Leveraging User Reviews to Improve Accuracy for Mobile App Retrieval

Dae Hoon Park
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
dpark34@illinois.edu

Mengwen Liu
College of Computing and Informatics
Drexel University
Philadelphia, PA 19104, USA
ml943@drexel.edu

ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
czhai@cs.illinois.edu

Haohong Wang
TCL Research America
2870 Zanker Road
San Jose, CA 95134, USA
haohong.wang@tcl.com

ABSTRACT

Smartphones and tablets with their apps pervaded our everyday life, leading to a new demand for search tools to help users find the right apps to satisfy their immediate needs. While there are a few commercial mobile app search engines available, the new task of mobile app retrieval has not yet been rigorously studied. Indeed, there does not yet exist a test collection for quantitatively evaluating this new retrieval task. In this paper, we first study the effectiveness of the state-of-the-art retrieval models for the app retrieval task using a new app retrieval test data we created. We then propose and study a novel approach that generates a new representation for each app. Our key idea is to leverage user reviews to find out important features of apps and bridge vocabulary gap between app developers and users. Specifically, we jointly model app descriptions and user reviews using topic model in order to generate app representations while excluding noise in reviews. Experiment results indicate that the proposed approach is effective and outperforms the state-of-the-art retrieval models for app retrieval.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*

General Terms

Algorithms, Design

1. INTRODUCTION

Nowadays, mobile apps occupy a large share of our everyday life. According to a recent analysis by Flurry, average American consumers spend about three hours (177 minutes) per day on mobile devices,[1] which is more than the average time spent on TVs (168 minutes). An analysis in 2013 shows that 80% of the time spent on mobile devices is inside apps and 20% is spent on the mobile web.[2] The time spent on the mobile web remained flat in 2014 while the time spent inside apps increased. While consumers spend much of their time inside apps, they constantly download new mobile apps.[3] This means that the role of app search and recommendation system remains important.

Meanwhile, the number of mobile apps in app store is explosively increasing so that search function in app store becomes essential. As of July 2014, there are about 1.3 million apps in Google Play app store and 1.2 million apps in Apple App Store.[4] The myriad apps made consumers extremely hard to find apps without search or recommendation functions. For example, Google Play does not list all of the apps. Instead, it lists only recommended or popular apps because finding an app through the long list does not make sense any more. Moreover, in an app developer's view, new or unpopular apps are barely discovered by consumers if they are not recommended by the app stores. Therefore, app search engine is definitely essential for both consumers and developers.

Thus, it is our goal to find apps based on a query given by a user. Specifically, given a user query that describes an aspect of an app, the desired search result would show a ranked list of apps where higher ranked apps are more likely to have the described aspect. For example, for a query "book a flight", we expect the search result to include apps such as "Expedia Hotels & Flights" and "Orbitz – Flights, Hotels, Cars" in high ranks since these apps meet the user's need quite well. However, if an app description is not written well, *e.g.*, too short or hardly useful, the retrieval system would not rank the app high even though the app is actually relevant to a query. In addition, app descriptions are written by app developers while search queries are made by

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

ACM 978-1-4503-3621-5/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767759 .

[1] http://www.flurry.com/blog/flurry-insights/mobile-television-we-interrupt-broadcast-again
[2] http://www.flurry.com/bid/95723/Flurry-Five-Year-Report-It-s-an-App-World-The-Web-Just-Lives-in-It
[3] http://www.flurry.com/blog/flurry-insights/app-install-addiction-shows-no-signs-stopping
[4] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

users, and this may result in vocabulary gap between them. Therefore, to improve accuracy for mobile app retrieval, we propose to leverage user reviews, which provide more information about an app in a user's vocabulary.

As an interesting new retrieval task, app retrieval has not yet been rigorously studied in the literature. Indeed, no test collection has ever been created yet for quantitatively evaluating this task. To address this problem, we conduct the first systematic study of effectiveness of both existing retrieval models and new retrieval models for this task, and we create the very first test collection for evaluating this new retrieval problem.

This paper makes the following contributions:

1. We introduce and study a novel research problem of mobile app retrieval leveraging user reviews. To the best of our knowledge, this is the first effort in this area, and no other research work studied the same problem as ours.

2. We propose a novel probabilistic topic model that jointly models user reviews and unstructured product information (product description) in order to obtain representation of apps. The model captures topics jointly from reviews and descriptions so that the topics reflect vocabulary of both users (user reviews) and developers (app descriptions). The model is unsupervised and general, so it can be applied to other domains where there are unstructured text about an entity and an associated set of unstructured text.

3. We create a new data set for evaluating the task and conduct experiments to show that the proposed method outperforms baseline approaches for mobile app retrieval. The mobile app data set is crawled from a major app store, Google Play. We let domain experts generate queries based on android forums. Then, we collect query relevance data via crowdsourcing service. The test collection is available at `http://timan.cs.uiuc.edu/downloads.html`. As far as we know, no previous research for mobile app retrieval has performed quantitative evaluations.

2. RELATED WORK

Recommendation systems are highly related to retrieval systems in that they rank objects to fulfill needs of users, and the recommendation systems are surveyed well in [10]. App recommendation systems have been studied by a few researchers [32, 30]. For example, Lin *et al.* [15] addressed cold start problem in app recommendation system by leveraging social network data. However, retrieval systems are different from recommendation systems mainly because a user explicitly expresses his or her needs in retrieval systems while the recommendation systems suggest items based on user profile without asking for the user's needs. Recommendation systems may be more convenient for users since a user does not have to input his or her needs, but they are likely to be less accurate than retrieval systems since they barely know the user's current needs. In addition, recommendation systems encounter a cold start problem when a user does not have his or her profile yet or when the system does not have enough transaction data yet. On the other hand, retrieval systems do not require such data, so there is no cold start problem for them.

There have been extensive studies for XML retrieval, which is related to our work since app data consist of elements such as app descriptions and user reviews. Some of the XML retrieval studies also support simple keyword queries [16] while some other XML retrieval studies support only structured queries [26]. However, our goal is to augment app descriptions with user reviews to obtain a better representation for an app, while XML retrieval focuses more on document structure in general. In addition, while retrieval unit in our task is clearly defined as a mobile app, in XML retrieval, every element is a retrievable unit [12], which makes XML retrieval different from our task.

Entity ranking and entity search is closely related to our problem [4]. While entity ranking usually focuses on exploiting rich structures [24, 19], Ganesan and Zhai [8] studied opinion-based entity ranking leveraging review text. In their work, a query is structured with preferences on different aspects of an entity. The known-type entities are then ranked based on aspect-based sentiment from user reviews, while we rank unknown-type apps based solely on query relevance. Product search is highly related to our work. Duan *et al.* [5] leveraged product specifications and user reviews to improve performance on keyword search in product database. However, while products usually have such structured specifications that characterize products, mobile apps usually have no structured specifications since there is no standardized features of apps, which makes the problem harder. Meanwhile, there has been some efforts to build commercial app search engines such as [3]. However, mobile app retrieval problem has not yet been studied rigorously with systematic experiments.

3. PROBLEM DEFINITION

In order to find an app, a user constructs a text query q, where we assume q represents the search intent of the user, and q is input to an app retrieval system. Then, the app retrieval system searches for apps that satisfy the user intent and shows the user a list of apps that are ranked according to their relevance to q, which conforms to the probability ranking principle [22]. Formally, we are given M apps $\boldsymbol{A} = \{a_1, ..., a_M\}$. For each app a_i, there is an unstructured app description d_i and user reviews that are concatenated to a single review document, r_i. Our goal is to retrieve a list of apps for each q based on their descriptions and/or reviews and rank them in order of the probability of relevance.

To the best of our knowledge, retrieval of entities exploiting opinionated content as well as entity description is a new problem that has not been well addressed yet in previous work. Kavita and Zhai [8] ranked entities leveraging user reviews but no associated descriptions. In [7] and [5], the researchers exploited user reviews and associated structured data to rank entities while we do not use structured data but unstructured description data about entities.

User reviews are good extra sources to find apps especially when an app description is too short or is poorly described. However, leveraging both app descriptions and user reviews is challenging because the two types of unstructured text are written by different authors in different views. Consequently, different topics are stressed and different vocabulary sets are used in the two types of data, which make them hard to combine. In addition, user reviews often contain content that does not address the entity's features; indeed, a huge portion of reviews is about installation problems or general sentiment on the whole app. Therefore, careful unification of the two different types of data is desired.

The main research questions we would like to answer in our study are:

1. *How can we create a test collection for evaluating this new retrieval task?* When query log data is not available, obtaining realistic queries is challenging and important for research.

2. *How effective are the existing general retrieval models for this task?* Since there is no previous work studied on this task, we do not know how well existing models will work for app retrieval.

3. *Can reviews help?* User reviews can be easily obtained at app stores, but it is unknown whether app retrieval systems can be improved leveraging them.

4. *Can topic models be used to more effectively combine descriptions and reviews?* Given two different types of text data, how can we effectively generate unified representation of descriptions and reviews?

4. TEST SET CREATION

4.1 Collecting Apps from App Stores

App descriptions and user reviews are available at multiple app stores such as Google Play, Apple App Store, and Amazon Appstore. Among them, we chose Google Play because it is one of the largest app stores with abundant reviews. In Google Play app store, there are 41 categories of apps, where each app is assigned with only one category. From each category, we crawled about one thousand popular apps on average, which are somehow ranked by Google, resulting in about 43 thousand apps in total. For each app, we crawled up to the first 50 user reviews, which are ranked by their helpfulness votes. To compare our methods with the search engine in Google Play, we also crawled top 20 retrieved apps and their reviews from Google Play for each search query. After all, we crawled 43,041 app descriptions (one description per app) and 1,385,607 user reviews in total (32.2 reviews per app on average). We pre-processed text in the descriptions and reviews in the following way. We first tokenized text into word tokens and lemmatized them using Stanford CoreNLP [17] version 1.3.5. We lowered word tokens and removed punctuation. Then, stopwords and word tokens that appear in less than five descriptions and five reviews were removed. Also, word tokens that appear in more than 30 percent of descriptions or reviews were removed since they do not carry important meanings. We finally have 18,559 unique word tokens (V), and the statistics of the resulting text data is shown in Table 1.

Table 1: Statistics of text data in app descriptions (D) and user reviews (R).

	D	R
Average number of tokens	94.1	176.4
Total number of tokens	4,051,366	7,592,779

4.2 Where Can We Obtain Realistic Queries?

In order to quantitatively evaluate how well the suggested methods perform, we need a query set and a query relevance data set. Unfortunately, there does not yet exist such test collection. We thus create our own test collection. However, collecting realistic queries that embed the needs of users for app search is not easy. Researchers who do not work for companies that provide an app search engine generally do not have access to the query log data, hence it is hard for them to know which apps users want to find and which apps are difficult to find. To collect such real queries, we propose to leverage an app forum.

There is an Android forum[5] where users frequently ask various kinds of questions about Android including Android apps. We employ Google search engine to find threads containing an exact phrase "looking for an app" in the forum in order to find posts about app search. This can be done by giving the following query to the Google search engine: *"looking for an app" site:forums.androidcentral.com*. Then, for each search result, we systematically determined to collect the post or not. We retained a post only if the user looks for an app and the thread includes one or more answers that recommend relevant apps because there are users who look for non-existing apps. The first sixty such posts were saved, and one of them with title "walkie talkie app" is shown below.

> I'm looking for an app that if I push a button and choose either of my kids android phones. I want it to just play on their phone without having to click on a voice file or do something interactive. Kind of like an intercom. Does anything like this exist?

Next, we asked domain experts to write a short search query (usually a couple of keywords) for each post, by pretending they were the authors who wrote those posts and want to search for the app at app stores. Examples of such generated queries are: "locate cell tower", "podcast streamer", "nightstand clock", "auto text while driving", and "music player for church". Please note that the collected queries may not precisely reflect representative queries in actual app stores, and collecting such queries is left as our future work. Meanwhile, one may be concerned that the queries are biased towards difficult ones since we obtain them from forum posts, where users post questions when they do not know the answers. The relevance data in the next section show that the queries are not "very" difficult.

4.3 Collecting Query Relevance Data

To judge whether a retrieved app is relevant to a query, we need human-labeled relevance data. However, labeling all the retrieved apps by humans is too expensive. We thus created a pool, which consists of top retrieved apps from different retrieval systems, and we employed a crowdsourcing service, CrowdFlower[6], to label them at affordable prices. Specifically, for each query, we pooled together the top 20 retrieved apps from each of the suggested methods with multiple parameter combinations. Then, for each of the (query, retrieved app) pairs, we made a question providing the short query, the entire post, and the link of the retrieved app, and we asked three annotators to label it. Each annotator was asked to read the query and entire question post, follow the link to the app store, read the app description, and then judge if the app satisfies the search intent on three relevance levels (no satisfaction at all (0), partial satisfaction (1), and perfect satisfaction (2)).

Collecting a high-quality gold standard data set through crowdsourcing is often difficult since there are lots of abusers. To control quality of the relevance judgments, we manually judged relevance of 120 (query, app) pairs and used them as quiz questions. Each annotator was allowed to start labeling the data set only if the annotator passes our quiz session with at least 75% of accuracy, where the quiz session consists of eight randomly selected quiz questions. We also inserted a random quiz question for every four (query, app) pairs in a random order, without telling which one is a quiz question. If the annotator's accuracy goes below 75% at any point of time, we removed all the answers from the annotator and asked other annotators to judge them. We paid five cents for each judgment, and each annotator was limited to judge up to 250 questions. When all the needed judgments were made, we verified the given links to the app store, and if the links are no longer valid, we removed the (query, app) pairs since the annotators may have made random answers when

[5]http://forums.androidcentral.com/

[6]http://www.crowdflower.com/

they encountered the broken links. The three resulting judgments for each (query, app) pair were averaged to be used as a relevance score. From the 60 queries, we discarded four queries that Google Play search engine could not retrieve relevant apps in top 10 apps.

Table 2: Statistics of relevance data for 56 queries. Some statistics include standard deviations followed by "±".

Avg. # of words in each query	4.04 ± 1.43
# of distinct (query, app) pairs judged	4,534
# of all judgments	13,602
Avg. # of (query, app) pairs for each query	81.0 ± 17.5
# of all annotators	272
Avg. # of judgments for each annotator	50.0 ± 48.1
Fleiss' kappa	0.39
Perfect agreement rate	0.71

The statistics of relevance data for the resultant 56 queries are shown in Table 2. To measure inter-annotator agreement, we employed Fleiss' kappa. The kappa value is 0.39, which can be interpreted as between "Fair agreement" and "Moderate agreement" according to [13]. Perfect agreement rate, the proportion of (query, app) pairs where all three annotators agree on judgment, is 0.71. To see how difficult each query is, we counted the number of judgments where at least two annotators judged as perfect relevant. For each query, there are 13.6 such relevant apps on average with standard deviation being 7.87, which means that the queries are not very difficult considering the size of the data set. This may be due to the fact that the forum post was uploaded a while ago so that the non-existing apps could have been released before we crawled the data set.

5. METHODS

In order to retrieve apps that best match a query q, we first try existing standard retrieval models based only on app descriptions, which is a typical information retrieval problem. Then, we add user reviews to the data set to see if they are useful for the task. To combine app descriptions with user reviews, we propose a topic model-based method as well as traditional methods.

5.1 Standard Text Retrieval

Despite the importance of app retrieval problem, it has not been answered how well standard text retrieval methods perform. We, therefore, employ existing state-of-the-art methods here.

5.1.1 Okapi BM25

The Okapi BM25 method has been one of the state-of-the-art methods for ad-hoc retrieval. As presented in [6], BM25 scores a document d with respect to q as follows:

$$score(q,d) = \sum_{w \in q \cap d} [\frac{(k_3+1)c(w,q)}{k_3+c(w,q)} \times \frac{(k_1+1)c'(w,d)}{k_1+c'(w,d)} \times \log \frac{N+1}{df(w)+0.5}] \quad (1)$$

where $c(w,q)$ is w's count in q, $df(w)$ is a document frequency of w in a corpus, N is the number of all documents in a corpus, and k_1, k_3, and b are parameters. A normalized count of w in d, $c'(w,d)$, is defined as

$$c'(w,d) = \frac{c(w,d)}{1-b+b\frac{N_d}{avl(d)}} \quad (2)$$

where $c(w,d)$ is w's count in d. N_d is the length of d, $avl(d)$ is the average length of d in a corpus. We use this model as one of the most popular text retrieval methods.

5.1.2 Query Likelihood Language Model (QL)

Query Likelihood retrieval model was introduced by Ponte and Croft in [21] using multiple Bernoulli to model documents. Instead of multiple Bernoulli, most researchers have focused on using multinomial to model documents since it was shown to perform better than multiple Bernoulli model [23]. Therefore, we use Query Likelihood method with multinomial model (unigram language model) in this paper. Query Likelihood scores a document d with respect to q as follows:

$$score(q,d) = \prod_{w \in q} p(w|d) \quad (3)$$

where $p(w|d)$ is a probability of w being in d. In order to avoid over-fitting and keep $p(w|d)$ from being zero, $p(w|d)$ is smoothed by Dirichlet smoothing technique and defined as

$$p(w|d) = \frac{N_d}{N_d+\mu} p_{ml}(w|d) + \frac{\mu}{N_d+\mu} p(w|\boldsymbol{D}) \quad (4)$$

where $\boldsymbol{D}$ is a set of all documents, and $p_{ml}(w|d)$ and $p(w|\boldsymbol{D})$ are estimated by maximum likelihood estimator (MLE), yielding $p_{ml}(w|d)=\frac{c(w,d)}{\sum_{w'} c(w',d)}$ and $p(w|\boldsymbol{D})=\frac{c(w,\boldsymbol{D})}{\sum_{w'} c(w',\boldsymbol{D})}$. Smoothing parameter μ enables the system to dynamically smooth $p_{ml}(w|d)$ based on the length of d, N_d. Consequently, Query Likelihood Language Model with Dirichlet smoothing is regarded as one of the state-of-the-art retrieval models, and we call it QL in this paper. Please refer to [31] for more information on smoothing language models.

5.1.3 Topic Model-based Approach

Traditional retrieval models such as BM25 and QL do not consider association among words, which makes the system unable to retrieve documents that do not contain a query word. If there exists a vocabulary gap between queries and documents, the retrieval system is not supposed to work well. To solve the problem, we focus on enriching document representation with topic models. Please note that techniques such as query expansion can be combined with our suggested methods.

A topic model is a probabilistic model that can find latent themes and their distributions in a document from a text collection, where a theme (topic) is a cluster of words whose occurrence in documents overlap frequently. Thus, even if a document d does not contains a certain word w, $p(w|d)$ can be high enough if d contains many words that are in the same topic as w. For example, even if a word "bistro" is not contained in a description for a restaurant finder app, the app can be retrieved if the description contains a word "restaurant" since the two words are likely to be in the same topic(s). The two most popular topic models are Probabilistic Latent Semantic Analysis (PLSA) [9] and Latent Dirichlet Allocation (LDA) [1]. PLSA has two main problems: (1) the number of parameters grows as the data set size grows, and (2) it does not generate a new document, which has not been seen in training data. Those problems are solved in LDA by utilizing Dirichlet allocation, and thus, we employ LDA in this work.

For app retrieval problem, we suggest to exploit LDA-based document model (LBDM) [27], which has been shown to effectively model documents. The LBDM-based retrieval system was shown in [28] to generally outperform a retrieval system with a more sophisticated topic model, Pachinko Allocation Model (PAM) [14], which captures correlations among topics. Thus, we employ LBDM as one of the baselines in this study. We still use the same scoring formula

as in (3), where the document language model $p(w|d)$ is replaced with LBDM. As presented in [27], $p(w|d)$ of LBDM involves a linear interpolation of MLE-estimated language model and LDA document model, which is defined as

$$p(w|d) = \lambda\Big[\frac{N_d}{N_d+\mu}p_{ml}(w|d) + \frac{\mu}{N_d+\mu}p(w|\boldsymbol{D})\Big] + (1-\lambda)p_{lda}(w|d) \quad (5)$$

where the LDA document model, $p_{lda}(w|d)$, is described in [27] in detail. As in equation (4), MLE-estimated document model, $p_{ml}(w|d)$, is smoothed with MLE-estimated corpus model, $p(w|\boldsymbol{D})$.

5.2 Retrieval with Descriptions and Reviews

App descriptions are not written perfectly by app developers. For example, some descriptions are too short, and some others may contain too much useless information for search. Luckily, abundant user reviews are available, which may be a good source to complement such descriptions. Another important reason to leverage user reviews is that both search queries and user reviews are written from a user's perspective while descriptions are written from a developer's perspective. Due to the nature of apps, app descriptions are usually written mainly about their features. However, app developers may not exactly know what terms users would like to use to describe the features. For example, an app description may contain a phrase "find nearby restaurants" to describe its feature. If a user searches for "food near me", which does not have any common terms with the description, the app will not be retrieved by simple keyword matching even though the two phrases are about the same feature. In such case, user reviews may play an important role to bridge vocabulary gap between app developers and users. If there is a user review containing a phrase such as "good app for locating food near me" and the retrieval system indexes the review as well, the app would be retrieved even when the description does not have such terms.

To leverage user reviews, we need to somehow combine representations of a description d and a concatenated user review r. Combining representations of two different data sets by simply adding words in them together may not be a good idea if the data sets have different characteristics. In this section, we describe how to combine them using our novel method as well as traditional methods.

5.2.1 BM25F

BM25F has been known as state-of-the-art for structured information retrieval. Regarding descriptions and reviews as different fields of a document, we can apply BM25F to our problem. Similar to [20], we replace $c'(w,d)$ in equation (1) with $c''(w,a)$, which is defined as

$$c''(w,a) = \frac{boost_d \cdot c(w,d)}{1-b_d+b_d\frac{|d|}{avl(d)}} + \frac{boost_r \cdot c(w,r)}{1-b_r+b_r\frac{|r|}{avl(r)}} \quad (6)$$

where $boost_d$ and $boost_r$ are weights for d and r, respectively, and b_d and b_r play the same role as b does for BM25. $|r|$ is a length of review r, and $avl(r)$ is the average length of r in a review corpus.

5.2.2 Combined Query Likelihood

To combine two different types of text data, it may be better to assign some portion of an app representation to description data and some other portion of it to user review data. Thus, the unigram language model for a description and a review, $p(w|d)$ and $p(w|r)$, respectively, can be combined as in [18] to build a unified language model for an app, $p(w|a)$, which is defined as

$$p(w|a) = (1-\eta)p(w|d) + \eta p(w|r) \quad (7)$$

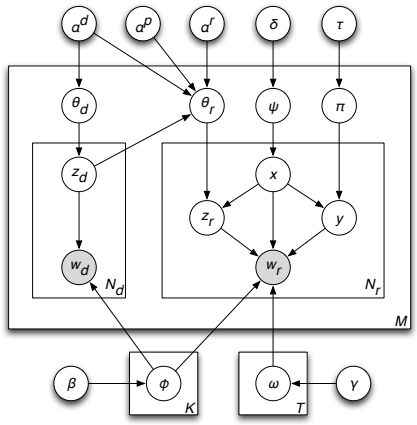

Figure 1: Graphical representation of AppLDA.

where η is a parameter to determine the proportion of review language model for $p(w|a)$. To score apps with respect to q, we follow the score function of QL. $p(w|d)$ and $p(w|r)$ are estimated by MLE and smoothed as in QL, and the resulting score function for q and a is defined as

$$score(q,a) = \prod_{w\in q} p(w|a) = \prod_{w\in q}\big[(1-\eta)p(w|d) + \eta p(w|r)\big]$$
$$= \prod_{w\in q}\Big[(1-\eta)\Big(\frac{N_d}{N_d+\mu_d}p_{ml}(w|d) + \frac{\mu_d}{N_d+\mu_d}p(w|\boldsymbol{D})\Big) + \eta\Big(\frac{N_r}{N_r+\mu_r}p_{ml}(w|r) + \frac{\mu_r}{N_r+\mu_r}p(w|\boldsymbol{R})\Big)\Big] \quad (8)$$

where $p(w|\boldsymbol{R})$ is a review corpus language model, N_r is the number of words in r, and μ_d and μ_r are Dirichlet smoothing parameters for d and r, respectively.

5.2.3 AppLDA: a topic model for app descriptions and user reviews

In our task, the role of topic model is similar to that of user reviews in that they both provide augmentation of vocabulary. In addition to bridging vocabulary gap, we design a topic model that can also remove noise in reviews. The key idea is to simultaneously model app descriptions and user reviews by sharing topics between the two different types of text and discarding parts of reviews if they don't share topics with app descriptions. Intuitively, when a user writes a review, the user would decide if he or she writes about a topic in app description or some other topics such as installation problems. Assuming that those other topics (review-only topics) do not help us retrieve relevant apps, we remove review texts that are about review-only topics in order to have a better estimation of app representations. We thus form review-only topics as well as shared topics to filter out review texts that are not useful for retrieval.

The graphical representation of AppLDA is depicted in Figure 1, and its generative process is described in Algorithm 1. The generation of app description by an app developer can be regarded as a typical topic modeling process that is explained for regular LDA in earlier this section. After an app description is generated, each word $w_{r,i}$ of review r with length N_r for an app a is written by a user. The user first chooses whether to write about topics that are shared with descriptions or some other topics that are far from the shared topics using switch $x_{r,i}$ according to a Bernoulli distribution $\boldsymbol{\psi}_a$, which is drawn from a Beta distribution with a symmetric tuple $\boldsymbol{\delta}$. If shared topics are chosen ($x_{r,i}=0$), the

Algorithm 1 Generative Process of AppLDA

for each shared topic z **do**
  draw $\boldsymbol{\phi}_z \sim Dirichlet(\boldsymbol{\beta})$
end for
for each review topic y **do**
  draw $\boldsymbol{\omega}_y \sim Dirichlet(\boldsymbol{\gamma})$
end for
for each app a with a description d and a review r **do**
  draw $\boldsymbol{\theta}_d \sim Dirichlet(\boldsymbol{\alpha}^d)$
  for each $i \in \{1, \ldots, N_d\}$ **do**
    draw $z_{d,i} \sim Multi(\boldsymbol{\theta}_d)$
    draw $w_{d,i} \sim Multi(\boldsymbol{\phi}_{z_{d,i}})$
  end for
  draw $\boldsymbol{\psi}_r \sim Beta(\boldsymbol{\delta})$
  draw $\boldsymbol{\theta}_r \sim Dirichlet(K \cdot \boldsymbol{\alpha}^p \cdot prior(\boldsymbol{\alpha}^d, \boldsymbol{z}_d) + \boldsymbol{\alpha}^r)$
  draw $\boldsymbol{\pi}_r \sim Dirichlet(\boldsymbol{\tau})$
  for each $i \in \{1, \ldots, N_r\}$ **do**
    draw $x_{r,i} \sim Bernoulli(\boldsymbol{\psi}_r)$
    if $x_{r,i} = 0$ **then**
      draw $z_{r,i} \sim Multi(\boldsymbol{\theta}_r)$
      draw $w_{r,i} \sim Multi(\boldsymbol{\phi}_{z_{r,i}})$
    else
      draw $y_{r,i} \sim Multi(\boldsymbol{\pi}_r)$
      draw $w_{r,i} \sim Multi(\boldsymbol{\omega}_{y_{r,i}})$
    end if
  end for
end for

user further specifies a shared topic $z_{r,i}$ from the topic distribution in r, $\boldsymbol{\theta}_r$, which is drawn from a Dirichlet distribution with an asymmetric vector $K \cdot \boldsymbol{\alpha}^p \cdot prior(\boldsymbol{\alpha}^d, \boldsymbol{z}_d) + \boldsymbol{\alpha}^r$. Here, K is the number of all shared topics, and $\boldsymbol{\alpha}^p$ is a symmetric vector. $prior(\boldsymbol{\alpha}^d, \boldsymbol{z}_d)$ is a distribution generated from topics in d, which is estimated by $\frac{N_{z,d}+\alpha^d}{N_d+K\alpha^d}$, where N with subscription and/or superscription means the number of words satisfying subscription/superscription conditions. For example, $N_{z,d}$ means the number of words assigned with z in d, and N_d is the number of words in d. Then, the user writes a word $w_{r,i}$ about the chosen shared topic according to a multinomial word distribution $\boldsymbol{\phi}_{z_{r,i}}$, which is drawn from a Dirichlet distribution with a symmetric vector $\boldsymbol{\beta}$. On the other hand, if the user chooses to write about topics that are far from shared topics ($x_{r,i} = 1$), the user further chooses a review topic $y_{r,i}$ according to a multinomial topic distribution $\boldsymbol{\pi}_r$, which is drawn from a Dirichlet distribution with a symmetric vector $\boldsymbol{\tau}$. Then, $w_{r,i}$ is chosen according to a word distribution $\boldsymbol{\omega}_{y_{r,i}}$, which is drawn from a Dirichlet distribution with a symmetric vector $\boldsymbol{\gamma}$. This process is repeated for all words in all app descriptions and user reviews for I iterations. Please note that all values in a symmetric vector are the same; *e.g.*, $\boldsymbol{\alpha} = \{\alpha, ..., \alpha\}$.

In order to guide the model to learn hidden topics in reviews, we use prior knowledge from topic distribution in app descriptions by $prior(\boldsymbol{\alpha}^d, \boldsymbol{z}_d)$. Intuitively, when a user writes a review about shared topics, the distribution of shared topics in reviews is likely to be at least somewhat similar to that in app descriptions. For example, if an app description is about finding nearby restaurants, the reviews are more likely to contain topics regarding restaurants than other topics such as finance or game topics. The prior knowledge in app descriptions is thus passed to reviews in the form of asymmetric prior distribution, $prior(\boldsymbol{\alpha}^d, \boldsymbol{z}_d)$, and this distribution is referred to draw topics in reviews. Here, the strength of the prior knowledge is controlled by the symmetric vector $K \cdot \boldsymbol{\alpha}^p$, and the prior knowledge is smoothed with the symmetric vector $\boldsymbol{\alpha}^r$. In other words, we can view this process as follows. A user is given a set of topics in an app description, and the user writes a review about the app referring to the topics in the description. Such prior knowledge can be employed via imposing asymmetric priors on the topic distributions of reviews. More information on applying asymmetric priors in a topic model can be found in [25].

The collapsed Gibbs sampling formulas to learn latent variables z_d, z_r, x, and y for an app a are as follows. Learning a topic of the ith word in d, $z_{d,i}$, is defined as

$$\begin{aligned} &p(z_{d,i}|\boldsymbol{W}^d, \boldsymbol{Z}^d_{\setminus d,i}, \boldsymbol{\alpha}^d, \boldsymbol{\beta}) \\ &\propto p(w_{d,i}|z_{d,i}, \boldsymbol{W}^d_{\setminus d,i}, \boldsymbol{Z}^d_{\setminus d,i}, \boldsymbol{\beta}) p(z_{d,i}|\boldsymbol{Z}^d_{\setminus d,i}, \boldsymbol{\alpha}^d) \\ &\propto \frac{N^{\setminus d,i}_{w_{d,i}|z_{d,i}} + \beta}{N^{\setminus d,i}_{z_{d,i}} + V\beta} \times \frac{N^{\setminus d,i}_{z_{d,i}|d} + \alpha^d}{N_d - 1 + K\alpha^d} \end{aligned} \tag{9}$$

where $\boldsymbol{W}^d$ is a set of all words in the description corpus, $\boldsymbol{Z}^d$ is all shared-topic assignments for those words in all descriptions, V is the size of vocabulary $\boldsymbol{V}$, and K is the number of all shared topics. Again, N with subscription and/or superscription means the number of words satisfying subscription/superscription conditions, and "$\setminus d, i$" means excluding d's ith data. To learn a shared topic ($x_{r,i} = 0$) for the ith word in r, $z_{r,i}$, we define the Gibbs sampling formula as

$$\begin{aligned} &p(x_{r,i} = 0, z_{r,i}|\boldsymbol{W}^r, \boldsymbol{Z}^r_{\setminus r,i}, \boldsymbol{Z}^d, \boldsymbol{X}_{\setminus r,i}, \boldsymbol{\alpha}^d, \boldsymbol{\alpha}^r, \boldsymbol{\alpha}^p, \boldsymbol{\delta}, \boldsymbol{\beta}) \\ &\propto p(x_{r,i} = 0|\boldsymbol{X}_{\setminus r,i}, \boldsymbol{\delta}) \times p(w_{r,i}|z_{r,i}, \boldsymbol{W}^r_{\setminus r,i}, \boldsymbol{Z}^r_{\setminus r,i}, \boldsymbol{\beta}) \\ &\quad \times p(z_{r,i}|\boldsymbol{Z}^r_{\setminus r,i}, \boldsymbol{Z}^d, \boldsymbol{\alpha}^d, \boldsymbol{\alpha}^r, \boldsymbol{\alpha}^p) \\ &\propto \frac{N^{\setminus r,i}_{x=0|r} + \delta}{N_r - 1 + 2\delta} \times \frac{N^{\setminus r,i}_{w_{r,i}|z_{r,i}} + \beta}{N^{\setminus r,i}_{z_{r,i}} + V\beta} \times \frac{N^{\setminus r,i}_{z_{r,i}|r} + K\alpha^p \frac{N_{z_{r,i}|d}+\alpha^d}{N_d+K\alpha^d} + \alpha^r}{(\sum_z N^{\setminus r,i}_{z|r}) + K(\alpha^p + \alpha^r)} \end{aligned} \tag{10}$$

where $\boldsymbol{W}^r$ is all words in the review corpus, and $\boldsymbol{Z}^r$ is all shared-topic assignments for those words in all reviews. On the other hand, to learn a review-only topic ($x_{r,i} = 1$) for the ith word in r, $y_{r,i}$, we define the Gibbs sampling formula as

$$\begin{aligned} &p(x_{r,i} = 1, y_{r,i}|\boldsymbol{W}^r, \boldsymbol{Y}_{\setminus r,i}, \boldsymbol{X}_{\setminus r,i}, \boldsymbol{\tau}, \boldsymbol{\delta}, \boldsymbol{\gamma}) \\ &\propto p(x_{r,i} = 1|\boldsymbol{X}_{\setminus r,i}, \boldsymbol{\delta}) \times p(w_{r,i}|y_{r,i}, \boldsymbol{W}^r_{\setminus r,i}, \boldsymbol{Y}_{\setminus r,i}, \boldsymbol{\gamma}) \\ &\quad \times p(y_{r,i}|\boldsymbol{Y}_{\setminus r,i}, \boldsymbol{\tau}) \\ &\propto \frac{N^{\setminus r,i}_{x=1|r} + \delta}{N_r - 1 + 2\delta} \times \frac{N^{\setminus r,i}_{w_{r,i}|y_{r,i}} + \gamma}{N^{\setminus r,i}_{y_{r,i}} + V\gamma} \times \frac{N^{\setminus r,i}_{y_{r,i}|r} + \tau}{(\sum_y N^{\setminus r,i}_{y|r}) + T\tau} \end{aligned} \tag{11}$$

where $\boldsymbol{Y}$ is a set of review-only topic assignments for all words in all reviews, and T is the number of all review-only topics.

Retrieval with AppLDA.

In order to retrieve apps relevant to a query q, we need document representations for apps, so we create a unigram language model for each a, $p_{lda}(w|a)$, which is defined as

$$\begin{aligned} p_{lda}(w|a) &= \sum_{z=1}^{K} p(w|z, \boldsymbol{W}^d, \hat{\boldsymbol{Z}}^d, \boldsymbol{\beta}) p(z|a, \hat{\boldsymbol{Z}}^d, \hat{\boldsymbol{Z}}^r, \boldsymbol{\alpha}^d, \boldsymbol{\alpha}^r, \boldsymbol{\alpha}^p) \\ &\propto \sum_{z=1}^{K} \frac{\hat{N}_{w|z} + \beta}{\hat{N}_z + V\beta} \times \frac{\hat{N}_{z|d} + \alpha^d + \hat{N}_{z|r} + K\alpha^p \frac{\hat{N}_{z|d}+\alpha^d}{N_d+K\alpha^d} + \alpha^r}{N_d + K\alpha^d + (\sum_z \hat{N}_{z|r}) + K(\alpha^p + \alpha^r)} \end{aligned} \tag{12}$$

where $\hat{\boldsymbol{Z}}^d$ and $\hat{\boldsymbol{Z}}^r$ are topics for descriptions and reviews estimated from AppLDA, respectively, and $\hat{N}$ with subscription is the estimated number of words satisfying the subscription condition. The formula can be interpreted as the unification

of LDA-estimated language models for descriptions and reviews, where the words that are not assigned with the shared topics are removed. In other words, the description and the cleaned review form a single unified document for each app, and the unified language model is used for retrieval. The AppLDA-estimated language model is combined with the MLE-estimated language models to define the score function for q and a as follows:

$$score(q,a) = \prod_{w \in q} p(w|a) = \prod_{w \in q} \Big((1-\lambda) p_{lda}(w|a) + \lambda \big[\frac{N_d + N_{x=0|r}}{N_d + N_{x=0|r} + \mu} p_{ml}(w|a) + \frac{\mu}{N_d + N_{x=0|r} + \mu} p(w|\boldsymbol{A}) \big] \Big) \tag{13}$$

where $N_{x=0|r}$ is the number of words assigned with shared topics in reviews, and $p_{ml}(w|a)$ is MLE-estimated language model for a's description and cleaned review, which is defined as

$$p_{ml}(w|a) = p(w|a, \boldsymbol{W}^d, \boldsymbol{W}^r, \hat{\boldsymbol{X}}) \propto \frac{N_{w|d} + N_{x=0,w|r}}{N_d + N_{x=0|r}} \tag{14}$$

and $p(w|\boldsymbol{A})$ is estimated by MLE for descriptions and cleaned reviews of all apps $\boldsymbol{A}$, and it is defined as

$$p(w|\boldsymbol{A}) = p(w|\boldsymbol{A}, \boldsymbol{W}^d, \boldsymbol{W}^r, \hat{\boldsymbol{X}}) \propto \frac{N_{w|\boldsymbol{D}} + N_{x=0,w|\boldsymbol{R}}}{N_{\boldsymbol{D}} + N_{x=0|\boldsymbol{R}}} \tag{15}$$

and μ is a Dirichlet smoothing parameter for MLE-estimated language models, and λ is a weight for MLE-estimated language models against the topic model-estimated language model. In order to estimate reliable values for LDA estimated language models, it is recommended to use multiple Markov chains in [27]. Similar to the results in [27], we found that three Markov chains with 100 Gibbs sampling iterations each show reasonably reliable performance, so we follow this setting.

6. EXPERIMENTS

In this section, we first describe how we set parameters for the experiments. Then, we qualitatively analyze the search results from the suggested methods, and we quantitatively evaluate the results using our test collection.

6.1 Parameter Setting

The following parameter values are used in the experiments unless otherwise specified. The parameters are tuned from our data set based on average of four NDCG measures specified in section 6.3. For BM25, we use the standard value k_3=1,000, and we set the parameters k_1=4.0 and b=0.4, which showed the best performance. For BM25F, we tune the parameters at our best, and we consider the following values: k_3=1,000, k_1=3.5, b_d=0.4, b_r=0.3, $boost_d$=0.6, and $boost_r$=0.4. For Query Likelihood Language Model (QL), μ is tuned to be 1,000. For Combined Query Likelihood (CombQL), the same μ is used, and we set μ_r=300 and η=0.4, which showed the best performance. For LBDM, K is tuned to be 300, and we set topic model parameters $\alpha=\frac{50}{K}$ and β=0.01, which is the common setting in the literature. Its retrieval parameters λ and μ are tuned to be 0.5 and 1,000, respectively. To see how well regular QL and LBDM perform with both data sets $\boldsymbol{D}$ and $\boldsymbol{R}$, we simply add words in reviews to the corresponding descriptions and used the merged documents as an input to QL and LBDM; we call these methods as QL($\boldsymbol{D}, \boldsymbol{R}$) and LBDM($\boldsymbol{D}, \boldsymbol{R}$). μ for QL($\boldsymbol{D}, \boldsymbol{R}$) is tuned to be 800. For LBDM($\boldsymbol{D}, \boldsymbol{R}$), K is tuned to be 300, and the same α and β values are used as for regular LBDM, and retriever parameters λ and μ are tuned to be 0.5 and 800, respectively. For our proposed model AppLDA, we set K=300 and T=30, which showed the best performance. We use the standard values for other topic model parameters: $\alpha^d=\alpha^r=\frac{50}{K}$, $\tau=\frac{50}{T}$, and $\beta=\gamma$=0.01. If one believes that the reviews have a specific amount of shared-topic proportion, then δ can be used as asymmetric prior. However, we let the model fully figure out the proportions, so we set δ=0.5 for symmetric vector $\boldsymbol{\delta}$, which is a small value. A larger value of $alpha^p$ lets the distribution of shared-topics in reviews be more similar to that in app descriptions; α^p is tuned to be 0.05. For retrieval parameters of AppLDA, we set λ=0.5 and μ=800, which showed the best performance.

6.2 Qualitative Analysis

Table 3: Top shared topics by AppLDA.

slot	check	photo	news
bonus	deposit	picture	article
win	mobile	pic	story
machine	account	gallery	break
spin	bank	image	read
casino	balance	effect	local
coin	banking	editing	latest
payout	credit	editor	content
slots	transfer	filter	fox
jackpot	transaction	edit	live

Table 4: Top review-only topics by AppLDA.

log	interface	ad	addictive
account	uus (ui)	pop	pass
login	design	annoying	addicting
sign	function	remove	enjoy
error	miss	advert	addict
password	lack	advertisement	challenge
website	user	rid	kill
connect	functionality	full	entertaining
server	improvement	seconds	interesting
access	clean	click	challenging

Table 5: Top topics in the reviews by LDA.

log	upgrade	purchase	note
account	battle	refund	support
check	spend	upgrade	pro
mobile	character	reinstall	developer
login	gold	bug	sync
password	rpg	force	program
sign	gameplay	year	tool
deposit	attack	month	dev
service	level	lose	tablet
bank	fight	block	draw

Table 3 and 4 show the biggest shared topics and review-only topics (measured by $\hat{N}_z$), respectively, estimated by AppLDA. At Google Play, 18 of 41 categories are game-related categories, so they are reflected in the topics; for example, the biggest shared topic is about "casino game", and the fourth review-only topic is about "sentiment towards games". The other biggest shared topics are about "mobile banking", "photo editor", and "news article", and each of them represents a feature of an app well. Review-only topics seem reasonable since the words in them are likely to appear more often in reviews than in descriptions. We also compare top review-only topics from AppLDA with top topics from regular LDA when we use only user reviews, which is shown in Table 5. Comparing the biggest topics of them, which are both about "account", it is shown that the topic in

Table 6: Top retrieved apps for a query "locate cell tower". Strikethrough indicates irrelevant apps.

QL	LBDM	CombQL	AppLDA
~~GPS Tracker Pro~~	zBoost Signal Finder	Cell Map	Cell Map
Tower Collector	3G 4G WiFi Map & Speedtest	zBoost Signal Finder	Map My Cell Tower
3G 4G WiFi Map & Speedtest	~~GPS Tracker Pro~~	Signal Finder	zBoost Signal Finder
~~Locate -Find Friends & Family~~	Tower Collector	Network Signal Info Pro	Signal Finder
~~Find My Phone~~	Map My Cell Tower	Tower Collector	Network Signal Info Pro
inViu OpenCellID	~~Locate -Find Friends & Family~~	Map My Cell Tower	Tower Collector
~~Signal Booster Reloaded~~	inViu OpenCellID	3G 4G WiFi Map & Speedtest	Cell Info Display
~~Signal Booster for Android~~	Signal Finder	~~Family Tracker: Locate Phones~~	3G 4G WiFi Map & Speedtest
~~Family Tracker: Locate Phones~~	~~Find My Phone~~	inViu OpenCellID	~~Sprint Family Locator~~
~~Digital Leash Locate~~	~~Family Tracker: Locate Phones~~	~~GPS Tracker Pro~~	~~GPS Tracker Pro~~

LDA is corrupted with "bank"-related words such as "bank" and "deposit" while the topic in AppLDA is about general accounts of apps. An "account problem" topic is more likely to appear in review-only topics while a "mobile banking" topic is more likely to appear in shared topics. AppLDA is able to separate such topics well by forming two different types of topics. Interestingly, AppLDA's shared-topics consist of words that do not carry sentiment while review-only topics often contains words carrying sentiment such as "miss", "lack", "annoying", and "addicting". This means that AppLDA separated the two different types of topics reasonably well; since regular LDA does not explicitly separate them, top topics in reviews contain few sentiment words, but several words of them are likely to appear often in app descriptions.

To show the difference of retrieved apps from different models, we retrieve apps for a query "locate cell tower" using the suggested methods. The top retrieved apps are showed in Table 6. According to the question post, the query looks for an app that locates a cell tower the phone is currently attached to. By comparing methods that do not leverage user reviews (QL and LBDM) with methods that do leverage user reviews (CombQL and AppLDA), we can see the effect of adding more review text. The relevant apps such as "Map My Cell Tower" and "zBoost Signal Finder" do not contain the query word "locate", which makes them hard to find. However, since the reviews of those apps contain phrases such as "Won't even locate my cell tower" and "Handy app to locate towers", CombQL and AppLDA could rank them high. While the reviews help bridge vocabulary gap by adding absent words from review text, topic model-based method also bridges vocabulary gap by connecting associated words. LBDM gave high scores to relevant apps such as "zBoost Signal Finder", "Map My Cell Tower", and "Signal Finder", which do not contain the word "locate" in their reviews, even though it does not leverage user reviews. Since the descriptions of those apps contain words such as "gps" and "map" that are in the same topics as "locate" is, they could be ranked high.

6.3 Quantitative Analysis

Since we average relevance judgments of three annotators, the aggregated relevance is defined as a real number in [0,2]. Hence, evaluation metrics such as Mean Average Precision (MAP), which requires binary relevance, cannot be employed. We instead employ Normalized Discounted Cumulative Gain (NDCG) [11] as the evaluation metric because NDCG is able to measure ranking performance on multiple-level relevance data. Specifically, we measure NDCG at 3, 5, 10, and 20 top retrieved apps to reflect diverse users' information needs. Unlike traditional web search, NDCG@3 might be quite important; it is common for app stores to show only one or a couple of retrieved apps on a smartphone screen. On the other hand, obtaining judgments for top 20 retrieved apps of retrieval systems with all combinations of parameter values is too expensive and not realistic in practice. We judged top 20 retrieved apps of the suggested retrieval systems with 22 parameter value combinations. The relevance data may be thus incomplete. However, it is shown in [29, 2] that ignoring unjudged documents is effective in such situations. Therefore, we alternatively employ induced NDCG at k judged apps, which shares the same philosophy as induced MAP in [29]. Induced NDCG ignores unjudged apps from the ranked list and is calculated in the same way as regular NDCG. We simply call it NDCG in this paper.

Table 7: NDCG evaluation results for mobile app retrieval. The first three methods exploit only app descriptions, $\boldsymbol{D}$, while the next five methods leverage user reviews, $\boldsymbol{R}$, as well as app descriptions, $\boldsymbol{D}$.

	N@3	N@5	N@10	N@20
BM25	0.578	0.550	0.527	0.537
QL	0.541	0.517	0.511	0.515
LBDM	0.584	0.563	0.543	0.565
BM25F	0.597	0.596	0.583	0.597
QL ($\boldsymbol{D}$,$\boldsymbol{R}$)	0.613	0.618	0.585	0.586
CombQL	0.637	0.624	0.602	0.593
LBDM($\boldsymbol{D}$,$\boldsymbol{R}$)	0.610	0.625	0.613	0.626
AppLDA	**0.651**†§	**0.656**†‡	**0.627**†	**0.634**†‡
Google Play	0.589	0.575	0.568	0.566

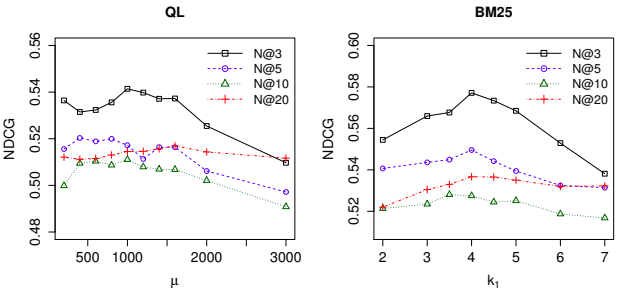

Figure 2: NDCG measures for different μ values of QL (left) and for different k_1 values of BM25.

We compare the performance of the suggested methods as well as Google Play's app search engine. Table 7 shows the evaluation results. †, ‡, and § are used to mark if the improvement for AppLDA is statistically (paired t-test with $p \leq 0.05$) significant in each measure over LBDM, CombQL, and LBDM($\boldsymbol{D}$,$\boldsymbol{R}$), respectively. As expected, LBDM outperforms BM25 and QL in all measures when only app descriptions are available because LBDM is able to recognize semantically related words. When only app descriptions are used, BM25 outperforms QL in all measures, and their performance on different parameter values is shown in Figure 2. However, when both descriptions and reviews

are used, QL($\boldsymbol{D}$,$\boldsymbol{R}$) and CombQL outperform BM25F in all measures except NDCG@20, which means QL and CombQL are more suitable to combine different data types for app retrieval than BM25F. It is clear to see that the models that leverage user reviews perform better than the models that use only descriptions. For example, BM25F outperforms BM25, and CombQL and QL($\boldsymbol{D}$,$\boldsymbol{R}$) outperform QL with a relatively big performance difference. In addition, AppLDA and LBDM($\boldsymbol{D}$,$\boldsymbol{R}$) outperform LBDM, which means that topic model's capability of bridging vocabulary is even amplified when user reviews are added. QL($\boldsymbol{D}$,$\boldsymbol{R}$) exploits user reviews by concatenating descriptions and user reviews, and it is outperformed by CombQL that combines description model and user review model with linear interpolation. This means that the review data set and description data set have their own characteristics, so they need to be combined without losing them. AppLDA outperforms CombQL and LBDM($\boldsymbol{D}$,$\boldsymbol{R}$), and the improvement is statistically significant in two measures and one measure, respectively. While CombQL does not score high in NDCG@10 and NDCG@20, LBDM($\boldsymbol{D}$,$\boldsymbol{R}$) does not score high in NDCG@3. AppLDA seems to complement such drawbacks of CombQL and LBDM($\boldsymbol{D}$, $\boldsymbol{R}$) by effectively modeling app descriptions and user reviews.

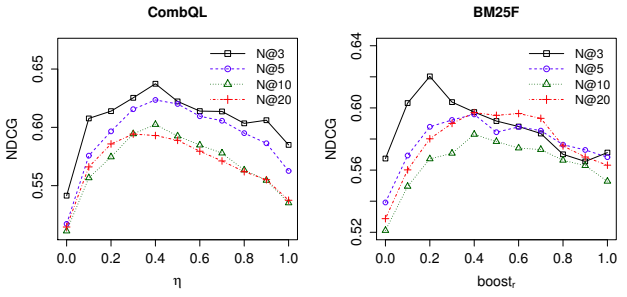

Figure 3: NDCG measures for different review weights (η) of CombQL (left) and for different review weights ($boost_r$) of BM25F when $boost_d = 1.0 - boost_r$ (right).

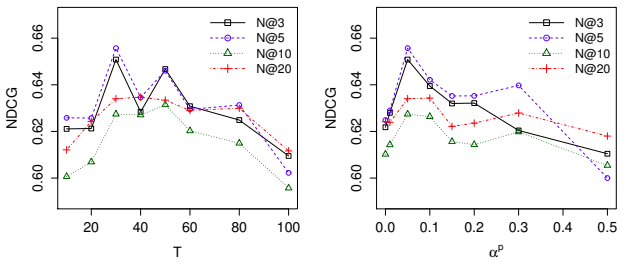

Figure 4: NDCG measures for AppLDA when different numbers of review-only topics (T) are used (left) and different prior weights (α^p) are used (right).

In order to understand the effects of leveraging reviews, we further investigate performance when different proportions of review representations are used. NDCG measures for different η values of CombQL and different $boosts_r$ values of BM25F are shown in Figure 3. η is a weight of review language model in CombQL, and $boost_r$ is a weight of normalized count of word in reviews. Here, we set $boost_d = 1.0 - boost_r$ for BM25F. When $\eta = 1.0$ or $boost_r = 1.0$, the models exploit only reviews while they exploit only descriptions when $\eta = 0.0$ or $boost_r = 0.0$. Surprisingly, for both models, using only reviews gives a better performance than using only descriptions; which means that review data set may be a better resource for app search than description data set. Since both reviews and queries are written by users, there may be less vocabulary gap between them, resulting in reviews being more useful for app retrieval than descriptions. Combining app descriptions and user reviews yields even better performance, which peaks when $\eta = 0.4$ and $boost_r$ is around 0.4. This means that descriptions and reviews supplement each other well.

Figure 4 shows AppLDA's performance when different numbers of review-only topics T are used and different values of α^p priors are used. It seems that about 30 to 50 review-only topics exist in the review data set, and setting too few or too many review topics harm the performance. The number of review-only topics is much smaller than that of shared topics (300). This is because there are various available features for each category of apps while the topics users mention other than app features converge to a small set of topics such as "installation problems" and "user interface". Meanwhile, α^p controls the amount of topic distribution priors for user reviews, and it is obtained from topic distribution of app descriptions. The priors are used to give clues when identifying topics in reviews under the assumption that the distribution of shared topics in reviews is likely to be similar to that of app descriptions. Indeed, it is shown in Figure 4 that adding priors is helpful while adding too much priors is rather harmful. The performance peaks at $\alpha^p = 0.05$ when $K = 300$, which means that giving about fifteen guiding words to a noise-removed review is generally desired, in other words.

7. CONCLUSION

In this paper, we conducted the first study of a new retrieval task, *i.e.*, mobile app retrieval. Since no test collection exists yet, we created the first one to evaluate this task. We used this test collection to systematically study the effectiveness of both existing retrieval models and a new model that we proposed to effectively leverage the companion review data with apps. Specifically, in order to combine different vocabulary sets in app descriptions and user reviews and to filter out noise in reviews, we proposed a topic model (AppLDA) that jointly models reviews and app descriptions and extracts aligned topics between reviews and descriptions. Evaluation results show that (1) BM25 outperforms QL when only descriptions are used while QL and CombQL generally outperforms BM25F when reviews are added, (2) leveraging reviews indeed helps app retrieval, (3) AppLDA significantly outperforms traditional retrieval models, and (4) adding priors in AppLDA helps align topics between app descriptions and user reviews.

Our work can be further extended in several ways: (1) collecting representative queries when query log data is not available can be further studied, (2) one can explore other directions such as query expansion and feedback-based models for app retrieval problem, and (3) one can identify other characteristics of mobile apps to design a better retrieval model. Our created test collection is made publicly available and thus enables further study of this new problem.

8. ACKNOWLEDGMENTS

This work is supported in part by a gift fund from TCL and by the National Science Foundation under Grant Number CNS-1027965.

9. REFERENCES

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[2] C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32. ACM, 2004.

[3] A. Datta, K. Dutta, S. Kajanan, and N. Pervin. Mobilewalla: A mobile application search engine. In *Mobile Computing, Applications, and Services*, pages 172–187. Springer, 2012.

[4] A. P. De Vries, A.-M. Vercoustre, J. A. Thom, N. Craswell, and M. Lalmas. Overview of the inex 2007 entity ranking track. In *Focused Access to XML Documents*, pages 245–251. Springer, 2008.

[5] H. Duan, C. Zhai, J. Cheng, and A. Gattani. Supporting keyword search in product database: A probabilistic approach. *Proc. VLDB Endow.*, 6(14):1786–1797, Sept. 2013.

[6] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 49–56. ACM, 2004.

[7] K. Ganesan and C. Zhai. Findilike: preference driven entity search. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 345–348. ACM, 2012.

[8] K. Ganesan and C. Zhai. Opinion-based entity ranking. *Information retrieval*, 15(2):116–150, 2012.

[9] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[10] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[11] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[12] J. Kamps, M. Marx, M. De Rijke, and B. Sigurbjörnsson. Xml retrieval: What to retrieve? In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 409–410. ACM, 2003.

[13] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[14] W. Li and A. McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584. ACM, 2006.

[15] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua. Addressing cold-start in app recommendation: latent user models constructed from twitter followers. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 283–292. ACM, 2013.

[16] Z. Liu, J. Walker, and Y. Chen. Xseek: a semantic xml search engine using keywords. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1330–1333. VLDB Endowment, 2007.

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[18] P. Ogilvie and J. Callan. Combining document representations for known-item search. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 143–150. ACM, 2003.

[19] J. Pehcevski, A.-M. Vercoustre, and J. A. Thom. Exploiting locality of wikipedia links in entity ranking. In *Advances in Information Retrieval*, pages 258–269. Springer, 2008.

[20] J. Pérez-Iglesias, J. R. Pérez-Agüera, V. Fresno, and Y. Z. Feinstein. Integrating the probabilistic models bm25/bm25f into lucene. *arXiv preprint arXiv:0911.5046*, 2009.

[21] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[22] S. E. Robertson. The probability ranking principle in ir. *Readings in information retrieval*, pages 281–286, 1997.

[23] F. Song and W. B. Croft. A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 316–321. ACM, 1999.

[24] A.-M. Vercoustre, J. A. Thom, and J. Pehcevski. Entity ranking in wikipedia. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1101–1106. ACM, 2008.

[25] H. M. Wallach, D. Minmo, and A. McCallum. Rethinking lda: Why priors matter. 2009.

[26] N. Walsh, M. Fernández, A. Malhotra, M. Nagy, and J. Marsh. Xquery 1.0 and xpath 2.0 data model (xdm). *W3C recommendation, W3C (January 2007)*, 2007.

[27] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185. ACM, 2006.

[28] X. Yi and J. Allan. A comparative study of utilizing topic models for information retrieval. In *Advances in Information Retrieval*, pages 29–41. Springer, 2009.

[29] E. Yilmaz and J. A. Aslam. Estimating average precision with incomplete and imperfect judgments. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 102–111. ACM, 2006.

[30] P. Yin, P. Luo, W.-C. Lee, and M. Wang. App recommendation: a contest between satisfaction and temptation. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 395–404. ACM, 2013.

[31] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM, 2001.

[32] H. Zhu, H. Xiong, Y. Ge, and E. Chen. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 951–960. ACM, 2014.