Panorama: A Semantic-Aware Application Search Framework

Di Jiang, Jan Vosecky, Kenneth Wai-Ting Leung, Wilfred Ng
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{dijiang, jvosecky, kwtleung, wilfred}@cse.ust.hk

ABSTRACT

Third-party applications (or commonly referred to the apps) proliferate on the web and mobile platforms in recent years. The tremendous amount of available apps in app marketplaces suggests the necessity of designing effective app search engines. However, existing app search engines typically ignore the latent semantics in the app corpus and thus usually fail to provide high-quality app snippets and effective app rankings. In this paper, we present a novel framework named Panorama to provide independent search results for Android apps with semantic awareness. We first propose the App Topic Model (ATM) to discover the latent semantics from the app corpus. Based on the discovered semantics, we tackle two central challenges that are faced by current app search engines: (1) how to generate concise and informative snippets for apps and (2) how to rank apps effectively with respect to search queries. To handle the first challenge, we propose several new metrics for measuring the quality of the sentences in app description and develop a greedy algorithm with fixed probability guarantee of near-optimal performance for app snippet generation. To handle the second challenge, we propose a variety of new features for app ranking and also design a new type of inverted index to support efficient Top-k app retrieval. We conduct extensive experiments on a large-scale data collection of Android apps and build an app search engine prototype for human-based performance evaluation. The proposed framework demonstrates superior performance against several strong baselines with respect to different metrics.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Text Mining

General Terms

Algorithms, Experimentation

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy.

Keywords

App, Topic Model, Query Processing

1. INTRODUCTION

In recent years, apps that are developed for the web and mobile devices have quickly become a million-dollar industry. Apple, Google and Microsoft have already opened their own app marketplaces. The absolute number of apps in existence, as well their rates of growth, are remarkable. For example, at the time of writing this paper, Google Play[1] claims that there are 400,000 apps available for the Android platform and the app growth rates are nearly 81% on a yearly basis. Given the potential of the market of apps, it is necessary to develop effective app search engines for the users, since 85% of app downloads are discovered through search[2]. However, we observe that two problems have not been satisfactorily solved in current app search engines. Consider the following two motivating examples:

Table 1: Google Play Snippet of *Angry Bird Seasons*

Description	Angry Birds Seasons: Cherry Blossom Festival! Angry Birds Seasons takes the captivating game play of the original to a whole new level! From Halloween to Chinese New Year, the birds are celebrating different festive seasons around the world! With more than 260 levels and regular free updates, these special episodes offer more challenging levels of pig-popping action and golden eggs to discover. ...
Snippet	Angry Birds Seasons: Cherry Blossom Festival! Angry Birds Seasons takes the captivating game play of the original to a wh...

PROBLEM 1. *(App Snippets)*
App snippet is the summary of the app's description. It needs to be concise and representative of the theme of the app description. However, since the app description is typically informal and lacks of careful organization, it is challenging to generate high-quality snippets. Commercial app search engines such as Google Play select the first several (about 166) characters of the app description as the app's snippet. Although this simple approach is workable in practice, it fails to guarantee generating high-quality app snippet

[1]Previously known as Android Market.
[2]http://blog.quixey.com/2012/02/02/search-is-king/.

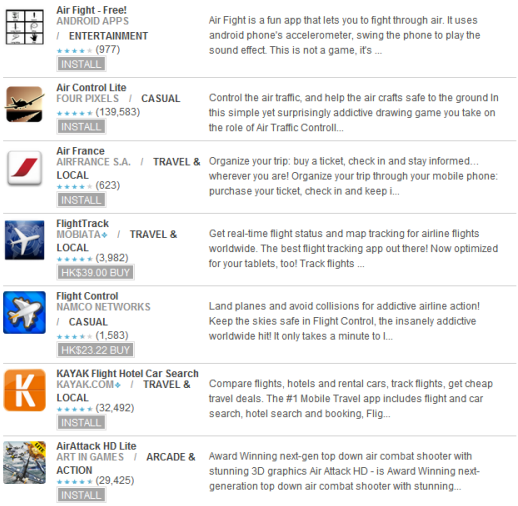


Figure 1: Google Play Search Result of *air fight*

in many cases. Consider the example shown in Table 1. The theme of the description is to highlight the app's differences from the previous versions. The last sentence in the description is highly relevant to the theme, however, it fails to be included in the snippet generated by Google Play. In contrast, the snippet generated by Google Play is not closely related to the theme and is not effective to reveal the real features of the app.

PROBLEM 2. *(App Ranking)*
App ranking is critical for app search engines. The high-ranked apps should be highly relevant to the user's information need. However, it is reported that mainstream app search engines still rely on the simplistic term matching technique in IR[3]*. This approach usually fails in the app search scenario, where the text information of many relevant apps only contains terms that have similar semantic meaning with the query terms. Consider a user who is looking for an app to emulate fighter jets and searches "air fight", the top seven results returned by Google Play of "air fight" is presented in Figure 1. The 7th app is a perfect match to the user's search intent. Although the description of the 7th app contains "air combat", Google Play fails to see the semantic similarity between the terms "fight" and "combat", thus, some irrelevant apps about air control are ranked higher than the relevant one.*

Although we only provide the motivating examples from Google Play, these two problems are also common in other commercial app search engines. In order to solve the problems, the semantics between words are needed to help capture the themes of app descriptions as well as provide effective app rankings. Although semantics can be obtained from information sources such as ontology or so, they are usually costly to obtain and cannot easily adapt to the highly dynamic app ecosystem. In this paper, we constrain "semantics" as the latent word relations that can be automatically obtained from the app corpus. We develop a semantic-aware app search framework named Panorama, which solves the challenging issues of how to discover the semantics from the app corpus as well as how to utilize the discovered semantics to solve the two aforementioned problems.

The Panorama framework is highlighted by three major components: the *App Topic Model Component*, the *App Snippet Generation Component* and the *App Ranking Component*. The *App Topic Model Component* contains the App Topic Model (ATM), which is specialized in app analysis. Since topic modeling has been shown effective to find the latent relations in data [3], the ATM is designed to discover the latent semantic by seamlessly integrating the text, the link as well as the category information in the app corpus. The *App Snippet Generation Component* utilizes different features to evaluate the *quality* of the sentences in app description and then employs a greedy algorithm to choose the high-quality sentences with redundancy reduction. The greedy algorithm is guaranteed to perform near-optimally with a fixed probability. In the *App Ranking Component*, we propose 12 features to evaluate an app's static quality from the three perspectives of app popularity, developer reputation and link prestige. Then the traditional TF-IDF score and topic score are further utilized to evaluate the relevancy of an app with respect to a search query. Since the traditional inverted index does not support the topic information, we design a new type of inverted index, which supports efficient Top-k app retrieval by evaluating the app's static quality, TF-IDF score and topic score simultaneously. Although we focus on Android apps in this paper, the techniques we proposed can be easily adapted to apps in other marketplaces. The contributions of this paper are summarized as follows:

- First, we propose the App Topic Model (ATM) that integrates the text, the link and the category information in order to discover the latent semantics from apps. To the best of our knowledge, it is the first topic model specialized in app analysis.
- Second, we propose three metrics to evaluate the quality of the sentences in app description. Based on the wide spectrum of metrics, a greedy algorithm which has nearly-optimal performance with a fixed probability is proposed to generate app snippets with redundancy reduction.
- Third, we propose 12 features to evaluate an app's static quality and utilize the TF-IDF score and the topic score to evaluate an app's relevance with respect to a search query. A new kind of inverted index is also proposed to support efficient Top-k query processing by evaluating the static quality score, the TF-IDF score and the topic score simultaneously.
- Fourth, we conduct extensive experiments on a large collection of Android apps and develop an app search engine prototype to perform human-based evaluations. Compared with some commercial app search engines, the proposed framework demonstrates comparable or even superior performance with regards to different metrics.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we discuss the app topic model. In Sections 4 and 5, we discuss the app snippet generation and app ranking. In Section 6, we present

[3]http://www.wired.com/gadgetlab/2012/02/chomp-apple-app-store/

Table 2: Example of Crawled App Information

Title	Angry Bird Seasons
Description	Angry Birds Seasons: Cherry Blossom Festival! Angry Birds Seasons takes the captivating game play of the original to a whole new level! From Halloween to Chinese New Year, the birds are celebrating different festive seasons around the world! With more than 260 levels and regular free updates, these special episodes offer more challenging levels of pig-popping action and golden eggs to discover. ...
Category	Arcade&Action
Developer	Rovio Mobile Ltd.
Average Rating	4.5
Rating Users	467,557
Price	Free
Installs	50,000,000 - 100,000,000 last 30 days
Same Developer	Angry Birds, Angry Birds Rio, Angry Birds Space Premium, ...
Also Viewed	Angry Monkey, Squibble, Angry Gran Free Game,...
Also Installed	AngryBirdsBackup Free, 3 Stars in Angry Birds, Save The Bird, ...

the experimental results. Finally, the paper is concluded in Section 7.

2. RELATED WORK

The paper is essentially related to three research fields: topic modeling, document summarization and inverted index. We now review them in the following three subsections.

2.1 Topic Modeling

Topic modeling is gaining momentum in text mining in recent years. Latent Dirichlet Allocation (LDA) is first proposed in [3] to analyze electronic documents. Topic models are reported to be effective to discover topics from academic articles [8] and web search query log [11]. Our work is also closely related to topic models that capture the links between documents. A generative model for hypertext document collections that explicitly models the generation of links is proposed in [9]. In [6], a joint probabilistic model is proposed for modeling the contents and inter-connectivity of document collections. Recently, Xia *et al.* [28] proposed the PLink-LDA, which utilizes the citation information as prior knowledge in the generative process of topic modeling. The ATM is essentially different from the previous ones, since it needs to incorporate three kinds of information: the text, the link and the category. Moreover, there are three different types of links in ATM while previous models usually have only one kind of link.

2.2 Document Summarization

Document summarization has been intensively studied in IR. Neto *et al.* [18] present a summarization procedure based on trainable machine learning algorithms which employs a set of features extracted directly from the original text. Varadarajan *et al.* [25] present a method to create query-specific summaries by identifying the most query-relevant fragments and combining them using the semantic associations within the document. Lin *et al.* [14] proposed a class of submodular functions for document summarization tasks. Shen *et al.* [23] proposed a new principled and versatile framework for multi-document summarization using the minimum dominating set. Our app snippet generation method is different in that some new metrics are proposed to handle the unique challenges in app analysis and our method has near-optimal performance guarantee with a fixed probability.

2.3 Inverted Index

The inverted index is a widely used data structure in search engines. The latest advancement of inverted index is to design new query processing paradigms to improve the efficiency. [24] shows how new accumulator trimming techniques combined with inverted list skipping can produce extremely high performance retrieval systems. Broder *et al.* [4] present an efficient query evaluation method based on a two level approach, which utilize early termination to achieve faster query processing. Recently, a set of new algorithms [7] [22] is proposed by utilizing an augmented inverted index structure called a block-max index to conduct high-performance query processing. We introduce the topic information into the traditional inverted index to build a new kind of inverted index. To the best of our knowledge, our work is the first one that discusses indexing the results of topic models.

3. APP TOPIC MODEL

Although it is reported in industry that topic modeling is useful for app search[4], little information has been published to reveal how to apply topic modeling to analyze apps. In this section, we propose the *App Topic Model* (ATM), which utilizes the crawled app information to discover the latent semantics, the *topics*, from the app corpus. To illustrate our ideas, we present an example of the crawled app information in Table 2 and the generative process of ATM in Algorithm 1. ATM makes use of the text information of app description, the link information (i.e., *Same Developer*, *Also Viewed* and *Also Installed*) and the category information (e.g., *Entertainment*, *Communication*, etc). When generating a sentence, the probability of choosing a specific topic subjects to the present app's topic distribution as well as the topic distributions of the apps that the present app is linked to. The line 22 of Algorithm 1 depicts how to generate the category information. We assume that there exists C app categories, $\eta_{1:C} = [\eta_1, ..., \eta_C]^T$ is a matrix with C K-dimensional logistic regression parameters as the rows, where η_C is a zero vector by default, so we only use $\eta_{1:C-1}$ as the parameters to

[4]http://chomp.com/us/about

be estimated. $\overline{z}$ is an average of $z_{1:N}$ over all observed words, where each z_n is a K-dimensional unit vector with only the ith entry being 1 if it denotes the ith topic. The category variable y can be considered as a sample generated from the discrete distribution $(p_1, ...p_{C-1}, 1 - \sum_{i=1}^{C-1} p_i)$, where $p_i = \frac{\exp(\eta_i^T \overline{z})}{1+\sum_{i=1}^{C-1}(\eta_i^T \overline{z})}$.

Algorithm 1 Generative Process of App Topic Model

for each topic $k \in 1, ..., K$ **do**
  draw a word distribution $\phi_k \sim$ Dirichlet(β);
end for
for each app $d \in 1, ..., D$ **do**
  draw d's topic distribution $\theta_d \sim$ Dirichlet(α);
  draw d's link type distribution $\theta'_d \sim$ Dirichlet(Ω);
  for each sentence s in d **do**
    if the link existence indicator $I_d > 0$ **then**
      draw $b \sim$ Bernoulli(o);
      draw link type $c \sim$ Multinomial(θ'_d)
      draw link $l \sim$ Uniform(π_c);
      if $b = 1$ **then**
        draw a topic $z \sim$ Multinomial(θ_d);
      else
        draw a topic $z \sim$ Multinomial(θ_{d_l});
      end if
    else
      choose a topic $z \sim$ Multinomial(θ_d);
    end if
    generate words $w \sim$ Multinomial(ϕ_z);
  end for
  choose the app category y according to multi-class logistic regression $y \sim LR(\frac{\exp(\eta_i^T \overline{z})}{1+\sum_{i=1}^{C-1}(\eta_i^T \overline{z})})$, $i \in [1, ..., C-1]$
end for

We utilize the mean field variational inference method [26] to estimate the parameters of ATM. The joint likelihood of generating the whole corpus $\mathbf{D}$ given the hyper-parameters is as follows:

$$P(\mathbf{D}|\alpha, \beta, \Omega, o) = \int \prod_{d=1}^{D} p(\theta_d|\alpha) \times \prod_{d=1}^{D} p(\theta'_d|\Omega) \times \prod_{d=1}^{D} \prod_{s=1}^{S_d} \sum_{b_{ds}} p(b_{ds}|o) \sum_{c_{ds}} p(c_{ds}|\theta'_d) \sum_{l_{ds}} p(l_{ds}|\pi_{dc_{ds}}) \sum_{z_{ds}} \left(p(z_{ds}|\theta_d)^{b_{ds}} p(z_{ds}|\theta_{d_{l_{ds}}})^{1-b_{ds}}\right) \times \prod_{d=1}^{D} \prod_{s=1}^{S_d} \prod_{n=1}^{W_{ds}} p(w_{dsn}|\phi_{z_{ds}}) \times \prod_{k=1}^{K} p(\phi_k|\beta) \times \prod_{d=1}^{D} p(y_d|\mathbf{Z}_d, \eta_{1:C-1}) d\theta_d. \tag{1}$$

We then minimize the KL-divergence between factorized variational posterior distribution and true posterior probability of the corpus by taking derivatives of the loss function with respect to variational parameters. The solution is listed as follows:

E-step:

$$\gamma_{dk} = \alpha_k + \sum_{s=1}^{S_d} \rho_{ds} \hat{\phi}_{dsk} + \sum_{d'=1}^{D} \sum_{c=1}^{C_{d'}} \sum_{l=1}^{L_{d'c}} I[l = d] \sum_{s=1}^{S_d} \psi_{d'scl} \cdot (1 - \rho_{d's}) \hat{\phi}_{d'sk}, \tag{2}$$

where γ is the posterior Dirichlet parameter corresponding to the representations of documents in the topic simplex.

$$\hat{\phi}_{dsk} = \exp\big\{\rho_{ds}(\Psi(\gamma_{dk}) - \Psi(\sum_{k=1}^{K} \gamma_{dk})) + (1 - \rho_{ds}) \sum_{c=1}^{C} \sum_{l=1}^{L_{dc}} \psi_{dscl} \sum_{n=1}^{W_{ds}} \big(\Psi(\hat{\lambda}_{kw_n}) - \Psi(\sum_{v=1}^{V} \hat{\lambda}_{kv}) + \frac{1}{N} \sum_{i=1}^{C-1} (\eta_{ik} y_i - \exp(\eta_{ik})/\xi_d)\big), \tag{3}$$

where $\hat{\phi}$ is the variational parameter corresponds to z and $\Psi(\cdot)$ is the digamma function, i.e., the first derivative of the log Gamma function.

$$\rho_{ds} = \big(1 + (X)^{-1}\big)^{-1}, \tag{4}$$

where ρ is the variational parameter corresponds to b and X is defined as follows:

$$X = \exp\{\sum_{k=1}^{K} \big(\Psi(\gamma_{dk}) - \Psi(\sum_{k}^{K} \gamma_{dk})\big) \hat{\phi}_{dsk} - \log o \sum_{k=1}^{K} \sum_{c=1}^{C} \sum_{l=1}^{L_{dc}} \psi_{dscl} \hat{\phi}_{dsk} \big(\Psi(\gamma_{d_l k}) - \Psi(\sum_{k}^{K} \gamma_{d_l k})\big) + \log o - \log(1 - o)\}. \tag{5}$$

$$\psi_{dscl} = \pi_{dcl} \exp\{(1 - \rho_{ds}) \sum_{k=1}^{K} \big(\Psi(\gamma_{d_l k}) - \Psi(\sum_{k}^{K} \gamma_{d_l k})\big) \hat{\phi}_{d_l sk}\}, \tag{6}$$

where ψ is the variational parameter corresponds to l.

$$\Delta_{dc} = \Omega_c + \sum_{s=1}^{S_d} v_{dsc}, \tag{7}$$

where Δ is the variational parameter corresponds to θ'.

$$v_{dsc} = \exp\{\Psi(\Delta_{dc}) - \Psi(\sum_{c=1}^{C} \Delta_{dc}) + \sum_{k=1}^{K} (1 - \rho_{ds}) \sum_{l'}^{L_{dc}} \psi_{dscl'} (\Psi(\gamma_{d_{l'} k}) - \Psi(\sum_{k=1}^{K} \gamma_{d_{l'} k})) \hat{\phi}_{d_{l'} sk} - 1\}, \tag{8}$$

where v is the variational parameter corresponds to c.

$$\xi_d = 1 + \frac{1}{N} \sum_{i=1}^{C-1} \sum_{k=1}^{K} \sum_{s=1}^{S_d} \hat{\phi}_{dsk} \exp(\eta_{ik}), \tag{9}$$

where ξ is the variational parameter corresponds to y.

M-step:

$$\hat{\lambda}_{kv} = \beta_v + \sum_{d=1}^{D} \sum_{s=1}^{S_d} \sum_{n=1}^{W_{ds}} I(w_{dsn} = v) \hat{\phi}_{dsk}, \tag{10}$$

where $\hat{\lambda}$ is the variational parameter corresponds to ϕ.

$$\eta_{ik} = \log \frac{\sum_{d=1}^{D} \sum_{s=1}^{S_d} y_{di} \hat{\phi}_{dsk} / S_d}{\sum_{d=1}^{D} \sum_{s=1}^{S_d} \hat{\phi}_{dsk} / (S_d \xi_d)}. \tag{11}$$

Through applying the ATM to the app corpus, each app is represented by a topic vector $(\theta_1, \theta_2, ..., \theta_n)$ where θ_k is

a real number that indicates the app's endorsement for the kth topic. The discovered topics $(\phi_1, \phi_2, ..., \phi_k)$ are different multinomial distributions over the terms in the corpus. In the following two sections, we discuss how to utilize the results of ATM for semantic-aware app snippet generation as well as semantic-aware app ranking.

4. APP SNIPPET GENERATION

In this section, we discuss the approach of generating high-quality app snippets. We propose three metrics (namely, centrality, formality and usefulness) in Subsections 4.1, 4.2 and 4.3. In Subsection 4.4, we proposed a greedy algorithm to generate app snippets and discuss its performance guarantee.

4.1 Centrality

Centrality assumes that the most central sentences in the app description give the necessary and sufficient amount of information related to the theme of the app description. In this metric, each sentence is represented by the TF-ISF[5] vectors of all words and the TF-ISF value of a term t is calculated as follows:

$$TF-ISF(t) = \sqrt{F_t} \times \big(\log(\frac{|S|}{|S_t|+1}) + 1\big), \tag{12}$$

where F_t is the frequency of t in the sentence, $|S|$ is the total number of sentences in the app description and $|S_t|$ is the number of sentences containing t. Finally, the centrality of a sentence s is obtained by averaging the cosine similarities between the present sentence and all other sentences.

$$Centrality(s) = \frac{\sum_{s' \in S \wedge s' \neq s} Sim(s, s')}{|S| - 1}. \tag{13}$$

4.2 Formality

The informal sentences usually contain subjective comments and are usually irrelevant to the app's functionality. Thus, a metric is needed to distinguish the formal sentences from the informal ones, since The formality metric assumes that formal sentences are more informative [10]. The non-deictic (ND) category of words, whose frequency is expected to increase with the formality of a text, includes the nouns, adjectives, prepositions and articles. The deictic category (D), whose frequency is expected to decrease with increasing formality of speech-styles, consists of the pronouns, verbs, adverbs, and interjections. The formality metric is obtained by adding up the frequencies of the formal categories, subtracting the frequencies of the deictic categories and normalize to a value between 0 and 1. Specifically, we design a normalized version of the formula proposed in [10] and calculate the formality of a sentence s as follows:

$$Formality(s) = \frac{(\sum_{x \in ND} freq(s_x) - \sum_{y \in D} freq(s_y) + 100)}{100 + length(s)}. \tag{14}$$

4.3 Usefulness

The usefulness metric evaluates whether or not a sentence discusses the description's theme from the perspective of topics [1]. To determine whether a sentence s is on-theme, we evaluate the probability that it is generated by the corresponding app's topic distribution. This metric is essentially different from centrality and formality, since this metric not only utilizes the local information but also incorporates the global semantics. We formulate the following equation to gauge the usefulness of the sentence s.

$$Usefulness(s) = \frac{(\sum_{t \in s} \sum_{k=1}^{K} P(t|\phi_k) \cdot \theta_k)}{length(s)}, \tag{15}$$

where θ_k is the app's endorsement for the kth topic. $length(s)$ is the number of terms in the sentences and provides length normalization so that sentences having different lengths are comparable.

4.4 App Snippet Generation Algorithm

In this subsection, we discuss an algorithm of generating the snippet for an app. Besides the three proposed metrics, we also use some conventional metrics that are intensively used in text summarization to evaluate the *quality* of a sentence. Overall, we utilize the following seven factors:

- Centrality.
- Formality.
- Usefullness.
- Whether starts with the app name.
- Cosine similarity with the app name.
- The sequence of the sentence in description.
- Sentence length.

The weight of each factor listed above is determined by the training data prepared by human judges (see Section 6). We then utilize a linear function to combine these factors and calculate the quality of the sentence s, $Quality(s)$, which is further normalized between to a real number between 0 and 1. App snippet generation aims to select a small set of sentences, which are representative of the app's theme and diverse to cover different facets of the app's features. Therefore, we formulate it as a maximization problem:

maximize:

$$OBJ(S')_{S' \in S} = \sum_{s \in S'} Quality(s) - \lambda \sum_{i,j \in S': i \neq j} Sim(i, j),$$

subject to:

$$\sum_{i \in S'} l_i \leq L.$$

where S is the set of all sentences in the app description. S' is the set of selected sentences, λ is a positive number determining the weight of redundancy penalty, l_i is the length of the ith sentence and L is the length limit. $Sim(i, j)$ evaluates the cosine similarity between sentences s_i and s_j. Finding the optimal solution of the maximization problem is NP-hard [16], thus, we propose a greedy approach of Algorithm 2 to solve it. The algorithm sequentially finds sentence x with the largest ratio of objective function gain to cost, the length of the sentence. If adding x increases the objective function value while not violating the length limit, it is then selected and otherwise bypassed. After the sequential selection, set G is compared to the singleton that is within

[5]TF is short for *term frequency* and the ISF is short for *inverse sentence frequency*.

Algorithm 2 App Snippet Generation Algorithm

$G = \emptyset$
$U = S$
while $U \neq \emptyset$ **do**
  $x \leftarrow \arg\max_{s \in U} \frac{OBJ(G \cup \{s\}) - OBJ(G)}{l_s}$
  if $\sum_{i \in G} l_i + l_x \leq L$ and $OBJ(G \cup \{x\}) - OBJ(G) \geq 0$ **then**
    $G \leftarrow G \cup \{x\}$
    $U \leftarrow U \backslash \{x\}$
  end if
end while
$s^* \leftarrow \arg\max_{s \in S, l_s \leq L} OBJ(s)$
return $G = \arg\max_{S \in \{\{s^*\}, G\}} OBJ(S)$

the length limit and has the largest objective value, and the larger of the two becomes the final output.

The last step of Algorithm 2 ensures that we are able to obtain a constant approximation factor if the objective function is monotone and submodular. If OBJ is monotone and submodular, it is easy to prove that the greedy algorithm has a near-optimal performance and a constant approximation ratio of $\left(1 - e^{-\frac{1}{2}}\right)$. However, OBJ is submodular but does not guarantee to be always monotone, we show that the greedy algorithm can work nearly-optimally with a bounded probability. Let S_L be the largest number of sentences of a feasible solution, the value of Sim is bounded and are independently identically distributed with mean μ and the value of $Quality(s)$ is independently identically distributed with mean μ'. Algorithm 2 works near-optimally with a probability of at least $1 - \exp\left\{\frac{-2(\mu' - \lambda(2S_L - 1)\mu)^2}{(1+\lambda(2S_L-1))^2} + \ln S_L\right\}$.

5. APP RANKING

In this section, we discuss how to rank the apps with respect to a search query. In Section 5.1, we discuss the features that are used to evaluate an app's static quality. In Sections 5.2 and 5.3, we discuss the TF-IDF score and the topic score in order to evaluate the app's relevance to a search query. Finally, in Section 5.4, we discuss the structure of the inverted index to support efficient online query processing.

5.1 Static Quality Score

Similar to the documents in general web search, each app also has their own static quality that is independent of any search query. In this subsection, we study the apps' static quality from three different perspectives: the apps' popularity, the corresponding developer's reputation and the apps' prestige in the app network.

5.1.1 App Popularity

We evaluate an app's popularity by the rating, the amount of users as well as the last 30 days installment. Among the three popularity factors, the rating needs to be further processed. App marketplaces usually use the raw average rating to evaluate the quality of the app. However, this indicator is easily skewed by a small number of ratings (or even a single rating) given to any particular app. Therefore, we use *Bayesian average* (BA) to incorporate the "wisdom of the crowd". The logic of Bayesian average is as follows. When an app has received many ratings, that data is considered more "reliable". When an app has received very few ratings, its rating should approximate the average rating for all apps. The equation of BA rating is given by:

$$BA = \frac{A_N \cdot A_R + Sum}{N + A_N}, \tag{16}$$

where N is the number of ratings given to this app, Sum is the sum of all ratings given to this app, A_N is is the average number of ratings for all apps and A_R is the average rating for all apps. In summary, we use the following three features to evaluate an app's popularity:

- BA Rating: the Bayesian average rating.
- User Amount: the original value.
- Last 30 Days Installments: the original value.

5.1.2 Developer Reputation

We now consider the static quality factors that originate from the app developers. It is straightforward that the quality of an app subjects to the expertise of the developers. Therefore, incorporating the developer's information help obtain a more objective perception of an app's quality. We include the developer's reputation in an implicit way, i.e, we do not compute a reputation score for each developer. Instead, we use Bayesian average to update the original rating, user amount and last 30 days installment. Equation (17) naturally incorporates the developer's information by considering all the apps that the developer has developed so far.

$$BA = \frac{\hat{A}_N \cdot \hat{A}_R + Sum}{N + \hat{A}_N}, \tag{17}$$

where the definitions of N and Sum are the same as Equation (16). $\hat{A}_N$ is is the average number of ratings, user amounts or last 30 days installments for all apps from the same developer and $\hat{A}_R$ is the average ratings, average user amounts or average last 30 days installments for all apps from the same developer.

Given that each developer has some expertise in developing a specific type of apps such as *Entertainment* or *Communication*, a strategy with even finer granularity is to calculate Bayesian average based on both developers and categories. In this case, the $\hat{A}_N$ in Equation (17) is the average number of ratings, user amounts or last 30 days installments for all apps from both the same developer and the same category as the present app. $\hat{A}_R$ is the average ratings, average user amounts or average last 30 days installments for all apps from the same developer and the same category as the present app. In summary, we use the following features to evaluate an app's static quality from the perspective of developer reputation:

- D-BA Rating.
- D-BA User Amount.
- D-BA Last 30 Days Installments.
- DC-BA Rating.
- DC-BA User Amount.
- DC-BA Last 30 Days Installments.

where D-BA denotes the Bayesian average based on the developers and DC-BA denotes the Bayesian average based on both the developers and the categories.

5.1.3 Link Prestige

Three app relation graphs are established from the three perspectives of *Same Developer*, *Also Viewed* and *Also Installed*. We apply the standard PageRank algorithm on each graph to obtain the app's link prestige. We get the following three PageRank values for an app:

- P_r^D: PageRank value obtained from the Same Developer graph.
- P_r^V: PageRank value obtained from the Also Viewed graph.
- P_r^I: PageRank value obtained from the Also Installed graph.

By combining all the 12 features from app popularity, developer reputation and link prestige by a linear function, we can obtain the static quality score of each app, which is further normalized by the maximum one in the corpus.

5.2 TF-IDF Score

Now we discuss how to evaluate the relevance of an app with regard to a search query through the traditional IR techinques. The TF-IDF score is based on the conventional term matching technique commonly used in IR. Specifically, we utilize the BM25F [5] model to evaluate the relevancy of an app a with respect to a query term t. The BM25F score of a with respect to t is calculated by Equation (18):

$$Score_{BM25F}(t,a) = IDF_t \cdot TF_{BM25F}(t,a), \tag{18}$$

$$IDF_t = \log(\frac{N}{N_t}), \tag{19}$$

where N is the number of apps in the corpus and N_t is the number of apps containing the term t.

$$TF_{BM25F}(t,a) = \sum_r w_r \cdot TF_{BM25F}(t,a,r), \tag{20}$$

where r contains the fields of app name and app description. w_r is the weight of the field r. Similar to [19], we assign $w_{name} = 3$ and $w_{description} = 1$.

$$TF_{BM25F}(t,a,r) = \frac{f_{t,a,r} \times (k_1+1)}{f_{t,a,r} + k_1 \times ((1-b_r) + b_r \times (l_{a,r}/l_r))}, \tag{21}$$

where $f_{t,a,r}$ is the frequency of term t is a's region r, $l_{a,r}$ is the length of a's region r and l_r is the average length of region r in the corpus. k_1 and b_r are set to be default value 1.2 and 0.75 respectively [5]. Given a query q, the normalized TF-IDF score of a is calculated as follows:

$$Score_{BM25F}(q,a) = \frac{\sum_{t \in q} IDF_t \cdot TF_{BM25F}(t,a)}{\sum_{t \in q} 4 \cdot IDF_t \cdot (k_1+1)}. \tag{22}$$

5.3 Topic Score

TF-IDF score evaluates the apps' relevancy through the *exact matching* of the query terms. However, many relevant apps only have terms that have similar semantic meaning with the query terms. Recall the example in the introduction, when the query term is *fight*, the apps that contain the term *combat* can also be good candidates for the results. Now we discuss how to utilize the topics discovered by ATM to retrieve these apps as well. We use the following equation to calculate the topic score of app a with respect to a query term t:

$$Score_{ATM}(t,a) = P(a|t) \propto P(t|a)P(a), \tag{23}$$

where

$$P(t|a) = \sum_{k=1}^{K} P(t|\phi_k) \cdot \theta_{ak}. \tag{24}$$

Without any prior knowledge, we consider $P(a)$ to be equal for all apps. Thus, with respect to q, the normalized topic score of a is as follows:

$$Score_{ATM}(q,a) = \frac{\sum_{t \in q} \sum_{k=1}^{K} P(t|\phi_k) \cdot \theta_{ak}}{length(q)}, \tag{25}$$

where $length(q)$ is the total number of terms in the query.

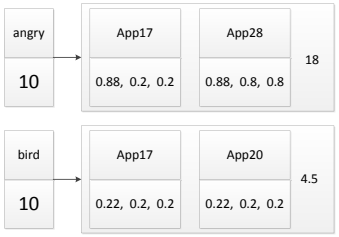
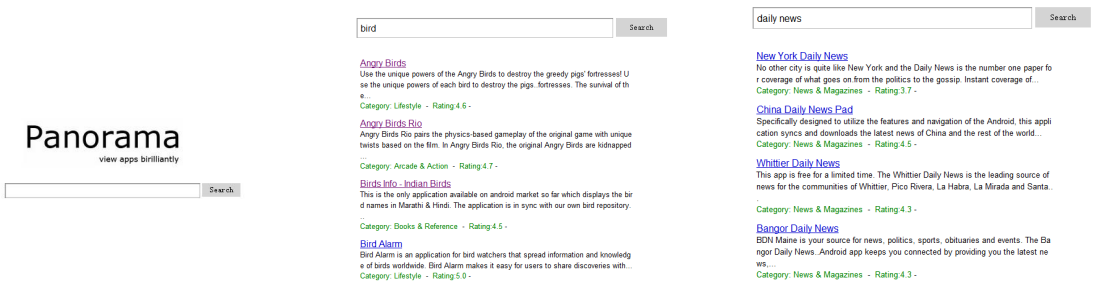


Figure 2: Example of the inverted index structure. The index contains the inverted lists of *angry* and *bird*. In the inverted list of *bird*, the IDF_{bird} is 10. Postings App17 and App20 form the first block. In App20, 0.22, 0.2 and 0.2 are TF_{BM25F}(bird, App20), $Score_{ATM}$(bird, App20) and $Quality$(App20), respectively.

5.4 Structure of Inverted Index

Search engines perform query processing based on the inverted index, which is a simple and efficient data structure that supports finding documents that contain a particular term. However, traditional inverted indices do not support the topic information. Although the query processing can be performed in a naive manner by aggregating the search results from traditional inverted indices and the topic information, this method would be very inefficient because significantly larger numbers of apps are needed to be retrieved in the intermediate results. We now discuss a new inverted index structure by incorporating the static quality score, TF-IDF score and topic score together. The new inverted index consists of many inverted lists (see Figure 2). Each inverted list is a list of postings and stores the IDF_t for the term t. Each posting takes the form of $(a_i, TF_{BM25F}(t,a_i), Score_{ATM}(t,a_i), Quality(a_i))$, where a_i is the app ID, $TF_{BM25F}(t,a_i)$ is the term frequency score, $Score_{ATM}(t,a_i)$ is the topic score and $Quality(a_i)$ is the app's static quality score. Similarly to the general web search engine, we assume that the query terms are disjunctive in default. We build the inverted index in the block-max paradigm [7] and utilize the local block-max WAND strategy [22] for query processing.

The important operation of efficient query processing is to estimate the upper bound score of the candidate document and skip the candidate document if it failed to be included



(a) Panorama Homepage (b) Panorama Search Result of *bird* (c) Panorama Search Result of *daily news*

Figure 3: User Interface of Panorama App Search Engine

in the Top-k results. This goal is achieved by grouping a number of documents as a *block*, which store the information about the upper bound of the impact of the postings in it. The scoring function can be rewritten as follows:

$$\begin{aligned} Score(q,a) &= w_1 \cdot Quality(a) \\ &+w_2 \cdot (\sum_{t\in q} \cdot Score_{BM25F}(t,a))/(\sum_{t\in q} 4 \cdot IDF_t \cdot (k_1+1)) \\ &+w_3 \cdot \sum_{t\in q} Score_{ATM}(t,a)/length(q) \\ \leq (\sum_{t\in q} &(w_1 \cdot 4 \cdot IDF_t \cdot Quality(d) + w_2 \cdot IDF_t \cdot \frac{TF_{BM25F}(t,a)}{k_1+1} \\ &+w_3 \cdot 4 \cdot IDF_{max} \cdot Score_{ATM}(t,a))/\sum_{t\in q} 4 \cdot IDF_t, \end{aligned} \tag{26}$$

where IDF_{max} is the largest IDF score in the inverted index. Specifically, we calculate the impact of a posting $(a, TF_{BM25F}(t,a), Score_{ATM}(t,a), Quality(a))$ in the inverted index by Equation(27). We then store the maximum impact of the postings in a block as the upper bound impact and store it as the meta information in the block.

$$\begin{aligned} w_1 \cdot 4 \cdot IDF_t \cdot Quality(d) + w_2 \cdot IDF_t \cdot \frac{TF_{BM25F}(t,a)}{k_1+1} \\ +w_3 \cdot 4 \cdot IDF_{max} \cdot Score_{ATM}(t,a) \end{aligned} \tag{27}$$

During query processing, the upper bound score of an app in the block can be dynamically estimated via $\frac{bound}{\sum_{t\in q} 4 \cdot IDF_t}$. Empirically, the strategy discussed above achieves a good performance in upper bound estimation. However, an app's upper bound can be underestimated in two cases: (1) when one inverted list ends up and the other inverted lists still have candidates. (2) when one app's static quality score is much larger than the block's combined score. Figure 2 shows an example of this scenario. Assume that the weights for TF-IDF score, topic score and static quality score be 0.5, 0.3 and 0.2, respectively. It is straightforward to obtain that the upper bounds of the first blocks of *angry* and *bird* are 18 and 4.5, respectively. When utilizing the two block upper bound to estimate a document's upper bound, we get $\frac{18+4.5}{40+40} = 0.28125$. However, the real score of App28 is $0.5 \times \frac{0.88\times 10}{4\times 10\times 2.2\times 2} + 0.3 \times \frac{0.8}{2} + 0.2 \times 0.8 = 0.305$. So App28's upper bound score is underestimated. This case happens when a document's static quality score is much larger than a block's combined score. App28's static quality score is $0.2 \times 0.8 = 0.16$. In contrast, the combined score of *bird*'s first block is 0.07625. Thus, we also store the maximum static score and maximum topic score in blocks to check whether this situation actually happens. If this happens, we use the maximum static score and maximum topic score to estimate the upper bound score, otherwise, we use the score given by Equation (27).

6. EXPERIMENTS

In this section, we present the experimental results. The experimental data are crawled from commercial app search engines such as Google Play[6], appgravity[7] and AppBrain[8] in March 2012. In Section 6.1, we present some topics discovered by ATM and quantitatively compare ATM with some existing topic models. In Section 6.2, we compare the proposed app snippets generation algorithm with open source software and several commercial app search engines. In Section 6.3, we compare the app ranking of the proposed framework with those generated by conventional IR models and several commercial app search engines. Finally, in Section 6.4, we evaluate the query processing efficiency of the proposed inverted index structure.

6.1 Evaluation of ATM

An informal but important measure of the success of probabilistic topic models is the plausibility of the discovered search topics. For simplicity, we use the fixed symmetric Dirichlet distribution like [8], which demonstrates good performance in our experiments. Hyperparameter setting is well studied in probabilistic topic modeling and is beyond the scope of this paper. Interested readers are invited to refer [27] to find a more detailed discussion. Some topics discovered by ATM are presented in Table 3. We can see that the discovered topics consist of semantically coherent terms and is effective to cover different facets of the apps' features.

We further quantitatively evaluate the performance of ATM by the metric of perplexity [21]. Perplexity is a measure of the ability of a model to generalize to unseen data. Better generalization performance is indicated by a lower perplexity. Through an extensive survey, we do not find any probabilistic models that are designed for analyzing apps, thus we carefully choose several state-of-the-art models that are general enough to be applied to app analysis. Specifically, we choose the LDA [3], JPMCC model [6], Link-LDA [17] and PLink-LDA [28] as the baselines. We compare ATM with

[6]https://play.google.com/store
[7]http://appgravity.com/.
[8]http://www.appbrain.com/.

Table 3: Examples of Topics Discovered by ATM.

Topic 1		Topic 2		Topic 3		Topic 4		Topic 5	
water	0.16628	battery	0.08196	aging	0.05714	messages	0.04268	sniper	0.07575
shower	0.06572	settings	0.02732	photo	0.04285	contacts	0.03658	game	0.06060
ooze	0.04814	saving	0.02185	camera	0.02857	send	0.03048	shooting	0.04545
flow	0.04757	mode	0.02185	face	0.02857	friends	0.03048	gun	0.04545
drop	0.02258	background	0.01639	mustache	0.02857	messenger	0.03048	camp	0.04545
splash	0.02155	power	0.01092	glasses	0.02857	chat	0.02439	score	0.03030
fluid	0.02152	screen	0.00546	wrinkles	0.01428	sms	0.02439	head	0.01515
ducky	0.02132	brightness	0.00546	skin	0.01428	notifications	0.01219	rifle	0.01515
rubber	0.02067	restore	0.00546	beard	0.01428	group	0.01219	weapons	0.01515
games	0.02032	consuming	0.00546	monocle	0.01428	voice	0.00609	reloaded	0.01515

the baselines by a ten-fold cross validation on a dataset of 10,000 apps and use Equation (28) to calculate the perplexity of each model.

$$Perplexity(\mathcal{M}) = (\prod_{d=1}^{D}\prod_{i=1}^{N_d} p(w_i|\mathcal{M}))^{\frac{-1}{\sum_{d=1}^{D}(N_d)}}, \quad (28)$$

where $\mathcal{M}$ is the model that is learned from the training process and w_i is the word in the document. Figure 4(a) shows the average perplexity for each model. We can observe that ATM achieves significantly lower perplexity, indicating that ATM provides a better fit for the app data than other models.

Like [29], we also use KL-divergence to evaluate the distinctiveness of the discovered topics. The larger the average KL-divergence is, the more distinct the search topics are. We show the average distance of term distributions of all pairs of search topics measured by KL-divergence in Figure 4(b). We find that the topics discovered by ATM show the highest KL-divergence than those of the baselines. By incorporating both the three kinds of link information and the category information, the word distributions in the topics discovered by ATM are more distinctive. The result indicates that ATM is superior in finding different facets of an app's theme.

6.2 App Snippet Quality

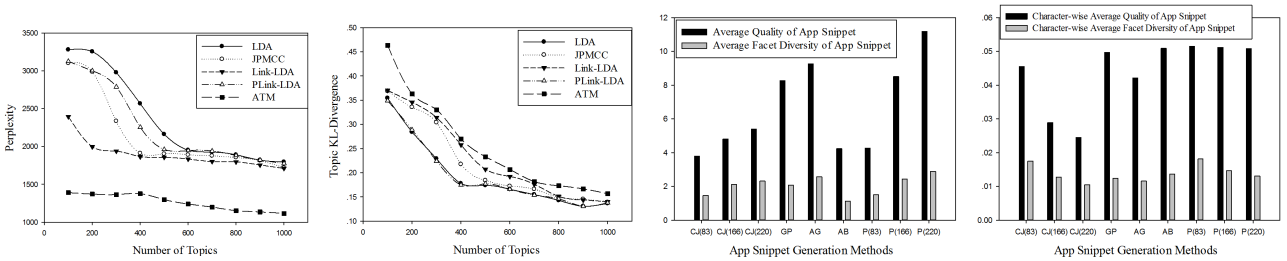
We compare the app snippet generation algorithm with the open source software Classifier4J [9] and three commercial app search engines: Google Play, appgravity and AppBrain. We choose the three app search engines because they are focused on searching Android apps and thus is comparable with the proposed framework. By the time of writing this paper, Google Play and appgravity simply utilize the first several characters (about 166 characters for Google Play and about 220 characters for appgravity) as an app's snippet. AppBrain seems to utilize a more sophisticated algorithm to generate the app snippet, which typically has a length of about 83 characters. In summary, we compare the following five methods in the experiments:

1. CJ: The summarizer algorithm implemented in Classifier4J. The number in the parentheses is the amount of characters in the snippet.
2. GP: App snippet of Google Play.
3. AG: App snippet of appgravity.
4. AB: App snippet of AppBrain.
5. P: Panorama's app snippet generation algorithm. The number in the parentheses is the amount of characters in the snippet.

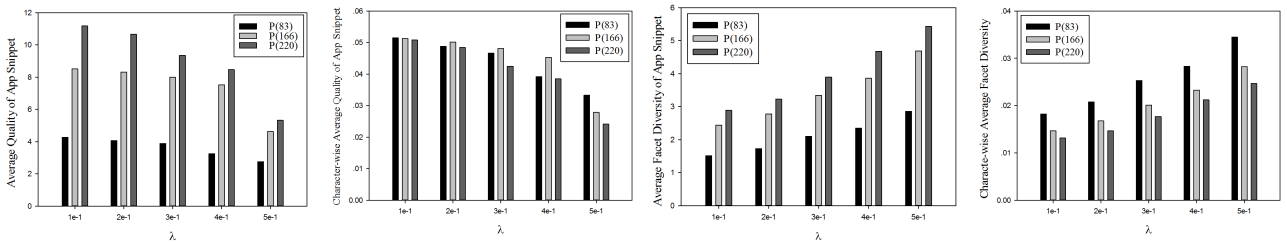
[9]classifier4j.sourceforge.net

The experimental data is prepared by crawling the information of the top 50 apps of 1,000 search queries from Google Play, appgravity and AppBrain. Then we select 1,000 apps which appear in all the three search engines' search results as the experimental data. The ground truth of the *sentence quality* is prepared by four human judges, who manually rate each sentence in the app description by a scale ranging from 0 to 5, where 0 indicates that the sentence is totally useless and 5 indicates that the sentence has very high quality and is representative of the app's real functionality. Besides assigning a quality score to each sentence, we also assign a a *facet label* to each sentence in order to indicate which facet this sentence covers. For example, if the sentences $s1$ and s_2 are about the app's UI design while the sentence s_3 is about the app's battery consumption, then s_1 and s_2 are assigned the same facet label while s_3 is assigned a different one. Out of the 1,000 manually prepared apps, we utilize 200 apps as the training data of RankSVM [12] to estimate the weight of each metric of sentence quality. The remaining 800 apps are further utilized to compare the proposed algorithm with the baselines in terms of the snippets' quality and facet diversity.

From the learned weights in Table 4, we observe that all the three proposed metrics have relatively large absolute values, showing that they are effective in estimating a sentence's quality. In our experiments, the snippet's quality is calculated as the sum of the quality score of the sentences in the snippet. From Figure 4(c), we can observe that the proposed app snippet generation algorithm achieves the best performance against the baselines which generate snippets of the same length. The result again verifies that the features we utilize are effective to reveal a sentence's real quality. Meanwhile, the greedy algorithm is effective to select the high-quality sentences as the snippet. The result also shows that by including the penalty on redundant information, the sentences in the resultant snippet of the proposed algorithm are essentially more diverse than those selected by the three commercial search engines. Since the snippets of the baselines are different in length, longer snippet tends to have a larger quality score and facet diversity. Therefore, we also calculate the two metrics on a character basis, i.e., dividing the original values of the two metrics by the amount of the characters in the snippet. The results of character-wise evaluation are presented in Figure 4(d). It is shown that Panorama also achieves the best performances when using the character-wise metrics. This result further verifies the



(a) Perplexity Evaluation (b) KL-Divergence Evaluation (c) Quality and Facet Diversity of App Snippet (d) Character-wise Quality and Facet Diversity of App Snippet



(e) Snippet Quality (f) Character-wise Snippet Quality (g) Facet Diversity (h) Character-wise Facet Diversity

Figure 4: Performance Comparison

superiority of the proposed app snippet algorithm in generating character-limited app snippets.

Table 4: Parameter Setting of Sentence Quality

Metrics	Weight
Centrality	1.0346936
Formality	0.5272671
Usefulness	1.0134872
Starts with Name	0.5581256
Cosine Similarity with Name	0.3146386
The sequence of sentence	-0.071374
The length of sentence	0.0063199

The parameter λ is the weight of the redundancy penalty in app snippet generation. Figure 4 (e) to (h) present the results of Panorama's snippet quality and facet diversity with different λ. The larger the value of λ, the lower the snippet quality and the higher the facet diversity. In our experiments, we set $\lambda = 0.1$, which achieves the highest character-wise quality and fairly good character-wise facet diversity.

6.3 App Ranking Quality

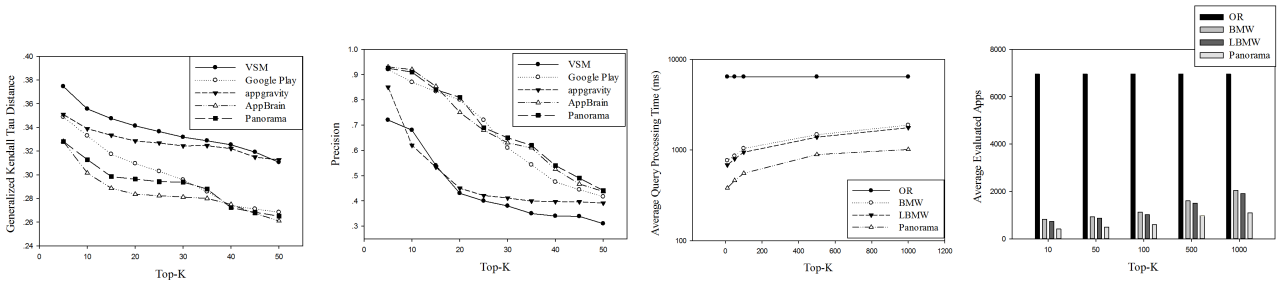
In this section, we evaluate the performance of Panorama in app rankings. The baselines we choose are as follows:

1. VSM: The traditional vector space model, using BM25F to calculate the relevance score.
2. Google Play: The app ranking of Google Play.
3. appgravity: The app ranking of appgravity.
4. AppBrain: The app ranking of AppBrain.

We build the VSM baseline and Panorama with a corpus of 2,2974 apps. In order to set the parameters of static quality score, four human judges are invited to rate 1,000 apps on a score scale of 0 to 5. Based on the manually rated apps, we train a RankSVM model to obtain the weights of different features (see Table 5).

We first prepare the *pseudo ground truth* by ranking aggregation. We utilize the Border's method [20] to aggregate the app ranking lists of the three commercial search engines for each search query. The apps in the aggregated rank list are ranked by their Border scores. We then utilize the aggregated ranking list as the *pseudo ground truth* of the corresponding query. Out of the total 1,000 search queries, we choose 200 of the aggregated ranking lists to train a RankSVM model to estimate the weights of static quality score, TF-IDF score and topic score. The remaining 800 search queries' ranking lists are further utilized to compare the Panorama with the baselines. We employ the generalized Kendall's Tau distance [15] to evaluate the correlation between the *pseudo ground truth* and the ranking lists generated by the methods under study. The larger the generalized Kendall's Tau distance, the smaller the correlation between the *pseudo ground truth* and the ranking list under study. The comparison result is presented in Figure 5(a). We observe that AppBrain demonstrates the highest correlation with the *pseudo ground truth* and achieves the lowest Kendall's Tau distance. From the Top-5 to Top-20 results, Panorama outperforms VSM, Google Play and appgravity. The result shows that Panorama can perform high-quality app rankings by combining the static quality score, TF-IDF score and topic score. The obtained weights of the three scores also verify that the static quality score and topic score are important features for determining the final rank of an app.

Since the *pseudo ground truth* is essentially biased towards the results of three commercial search engines, we also con-



(a) Ranking Quality (Pseudo Ground Truth)

(b) Ranking Quality (Human Judgment)

(c) Average Query Processing Time

(d) Average Evaluated Apps

Figure 5: Evaluation of App Ranking and Query Processing

duct a user-based evaluation similar to [13]. Each user is invited to conduct 50 searches on the four baseline search engines as well as the Panorama. The users are also asked to perform relevance judgment on the top 50 results for each query by filling in a score for each app to reflect the relevance of the app to the user's underlying need. The score indicates three levels of relevancy (*Good, Fair and Poor*). Apps rated as *Good* are considered relevant (positive samples), while those rated as *Poor* are considered irrelevant to the user's needs (negative samples). The apps rated as *Fair* are treated as unlabeled. Apps rated as *Good* (relevant apps) are used to compute Top-k precisions. The average Top-k precision of different approaches are presented in Figure 5 (b). From the result, we observe that Panorama significantly outperforms VSM and generates comparable or even better ranking quality with commercial search engines such as Google Play and AppBrain.

Table 5: Parameter Setting of App Static Quality

Features	Weight
BA-Rating	1.0343661
User Amount	2.3543382
Last 30 Days Installments	2.9078324
D-BA Rating	0.4830251
D-BA User Amount	0.6597734
D-BA 30 days installments	2.0066383
DC-BA Rating	0.4037651
DC-BA User Amount	0.6243585
DC-BA 30 days installments	1.9364556
P_r^D	0.0125456
P_r^V	0.2149537
P_r^I	0.2745393

6.4 Query Processing Efficiency

We proceed to evaluate the efficiency of the proposed inverted index. We prepare the baselines by a separate-index organization. Specifically, we first build an inverted index based on the TF-IDF information, i.e., constructing the traditional inverted index. Meanwhile, we build another inverted index based on the topic information. When a search query is submitted, we first retrieve the relevant documents from the two inverted indexes separately. Then we combine the two intermediate results with the static quality scores to generate the final Top-k results. Certainly, in this scenario, we need to retrieve more apps than k in the intermediate results, in order to guarantee that the scores of the real Top-k apps are correctly computed. Note that in this paper, we focus on returning the results that are exactly the same as the OR exhaustive query processing. When applying the baselines based on the separate-index organization, we gradually increase the number of apps retrieved in the intermediate results until the final result is the same as that of the exhaustive query processing, and then we gauge the corresponding method's efficiency in query processing. In summary, we compare the Panorama with the following query processing strategies. *OR*: The exhaustive disjunctive query processing [2]. *BMW*: The block-max WAND pruning strategy [7]. *LBMW*: The local block-max WAND strategy [22] on the separate-index organization.

We utilize 1,000 search queries whose length ranges from 1 to 10 terms to evaluate the average query processing time of the aforementioned methods. The results of average query processing time are presented in Figure 5(c). The results show that Panorama significantly outperforms the methods using the incremental index. The Panorama framework performs quite well even when k is equal to 1000, and the increase in consumed time is fairly moderate. The result verifies that simply adding another new inverted index based on the topic information is not efficient enough for online query processing, since the tedious two-phase computing paradigm consumes much time. The comparison results of average evaluated documents are presented in Figure 5(d). We observe that considerable performance gain can be obtained from Panorama's query processing strategy. The result indicates that the proposed method for document upper bound score estimation can achieve a large amount of skipping. Thus, the Panorama evaluates the least number of apps to generate the final Top-k results, indicating that more documents are effectively skipped on the new index structure.

7. CONCLUSION

The rapid growth rate of apps calls for a need of developing effective app search engines. In this paper, we propose a semantic-aware app search framework named Panorama to tackle two challenging problems in existing app searching: generating high-quality app snippet and generating good app ranking with respect to search queries. Panorama uses the App Topic Model (ATM) to discover the latent semantics from the app corpus. Based on the discovered semantics we propose a greedy algorithm with a bounded performance guarantee to generate concise and informative app snip-

pets. Panorama goes beyond the conventional term matching techniques and extends the horizon of app search by incorporating the latent semantics for app ranking. In order to support efficient query processing, a new index structure consisting of static app quality, TF-IDF score and topic score are further proposed. Extensive experiments on a large-scale dataset verify the effectiveness of the Panorama framework, which demonstrates superior performance against some commercial app search engines with respect to different metrics.

8. ACKNOWLEDGEMENTS

This work is partially supported by GRF under grant numbers HKUST 617610 and 618509. We are grateful to Heng Wang and Hao Li for the help with the experiments. We also wish to thank the anonymous reviewers for their comments.

9. REFERENCES

[1] J. Allan, R. Gupta, and V. Khandelwal, *Topic models for summarizing novelty*, ARDA Workshop on Language Modeling and Information Retrieval. Pittsburgh, Pennsylvania, 2001.

[2] V. Anh and A. Moffat, *Structured index organizations for high-throughput text querying*, String Processing and Information Retrieval, Springer, 2006, pp. 304–315.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, *Latent dirichlet allocation*, The Journal of Machine Learning Research (2003).

[4] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, *Efficient query evaluation using a two-level retrieval process*, Proceedings of the CIKM conference, 2003.

[5] S. Buttcher, C. Clarke, and G. V. Cormack, *Information retrieval: Implementing and evaluating search engines*, The MIT Press, 2010.

[6] D. Cohn and T. Hofmann, *The missing link-a probabilistic model of document content and hypertext connectivity*, Advances in neural information processing systems (2001), 430–436.

[7] S. Ding and T. Suel, *Faster top-k document retrieval using block-max indexes*, Proceedings of the SIGIR conference, 2011.

[8] T. L. Griffiths and M. Steyvers, *Finding scientific topics*, Proceedings of the National Academy of Sciences of the United States of America **101** (2004).

[9] A. Gruber, M. Rosen-Zvi, and Y. Weiss, *Latent topic models for hypertext*, Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, 2008.

[10] F. Heylighen and J. M. Dewaele, *Formality of language: definition, measurement and behavioral determinants*, Interner Bericht, Center Leo Apostel, Vrije Universiteit Brussel (1999).

[11] D. Jiang, J. Vosecky, K.W.T. Leung, and W. Ng, *G-wstd: A framework for geographic web search topic discovery*, Proceedings of the 21st ACM international conference on Information and knowledge management, ACM, 2012, pp. 1143–1152.

[12] T. Joachims, *Optimizing search engines using clickthrough data*, Proc. of the SIGKDD Conference, 2002.

[13] K. W. T. Leung, D. L. Lee, and W. C. Lee, *Personalized web search with location preferences*, Data Engineering (ICDE), 2010 IEEE 26th International Conference on, IEEE, 2010, pp. 701–712.

[14] H. Lin and J. Bilmes, *A class of submodular functions for document summarization*, The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT), Portland, OR, June, 2011.

[15] J. Mazurek, *Evaluation of ranking similarity in ordinal ranking problems*, Acta academica karviniensia, 119–128.

[16] R. McDonald, *A study of global inference algorithms in multi-document summarization*, Advances in Information Retrieval (2007), 557–564.

[17] R. Nallapati and W. Cohen, *Link-plsa-lda: A new unsupervised model for topics and influence of blogs*, International Conference for Weblogs and Social Media, 2008.

[18] J. Neto, A. Freitas, and C. Kaestner, *Automatic text summarization using a machine learning approach*, Advances in Artificial Intelligence (2002), 205–215.

[19] J. R. Perez-Aguera, J. Arroyo, J. Greenberg, J. P. Iglesias, and V. Fresno, *Using bm25f for semantic search*, Proceedings of the 3rd International Semantic Search Workshop, ACM, 2010, p. 2.

[20] M. E. Renda and U. Straccia, *Web metasearch: rank vs. score based rank aggregation methods*, Proceedings of the 2003 ACM symposium on Applied computing, ACM, 2003, pp. 841–846.

[21] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, *The author-topic model for authors and documents*, Proceedings of the UAI conference, 2004.

[22] D. Shan, S. Ding, J. He, H. Yan, and X. Li, *Optimized top-k processing with global page scores on block-max indexes*, Proceedings of the WSDM conference, 2012.

[23] C. Shen and T. Li, *Multi-document summarization via the minimum dominating set*, Proceedings of the 23rd International Conference on Computational Linguistics, Association for Computational Linguistics, 2010, pp. 984–992.

[24] T. Strohman and W. B. Croft, *Efficient document retrieval in main memory*, Proceedings of the SIGIR conference, 2007.

[25] R. Varadarajan and V. Hristidis, *A system for query-specific document summarization*, Proceedings of the 15th ACM international conference on Information and knowledge management, ACM, 2006, pp. 622–631.

[26] M. J. Wainwright and M. I. Jordan, *Graphical models, exponential families, and variational inference*, Foundations and TrendsÂő in Machine Learning **1** (2008), no. 1-2, 1–305.

[27] H. M. Wallach, *Structured topic models for language*, Unpublished doctoral dissertation, Univ.of Cambridge (2008).

[28] H. Xia, J. Li, J. Tang, and M. F. Moens, *Plink-lda: Using link as prior information in topic modeling*, Database Systems for Advanced Applications, Springer, 2012, pp. 213–227.

[29] Z. Yin, L. Cao, J. Han, C. Zhai, and T. Huang, *Geographical topic discovery and comparison*, Proceedings of the WWW conference, 2011.