Q|}\sum_{q\in Q} rank_q,$$

where $Q = \{q\}$ is the evaluation query set, $rank_q$ is the rank of the first clicked document for query q. To address click position bias, we use propensity-weighted MRR and NACP which tend to be more consistent with online experiment results [23, 40].

We tune each ranking model on the search validation dataset, and report evaluation results on the test dataset. We performed statistical significance tests using paired t-test with 0.01 as the p-value threshold. In all tables we denote a statistically significant increase and decrease compared to the baseline with $\triangle$ and $\triangledown$ respectively.

5.3 Baselines

We compare the following semantic matching baseline models with ours. Note that more advanced models like recurrent neural networks and transformers are not applicable to our problem. As mentioned in Section 3.3, to protect user privacy, our dataset only contains k-anonymized words. Character n-grams extracted from these words retain no sequence information for model training.

W2V We compare our work to Word2Vec (W2V) [25] since both methods train text embeddings without needing access to search logs. We use a straightforward approach to apply Word2Vec to ranking: we compute the Word2Vec embeddings for query q and document d, producing $emb(q)$ and $emb(d)$. We can then take the dot product of these embeddings to produce a similarity score:

$$sim(q, d) = emb(q) \cdot emb(d).$$

We train the embeddings using the Word2Vec skip-gram approach on the 107 million document titles extracted from the search

dataset $\mathcal{D}_s$. We retain the 500,000 most frequent n-grams as the vocabulary and choose 90 as the dimensionality of these embeddings. We use the full document title as the context, since the text does not retain word sequence information.

DSSM The Deep Structured Semantic Model (DSSM) [18] is a commonly used semantic matching model for ranking and we include it in our comparison. The DSSM architecture is implemented using TF-Ranking [30], and is trained on the search dataset $\mathcal{D}_s$. We choose the softmax ranking loss function as our training objective [30]. We use a dimension size of 159 for the character n-gram embeddings and ReLU as the activation function. The hidden layer dimensions, learning rate, dropout rate and vocabulary size are tuned via grid search.

5.4 Features

We compare different sets of features combined in LambdaMART. We list all the studied features in Table 2 and describe them in groups below.

Table 2: The full set of features used to train LTR models.

Feature	Description
Lexical text matching features (TM)	
Overlap	# of query terms in the document title.
Normalized	Same as *Overlap*, but after lexical normalization.
BM25V	A variant of BM25 [34]
Activity-based features (ACT)	
Last Access	Time since last access.
Last Edit	Time since last edit.
Doc Age	Time since the document was created.
Activity-based semantic matching models (Section 4)	
CONCAT$_{sim}$	Output of CONCAT: $sim(q, d)$.
CONCAT$_{rep}$	State of last hidden layer of CONCAT: h_k.
SIAM$_{sim}$	Output of SIAM: $sim(q, d)$.
SIAM$_{rep}$	State of last hidden layers of SIAM: h_k^q, h_k^d.
Other semantic matching models (Section 5.3)	
W2V$_{sim}$	Output of W2V: $sim(q, d)$.
W2V$_{rep}$	Embeddings of q and d: $emb(q)$, $emb(d)$.
DSSM$_{sim}$	Output of DSSM: $sim(q, d)$.
DSSM$_{rep}$	State of last hidden layer of DSSM: h_k.

Lexical text matching features (TM) These features perform various forms of lexical matching of the query and document.

Activity-based features (ACT) We use three simple but effective activity-based features, that characterize document recency, namely *Last Access*, *Last Edit*, and, *Doc Age*.

Semantic matching features We extract the similarity scores as well as the internal representations from our proposed and baseline Semantic Matching Models. The similarity scores and internal representations are denoted with subscript *sim* and *rep* respectively in Table 2. Using both *sim* and *rep* is denoted using subscript *both* in our experiment result tables, e.g., CONCAT$_{both}$ includes both CONCAT$_{sim}$ and CONCAT$_{rep}$. We use a dimension size of 159 for character n-gram embeddings. We cap the vocabulary size at 500k by mapping less frequent character n-gram to out-of-vocabulary embeddings. We tune the hidden layer size, learning rate and activation functions.

Table 3: Ranking performance of Semantic Matching Models and activity-based features. The table reports relative performance with respect to W2V$_{sim}$ baseline. △/▽ indicates statistically better/worse results over the baseline.

Model	MRR	NACP
W2V$_{sim}$	+0.00%	+0.00%
DSSM$_{sim}$	+1.64%$^{\triangle}$	+2.92%$^{\triangle}$
SIAM$_{sim}$	+6.40%$^{\triangle}$	+6.34%$^{\triangle}$
CONCAT$_{sim}$	-5.56%$^{\triangledown}$	-3.28%$^{\triangledown}$
Last Access	-1.10%	+1.84%
Doc Age	+2.01%$^{\triangle}$	+2.32%$^{\triangle}$
Last Edit	+2.08%$^{\triangle}$	+2.45%$^{\triangle}$
BM25V	+12.74%$^{\triangle}$	+11.76%$^{\triangle}$
TM	+15.90%$^{\triangle}$	+14.77%$^{\triangle}$

6 RESULTS

6.1 Comparing Individual Models

In this section, we study the ranking performance of the Semantic Matching Models when used independently. Our results are displayed in Table 3. *TM* is a fine-tuned combination of all the lexical text matching features in Table 2 used by the production system. All of these models, with the exception of DSSM$_{sim}$ and *TM*, do not require any search training data.

First, we observe that the activity-based features perform reasonably well, reaching levels similar to Semantic Matching Models such as W2V and in the cases of *Doc Age* and *Last Edit*, performing even better. These results seem surprising as features like *Doc Age* and *Last Edit* are comparatively simple features. However, this can be explained by the fact that our task is to re-rank the top 5 results in Google Drive. Nevertheless, these findings reinforce our belief that activity-based signals can be useful for search tasks.

Second, we see that supervised embeddings DSSM and weakly supervised embeddings SIAM are more effective than the self-supervised W2V approach. The CONCAT model underperforms significantly, especially when compared to the very similar SIAM model. Interestingly, we find the CONCAT model performs significantly *better* on the co-access prediction task, improving Area Under Curve (AUC) by 10%. We hypothesize that the SIAM architecture has an inductive bias by forcing the representations of both query and document to be directly comparable via dot product. Conversely, for the CONCAT model, the concatenation of the embeddings allows the model to learn complex feature interactions between the texts t and t', which may learn useful patterns for co-access, but likely does not generalize well to generic semantic similarity tasks and also not well to ranking.

To further analyze this behavior we investigate to what extent the different semantic matching models are correlated with the lexical text matching and activity-based features. In Table 4 we present the Pearson correlation between each model's similarity score output and those features. The findings here suggest that W2V has the strongest correlation with lexical matching features but does not correlate at all with activity-based features, which is expected as the W2V model is trained on unlabeled text corpora

Table 4: Pearson Correlation between semantic matching scores and lexical matching and activity-based features.

Model	*BM25V*	*Normalized*	*DocAge*	*LastEdit*
$W2V_{sim}$	0.3014	0.3798	0.0168	0.0104
$SIAM_{sim}$	0.2347	0.3044	-0.1256	-0.1183
$CONCAT_{sim}$	0.0799	0.1126	-0.1478	-0.1305
$DSSM_{sim}$	0.1784	0.2401	-0.0792	-0.0774

not on any user logs. Furthermore, we see that CONCAT correlates weakly with lexical matching but much more strongly with activity-based features, compared with other semantic matching models. This indicates that the CONCAT architecture could capture activity-based patterns more than the other Semantic Matching Models. Finally, it seems that SIAM correlates strongly with both lexical matching features *and* activity-based features (when compared to other models), which explains its strong performance in Table 3.

Finally, it is clear that none of the semantic similarity models beat the simple ad-hoc retrieval baselines *BM25V* and *TM*. Because of the strong performance of the lexical matching features, we next investigate whether Semantic Matching Models could improve ranking performance on top of *TM* in the next section.

6.2 Combined With Lexical Matching Features

We incorporate the semantic matching models with the text matching features in a LambdaMart ranker. The results are displayed in Table 5. For all Semantic Matching Models, we find that adding both the similarity score (*sim*) and the representation (*rep*) as features performs the best. Thus, we only report results when using both *sim* and *rep* features (denoted by subscript *both*) in the rest of this paper due to space limitation.

Table 5: Ranking performance (relative to the *TM* baseline) of Semantic Matching Models when combined with lexical text matching features.

Model	MRR	NACP
TM	+0.00%	+0.00%
$TM + W2V_{both}$	$+1.46\%^{\Delta}$	$+1.52\%^{\Delta}$
$TM + DSSM_{both}$	$+1.59\%^{\Delta}$	$+2.08\%^{\Delta}$
$TM + CONCAT_{both}$	$+2.67\%^{\Delta}$	$+3.53\%^{\Delta}$
$TM + SIAM_{both}$	$+2.88\%^{\Delta}$	$+3.37\%^{\Delta}$

In all cases, incorporating Semantic Matching Models together with lexical text matching features improves the ranking performance. Moreover, we observe that our weakly supervised activity-trained semantic models (CONCAT and SIAM) significantly outperform (p-value < 0.01, t-test) both the unsupervised (W2V) and the supervised (DSSM) methods. We believe this could be explained by that our Semantic Matching Models could capture some activity-based patterns in addition to semantic similarity, as we discussed in the correlation analysis (Table 4) of Section 6.2. Next, we find there does not seem to be a discernible difference between the CONCAT and SIAM architectures. This indicates that the difference in ranking performance between CONCAT and SIAM that we observed in the previous section (see Table 3) is largely overcome by incorporating text matching features in the model.

Table 6: Ranking performance of Semantic Matching Models (relative to the *TM* + *ACT* baseline) when combined with lexical text matching features and activity-based features.

Model	MRR	NACP
TM + *ACT*	+0.00%	+0.00%
$TM + ACT + W2V_{both}$	+0.15%	$+0.35\%^{\Delta}$
$TM + ACT + DSSM_{both}$	$-1.39\%^{\nabla}$	$-2.18\%^{\nabla}$
$TM + ACT + CONCAT_{both}$	$+0.46\%^{\Delta}$	$+0.93\%^{\Delta}$
$TM + ACT + SIAM_{both}$	+0.13%	+0.29%

6.3 Combined With All Hand-crafted Features

In this section we explore what benefit the Semantic Matching Models have on top of hand-crafted text matching and activity-based features. The main results are provided in Table 6.

First, we find the improvement from Semantic Matching Models over *TM* + *ACT* becomes marginal, except for our CONCAT model. CONCAT outperforms (p-value < 0.01) all the other baseline semantic matching models (W2V, DSSM), and is the only model that provides a statistically significant improvement over *TM* + *ACT* on both MRR and NACP, achieving nearly +1% improvement on NACP.

Our further analysis in Table 7 shows CONCAT is especially helpful for the hard queries on which the activity-based features have less discriminative power. Specifically, we measure their discriminative power using the range of the *Last Access* feature defined as follows:

$$range(\textit{Last Access}) = \max\{\textit{Last Access}\} - \min\{\textit{Last Access}\}.$$

A lower range indicates the documents have similar activity-based features and thus the features are less discriminative for ranking the documents. Table 7 reports the improvement of CONCAT over *TM* + *ACT* for queries within different *Last Access* range. We find CONCAT improves over the strong *TM* + *ACT* baseline by +1.16% and +2.22% on MRR and NACP respectively, impacting more than 20% of the test queries.

Table 7: Relative performance improvements of CONCAT over *TM* + *ACT* for queries of different *range*(*Last Access*). % of queries indicates what percentage of the queries have a range within the specified threshold.

range (in days)	% of queries	MRR	NACP
2	10.7%	$+1.55\%^{\Delta}$	$+2.93\%^{\Delta}$
8	20.7%	$+1.16\%^{\Delta}$	$+2.22\%^{\Delta}$
18	30.1%	$+0.97\%^{\Delta}$	$+1.89\%^{\Delta}$
53	50.1%	$+0.67\%^{\Delta}$	$+1.33\%^{\Delta}$
198	90.1%	$+0.48\%^{\Delta}$	$+1.94\%^{\Delta}$
$+\infty$	100.0%	$+0.46\%^{\Delta}$	$+0.93\%^{\Delta}$

6.4 Training With Abundant Search Data

We regard our semantic matching models *weakly-supervised*, since they are trained on document co-access labels and do not require

any relevance labels or search clicks. In contrast, DSSM is a supervised model that relies on a large number of search clicks or relevance labels for training. That said, it plausible that with abundant search clicks or relevance labels, the supervised model DSSM could provide stronger ranking performance improvements than our weakly supervised models. Therefore, in this last experiment, we collect more search data (up to 23 million queries) to train DSSM, and compare Concat with DSSM in Table 8.

Before analyzing the results, we emphasize that our work aims to leverage co-access labels extracted from activity logs to improve ranking *when the search training data is limited*. This is often the case for cloud storage systems, in which users *access* documents much more frequently than they *search* for documents. We further collect more search data in order to investigate whether our *weakly supervised* semantic matching models trained only on co-access labels can provide similar performance as DSSM trained on abundant search clicks.

Table 8: Relative performance of DSSM trained with different size of search training data with respect to Concat. #Queries reports the number of queries used for training the semantic matching models. △/▽ indicates statistically better/worse results over Concat (p-value<0.01).

Model	#Queries	MRR	NACP
$TM + ACT + \text{Concat}_{both}$	0	+0.00%	+0.00%
$TM + ACT + \text{DSSM}_{both}$	31K	-1.84%$^{\triangledown}$	-3.15%$^{\triangledown}$
$TM + ACT + \text{DSSM}_{both}$	63K	-1.35%$^{\triangledown}$	-2.13%$^{\triangledown}$
$TM + ACT + \text{DSSM}_{both}$	1M	-0.03%	-0.13%
$TM + ACT + \text{DSSM}_{both}$	23M	+0.28%	+0.54%

In Table 8, we find that Concat still significantly outperforms DSSM, when the search training data contains less than 1 million queries. When training with 1 million to 23 million queries, DSSM becomes comparable or slightly better than Concat, however, the differences are quite small and not statistically significant (p-value < 0.01). These results are exciting, as they indicate that, even in cases with sparse and limited search logs, activity-based semantic matching models could provide an adequate substitute.

6.5 Discussion

It is clear from Table 5 and Table 6, that our semantic matching models trained on document co-access labels can effectively improve ranking performance, and they significantly outperform W2V (text representations learned from unlabeled text corpora) and DSSM (text representations learned from clickthrough data). Moreover, Table 8 suggests that Concat can provide comparable performance to DSSM even when DSSM is trained with orders of magnitude more search queries.

The concatenation structure of Concat allows complex feature interaction between the two text inputs. Because of this, Concat significantly outperforms the Siamese network model Siam in co-access prediction. Concat is also better than Siam at capturing effective signals from the co-access labels that are not fully covered by the lexical and activity-based features (Table 6). However, we find Concat is less effective than Siam at capturing textual similarity according to the correlation analysis (Table 4). We believe this is because the Siamese structure forces Siam to learn representations for text matching, and therefore it's more effective when transferred from the co-access prediction task to the search ranking task. Thus, when the lexical text matching features are absent, Siam significantly outperform Concat (Table 3).

Our offline experiments using large scale Google Drive search and activity logs provide strong evidence that activity logs are an adequate substitute for search logs when training semantic matching models. This is especially important for cloud storage systems or other domain-specific applications, where search logs may not be available in large quantity or even at all. For example, when search is first introduced as a feature to a cloud storage system, search logs will be non-existent, but activity logs are abundant. As another example, a small enterprise may not have enough search traffic to build a specialized semantic matching model, however it may have enough activity data to do so.

7 CONCLUSION

In this paper, we demonstrate that leveraging user activity, e.g., document access, editing and sharing, can significantly improve the quality of cloud storage search. In cloud storage search, users often eschew using search altogether, opting out for navigation instead. This makes it challenging to leverage click data as it may not be available in the quantity necessary to train ranking models.

Compared to *search logs* in cloud storage system, user *activity logs* are always available in abundant quantities, as they capture any user interaction with the stored documents, however they are not directly tied to search intents and information needs. To this end, we introduce a novel method for automatically learning text embeddings from activity logs, by employing *document co-access* as a label for document similarity.

Our experiments demonstrate that such embeddings can significantly outperform standard semantic matching approaches, and can be combined with other features to further improve performance when search data is limited. Furthermore, combining hand-crafted features with activity-trained text embeddings provides the best of both worlds, and offers significant improvements over either using hand-crafted features or embeddings individually.

To the best of our knowledge, we are the first to examine the use of activity logs in a large scale cloud storage search engine. As more document collections are moving to the cloud, the benefits of activity logs for improving cloud storage search is an important finding that opens up several directions for future work.

As an example, in this work we do not use more complex text embedding models such as recurrent neural networks or transformers, due to privacy constraints that are in place to protect the users. In future work, we would like to explore ways of using such models while retaining user privacy.

As another example, in future work we would also like to consider more expressive models of user activity, such as recurrent neural networks to model the historical activity of a user in order to predict future document usage. This can be helpful in reducing the noise in user activity data, and contribute to further improvements in search quality.

REFERENCES

[1] Qingyao Ai, Susan T. Dumais, Nick Craswell, and Dan Liebling. Characterizing email search using large-scale behavioral logs and surveys. In *Proc. of WWW*, pages 1511–1520, 2017.

[2] Tarfah Alrashed, Ahmed Hassan Awadallah, and Susan Dumais. The lifetime of email messages: A large-scale analysis of email revisitation. In *Proc. of CHIIR*, pages 120–129. ACM, 2018.

[3] Michael Bendersky, Xuanhui Wang, Donald Metzler, and Marc Najork. Learning from user interactions in personal search via attribute parameterization. In *Proc. of WSDM*, pages 791–799, 2017.

[4] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. Revisiting approximate metric optimization in the age of deep neural networks. In *Proc. of SIGIR*, pages 1241–1244, 2019.

[5] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[6] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. Rank by time or by relevance?: Revisiting email search. In *Proc. of CIKM*, pages 283–292, 2015.

[7] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, pages 1–24, 2011.

[8] Suming J. Chen, Zhen Qin, Zac Wilson, Brian Calaci, Michael Rose, Ryan Evans, Sean Abraham, Donald Metzler, Sandeep Tata, and Michael Colagrosso. Improving recommendation quality in google drive. In *Proc. of KDD*, page 2900–2908, 2020.

[9] Daniel Cohen, Scott M. Jordan, and W. Bruce Croft. Learning a better negative sampling policy with deep neural networks for search. In *Proc. of ICTIR*, pages 19–26. ACM, 2019.

[10] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proc. of WSDM*, pages 126–134. ACM, 2018.

[11] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. In *Proc. of SIGIR*, pages 65–74. ACM, 2017.

[12] Dotan Di Castro, Zohar Karnin, Liane Lewin-Eytan, and Yoelle Maarek. You've got mail, and here is what you could do with it!: Analyzing and predicting actions on email messages. In *Proc. of WSDM*, pages 307–316. ACM, 2016.

[13] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: Understanding personal cloud storage services. In *Proc. of IMC*, pages 481–494. ACM, 2012.

[14] Susan Dumais, Edward Cutrell, Jonathan J Cadiz, Gavin Jancke, Raman Sarin, and Daniel C Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *Proc. of SIGIR*, pages 72–79, 2003.

[15] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001.

[16] Marguerite Fuller, Liadh Kelly, and Gareth JF Jones. Applying contextual memory cues for retrieval from personal information archives. In *Proc. of Personal Information Management Workshop*, 2008.

[17] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proc. of CIKM*, pages 55–64, 2016.

[18] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proc. of CIKM*, pages 2333–2338, 2013.

[19] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proc. of KDD*, pages 133–142, 2002.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proc. of NeurIPS*, pages 3149–3157. Curran Associates Inc., 2017.

[21] Jinyoung Kim and W Bruce Croft. Ranking using multiple document types in desktop search. In *Proc. of SIGIR*, pages 50–57, 2010.

[22] Weize Kong, Mike Bendersky, Marc Najork, Brandon Vargo, and Mike Colagrosso. Learning to cluster documents into workspaces using large scale activity logs. In *Proc. of KDD*, page 2416–2424, 2020.

[23] Lihong Li, Shunbao Chen, Jim Kleban, and Ankur Gupta. Counterfactual estimation and optimization of click metrics in search engines: A case study. In *Proc. of WWW*, pages 929–934, 2015.

[24] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NeurIPS*, pages 3111–3119. NeurIPS, 2013.

[26] Bhaskar Mitra and Nick Craswell. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, 2018.

[27] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML*, pages 807–814. Omnipress, 2010.

[28] Kanika Narang, Susan T. Dumais, Nick Craswell, Dan Liebling, and Qingyao Ai. Large-scale analysis of email search and organizational strategies. In *Proc. of CHIIR*, pages 215–223. ACM, 2017.

[29] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[30] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, et al. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *Proc. of KDD*, pages 2970–2978, 2019.

[31] Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. Multitask mixture of sequential experts for user activity streams. In *Proc. of KDD*, page 3083–3091, 2020.

[32] Zhen Qin, Zhongliang Li, Michael Bendersky, and Donald Metzler. Matching cross network for learning to rank in personal search. In *Proc. of WWW*, page 2835–2841, 2020.

[33] Filip Radlinski and Thorsten Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Proc. of AAAI*, pages 1406–1412. AAAI Press, 2006.

[34] Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

[35] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proc. of WWW*, pages 373–374. ACM, 2014.

[36] Latanya Sweeney. K-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, October 2002.

[37] Sandeep Tata, Alexandrin Popescul, Marc Najork, Mike Colagrosso, Julian Gibbons, Alan Green, Alexandre Mah, Michael Smith, Divanshu Garg, Cayden Meyer, and Reuben Kan. Quick access: Building a smart experience for google drive. In *Proc. of KDD*, pages 1643–1651. ACM, 2017.

[38] Brandon Tran, Maryam Karimzadehgan, Rama Kumar Pasumarthi, Michael Bendersky, and Donald Metzler. Domain adaptation for enterprise email search. In *Proc. of SIGIR*, pages 25–34, 2019.

[39] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. Learning latent vector spaces for product search. In *Proc. of CIKM*, pages 165–174, 2016.

[40] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. Learning to rank with selection bias in personal search. In *Proc. of SIGIR*, pages 115–124, 2016.

[41] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. Position bias estimation for unbiased learning to rank in personal search. In *Proc. of WSDM*, pages 610–618, 2018.

[42] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proc. of CIKM*, pages 1313–1322, 2018.

[43] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, June 2010.

[44] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proc. of SIGIR*, pages 55–64. ACM, 2017.

[45] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. Neural ranking models with multiple document fields. In *Proc. of WSDM*, pages 700–708, 2018.