Multi-Task Learning for Email Search Ranking with Auxiliary Query Clustering

Jiaming Shen[1*], Maryam Karimzadehgan[2], Michael Bendersky[2], Zhen Qin[2], Donald Metzler[2]
[1]Department of Computer Science, University of Illinois Urbana-Champaign, IL, USA
[2]Google Inc., Mountain View, CA, USA
[1]js2@illinois.edu [2]{maryamk, bemike, zhenqin, metzler}@google.com

ABSTRACT

User information needs vary significantly across different tasks, and therefore their queries will also differ considerably in their expressiveness and semantics. Many studies have been proposed to model such query diversity by obtaining query types and building query-dependent ranking models. These studies typically require either a labeled query dataset or clicks from multiple users aggregated over the same document. These techniques, however, are not applicable when manual query labeling is not viable, and aggregated clicks are unavailable due to the private nature of the document collection, *e.g.*, in email search scenarios. In this paper, we study how to obtain query type in an unsupervised fashion and how to incorporate this information into query-dependent ranking models. We first develop a hierarchical clustering algorithm based on truncated SVD and varimax rotation to obtain coarse-to-fine query types. Then, we study three query-dependent ranking models, including two neural models that leverage query type information as additional features, and one novel multi-task neural model that views query type as the label for the auxiliary query cluster prediction task. This multi-task model is trained to simultaneously rank documents and predict query types. Our experiments on tens of millions of real-world email search queries demonstrate that the proposed multi-task model can significantly outperform the baseline neural ranking models, which either do not incorporate query type information or just simply feed query type as an additional feature.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**;

KEYWORDS

Email Search; Neural Ranking Model; Multi-Task Learning; Query Clustering

*Work done while interning at Google.

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10.
https://doi.org/10.1145/3269206.3272019

ACM Reference Format:
Jiaming Shen[1*], Maryam Karimzadehgan[2], Michael Bendersky[2], Zhen Qin[2], Donald Metzler[2]. 2018. Multi-Task Learning for Email Search Ranking with Auxiliary Query Clustering. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3269206.3272019

1 INTRODUCTION

User search queries come in very different, diverse flavors. For example, queries can be keywords, combinations of phrases, or just natural language sentences [17]. Queries may also differ in length, grammatical structure, and ambiguity [25]. Broder [4] showed that web search queries could be navigational, informational, or transactional, and different types of queries will prefer different ranking criteria.

Based on these observations, efforts have been made on obtaining different query types and building query-dependent ranking models. For example, Wen et al. [41] proposed to cluster queries based on their clicked documents in search logs and treated each query cluster as the query type. Kang and Kim [22] classified queries into two categories using training data and then built two separate ranking models for two categories. Geng et al. [17] extracted k-nearest neighbors of each user-issued query in a labeled query pool and then constructed a ranking model based on this query subset. These methods have proven to be useful in the context of web search where either a labeled query set (for query classification) or click information aggregated across users is available. However, it is challenging to obtain/leverage such query type information or to build a query-dependent ranking model in a search scenario where manual query labeling is not viable due to privacy, and aggregated click data is unavailable.

With web mail services offering larger and larger cloud storage, email search is quickly becoming an important research topic in information retrieval [2, 9, 39, 40, 42]. In email search, users can only issue queries over their own private email collections. As shown in [9, 23], email search queries also exhibit a lot of diversity in form, as well as in intent. For example, for some queries like "recent water bill" and "chase bank statement", a reverse chronological ordering strategy would be preferable. On the other hand, for queries such as "neural model papers" and "UMAI proposal", relevance and content-based ranking would work better. As a result, some features (*e.g.*, the recency of an email) will play a different role in the ranking model for different query types, and using

a single universal ranking function is not suitable to model such query diversity.

In this paper, we propose a query-dependent ranking framework that leverages auxiliary query clustering for email search ranking. To overcome the challenges that neither labeled queries nor aggregated clicks are available in email search, we apply an unsupervised hierarchical clustering algorithm based on truncated SVD [12] and varimax rotation [21] to obtain coarse-to-fine query types. The key idea is that queries of the same type will share the same or similar query attributes [2]. Therefore, we can obtain query types by clustering queries based on these attributes. For example, we will cluster queries containing travel-related bigrams like "Uber trip" or "Lyft itinerary" and treat them as the same type.

Once we have this query type information, intuitively, we can view each query's cluster as a feature and feed it into a ranking model. Based on this idea, we propose a pairwise neural ranking model. Given a query and a pair of documents with precisely one click, this model will directly learn the query/document representation and predicts the clicked document. As the query cluster is integrated in the query features, the model is trained to leverage such cluster information. The main drawback of this model is that it fails to distinguish between query cluster features and other query features. To deal with this problem, we propose the second model based on the wide and deep architecture [10] and feed cluster-enhanced cross-product features into the wide part of network. In such a way, this model will treat the query cluster feature differently from other query features.

These two models view the query type as an input feature, same as other query features. However, both pervious research [3, 10] and our own experiment results show that neural networks can be inefficient in modeling the interactions between features directly from input layers. To solve this problem and to better leverage the query type information, we design our third ranking model using multi-task learning. This model views query cluster as the output label for auxiliary query cluster prediction task. Specifically, the model adopts a multi-task objective function and is trained to simultaneously rank documents and predicts query clusters. In such a way, the query cluster information can be propagated to all intermediate layers of neural models and also helps the model to learn better query/document representations. Our experiment on tens of millions of real-world queries also demonstrates the effectiveness of this approach.

In summary, this paper makes the following contributions:

(1) We develop a novel hierarchical clustering algorithm based on truncated SVD and varimax rotation. It works with high-dimensional input and returns multiple coarse-to-fine query types in a top-down fashion.
(2) We propose three query-dependent ranking models including a novel multi-task learning model that effectively leverages the query type information for improving the ranking performance. To the best of our knowledge, this is the first work that uses multi-task learning in the email search ranking.
(3) We conduct a thorough evaluation of our proposed models using tens of millions of real-world email search queries. The experiment results show that our multi-task learning model can significantly outperform other baseline neural ranking models.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3.1 formalizes our problem and Section 3.2 presents our hierarchical query clustering algorithm. Then, we discuss three query-dependent ranking models In Section 3.3 and Section 3.4. In Section 4, we report and analyze the experimental results. Finally, we conclude the paper and discuss some future directions in Section 5.

2 RELATED WORK

Email Search. With web mail sources offering larger and larger cloud storage, there is large amount of emails stored in the cloud. This large amount of data calls for effective email search capabilities. Compared with web search, email search has two unique challenges. First, we cannot obtain explicit relevance judgments in the form of labeled ⟨query, document⟩ pairs due to the privacy issue.. To solve this problem, we use the click data as implicit relevance judgments. Second, we cannot directly leverage the "wisdom of crowds" to aggregate clicks on the same document across different users. Consequently, many learning methods based on co-click data in web search are no longer applicable. To deal with this problem, Bendersky et al. [2] proposed an attribution parameterization method that uses click-through rate of query and document attributes (*e.g.*, frequent n-grams) to learn a ranking model. Zamani et al. [42] developed a context-aware wide and deep network model for semantic matching. In this paper, we propose a neural model that can leverage query cluster information and show the new model can significantly outperform previous approaches.

Query-dependent Ranking Models. Since user's information need can vary across different tasks, search queries can be grouped into different types. Broder [4] showed that web search queries can be navigational, informational, or transactional. Similarly, Carmel et al. [9] showed that in email search, for some queries users prefer returned emails sorted by time, while for other queries they prefer returned emails ranked by textual relevance. Intuitively, we would like to leverage such query type information and exploit different ranking models for different query types.

To build such query-dependent ranking models, we need to obtain the query type first. Many efforts have been made on query classification [1, 8, 35], which require a labeled set of queries for each query type. Such query labeling is costly, and thus some studies proposed to obtain query types by unsupervised query clustering instead of supervised query classification. Wen et al. [41] used the DBSCAN algorithm to cluster queries based on user logs. Sadikov et al. [33] applied a probabilistic model to cluster query refinements based on document co-click and session co-occurrence. Geng et al. [17] proposed to extract k-nearest neighbors of the user issued query in a labeled query pool first and then to construct a

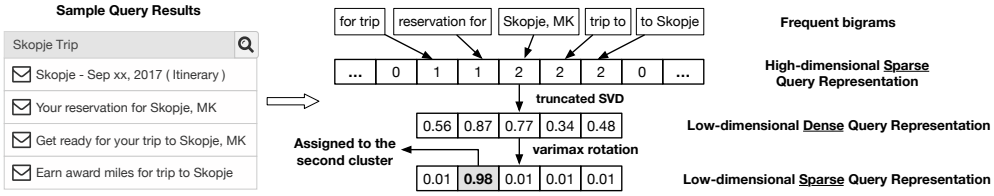


Figure 1: An illustrative example showing the query raw representation and multiple low-dimensional representations learned during query clustering process.

ranking model based on this query subset. All these methods require either labeled queries or cross-user co-click data, which is difficult or even impossible to obtain in the email search setting. In this paper, we propose a completely unsupervised hierarchical clustering method and seamlessly incorporate query clusters of different granularities in a neural ranking model.

Neural Ranking Models. Classical learning to rank models [6, 7] typically require hand-crafted features as input. Neural models, on the other hand, attempt to directly learn query/document representations and avoid tedious feature engineering. Salakhutdinov and Hinton [34] proposed to adopt auto-encoder architecture which learns binary representations of documents. More recently, Huang et al. [20] proposed DSSM architecture that represents the query and document as a bag-of-character-trigrams and trains on clickthrough data. As a special type of Siamese architecture [5], DSSM model uses a fully connected layer for query and document models. More sophisticated architectures are also explored, including those using convolutional layers [16, 36], recurrent layers [30, 31], and recursive layers [38]. Besides those representation-based models, other interaction-based models are proposed. Lu and Li [27] proposed DeepMatch which first compares different parts of the query with different parts of the document and then uses a feed forward network for computing the relevance score. Hu et al. [19] used stacked convolutional neural network models for matching two sentences. Other interaction-based models include [29, 32]. For a complete literature review on neural ranking models, please refer to [28].

3 METHODOLOGY

We first formulate our problem and define notations. Then we describe our hierarchical query clustering algorithm. Finally, we present three query-dependent ranking models, including one novel multi-task neural ranking model that is trained to rank documents and predict query types, simultaneously.

3.1 Problem Formulation

Let $\mathbb{Q} = \{Q_1, \ldots, Q_M\}$ denote a set of M queries. Each $Q_i = (\mathbf{q_i}, [(\mathbf{d_{i1}}, c_{i1}), (\mathbf{d_{i2}}, c_{i2}), \ldots, (\mathbf{d_{iN}}, c_{iN})])$ represents the i^{th} query in the query set that contains the query feature $\mathbf{q_i}$ and the document feature $\mathbf{d_{ij}}$ along with its click information c_{ij} for each document. If the j^{th} document is clicked for the i^{th} query, we set $c_{ij} = 1$, otherwise, we set $c_{ij} = 0$. Given the training query set $\mathbb{Q}$, we learn a model to find a scoring function $f(\cdot)$ that can minimize the empirical loss defined as:

$$L_{\mathbb{Q}}(f) = \frac{1}{|\mathbb{Q}|} \sum_{Q_i \in \mathbb{Q}} l(Q_i, f),$$

where $l(Q_i, f)$ denotes the loss of function f on query Q_i.

In this paper, we use the following pairwise loss function based on the pairwise topology described in [13], where it was found to be useful for weakly supervised learning:

$$\begin{aligned} l(Q_i, f) &= \sum_{\langle \mathbf{d_{ia}}, \mathbf{d_{ib}} \rangle \in Q_i} -y_{ab} \log(p_{ab}) - (1 - y_{ab}) \log(1 - p_{ab}), \\ p_{ab} &= f(\langle \mathbf{q_i}, \mathbf{d_{ia}}, \mathbf{d_{ib}} \rangle), \end{aligned}$$

where y_{ab} equals to 1 if $\mathbf{d_{ia}}$ is more relevant than $\mathbf{d_{ib}}$ (namely, $c_{ia} = 1$ and $c_{ib} = 0$) and equals to 0, otherwise.

Given a new query $Q_t = (\mathbf{q_t}, [\mathbf{d_{t1}}, \mathbf{d_{t2}}, \ldots, \mathbf{d_{tN}}])$, we use the learned $f(\cdot)$ to score each document in Q_t as follows:

$$score(\mathbf{d_{ta}}) = \frac{1}{N} \sum_{\mathbf{d_{tb}} \neq \mathbf{d_{ta}}} f(\langle \mathbf{q_t}, \mathbf{d_{ta}}, \mathbf{d_{tb}} \rangle).$$

After we obtain the score for each document, we can either directly rank documents based on these scores or feed them as additional ranking signals into a learning-to-rank framework.

3.2 Hierarchical Query Clustering

In this section, we describe how we represent the email search query and how we conduct query clustering based on the query representation.

Query Representation. Due to privacy issues in email search, we can only access very limited query features that do not reveal any information about the user, such as frequent n-grams[1]. Therefore, we follow the approach in [17] and first apply a reasonable baseline ranker (*e.g.*, BM25) to obtain a pre-ranked document list. Then, we aggregate the features of top-ranked documents in this pre-ranked list and use them to enhance the query representation. The philosophy is that although these documents may not be in the optimal ranking order, they can still provide useful information for a query. A similar idea is proved useful in [13].

We explain the details by following an example in Figure 1. Suppose a user issues a query "Skopje trip" and obtains a preliminarily ranked email list. Based on the top 4 emails in this ranked list, as well as the query itself, we can collect five frequent 2-grams, one from the query and the other four from documents. Then we represent this query using a high-dimensional sparse vector. For example, the bigram "to Skopje" appears twice in the top 4 ranked emails and thus the feature value corresponding to this bigram equals to 2.

[1]All available feature details are presented in the experiment section.

Query Clustering. As the generated query feature vector is of high dimensionality but very sparse, we need to design an algorithm which can work with high-dimensional input and can leverage such sparsity. Also, we want our algorithm to scale to billions of examples with potentially thousands of clusters. Traditional clustering algorithms such as k-means and DBSCAN can not satisfy all these requirements at the same time [18], and therefore, we design a hierarchical clustering algorithm which returns multiple coarse-to-fine query clusters by learning low-dimensional query representations.

At the high level, query clusters are constructed recursively in a top-down manner. Initially, all queries are put in the root, denoted as the level-0 cluster. Then, to assign each query to one of the level-1 clusters, we do the following two steps. First, we train a truncated SVD model [12] based on the query high-dimensional sparse representation. We choose to use truncated SVD model because it can leverage the sparsity of input query feature and outputs a dense low-dimensional representation of this query, as shown in Figure 1. Second, we apply a varimax rotation [21] on top of the dense low-dimensional query representation. The varimax rotation tries to project each query vector into as few axes as possible which returns a sparse low-dimensional representation of the query. Each dimension in the final sparse low-dimensional query vector represents a level-1 cluster and the dimension with the highest score is taken as the query's level-1 cluster assignment. For example, the sample query in Figure 1 is put in the second level-1 cluster as it has the highest score in the second dimension.

After all queries are put into one of the level-1 clusters (*e.g.*, cluster-1, cluster-2, ...), we proceed to construct all level-2 clusters (*e.g.*, cluster-1.1, cluster-1.2, cluster-2.1, cluster-2.2, ...) under each level-1 clusters. To achieve this, we learn a new truncated SVD model based on all queries that reside in one specific level-1 cluster and apply the varimax rotation again. We essentially re-learn a new sparse low-dimensional query representation because we want to take advantage of the fact that different subsets of the data may be best described by their own distance metrics. Also, note that we can construct the level-2 clusters under one specific level-1 cluster (*e.g.*, cluster-1) without any knowledge on other level-1 clusters (*e.g.*, cluster-2, cluster-3, ...). Therefore, we can implement this hierarchical clustering algorithm in a distributed fashion and allow it to scale to billions of examples.

Algorithm 1 summarizes our divisive hierarchical query clustering algorithm (non-recursive version). Given a collection of queries $\mathbb{Q}$, the depth of hierarchy D and the number of branches in each level B, it first puts all queries in the root cluster node. Then, for each selected cluster c of depth d, if the number of queries in it passes the minimal threshold (line 7), the algorithm will fit the subspace based on all the queries in c and then learn their sparse low-dimensional representations F_c. Otherwise, this leaf cluster c will be pruned. After that, it constructs B sub-clusters and assigns each query $q_i \in S_c$ to one of them based on F_c. In such a way, each query will be assigned to at most one cluster per hierarchy level.

Algorithm 1: Divisive Hierarchical Query Clustering

Input: A collection of queries $\mathbb{Q}$; the depth of hierarchy D; the number of branch in each level B; the minimal required examples in each leaf node E.
Output: A hierarchical cluster tree T of queries $\mathbb{Q}$.

```
1  Assign all queries to root cluster at depth d = 0;
2  Initialize T ← ∅;
3  for depth d from 0 to D do
4      for cluster c at depth d do
5          S_c ← queries assigned to c;
6          if d = D then
7              if |S_c| < E then
8                  T ← T − {c} // Prune this leaf node;
9              Continue;
10         F_c ← VARIMAX(TRUNCATED-SVD(S_c, B));
11         N_1, ..., N_B ← QUERY-ASSIGN(F_c);
12         for sub cluster index i from 1 to B do
13             T ← T ∪ N_i;
14 Return hierarchical cluster tree T;
```

3.3 Pairwise query-dependent ranking models

We describe two pairwise neural ranking models in this section. The first one is motivated by the semantic matching model in [20], and especially its application in the email search setting [42]. Instead of using the *pointwise* topology in [20], we adopt a *pairwise* topology which is proved to be effective in the weakly supervised setting [13]. Following we refer to such Deep Pairwise Ranking Model as DPRM.

The network architecture of DPRM is shown in Figure 2(a). The input includes a query q and a pair of documents associated with this query. Specifically, one document is clicked while the other one is not clicked. The raw input feature of query and documents are described in Section 4.1. Notice that some features are sparse categorical features (e.g., frequent subject n-grams) while the others are dense continuous features (e.g., email's recency). We feed the sparse features (for both query and documents) into an embedding layer which directly learns their dense representations[2].

After obtaining the query/document representations, we concatenate them and feed them into a stack of hidden layers. These hidden layers are fully connected, and the representation in $(i+1)^{th}$ hidden layer is obtained by mapping the representation in i^{th} hidden layer as follows:

$$\mathbf{h_{i+1}} = g(\mathbf{W_i^T} \cdot \mathbf{h_i} + \mathbf{b_i}),$$

where $\mathbf{W_i}$ is the transformation matrix from layer i to layer $i+1$, the $\mathbf{b_i}$ is the bias vector, and the activation function $g(\mathbf{x}) = \max(\mathbf{x}, 0)$ is the rectifier linear unit (ReLU) applied on each dimension of $\mathbf{x}$. Finally, in the output layer, we use one fully connected neuron with sigmoid activation function $\sigma(z) = \frac{1}{1+e^{-z}}$ to calculate the probability that document d_A

[2] We will replace all sparse features with frequency less than a threshold (in the training corpus) with a special unknown token UNK. Then, during the testing time, an unseen sparse feature will also be treated as UNK and thus has the same embedding of UNK.

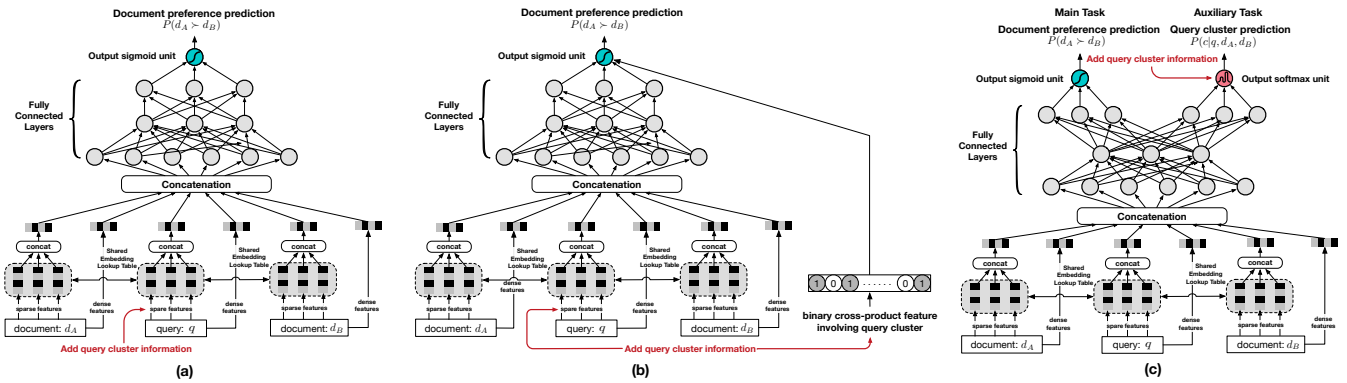


Figure 2: The network architecture of (a) QC-DPRM. (b) QC-WDPRM. (c) QC-MTLRM. Note that the query cluster information in QC-DPRM and QC-WDPRM contributes in a "bottom-up" fashion from the input layer, while the QC-MTLRM models it as the label for auxiliary task thus enables it to contribute in a "top-down" fashion.

is preferred (i.e., more likely to be clicked) than document d_B. To incorporate query cluster information in DPRM, we treat each query's clusters as additional sparse features and feed them into the embedding layer. Similarly, we refer to such Query Cluster aware DPRM as QC-DPRM.

One limitation of QC-DPRM is that it models the sparse query cluster feature and other sparse query features (e.g., query language) in the same manner. To explicitly differentiate these two types of features, we propose the second model based on the wide & deep model similar to the one in [10]. Our model architecture is shown in Figure 2(b). The key idea is to jointly train a wide linear model (with potential cross-product features) as well as a deep neural model, and therefore combine the benefits of memorization and generalization. This simple approach is proved useful for building recommender system [10], and modeling query context features [42].

In the context of email search ranking, we manually design a set of binary cross-product features involving query clusters and feed them into *the wide part of the model*. These cross-product features can capture important feature interactions and add non-linearity to the wide linear model. For example, a cross-product feature "query_cluster=1 AND language=English" is activated (*i.e.*, equals to 1) if and only if this query is in cluster-1 and is of language English. By requiring each cross-product feature in the wide part of model to contain one query cluster feature, we enable the model to differentiate between the query cluster feature and other sparse query features. We refer to such model as QC-WDPRM in the remainder of this paper.

3.4 Multi-task query-dependent ranking model

The QC-DPRM model and QC-WDPRM model essentially treat the query cluster information as an input feature. However, both previous research [3, 10] and our own experiment results show that neural networks can be inefficient in modeling the interactions between features directly from input layers. To solve this problem and better leverage the query type information, we take an alternative approach in this section and treat the query cluster as the "label" of an auxiliary query cluster prediction task. We propose our third ranking model based on the multi-task learning framework. Given a query and a pair of documents, this model simultaneously predicts which document is more likely to be clicked and which cluster(s) this query will reside in. The query cluster information will be propagated to every intermediate layer in a top-down fashion, which allows our model to learn a better representation of query and documents. As shown in [26], good query/document representations learned by incorporating more knowledge from different tasks can significantly improve the ranking performance.

The architecture of our proposed query-dependent multi-task ranking model is shown in Figure 2(c). The lower layers are shared across two tasks. The top layers are different in each task. For the task of document preference prediction, we use fully connected hidden layers and an output layer composed of one single neuron. The output layer returns the probability that document A is preferred to document B. Given a query q, a pair of documents $\langle d_A, d_B \rangle$ with precisely one clicked, we define the following log-loss for this query:

$$l^{rank}(q) = -y_{ab} \log(p_{ab}) - (1 - y_{ab}) \log(1 - p_{ab}), \quad (1)$$

$$p_{ab} = P(d_A \succ d_B), \quad (2)$$

where y_{ab} equals to 1 if document d_A is clicked and equals to 0 otherwise.

For the task of predicting query clusters, we use one additional fully connected hidden layer and a softmax output layer, on top of the shared layers. The dimensionality of the final softmax layer is the same as the number of query clusters, and it outputs the query cluster distribution. We use the cross-entropy loss as the objective for this task:

$$l^{cluster}(q) = -\sum_{c \in C} p_c \log \hat{p_c}, \quad (3)$$

$$\hat{p_c} = p(c|q, d_A, d_B), \quad (4)$$

where p_c is the true distribution on query cluster space[3].

In order to combine these two tasks, we define a joint multi-task objective function, as follows:

$$L(\Theta) = \frac{1}{|Q|} \sum_{q \in Q} \left(l^{rank}(q) + \lambda \cdot l^{cluster}(q) \right), \quad (5)$$

where Θ represents all the neural network parameters and λ, named `mix_rate`, is used to balance the ranking loss and the query cluster prediction loss. We study how λ influences the model performance in Section 4.2.

Note that the above objective function can be transformed into the following form:

$$L(\Theta) = \frac{1}{|Q|} \sum_{q \in Q} l^{rank}(q) + \lambda \left(\frac{1}{|Q|} \sum_{q \in Q} l^{cluster}(q) \right). \quad (6)$$

The first term on the right-hand side of the equation is essentially the objective function used in QC-DPRM and QC-WDPRM. The second term derived originally from the loss on query cluster prediction can be seen as a regularization (as it will influence how feature representations are learned) and is controlled by the parameter λ. In the remainder of this paper, we refer this multi-task ranking model as QC-MTLRM.

4 EXPERIMENTS

In this section, we first describe the data sources and features used in our experiments. Then, we evaluate our proposed models and study the influence of some important model hyper-parameters. Finally, we feed the output of each model as an additional signal into an end-to-end ranking pipeline, and explore how much improvement can be achieved.

4.1 Data

To the best of our knowledge, there is no publicly available *large-scale* email search dataset for training neural ranking models, probably because it is too private and too sensitive. Therefore, in this paper, we use the data derived from the search click logs of a commercial email service. The training set contains approximately 66 million queries, and there are about 4 million and 9 million queries in the development and testing set, respectively. All queries in the development set are issued strictly later than all queries in the training set, and all queries in the testing set are issued strictly later than all queries in the development set[4]. We construct datasets in such a way to avoid the potential data leakage problem. Each query has six candidate documents with precisely one clicked[5], and the purpose is to rank these six documents for the given query. Such large scale dataset also shows the scalability of our proposed models.

The original datasets are anonymized based on k-anonymity approach [37], and thus the only content features we can access are the query and document n-grams that are frequent in the whole corpus. We use those frequent n-grams, as well as some other category and structure features [2] for query clustering. Besides these features, we also use the situational features including the language and timestamp of search requests to learn the ranking models, as described in [42]. All features are summarized in Table 1.

We use click data as ground truth labels to learn and evaluate our proposed ranking models, which is a standard practice for email search [2, 9, 39, 42]. Furthermore, we apply the position bias correction techniques [11, 39, 40] to reduce the noise in click data during our model training.

4.2 Query-dependent Ranking Model Evaluation

4.2.1 *Experimental Setup.*

We implement our neural network models using TensorFlow. We set the depth of hierarchy D to be 3 and the number of branches B to be 7. We analyze the effect of these clustering hyper-parameters in Section 4.2.3. We tune the neural network hyper-parameters over the development set as follows: we sweep the learning rate between {0.05, 0.08, 0.1, 0.15, 0.3}, the dropout rate between {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}, the number of hidden layers between {3, 4}, the size of first hidden layers between {64, 128, 256, 512, 768, 1024}, the embedding dimension between {20, 30, 40, 50, 100}, and the optimization algorithm between {Adagrad [14], Adam [24]}. For each model, we select the combination of hyper-parameters that have the best performance on the development set and report its result on the testing set.

4.2.2 *Evaluation Metric.*

We evaluate model performance using mean reciprocal rank (MRR). Since each query has precisely one clicked document, the MRR is calculated as follows:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i},$$

where Q denotes the evaluation set and $rank_i$ represents the rank position of the clicked document for i^{th} query in the evaluation set.

Similar to [42], we also use success@1 and success@5 to evaluate our models. The success@k measures the percentage of queries for which the clicked message is ranked within the top-k results[6][9]. Finally, to compare the performance of multiple models, we conduct statistical significance test using the two-tailed paired t-test with 99% confidence level.

4.2.3 *Results and Discussion.*

In this section, we first show some query n-gram clusters. Then, we empirically study whether and how the query cluster feature can help improve the ranking results. Finally, we

[3] As we are using a hierarchical clustering algorithm, each query may reside in multiple query clusters (*e.g.*, cluster-1, cluster-1.1, cluster-1.1.2). Suppose we have 100 valid query clusters (of different granularities), then this is essentially a distribution over these 100 query clusters (with only three non-zero values).

[4] Namely, we will not train a model using queries issued in May 2018 and test its performance on queries issued in April 2018.

[5] These six candidates are presented in the dropdown menu while users type their queries in the search bar but before they click the search bottom. When users find the target email, the system will direct them to the exact email and thus generates precisely one click.

[6] In fact, success@1 is the precision of document ranked at the first position, and success@5 is the accuracy of the system in *not* retrieving the correct (clicked) document at the last position.

Table 1: Summary of the query and document features used for clustering queries and learning ranking models.

Feature Type	Descriptions	Usage
Content	List of frequent n-grams appearing in the query text and the email subject *e.g., "Class schedule on Friday morning" → ["class schedule", "Friday morning"].*	Cluster queries Learn ranking models
Category	Small set of commonly used email labels *e.g., Promotions, Forums, Purchases, and etc.* (see [2] for detailed label examples)	Cluster queries Learn ranking models
Structure	Frequent machine-generated email subject templates *e.g., Your trip confirmation number 12345 → Your trip confirmation number ** (see [2] Table 2 for more details on structure features)	Cluster queries Learn ranking models
Situational	Temporal and Geographical features of current search request *e.g., Friday, 8:00pm, USA, Japan* (see [42] for more details)	Learn ranking models

Table 2: Ranking performance of each proposed query-dependent ranking model. Relative performances compared with the baseline DPRM model are shown in parentheses. The superscript * means the improvement is statistically significant.

Method	MRR	success@1	success@5
DPRM	0.6698	0.4874	0.8861
QC-DPRM	0.6697 (-0.01%)	0.4873 (-0.02%)	0.8864 (+0.03%)
QC-WDPRM	0.6699 (+0.01%)	0.4875 (+0.02%)	0.8862 (+0.01%)
QC-MTLRM	**0.6748 (+0.70%)***	**0.4939 (+1.32%)***	**0.8875 (+0.17%)***

analyze how some important hyper-parameters affect the model performance.

Query n-gram clusters. Due to the private nature of email search queries, we cannot show the query clustering results directly. Instead, we show the most distinctive n-gram features in Figure 3a. The distinctiveness of each n-gram s in cluster c is defined as $\frac{cnt(s|c)}{cnt(s)}$, where $cnt(s)$ is the occurrence count of s in the entire query set, and $cnt(s|c)$ is the occurrence count of s in the queries of cluster c. These distinctive n-grams give us a hint about a query cluster's topic. As we can see, the queries about topic *travel* are clustered together at the first level, which then be divided further into subtopics including *car rental*, *air flight*, and *ride sharing*.

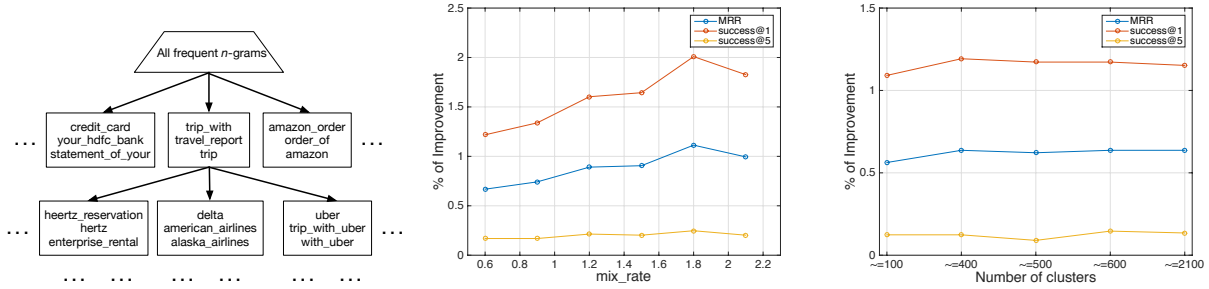
Effect of query cluster as raw feature. To understand how the query cluster may contribute to the final ranking model as a raw input feature, we first feed it into the baseline DPRM model as an additional query feature. Since there are hundreds and thousands of query clusters but each query can be assigned to at most "number of tree depth" clusters, we treat query cluster as a sparse feature and employ an embedding layer on top of it. Similarly, we generate cross-product features involving query cluster and feed them into the wide part of QC-WDPRM.

Table 2 reports the relative improvements achieved by the above two query cluster enhanced models. We can see that simply adding query cluster as a raw feature cannot improve the baseline DPRM model. The reason is that the DPRM architecture cannot differentiate between the query cluster feature and some other sparse query features. As for the QC-WDPRM model, it does allow query cluster features to contribute differently compared to other sparse query features via the feature crosses in the wide part. However, the query cluster is still treated as an input-level feature in the QC-WDPRM model.

Effect of query cluster as separate label. To reflect the intuition that query cluster information should guide how other features contribute to the ranking, we proposed the QC-MTLRM model. Instead of considering the query cluster as a feature, the QC-MTLRM model treats the query clusters as a separate "label" and learns how other query/document features can predict such a label. Results in Table 2 show that the QC-MTLRM model significantly outperforms the already-strong baseline DPRM model and can achieve 0.7% and 1.32% improvements in terms of MRR and success@1, respectively. As we evaluate our models on tens of millions of search queries, although the numbers may appear small, the improvements in practice are quite substantial. In fact, a +1% MRR improvement would be considered as a highly significant launch.

Effect of `mix_rate`. During the training, QC-MTLRM optimizes a joint loss including the log-loss of document preference prediction and the cross-entropy loss of query cluster prediction. To balance these two losses, we introduce a hyper-parameter `mix_rate` in Eq. (6). In all previous experiments, we tune this hyper-parameter using a development set and then test the performance of QC-MTLRM with this `mix_rate` fixed. As a record, the optimal `mix_rate` tuned on development set is 0.9. In this experiment, however, we intend to study how this hyper-parameter will influence the ranking performance and how QC-MTLRM is sensitive to the choice of `mix_rate`. Therefore, we train multiple QC-MTLRM models with different `mix_rate` and directly report their performance on the testing data. The results are shown in Figure 3b. First, we notice that a wide range of `mix_rate` choices can help improve QC-MTLRM's ranking performance (*i.e.*, give us positive relative improvements). Second, we find that the performance of QC-MTLRM first increases as `mix_rate` increases until it reaches about 1.8 and then starts to decrease when we further increase `mix_rate`. Notice that even the previous optimal `mix_rate` tuned on development set is different from the optimal value 1.8 obtained directly on test set, our QC-MTLRM model can still outperform baseline DPRM model. We hypothesize that if we further increase the size of development set which gives more reliable estimation of `mix_rate`, the relative improvement of QC-MTLRM model over DPRM model will be more significant.

Effect of query cluster number. One important factor of QC-MTLRM is the number of query clusters. To have an insight into how this factor may influence the ranking performance, we train multiple QC-MTLRM models by varying



(a) (b) (c)

Figure 3: (a) Most distinctive frequent n-grams for each query cluster. (b) Relative improvement of QC-WDPRM over DPRM v.s. `mix_rate`. (c) Relative improvement of QC-WDPRM over DPRM v.s. cluster number.

the query cluster number and fixing all the other hyper-parameters. The result is plotted in Figure 3c. As we can see, the performance boost appears to not be very sensitive to the number of query clusters. We suspect the reason for this phenomenon is that around 100 query clusters have already covered most of the important data-dependent information.

4.3 End-to-End Ranking Model Evaluation

Nowadays, the production-level search engines commonly learn a final ranking function by leveraging multiple signals, which is also widely adopted in personal search scenario [2, 39, 42]. Therefore, in this experiment, we study whether we can use the output of QC-MTLRM model (as a separate feature) alongside many other ranking signals to learn a global ranking function and improve the final performance.

4.3.1 *Experimental Setup.*

Following [42], we use an adaptive learning-to-rank framework based on multiple adaptive regression tree (MART) algorithm [15]. Again, we apply the position bias correction techniques [11, 39] during the model training.

4.3.2 *Evaluation Metric.*

To evaluate the results from online production system, we use weighted mean reciprocal rank (WMRR) proposed in [39] and weighted average click position (WACP) as our primary metrics. They are calculated as follows:

$$WMRR = \frac{1}{\sum_{i=1}^{|Q|} w_i} \sum_{i=1}^{|Q|} w_i \cdot \frac{1}{rank_i},$$

$$WACP = \frac{1}{\sum_{i=1}^{|Q|} w_i} \sum_{i=1}^{|Q|} w_i \cdot rank_i,$$

where w_i denotes the bias correction weight, and it is inversely proportional to the probability of observing a click at the clicked position. We set those weights using result randomization, as described in [39]. Here, the lower WACP number indicates the better model performance[7].

4.3.3 *Results and Discussion.*

[7] As the lower WACP number indicates the better model performance, a model reduces 4% WACP is better than another one which reduces 3% WACP.

Table 3: Relative improvements achieved by adding the output of each proposed model as a separate signal to our current personal ranking model, compared with the performance of the ranking model with only DPRM model output as signal (denoted as LTR). The superscript $*$ means the improvement is statistically significant over the LTR and the superscript $**$ indicates the improvement is statistically significant over both the LTR and the LTR+DPRM.

Method	WMRR	WACP
LTR + DPRM	+2.35%*	-3.24%*
LTR + QC-DPRM	+2.32%*	-3.20%*
LTR + QC-WDPRM	+2.35%*	-3.28%*
LTR + QC-MTLRM	**+2.52%****	**-3.41%****

In this experiment, we first adopt a general learning to rank (LTR) framework with many standard email search signals [9] as well as situational context signals [42]. We then feed the output of each of the DPRM with/without query cluster, the QC-WDPRM with query cluster, and the QC-MTLRM with query cluster as an additional feature into the learning to rank framework. In such an end-to-end setting, we can understand how much value will be added to the final ranking model.

We show the relative improvement achieved by each model in Table 3. First, we can see that adding the DPRM feature can significantly improve the overall ranking performance which confirms the previous finding [42]. Then, we notice that the performance boost by introducing QC-DPRM is less than simply using the raw DPRM feature without query cluster, which may be caused by the noise in query clusters. Finally, we find that adding QC-MTLRM feature can achieve the best performance boost among all the considered models.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we study how to improve email search ranking by capturing query type information in an unsupervised fashion and constructing query-specific ranking models. We first develop a hierarchical clustering algorithm based on truncated SVD and varimax rotation to obtain query type information. Then, we propose three query-dependent neural ranking models. The first two models leverage query type information as additional features, while the third one views query cluster as additional label and conducts document ranking and query cluster prediction simultaneously using multi-task learning.

This multi-task learning scheme enables query type information to affect all intermediate layers of the neural model and guides the model to learn better query/document representations. We evaluate our proposed approach on over 75 million real-world email search queries. The experiments demonstrate the effectiveness of our query clustering algorithm and show that our novel multi-task model can significantly outperform the baseline models which either do not incorporate query type information or just simply treat the query type as an additional feature.

In the future, we would like to further study the following directions: (1) We may apply the current hierarchical clustering method to cluster users and similarly build a user-specific multi-task ranking model for email search. (2) We can extend the pairwise ranking paradigm to listwise ranking paradigm and it would be interesting to see how the query type information would help in listwise ranking models. (3) We will explore how to choose the number and granularities of clusters that are most suitable for learning query-dependent ranking models.

REFERENCES

[1] Steven M. Beitzel, Eric C. Jensen, Abdur Chowdhury, and Ophir Frieder. 2007. Varying approaches to topical web query classification. In *SIGIR*.

[2] Michael Bendersky, Xuanhui Wang, Donald Metzler, and Marc Najork. 2017. Learning from User Interactions in Personal Search via Attribute Parameterization. In *WSDM*.

[3] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed Huai hsin Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM*.

[4] Andrei Broder. [n. d.]. A taxonomy of web search. In *ACM SIGIR Forum*.

[5] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using A "Siamese" Time Delay Neural Network. In *IJPRAI*.

[6] Christopher J.C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview.

[7] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In *ICML*.

[8] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. 2009. Context-aware query classification. In *SIGIR*.

[9] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by Time or by Relevance?: Revisiting Email Search. In *CIKM*.

[10] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Gregory S. Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. *CoRR* abs/1606.07792 (2016).

[11] Nick Craswell, Onno Zoeter, Michael J. Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *WSDM*.

[12] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by Latent Semantic Analysis. *JASIS* (1990).

[13] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR*.

[14] John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In *COLT*.

[15] Jerome H. Friedman. 1999. Greedy Function Approximation: A Gradient Boosting Machine.

[16] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. 2014. Modeling Interestingness with Deep Neural Networks. In *EMNLP*.

[17] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Harry Shum. 2008. Query dependent ranking using K-nearest neighbor. In *SIGIR*.

[18] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[19] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional Neural Network Architectures for Matching Natural Language Sentences. In *NIPS*.

[20] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*.

[21] Henry F Kaiser. [n. d.]. The varimax criterion for analytic rotation in factor analysis. *Psychometrika* ([n. d.]).

[22] In-Ho Kang and Gil-Chang Kim. 2003. Query type classification for web document retrieval. In *SIGIR*.

[23] Jin Young Kim, Nick Craswell, Susan T. Dumais, Filip Radlinski, and Fang Liu. 2017. Understanding and Modeling Success in Email Search. In *SIGIR*.

[24] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[25] Hang Li, Gu Xu, W. Bruce Croft, Michael Bendersky, Ziqi Wang, and Evelyne Viegas. 2012. QRU-1: A Public Dataset for Promoting Query Representation and Understanding Research. In *WSDM Workshop*.

[26] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015. Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval. In *HLT-NAACL*.

[27] Zhengdong Lu and Hang Li. 2013. A Deep Architecture for Matching Short Texts. In *NIPS*.

[28] Bhaskar Mitra and Nick Craswell. 2017. Neural Models for Information Retrieval. *CoRR* (2017).

[29] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *WWW*.

[30] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Kreidieh Ward. 2014. Semantic Modelling with Long-Short-Term Memory for Information Retrieval. *CoRR* (2014).

[31] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Kreidieh Ward. 2016. Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* (2016).

[32] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. 2016. A Study of MatchPyramid Models on Ad-hoc Retrieval. *CoRR* (2016).

[33] Eldar Sadikov, Jayant Madhavan, Lu Wang, and Alon Y. Halevy. 2010. Clustering query refinements by user intent. In *WWW*.

[34] Ruslan Salakhutdinov and Geoffrey E. Hinton. 2009. Semantic hashing. *Int. J. Approx. Reasoning* (2009).

[35] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. 2006. Building bridges for web query classification. In *SIGIR*.

[36] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *WWW*.

[37] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (2002).

[38] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*.

[39] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*.

[40] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. (2018).

[41] Ji-Rong Wen, Jian-Yun Nie, and HongJiang Zhang. 2002. Query clustering using user logs. *ACM Trans. Inf. Syst.* (2002).

[42] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *WWW*.