Target Apps Selection: Towards a Unified Search Framework for Mobile Devices

Mohammad Aliannejadi
Faculty of Informatics
Università della Svizzera italiana (USI)
mohammad.alian.nejadi@usi.ch

Hamed Zamani
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
zamani@cs.umass.edu

Fabio Crestani
Faculty of Informatics
Università della Svizzera italiana (USI)
fabio.crestani@usi.ch

W. Bruce Croft
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
croft@cs.umass.edu

ABSTRACT

With the recent growth of conversational systems and intelligent assistants such as Apple Siri and Google Assistant, mobile devices are becoming even more pervasive in our lives. As a consequence, users are getting engaged with the mobile apps and frequently search for an information need in their apps. However, users cannot search within their apps through their intelligent assistants. This requires a *unified mobile search* framework that identifies the target app(s) for the user's query, submits the query to the app(s), and presents the results to the user. In this paper, we take the first step forward towards developing unified mobile search. In more detail, we introduce and study the task of *target apps selection*, which has various potential real-world applications. To this aim, we analyze attributes of search queries as well as user behaviors, while searching with different mobile apps. The analyses are done based on thousands of queries that we collected through crowdsourcing. We finally study the performance of state-of-the-art retrieval models for this task and propose two simple yet effective neural models that significantly outperform the baselines. Our neural approaches are based on learning high-dimensional representations for mobile apps. Our analyses and experiments suggest specific future directions in this research area.

1 INTRODUCTION

Recent years have witnessed a rapid growth in the use of mobile devices, enabling people to access the Internet in various contexts. More than 77% of Americans now own a smartphone[1], with an increasing trend in terms of the time people spend on their phones. As of 2016, the average U.S. user spends 5 hours on mobile devices per day, with just 8% of it spent in the phone's browser. In fact, people spend most of their time (72%) using apps that have their own search feature[2]. Moreover, Google Play Store now features more than 3.5 million apps and users install an average of 35 mobile apps on their phones, using half of them regularly[3].

More recently, with the release of conversational search systems such as Google Assistant, Apple Siri, and Microsoft Cortana, people are experiencing mobile search through a single voice-based interface. These systems introduce several research challenges. Given that people spend most of their times in apps and, as a consequence, most of their search interactions would be with apps (rather than a browser), one limitation is that users are unable to use a conversational system to search within many apps. This suggests the need for a *unified search framework* that *replaces all the search boxes in the apps, with a single search box*. With such a framework, the user can submit a query through this system which will identify the target app(s) for the issued query. The query is then routed to the identified target apps and the results are displayed in a unified interface.

In this work, we are particularly interested in taking the first step towards developing a unified search framework for mobile devices by introducing and studying the task of *target apps selection*, which is defined as identifying the target app(s) for a given query. To this end, we built a collection of cross-app search queries through crowdsourcing, which is released for research purposes[4]. Our crowdsourcing experiment consists of two parts: we initially asked crowdworkers to explain their latest search experience on their smartphones and used them to define various realistic mobile search tasks. Then, we asked another set of workers to select the apps they would choose to complete the tasks as well as the query they would submit. We investigate various aspects of user behaviors while completing a search task. For instance, we show that users choose to complete most of the search tasks using two apps. In addition, we demonstrate that for the majority of the search tasks, most of the users prefer *not* to use Google Search.

From the lessons learned from our data analysis, we propose two simple yet efficient neural target apps selection (NTAS) models. Our first model looks at the problem as a ranking task and produces a score for a given query-app pair. We study two different training settings for this model. Our second framework, on the

[1] http://www.pewinternet.org/fact-sheet/mobile/

[2] http://flurrymobile.tumblr.com/post/157921590345/us-consumers-time-spent-on-mobile-crosses-5

[3] https://www.thinkwithgoogle.com/advertising-channels/apps/app-marketing-trends-mobile-landscape/

[4] Available at http://aliannejadi.com/unimobile.html

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210039

other hand, casts the problem as a multi-label classification task. Both neural approaches, called NTAS, learn a high-dimensional representation for each app. Our experiments demonstrate that our model significantly outperforms a set of state-of-the-art models in this task.

In summary, the main contributions of this paper are:

- Designing and conducting two crowdsourcing tasks for collecting cross-app search queries for real-life search tasks. The tasks and queries are publicly available for research purposes.
- Presenting the first study of user behaviors while searching with different apps as well as their search queries. In particular, we study the attributes of the search queries that are submitted to different apps and user behaviors in terms of the apps they chose to complete a search task.
- Proposing two neural models for target apps selection.
- Evaluating the performance of state-of-the-art retrieval models for this task and comparing them against our proposed method.

Our analyses and experiments suggest specific future directions in this research area.

2 RELATED WORK

While the study of unified mobile search is a new research area, it has roots in previous research. Our work is related to the areas of mobile IR, federated, and aggregated search. Moreover, relevant research has been done in the area of proactive IR where a system aims to provide personalized information cards to users based on their context. Other relevant works can be found in the areas of query classification, neural networks, and crowdsourcing. In the following, we summarize the related research in each of these areas.

Mobile IR. One of the main goals of mobile IR is to enable users to carry out all the classical IR operations using a mobile device [14]. One of the earliest studies on mobile IR was done by Kamvar and Baluja [20] where they did a large-scale mobile search query analysis. They found mobile searches were less diverse in terms topic. In another study, Church et al. [11] argued that the conventional Web-based approaches fail to satisfy users' information needs. In fact, Song et al. [33] found significant difference in search patterns done using iPhone, iPad, and desktop. In a more recent study, Guy [18] conducted an analysis on mobile spoken queries as opposed to typed-in queries. They found that spoken queries are longer and closer to natural language. These findings were in line with an older study by Crestani and Du [13].

More recently, research has been done on various topics in mobile IR such as app and venue recommendation as well as app search [1, 26, 27]. For instance, Shokouhi et al. [32] studied query reformulation patterns in mobile query logs and found that users do not tend to switch between voice and text while reformulating their queries. Park et al. [27] represented apps using online reviews for improved app search on the market. Williams et al. [35] leveraged mobile user gesture interactions, such as touch actions, to predict good search abandonment on mobile search. Park et al. [26] inferred users implicit intentions from social media for the task of app recommendation. Harvey and Pointon [19] found that fragmented attention of users while searching on-the-go, affects their search objective and performance perception. In contrast to the prior work, we explore how users behave while searching with different apps. To do this, we study the attributes of search queries assigned to different apps.

A few industrial systems exist aiming to provide users with unified mobile search. Apple Spotlight[5] is the most popular example of such systems that is available on iOS devices. Also, Sesame Shortcuts[6] is an Android app that creates easy-to-access shortcuts to the installed apps. The shortcuts are also accessible via keyword-based queries. Despite the existence of these systems, research on cross-app search has not yet been done.

Proactive IR. The aim of proactive IR systems is to anticipate users' information needs and proactively present information cards to them. Shokouhi and Guo [31] analyzed user interactions with information cards and found that the usage patterns of the cards depend on time, location, and user's reactive search history. Benetka et al. [7] showed that information needs vary across activities as well as during the course of an activity. They proposed a method to leverage users' check-in activity for recommending information cards. Our work focuses on the queries that users issue in different apps. Queries can express complex information needs that are impossible to infer from context.

Federated and aggregated search. A unified mobile search system distributes a search query to a limited number of apps that it finds more relevant to a search query. There is a considerable overlap between the target apps selection task and federated/aggregated search. In federated search, the query is distributed among uncooperative resources with homogeneous data; whereas in aggregated search, the content is blended from cooperative resources with heterogeneous data [4]. Given the uncooperative environment of most federated search systems, Callan and Connell [8] proposed a query-based sampling approach to *probe* various resource providers and modeled them based on the returned results. In most aggregated search systems, on the other hand, different resources are parts of a bigger search system and thus cooperative. Moreover, an aggregated search system can even access other metadata such as users' queries and current traffic [4]. Diaz [16] proposed modeling the query dynamics and collection to detect news queries for integrating the news *vertical* into the result page. This work was later extended by Arguello et al. [6] to include images, videos, and travel information. In this work, we assume an uncooperative environment because the contents of apps are not accessible to the unified search system. Moreover, given the existence of various content types in different apps, we assume the documents to be heterogeneous.

Query classification. Our work is also related to the research in query classification where different strategies are taken to assign a query to predefined categories. Kang and Kim [21] defined three types of queries arguing that search engines require different strategies to deal with the queries belonging to each of the classes. Shen et al. [30] introduced an intermediate taxonomy used to classify queries to specified target categories. Cao et al. [9] leveraged conditional random fields to incorporate users' neighboring queries

[5] https://en.wikipedia.org/wiki/Spotlight_(software)

[6] http://sesame.ninja/

Table 1: Distribution of crowdsourcing search task categories.

Search Category	% of tasks
General Information & News	13%
Video & Music	12%
Image	9%
Social Networking	9%
App	9%
File & Contact	8%
Online Shopping	13%
Local Services & Navigation	15%
Email & Event	12%

in a session as context. More recently, Zamani and Croft [39] studied word embedding vectors for the query classification task and proposed a formal model for query embedding estimation.

Neural IR. The recent and successful development of deep neural networks for various tasks has also impacted IR applications. In particular, neural ranking models have recently shown significant improvements in a wide range of IR tasks, such as ad-hoc retrieval [17], question answering [37], and context-aware retrieval [38]. These approaches often rely on learning high-dimensional dense representations that carry semantic information. They can be particularly useful to match queries and documents where no term overlap exists. We also take advantage of such latent high-dimensional representations in our models for representing mobile apps.

Crowdsourcing. Although there has been a large body of research in IR related to crowdsourcing, we can only mention the most relevant works. Alonso and Stone [2] explored building a query log that is coupled with query annotations describing the search tasks. Arguello et al. [5] used crowdsourcing to collect spoken queries for predefined search tasks. In our work, we combine these two approaches (see Section 3).

3 CROWDSOURCING

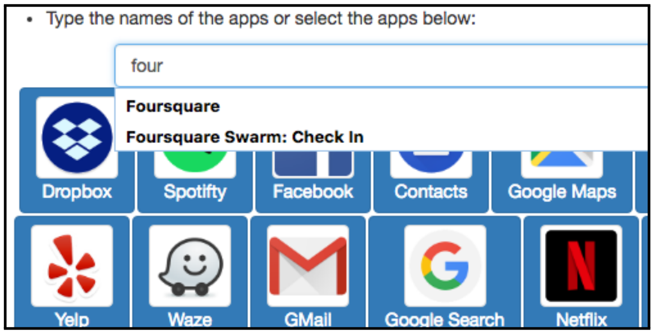
In this section, we describe how we collected *UniMobile*, which is, to the best of our knowledge, the first dataset on cross-app mobile search queries. We started by creating a number of Human Intelligence Tasks (HITs) on Amazon Mechanical Turk[7], asking workers to describe their latest mobile search experience in detail. The answers helped us to define fine-grained diverse naturalistic mobile search tasks. Then, we launched another task asking workers to assume they wanted to complete a given search task on their smartphones. They had to submit their search queries as well as the apps they would choose to complete each task.

Task definition. In the first crowdsourcing task, we described the category of search, giving them a handful of general examples. Furthermore, we also asked them to give us the context and background of their search, as well as the queries and the apps they used to do the search. Finally, we provided a complete example of a valid answer. We launched this job for most of search categories listed in Table 1. The HIT payment was $0.10 and the workers were based in the U.S. with an overall acceptance rate of 75% or higher. The average work time was 246 seconds with 135 workers completing 169 HITs resulting in an average of 92 terms per HIT. The workers provided enough details about the context and background of their search that enabled us to generalize the task to the level that we would get a wide range of queries on the same task. For example, one worker submitted the following answer:

> "I was searching for a new refrigerator to buy. The first thing I did was search for the best refrigerators of 2017 and then narrow down my search for exactly the type of refrigerator that I was looking for..."

Then, we used this answer to define a more general search task:

> "Consider one of the oldest appliances in your home. You have been thinking of changing it for a while. Now, it's time to order it online."



- Type the names of the apps or select the apps below:

four

Foursquare

Foursquare Swarm: Check In

Figure 1: HIT interface for choosing apps. The workers could enter an app's name or click on an app's icon.

Query and app pairs. The second crowdsourcing task consisted of 206 individual search task descriptions, mostly extracted from the answers we got in the first task. Table 1 lists the distribution of the tasks. In the definition of tasks, our aim was to cover various aspects of mobile information seeking as mentioned in [10]. We asked the workers to read the search task description very carefully and assume that they wanted to perform it using their own mobile device. Then, we asked them to select one or more apps from a given list. Alternatively, they could type the name of the app they would choose for that search task. We provided an auto-complete feature for entering the apps' names in order to make it easier for the users to type the name of their favorite apps. Figure 1 shows the interface we designed for this HIT. Since we restricted the HIT to be done only by workers in the U.S., we chose the list of apps from the most popular Android apps in the U.S. market. Note that the apps were randomly shuffled and displayed to each worker to prevent any position bias. These apps are listed as follows: Google Search, Gmail, Play Store, Facebook, Instagram, Google Maps, YouTube, Amazon, Twitter, Spotify, Waze, Pinterest, WhatsApp, File Manager, Netflix, Yelp, Contacts, Dropbox.

As incentive, we paid $0.05 for every HIT assignment. We also encouraged the workers to complete a survey for a $0.05 bonus. Our aim was to understand the workers' background and familiarity with mobile devices. We asked the workers to perform the task using their mobile devices' browsers and tracked their keyboard

[7] http://www.mturk.com

Table 2: Statistics of UniMobile.

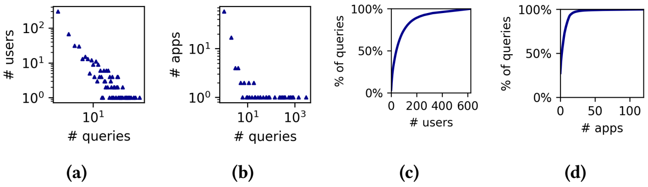
# queries	5,812
# unique queries	5,567
# users	625
# search tasks	206
# unique apps	121
# unique first apps	70
# unique second apps	89
Mean unique apps per task	7.51 ± 10.57
Mean query per user	9.30 ± 20.30
Mean query per task	28.21 ± 12.72
Mean query terms	4.21 ± 2.45
Mean query characters	24.83 ± 12.88

keystrokes to prevent them from copying any text from the task description. The average work time for this task was 85 seconds with 91% of the workers completing the survey. The key statistics of the survey were that 59% of the workers used Android and 55% used a mobile device as the primary device to connect to the Internet. Moreover, 83% of the workers believed they use their mobile device more than two hours a day and 41%, more than four hours a day. After launching several batches, we went through all the submitted answers for quality control and we observed that following crowdsourcing task design guidelines of [22] helped us achieve a very high assignment approval rate (99%). We have made the collection publicly available for research purposes. The released data consists of the tasks that we defined through the first set of HITs as well as user queries in the second set of HITs, together with their corresponding ranked list of apps. The data can be used to study how users are engaged in searching with different apps. Also, the release of the defined tasks provides the opportunity to conduct a similar study in a lab setting on participants' mobile phones and compare the findings with this work.

4 DATA ANALYSIS

In this section, we present a thorough analysis of UniMobile, to understand how users issue queries in different apps, and which apps they choose to complete search tasks. With the definition of 206 mobile search tasks, we were able to collect 5,812 search queries and their target apps. Overall, queries were assigned to 121 unique apps. Table 2 lists all the details of our dataset. In the following, we analyze different aspects of the data.

How apps are distributed. Figure 2 shows the distribution of queries with respect to users and apps in UniMobile. As we can see in Figures 2a and 2c, while there exist 173 users who submitted only one query, 110 users account for 80% of the queries and 239 users account for 95% of the queries. Also, we see in Figures 2b and 2d that the distribution of apps follows a power-law. In particular, 9 apps account for more than 80% and 17 apps account for more than 95% of the queries. Figure 3 shows how queries are distributed with respect to the top 17 apps. As we can see, while Google Search[8], that is mainly targeted for Web search, constitute 39% of total app selections, users opt to perform the majority (61%) of their search tasks using other apps. Moreover, the variety of apps ranges from apps dealing with local phone data (e.g., Contacts and Calendar) to social media apps (e.g., Facebook and Twitter) indicating that they cover a wide range of search tasks.



(a) (b) (c) (d)

Figure 2: The distribution of number of queries with respect to apps and users.

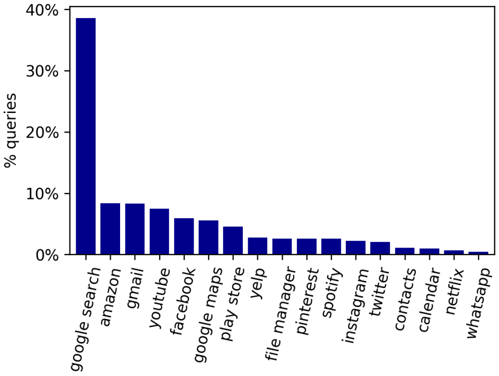
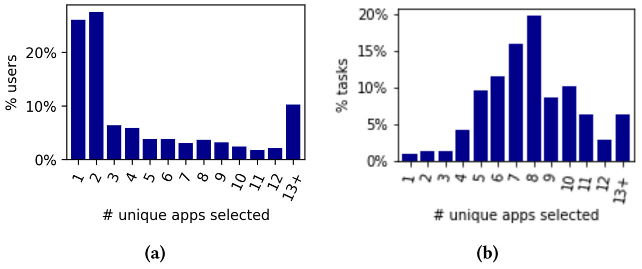


Figure 3: Number of queries per app for top 17 apps.

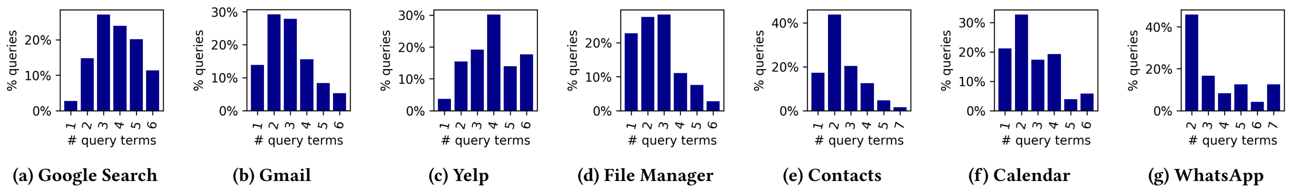


(a) (b)

Figure 4: Distribution of unique apps per user and task.

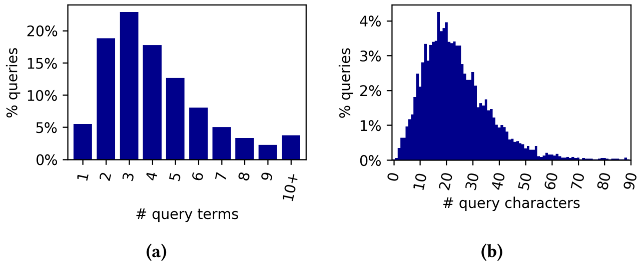
How apps are selected. Here we are interested in finding out how users behave while choosing an app to perform a search task. Although users assign two apps while submitting 72% of the queries, they choose only one app for 21% of the queries and choose more than two apps for only 7% of the queries. We also analyze how many different apps users select while doing the tasks. Figure 4a shows the distribution of unique apps per users illustrating how many users selected a certain number of different apps. As we can see, a quarter of users preferred to search using two unique apps. On the other hand, Figure 4b plots the same distribution with respect to the tasks, that is how many unique apps were selected for each task. We see an entirely different distribution where the average number of unique apps per task is 7.51, showing that the given

[8] The term "Google Search" is also used to refer to the Google Chrome app.



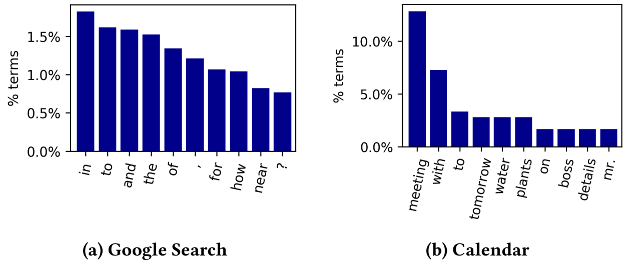
(a) Google Search (b) Gmail (c) Yelp (d) File Manager (e) Contacts (f) Calendar (g) WhatsApp

Figure 5: Histogram of number of query terms per app. Despite the small size, we can see the radically different distributions.



(a) (b)

Figure 6: Query length distribution with respect to number of terms and characters.



(a) Google Search (b) Calendar

Figure 7: Distribution of top query unigrams for two sample apps.

search tasks can be addressed using multiple apps. As we compare the two distributions in Figures 4a and 4b, we can conclude that while the majority of search tasks can be addressed using multiple apps, users usually limit their choice to a personal selection of apps. Therefore, a system can define a set of candidate apps which then can be narrowed down considering user’s personal preference.

Furthermore, we analyze users’ choice of Google Search, observing that it is selected as the first app in 39% of the queries while 46% as the second app. The users chose Google Search as the third app in 30% of the queries with three selected apps. This indicates that, according to UniMobile, in most cases (61%), users prefer to open a more specific app than Web search apps such as Google Search. We also analyze users collective app selection behavior with respect to the tasks. For each task, we count how often each app is selected and sorted them. Our aim is to find out how often users decide to perform their search tasks using Google Search. According to our study, in 14% of the tasks, no user selected Google Search, while in 35% of the tasks Google Search was the most selected app. Moreover, in 68% of the tasks it was among the top two most frequently selected apps, and in 78% of the tasks it was among the top three. Considering the categorical distribution of apps in Table 1 where only 13% of the tasks were in the category of General Information & News, we see that Google Search attracts many queries from the tasks that can be done using a more specific app. Given the integrity and aggregation of various search services such as image, video, location, and online shopping and easy access to them in one app, this observation is not surprising. Nevertheless, we see that for 86% of the tasks, most users prefered other apps. This suggests that a unified mobile search system has a high potential of simplifying and improving users search experience.

How queries differ among apps. We analyze different attributes of queries with respect to their corresponding apps, to understand how different users formulate their information needs into queries using different apps. After tokenizing the queries, the average query terms per query is 4.21. We analyze the distribution of number of query terms per app, observing different distributions for every app, some of which are shown in Figure 5. This difference is more obvious if we compare Google Search with personal or local apps such as Contacts. In particular, Google Search has an average of 4.82 query terms while Contacts has an average of 2.67, which is considerably less than other apps. This can be explained if we consider the type of information users usually look for using Contacts app. The queries usually consist of one of the stored names on the phone, followed by terms such as “email,” “address,” “info,” and “contact.” Moreover, Figure 6 plots the distribution of query length with respect to terms and characters on the whole dataset.

Figure 5 demonstrates the distribution of number of query terms for 7 apps. In this figure, we only include the apps that exhibit a considerably different distribution from the average. As shown, Google Search query terms peak at 3 while personal apps such as Contacts, Calendar, and Gmail peak at 2. This indicates that the structure of queries vary depending on the target app. We can also see the difference in the most frequent unigrams for two example apps in Figure 7 where we see that while stopwords are the most frequent unigrams used in queries submitted to Google Search, for a specific personal app such as Calendar, a domain-specific term such as “meeting” accounts for more than 15% of the total distribution. This suggests that while considering domain-specific terms is crucial to predicting the target app, taking into account the query structure is also important. For instance, as we see in Figure 7a, the question mark is among the top query unigrams submitted to Google Search, suggesting that many of the queries are submitted in the form of a question. In contrast, as we mentioned earlier, the structure of contact queries are mostly in the form of “<proper noun> + <information field>,” as in “sam email.”

Query overlap. Here we study query overlap or query similarity over the queries using a simple function used in previous studies done on large-scale query logs (e.g., [12]). We measure the query overlap at various degrees and use the similarity function $\text{sim}(q_1, q_2) = |q_1 \cap q_2|/|q_1 \cup q_2|$. This function simply measures the overlap of query terms. We observed 70% of queries overlapping with at least another query at the similarity threshold of > 0.25. Higher thresholds lead to significantly lower similar queries; with thresholds > 0.50 and > 0.75 we observe that 24% and 9% of queries were similar, respectively. Similar to previous analyses, in Table 3 we observe a different level of query overlap in queries associated with different apps. The least query overlap is observed for Facebook queries. This could be due to the personal environment of Facebook. The highest query overlap is observed in Play Store queries. We observed the presence of some domain-specific terms such as "app" in many queries which results in higher query similarity. The observed difference in query overlap for every app suggests that various factors influence the way users formulate their queries. For example, apps that provide more focused information, receive more similar queries. On the other hand, more personal apps receive a diverse set of queries as they reflect personal information needs which can be totally different from one user to the other.

Summary. Our analyses first showed that users' queries are mainly targeted to a few apps; however, these apps are very different in terms of their content. Moreover, we showed that users often choose two different apps for a single query, suggesting that many users submit the same query in multiple apps. Also, we showed that different users select an average of more than 7 apps for each task, with Google Search being the top selected app in only 35% of the cases. This again indicates the necessity of a unified search system on mobile devices. Finally, we analyzed the queries issued in different apps and found notable differences. For instance, we showed that query lengths, unigram distribution, and query overlap differ among apps. This suggests that the query structure needs to be taken into account while representing the apps.

Table 3: The percentage of similar queries at different similarity thresholds considering only the queries associated with every app.

App	% of similar queries		
	> 0.25	> 0.50	> 0.75
All apps	70%	24%	9%
Google Search	63%	19%	6%
Amazon	38%	8%	3%
Gmail	57%	14%	7%
YouTube	49%	20%	7%
Google Maps	46%	3%	1%
Facebook	30%	9%	1%
Play Store	61%	26%	14%

5 NEURAL TARGET APPS SELECTION

Assume that a user aims to submit a query q to a set of mobile apps $\{a_1, a_2, \cdots, a_n\}$, called the target apps. Note that the size of this set could be equal to 1. The task of *target apps selection* is defined as ranking the mobile apps in response to the query q, such that the target apps appear in higher ranks. In this section, we propose our methodology to tackle the target apps selection task. To this end, we propose two *general* frameworks based on neural networks. Our first framework, called NTAS1, is given a query and a candidate app and produces a retrieval score. We study both pointwise and pairwise training settings for this framework. Our second framework, called NTAS2, is given a query as the input and produces a probability distribution indicating the probability of each app being targeted, for all apps.

One of the main challenges in this task is that it is not obvious how to represent each app. For example, although the apps' descriptions would be used for app representation in the app selection task [27], it cannot be used in the target apps selection. Because the queries that can be searched in a specific app do not match with the content of the app's description. To address this issue, our frameworks learn a high-dimensional representation for each app, as part of the network. The following subsections describe these two frameworks in more detail.

5.1 NTAS1: App Scoring Model

NTAS1 outputs a retrieval score for a given query q and a candidate app a. Formally, NTAS1 can be defined as follows:

$$\text{score} = \psi(\phi_Q(q), \phi_A(a)) ,$$

where $\psi(\cdot, \cdot) \in \mathbb{R}$ is a scoring function for the given query representation $\phi_Q(q) \in \mathbb{R}^m$ and app representation $\phi_A(a) \in \mathbb{R}^n$. Various neural architectures can be employed to model each of the three components in the NTAS1 framework.

We implement the component $\phi_Q(q)$ with two major functions: an embedding function $\mathcal{E} : V \rightarrow \mathbb{R}^d$ that maps each vocabulary term to a d-dimensional embedding space, and a global term weighting function $\mathcal{W} : V \rightarrow \mathbb{R}$ that maps each vocabulary term to a real-valued number showing its global importance. The query representation function ϕ_Q represents a query $q = \{w_1, w_2, \cdots, w_{|q|}\}$ as follows:

$$\phi_Q(q) = \sum_{i=1}^{|q|} \widehat{\mathcal{W}}(w_i) \cdot \mathcal{E}(w_i) , \tag{1}$$

which is the weighted element-wise summation over the terms' embedding vectors (hence, $m = d$). $\widehat{\mathcal{W}}$ is the normalized global weights computed using a softmax function as follows:

$$\widehat{\mathcal{W}}(w_i) = \frac{\exp(\mathcal{W}(w_i))}{\sum_{j=1}^{|q|} \exp(\mathcal{W}(w_j))} .$$

This is a simple yet effective approach for query representation based on the bag of words assumption, which has been proven to be effective for the ad-hoc retrieval task [15]. Note that the matrices $\mathcal{E}$ and $\mathcal{W}$ are the network parameters in our model and are learned to provide task-specific representations.

The app representation component ϕ_A is simply implemented as a look-up table. In other words, our neural model consists of an app representation matrix $\mathcal{A} \in \mathbb{R}^{N \times n}$ where N denotes the total number of apps and the i^{th} row of this matrix is a n-dimensional representation for the i^{th} app. Therefore, $\phi_A(a)$ returns a row of the matrix $\mathcal{A}$ that corresponds to the app a.

To model the function ψ, following Zamani et al. [40], we feed the Hadamard product (which enforces $m = n$) of the learned query and app representations into a fully-connected feed-forward network with two hidden layers. This network produces a single output as the score assigned to the given query-app pair. We use rectified linear unit (ReLU) as the activation function in the hidden layers of the network. To prevent overfitting, the dropout technique [34] is employed.

We study both pointwise and pairwise learning settings for our NTAS1 model.

Pointwise learning. In a pointwise setting, we use mean squared error (MSE) as the loss function. MSE for a mini-batch b is defined as follows:

$$\mathcal{L}_{MSE}(b) = \frac{1}{|b|} \sum_{i=1}^{|b|} (y_i - \psi(\phi_Q(q_i), \phi_A(a_i)))^2 ,$$

where q_i, a_i, and y_i denote the query, the candidate app, and the label in the i^{th} training instance of the mini-batch. For this training setting, we use a linear activation for the output layer.

Pairwise learning. NTAS1 can be also trained using a pairwise setting. Therefore, each training instance consists of a query, a target app, and a non-target app. To this end, we employ hinge loss (max-margin loss function) that has been widely used in the learning to rank literature for pairwise models [23]. Hinge loss for a mini-batch b is defined as follows:

$$\mathcal{L}_{Hinge}(b) = \frac{1}{|b|} \sum_{i=1}^{|b|} \max \{0, \epsilon - \text{ sign}(y_{i1} - y_{i2}) \\ (\psi(\phi_Q(q_i), \phi_A(a_{i1})) - \psi(\phi_Q(q_i), \phi_A(a_{i2})))\} ,$$

where ϵ is a hyper-parameter determining the margin of hinge loss, a linear loss function that penalizes examples violating the margin constraint. To bound the output of the model to the $[-1, 1]$ interval, we use tanh as the activation function for the output layer, in the pairwise training setting. The parameter ϵ is also set to 1, which works well when the predicted scores are in the $[-1, 1]$ interval.

5.2 NTAS2: Query Classification Model

Unlike NTAS1 that predicts a score for a given query-app pair, our second framework computes the probability of each app being targeted by a given query. In more detail, NTAS2 is modeled following score = $\gamma(\phi_Q(q))$, where the i^{th} element of γ denotes the probability of the i^{th} app being targeted, given the query representation $\phi_Q(q)$. n is the total number of apps.

To implement NTAS2, we represent each query via a weighted element-wise average as explained in Equation (1). γ is modeled using a fully-connected feed-forward network with the output dimension of n. ReLU is employed as the activation function in the hidden layers, and a softmax function is applied on the output layer to compute the probability of each app being targeted by the query.

To train NTAS2, we use a cross-entropy loss function which for a mini-batch b is defined as:

$$\mathcal{L}_{ce}(b) = \frac{1}{|b|} \sum_{i=1}^{|b|} \sum_{j=1}^{n} (p(a_j|q_i) \log \gamma(\phi_Q(q_i))) .$$

Similar to NTAS1, we use dropout to regularize the model.

6 EXPERIMENTS

In this section, we evaluate the performance of the proposed models in comparison with a set of state-of-the-art IR models. We also study the performance of the models with respect to tasks and users.

6.1 Experimental Setup

Dataset. We evaluated the performance of our proposed models on the UniMobile dataset. We followed two different strategies to split the data: (1) In *UniMobile-Q*, we randomly selected 70% of the *queries* for training, 10% for validation, and 20% for test set (2) In *UniMobile-T*, we randomly split the tasks (rather than queries). To do so, we randomly selected 70% of the *tasks* for training, 10% for validation, and 20% for test set. To minimize random bias, for each splitting strategy we repeated the process five times. The hyper-parameters of the models were tuned based on the results on the validation sets. Therefore, we repeated all the experiments five times and reported the average performance.

Evaluation metrics. Effectiveness was measured by five standard evaluation metrics: mean reciprocal rank (MRR), precision of the top 1 retrieved app (P@1), normal discounted cumulative gain for the top 1, 3, and 5 retrieved apps (nDCG@1, nDCG@3, nDCG@5). We determined the statistically significant differences using the two-tailed paired t-test with Bonferroni correction at a 95% confidence interval ($p < 0.05$). In the ranked list of apps associated to every query, we assigned the score of 2 to the *first* relevant app and 1 to the rest of relevant apps, to differentiate between a model that is able to rank the first relevant app higher and a model that is not.

The choice of evaluation metrics was motivated by considering three different aspects of the task, inspired by data analysis. We chose MRR considering scenarios where a user is looking for relevant information only in one app, and so they would stop scanning the search results as soon as they find the first relevant document. We reported P@1 and nDCG@1 to measure the performance for scenarios that a user only checks the first result. Given that many search tasks need to be addressed using more than one app, it is crucial to evaluate a system with respect to more than one relevant app in the top-k results. nDCG@3 allowed us to evaluate our approach when a user scans the top 3 results. Since we found that most of the queries were assigned to one or two apps (see Section 4), nDCG@3 measures how well a system is able to place the two relevant apps among the top 3 results. We also used nDCG@5 to evaluate top 5 results on a single screen, given the size of a typical smartphone.

Compared methods. We compared the performance of our model with the following methods:

- *StaticRanker:* For every query we ranked the apps in the order of their popularity in the training set as a static (query independent) model.
- *QueryLM, BM25, BM25-QE:* For every app we aggregated all the relevant queries from the training set to build a document representing the app. Then we used Terrier [25] to index the documents. QueryLM uses the language model retrieval model [29].

Table 4: Performance comparison with baselines on UniMobile-Q and UniMobile-T. The superscript * denotes significant differences compared to all the baselines.

Method	UniMobile-Q Dataset					UniMobile-T Dataset				
	MRR	P@1	nDCG@1	nDCG@3	nDCG@5	MRR	P@1	nDCG@1	nDCG@3	nDCG@5
StaticRanker	0.6485	0.5293	0.4031	0.4501	0.5144	0.6718	0.5507	0.4247	0.4853	0.5446
QueryLM	0.5867	0.3803	0.3068	0.4676	0.5508	0.5178	0.3272	0.2619	0.3716	0.4503
BM25	0.7523	0.6233	0.4915	0.6298	0.6859	0.6780	0.5244	0.4101	0.5392	0.5992
BM25-QE	0.6948	0.5177	0.4116	0.5909	0.6498	0.6256	0.4276	0.3312	0.5015	0.5704
k-NN	0.7373	0.6031	0.4794	0.6091	0.6633	0.6879	0.5414	0.4287	0.5413	0.6003
k-NN-AWE	0.7420	0.6081	0.4842	0.6156	0.6682	0.6984	0.5551	0.4407	0.5560	0.6117
LambdaMART	0.7313	0.6127	0.4864	0.6110	0.6426	0.6749	0.5469	0.4323	0.5419	0.5704
NTAS1-pointwise	0.7591*	0.6214	0.4897	0.6328	0.6934*	0.7047*	0.5582*	0.4493*	0.5506*	0.6258*
NTAS1-pairwise	**0.7661***	**0.6285***	**0.5012***	**0.6364***	**0.7018***	**0.7192***	0.5661*	**0.4709***	**0.5941***	**0.6471**
NTAS2	0.7638*	0.6271*	0.4996*	0.6351*	0.6976*	0.7144*	**0.5723***	0.4608*	0.5689*	0.6334*

For BM25-QE, we adopted Bo1 [3] model for query expansion. We used the Terrier implementation of these methods.

- *k-NN, k-NN-AWE:* To find the nearest neighbors in k nearest neighbors (k-NN), we considered the cosine similarity between TF-IDF vectors of queries. Then, we took the labels (apps) of the nearest queries and produced the app ranking. As for k-NN-AWE, we computed the cosine similarity between the average word embedding (AWE) of the queries obtained from GloVe [28] with 300 dimensions.
- *LambdaMART:* For every query-app pair, we used the scores obtained by BM25, k-NN, and k-NN-AWE as features to train LambdaMART [36] implemented in RankLib[9]. For every query, we considered all irrelevant apps as negative samples.

6.2 Results and Discussion

In the following, we evaluate the performance of NTAS1 and NTAS2 trained on both data splits. We further analyze how other baseline models perform comparing their performance on both splits together with other methods.

Performance comparison. Table 4 lists the performance of our proposed methods as well as the compared methods. As we can see, the performance of all methods drops when we use UniMobile-T data splits, except for StaticRanker. StaticRanker gives us an idea of how much the test set is biased towards more popular apps. For example, we see that StaticRanker performs better on UniMobile-T suggesting that it consists of more popular apps. As we compare the relative performance drop between the two data splits, we see that among other baselines, k-NN-AWE is more robust with the minimum relative drop (−8.4% on average). QueryLM, on the other hand, is the least robust model with the maximum relative drop (−16% on average). This indicates that k-NN-AWE is able to capture similar queries for unseen tasks using a pre-trained word embedding, whereas QueryLM relies heavily on the indexed queries.

Among the baselines tested on UniMobile-Q, we see that BM25 performs best in terms of all evaluation metrics. Given that UniMobile-Q contains queries belonging to the same tasks both in training and test sets, this shows that when more similar queries exist in the index, BM25 is able to rank the apps more effectively. However, on UniMobile-T, k-NN-AWE performs best in terms of all metrics. Given that UniMobile-T *does not* contain queries belonging to the same task in training and test sets, this suggests that leveraging a pre-trained word embedding helps k-NN capture query similarities more effectively when the queries are less similar, leading to a better generalization. This can also be seen when comparing the performance of k-NN and k-NN-AWE, given that k-NN-AWE consistently outperforms k-NN. Regarding LambdaMART, we see that even though it benefits from multiple features, it does not perform as well as k-NN-AWE and BM25 on UniMobile-Q. On the contrary, we see that it performs better on UniMobile-T showing that the AWE-based feature improves its generalization.

As we can see, NTAS1-pairwise and NTAS2 outperform all the methods, on both data splits, in terms of all evaluation metrics. All the improvements are statistically significant suggesting that using queries to learn the app representation helps our approach learn the similarities more effectively. Considering the relative difference on the two data splits, we observe that our proposed approaches also show a drop. Compared to other methods (except for StaticRanker), we observe that NTAS1-pairwise and NTAS2 consistently have a lower relative drop across UniMobile-Q and UniMobile-T, indicating that the trained app embedding is an effective way to represent mobile apps based on the queries that are assigned to them. Among our proposed methods, NTAS1-pairwise has the least relative drop (−7.4% on average), suggesting that a pairwise setting leads to a higher generalization.

Representation analysis. We reduce the dimensionality of the learned app representations by projecting them to a two-dimensional space using t-Distributed Stochastic Neighbor Embedding (t-SNE) [24]. Figure 8 shows the proximity of the representation of different apps[10] being grouped in some clusters. For instance, all social media apps are placed close to each other. Also, we see that location search and navigation apps are in another cluster. Interestingly, Gmail is close to File Manager, Contacts, and WhatsApp. People usually search for attachments or their contacts using Gmail, explaining their proximity. Google Search, on the other hand, belongs

[9] https://sourceforge.net/p/lemur/wiki/RankLib/

[10] Given space limitations, we could not include all the apps in this figure.

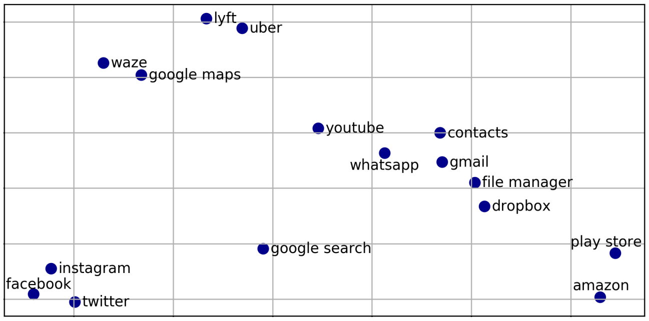
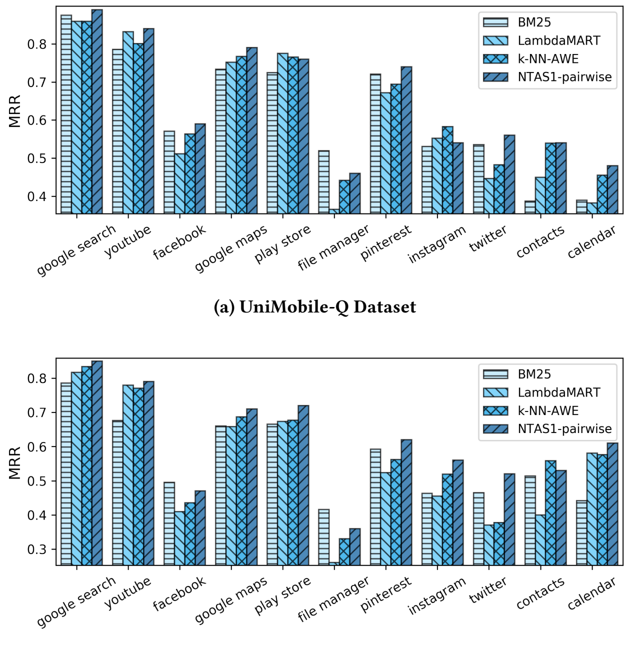


Figure 8: Proximity of different app representations learned by NTAS1 (pairwise). This plot is produced by reducing the dimensionality (using the t-SNE algorithm) of the app representations to two for visualization.

to no cluster. This could be due to the variety of queries people submit to Google Search, placing it somewhere in the center of all other apps. However, we cannot explain why YouTube is close to WhatsApp, or why Play Store is close to Amazon. Hence, Figure 8 shows that learning high-dimensional app representation using the queries submitted to them is effective, though perhaps not perfect.

Performance on apps. Here, we compute the mean performance of the queries targeted to a specific app and plot the result for each app in Figure 9. For the sake of visualization, we only compare the performance of NTAS1-pairwise with three other methods in terms of MRR. We see that all models perform well in ranking less personal apps such as Google Search, YouTube, and Google Maps. Since none of the models incorporate users’ personal data, this result is expected. This suggests users’ personal data can be leveraged to rank apps such as File Manager, Contacts, and Calendar higher. Also, users’ activities on their social media apps should be leveraged to provide a more effective personalized ranking. Moreover, we see that k-NN-AWE is more robust across the two data splits, compared to other baselines. In particular, it performs well in ranking Contacts suggesting that the proximity of contact names in the high-dimensional space of word embedding enables k-NN-AWE to outperform other models. Finally, it can be seen that NTAS1-pairwise is more robust across the two data splits, compared to other methods. Specifically, it outperforms all other methods for the majority of apps. However, NTAS1-pairwise performs worse than other methods for File Manager, on both data splits. This is mainly due to insufficient number of training data, given the diversity of the queries related to this app.

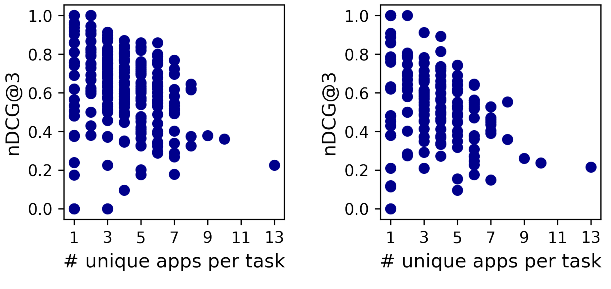
Performance on tasks. We are interested in seeing how methods perform differently with respect to different search tasks. To do so, we averaged the performance (nDCG@3) of all queries belonging to the same task. Then, we grouped the tasks by the total number of unique apps selected by users and plotted their results in Figure 10. Our intuition was that if different users chose several apps for a single task, it can be a sign that the task is more challenging for the models. We can see in the figure that as the number of unique apps per task raises, the models perform worse. Although, the negative correlation is not very strong (Pearson: −0.3049 and −0.3450 for UniMobile-Q and UniMobile-T, respectively), it is consistent with all models and evaluation metrics. This indicates that if a task can be done using multiple apps, their corresponding queries also become more difficult for a system. A multi-app task can be either very personal (i.e., every user chooses their own favorite app) or very general (i.e., it can be done using many apps). Therefore, one can explore incorporating users’ regular app usage patterns to perform a personalized target app selection.



(a) UniMobile-Q Dataset

(b) UniMobile-T Dataset

Figure 9: Performance comparison with respect to certain apps on both data splits.



(a) UniMobile-Q Dataset

(b) UniMobile-T Dataset

Figure 10: Negative correlation between the number of unique apps users selected for a task and performance.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced and studied the task of target apps selection, which was motivated by the growing interest in conversational search systems where users speak their queries to a unified voice-based search system. To this aim, we presented the first analysis of mobile cross-app search queries and user behaviors in terms of the apps they chose to complete different search tasks. We found that a limited number of popular apps attract most of the search queries. We further observed notable differences between queries submitted to different apps. We showed that query length and terms differ among apps. We also showed that, 39% of search queries were done in Google Search, and it was the top choice of users in 35% of the tasks. Given that more than 71% of the defined tasks could be done with the current features of Google Search, this indicates that users prefer to search using a more specific app. We carried out the experiments and analyses on the dataset of cross-app mobile queries that we collected through crowdsourcing.

Since the mobile information environment is uncooperative and the data is heterogeneous, representing each app for the target apps selection task is challenging. We proposed two models that learn high-dimensional latent representations for the mobile apps in an end-to-end training setting. Our first model produces a score for a given query-app pair, while the second model produces a probability distribution over all the apps given a query. We compared the performance of our proposed method with state-of-the-art retrieval baselines splitting data following two different strategies. Our approach outperformed all baselines significantly.

There are several directions for future work. We plan to conduct a follow-up study asking volunteers to install an app which will track their movement and sense their context. We will ask the volunteers to report their daily mobile search experiences using our app. This will enable us to study user behaviors while searching with different apps in the wild. Since we will not ask users to complete predefined search tasks, we expect to see different distribution of search tasks and selected apps. Moreover, a real unified mobile search system would have access not only to the users' personal selection of apps, but also to their daily app usage patterns. Incorporating such information into the ranking model is an interesting future direction. More importantly, mobile devices can be used to sense users' context. Another future direction is to study how the sensed contextual information can be leveraged to enhance a ranking model. Also, search results aggregation and presentation should be explored in the future, considering two important factors: high information gain and user satisfaction.

ACKNOWLEDGMENTS

This work was supported in part by the RelMobIR project of the Swiss National Science Foundation (SNSF), in part by the Center for Intelligent Information Retrieval, and in part by NSF grant #IIS-1160894. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Mohammad Aliannejadi and Fabio Crestani. 2017. Venue Appropriateness Prediction for Personalized Context-Aware Venue Suggestion. In *SIGIR.* 1177–1180.

[2] Omar Alonso and Maria Stone. 2014. Building a query log via crowdsourcing. In *SIGIR.* 939–942.

[3] Giambattista Amati. 2003. *Probability models for information retrieval based on divergence from randomness.* Ph.D. Dissertation. University of Glasgow, UK.

[4] Jaime Arguello. 2017. Aggregated Search. *Foundations and Trends in Information Retrieval* 10, 5 (2017), 365–502.

[5] Jaime Arguello, Sandeep Avula, and Fernando Diaz. 2016. Using Query Performance Predictors to Improve Spoken Queries. In *ECIR.* 309–321.

[6] Jaime Arguello, Jamie Callan, and Fernando Diaz. 2009. Classification-based resource selection. In *CIKM.* 1277–1286.

[7] Jan R. Benetka, Krisztian Balog, and Kjetil Nørvåg. 2017. Anticipating Information Needs Based on Check-in Activity. In *WSDM.* 41–50.

[8] James P. Callan and Margaret E. Connell. 2001. Query-based sampling of text databases. *ACM Trans. Inf. Syst.* 19, 2 (2001), 97–130.

[9] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. 2009. Context-aware query classification. In *SIGIR.* 3–10.

[10] Karen Church and Barry Smyth. 2009. Understanding the intent behind mobile information needs. In *IUI.* 247–256.

[11] Karen Church, Barry Smyth, Keith Bradley, and Paul Cotter. 2008. A large scale study of European mobile search behaviour. In *Mobile HCI.* 13–22.

[12] Karen Church, Barry Smyth, Paul Cotter, and Keith Bradley. 2007. Mobile information access: A study of emerging search behavior on the mobile Internet. *TWEB* 1, 1 (2007), 4.

[13] Fabio Crestani and Heather Du. 2006. Written versus spoken queries: A qualitative and quantitative comparative analysis. *JASIST* 57, 7 (2006), 881–890.

[14] Fabio Crestani, Stefano Mizzaro, and Ivan Scagnetto. 2017. *Mobile Information Retrieval.* Springer.

[15] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR.* 65–74.

[16] Fernando Diaz. 2009. Integration of news content into web results. In *WSDM.* 182–191.

[17] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM.* 55–64.

[18] Ido Guy. 2016. Searching by Talking: Analysis of Voice Queries on Mobile Web Search. In *SIGIR.* 35–44.

[19] Morgan Harvey and Matthew Pointon. 2017. Searching on the Go: The Effects of Fragmented Attention on Mobile Web Search Tasks. In *SIGIR.* 155–164.

[20] Maryam Kamvar and Shumeet Baluja. 2006. A large scale study of wireless search behavior: Google mobile search. In *CHI.* 701–709.

[21] In-Ho Kang and Gil-Chang Kim. 2003. Query type classification for web document retrieval. In *SIGIR.* 64–71.

[22] Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing user studies with Mechanical Turk. In *CHI.* 453–456.

[23] Hang Li. 2011. *Learning to Rank for Information Retrieval and Natural Language Processing.* Morgan & Claypool Publishers.

[24] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[25] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Douglas Johnson. 2005. Terrier Information Retrieval Platform. In *ECIR.* 517–519.

[26] Dae Hoon Park, Yi Fang, Mengwen Liu, and ChengXiang Zhai. 2016. Mobile App Retrieval for Social Media Users via Inference of Implicit Intent in Social Media Text. In *CIKM.* 959–968.

[27] Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. 2015. Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval. In *SIGIR.* 533–542.

[28] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP.* 1532–1543.

[29] Jay M. Ponte and W. Bruce Croft. 1998. A Language Modeling Approach to Information Retrieval. In *SIGIR.* 275–281.

[30] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. 2006. Building bridges for web query classification. In *SIGIR.* 131–138.

[31] Milad Shokouhi and Qi Guo. 2015. From Queries to Cards: Re-ranking Proactive Card Recommendations Based on Reactive Search History. In *SIGIR.* 695–704.

[32] Milad Shokouhi, Rosie Jones, Umut Ozertem, Karthik Raghunathan, and Fernando Diaz. 2014. Mobile query reformulations. In *SIGIR.* 1011–1014.

[33] Yang Song, Hao Ma, Hongning Wang, and Kuansan Wang. 2013. Exploring and exploiting user search behavior on mobile and tablet devices to improve search relevance. In *WWW.* 1201–1212.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958.

[35] Kyle Williams, Julia Kiseleva, Aidan C. Crook, Imed Zitouni, Ahmed Hassan Awadallah, and Madian Khabsa. 2016. Detecting Good Abandonment in Mobile Search. In *WWW.* 495–505.

[36] Qiang Wu, Christopher J. C. Burges, Krysta Marie Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Inf. Retr.* 13, 3 (2010), 254–270.

[37] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep Learning for Answer Sentence Selection. In *NIPS Deep Learning Workshop.*

[38] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *WWW.* 1531–1540.

[39] Hamed Zamani and W. Bruce Croft. 2016. Estimating Embedding Vectors for Queries. In *ICTIR.* 123–132.

[40] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. 2018. Neural Ranking Models with Multiple Document Fields. In *WSDM.* 700–708.