ADI: Towards a Framework of App Developer Inspection

Kai Xing, Di Jiang, Wilfred Ng, Xiaotian Hao

Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{kxing,dijiang,wilfred,xhao}@cse.ust.hk

Abstract. With the popularity of smart mobile devices, the amount of mobile applications (or simply called apps) has been increasing dramatically in recent years. However, due to low threshold to enter app industry, app developers vary significantly with respect to their expertise and reputation in the production of apps. Currently, there is no well-recognized objective and effective means to profile app developers. As the mobile market grows, it already gives rise to the problem of finding appropriate apps from the user point of view. In this paper, we propose a framework called *App Developer Inspector* (ADI), which aims to effectively profile app developers in aspects of their expertise and reputation in developing apps. ADI is essentially founded on two underlying models: the *App Developer Expertise* (ADE) model and the *App Developer Reputation* (ADR) model. In a nutshell, ADE is a generative model that derives the latent expertise for each developer and ADR is a model that exploits multiple features to evaluate app developers' reputation. Using the app developer profiles generated in ADI, we study two new applications which respectively facilitate app search and app development outsourcing. We conduct extensive experiments on a large real world dataset to evaluate the performance of ADI. The results of experiments demonstrate the effectiveness of ADI in profiling app developers as well as its boosting impact on the new applications.

Keywords: App Developer, Profiling, App Searching, App Development Outsourcing

1 Introduction

Nowadays, more and more people use smart phones as their primary communication tools. This trend leads to the booming of the mobile app market, which is estimated to reach 25 billion US$ by 2015 [1]. In the face of the lucrativeness of the app market, many people plunge themselves into developing mobile apps.

[1] http://www.prnewswire.com/news-releases/marketsandmarkets-world-mobile-applications-market-worth-us25-billion-by-2015-114087839.html

However, app development has relatively low threshold to enter and thus either individual amateurs or well-organized studios can release their apps to the market. In light of the huge discrepancy among app developers, we propose a formal framework that can effectively evaluate and comprehensively analyze app developers. The study and development of this framework can not only facilitate better app quality control but also promote new app development in the app ecosystem.

In this paper, we propose a framework named *App Developer Inspector* (ADI) for profiling app developers. ADI mainly consists of two underlying models, the *App Developer Expertise* (ADE) model and the *App Developer Reputation* (ADR) model, respectively profiling app developers' expertise and reputation. ADE is a novel generative model to derive app developers' expertise. ADR, which is a RankSVM-based model, undertakes the function of evaluating app developers' reputation. The expertise reveals which kind of app functionalities (such as sports games, communication applications and so on) an app developer is expert in. The reputation indicates a developer's overall trustworthiness and proficiency in developing high quality apps.

In order to demonstrate the use of app developer profiles obtained from ADI, we study the following two applications.

- **Facilitate App Search** App developers' reputation and expertise are both significant factors in developing effective app search engine. There exist many "Look-Alike" apps having similar names and descriptions with the high quality ones[5]. When users search for a popular high quality app, app search engines sometimes rank the "Look-Alike" apps high in the searching results without taking app developers' reputation into consideration.
- **App Development Outsourcing** App developer profiles are also useful in app development outsourcing. With the information of app developers' expertise, employers[2] are able to find proper developers to undertake app development.

We conduct exhaustive experiments on a large real world dataset to show the effectiveness of ADE and ADR. The results of experiments demonstrate that ADE performs well in generating app developers' expertise and ADR shows good performance in evaluating app developers' reputation. We also demonstrate the boosting impact of app developer profiles in facilitating app search and app development outsourcing by detailed experimentation.

In summary, the main contributions of this paper are as follows:

- To the best of our knowledge, this work is the first to systematically study how to profile app developers. We develop a new framework, *App Developer Inspector* (ADI), which seamlessly integrates multiple information sources to derive app developers' expertise and reputation.

[2] In this work, we refer companies or individuals who want to outsource app development as the employers.

- We conduct comprehensive experiments to verify the effectiveness of the ADI framework. The experimental results show that ADE effectively generates app developers' expertise in fine granularity and ADR objectively evaluates app developers' reputation.
- We show how to apply app developers' reputation in facilitating app search and verify the improvement by exhaustive experiments. For app development outsourcing, we propose a new method BMr-BR to recommend quality developers.

The rest of the paper is organized as follows. In Section 2, we review the related literature. In Section 3, we provide an overview of the *App Developer Inspector* framework. In Section 4, we propose ADE model to obtain app developers' expertise. In Section 5, we evaluate app developers' reputation with the ADR model. In Section 6, we introduce the applications of app developer profiles in facilitating app search and app development outsourcing. In Section 7, we present the experimental results. Finally, the paper is concluded in Section 8.

2 Related Work

Our framework involves a spectrum of techniques that are briefly discussed as follows.

Smart Phone App There are some works on smart phone apps but most of them focused on app security. In [25], a systematic study was presented to detect malicious apps on both official and unofficial Android Markets. Alazab *et al.* [1] used the Android application sandbox Droidbox to generate behavioral graphs for each sample and these provided the basis of development of patterns to aid in identifying malicious apps. X.Wei *et al.* [24] described the nature and sources of sensitive data, what malicious apps can do to the data, and possible enterprise solution to secure the data and mitigate the security risks. Di *et al.* [11] proposed a framework of utilizing semantic information for app search.

Topic Modeling In recent years, topic modelling is gaining momentum in data mining. Griffiths *et al.* [6] applied Latent Dirichlet Allocation (LDA) to scientific articles and studied its effectiveness in finding scientific topics. There follow more topic models that are proposed to handle the problems of document analysis that exist in specific domains. Kang *et al.* [13] proposed a topic-concept cube which supports online multidimensional mining of query log. Mei *et al.* [17] proposed a novel probabilistic approach to model the subtopic themes and spatiotemporal theme patterns simultaneously. Some recent work on query log analysis also studied the impact of temporal and spatial issues. Ha-Thuc *et al.* [7] proposed an approach for event tracking with emphasis on scalability and selectivity. Di *et al.* [9],[10] also studied the spatial issues in web search data with topic modeling. To the best of our knowledge, our work is the first one to systemically study how to utilize topic modeling to profile app developers' expertise.

Learn to Rank Our work is related to learning to rank techniques, which is a intensively studied area in information retrieval and machine learning. RankSVM was proposed in [12] to optimize search engines via clickthrough data. In [22] and[8], RankSVM had been successfully applied to identify quality tweets on the social network *Twitter* and used social network user profile to personalize web search result.

Expert Finding The application of app development outsourcing has some similarities with expert finding. Maarten de Rijke *et al.*[2] proposed two models to search experts on a given topic from an organization's document repositories. Although there are some similarities between expert finding and app development outsourcing, e.g. they all aim to find "expert", the difference between them is fundamental. Two models in [2] focused on mining experts from corporation documents while our app development outsourcing is intended to recommend proper and quality app developers.

3 Overview of ADI

In this section, we provide a general overview of the ADI framework, including its architecture and a glance of app developer profiles.

3.1 Architecture of ADI

As Figure 1 shows, there are two models, ADE and ADR, to generate app developer profiles. Before applying ADR, We first extract some features, such as popularity, good ratio, web site quality and so on, from multiple information sources, i.e., information from apps, users and developers. Then ADR model generates app developers' reputation on basis of the extracted features. Meanwhile, a novel generative model ADE is employed to compute app developers' expertise from app descriptions and categories. Finally, an app developer profile that consists of app developers' expertise and reputation is generated.

3.2 App Developer Profiles

In ADI, app developer profiles consists of two components, which are respectively app developers' reputation and app developers' expertise.

The first component, app developers' reputation, which indicates app developer's proficiency and trustworthiness in developing apps, is represented by a real value. We apply ADR to compute this value and higher the value is, higher reputation a developer achieves.

The other component, app developers' expertise, depicts functionality-based expertise of developers in app development. In app developers' expertise are also two subcomponents. One is a series of expertises, each of which is represented by a set of keywords. These keywords are generated by our ADE model to

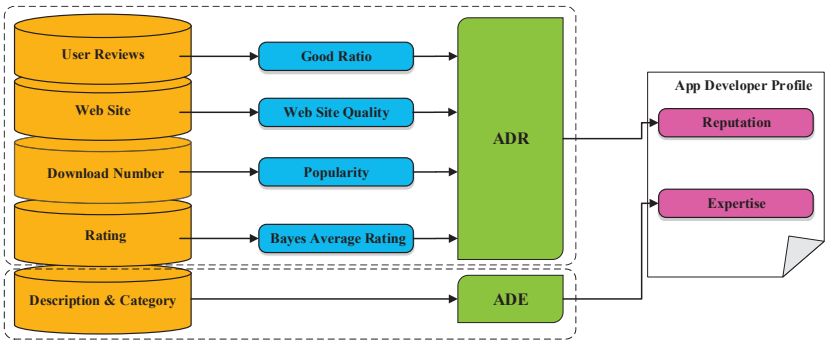


Fig. 1. Architecture of App Developer Inspector (ADI)

characterize a certain expertise. For each expertise, a proficiency is given to indicate how proficient a developer is in the expertise. Take Rovio (a famous game studio) as an example, in Figure 2 expertise 1 is represented by "game, birds, war..." and the corresponding proficiency is 0.7. The underlying reason of including this expertise part is that an app developer generally has more than one set of expertises and they may be in different proficiency level as well.

Fig. 2. Abstract Presentation of Rovio's profile

4 App Developer Expertise Model

In this section, we describe *App Developer Expertise* (ADE) model that derives the app developers' expertise in app development. The ADE aims to profile each developer's expertise in a concise and flexible way. We assume that each app developer has a Multinomial *expertise* (or in the metaphor of LDA, a topic)

distribution. We first group the app descriptions of the same developer as a document. Then, we filter out the non-informative words according to a stopword list provided in [14]. An interesting phenomenon in the app corpus is that developers are actually have implicit *links*, which can be obtained by analyzing the download records of the users. For example, if app a_1 developed by developer d_1 and app a_2 developed by developer d_2 are both downloaded by the same user, we create a link between d_1 and d_2. We utilize a $D \times D$ matrix M to store the link information and the entry $M[i, j]$ is computed by the number of times that i's apps have been downloaded with j's apps.

The generative process of this model is illustrated in Algorithm 1 and the notation used is summarized in Table 1.

Algorithm 1 Generative Process of ADE

for each topic $k \in 1, ..., K$ **do**
  draw a word distribution $\phi_k \sim$ Dirichlet(β);
  draw a category distribution $\phi'_k \sim$ Dirichlet(δ);
end for
for each document $d \in 1, ..., D$ **do**
  draw d's topic distribution $\theta_d \sim$ Dirichlet(α)
  sample a linked developer d_i with proportion to link weight of $l(d, d_i)$, then draw a document specific distribution θ_{d_i};
  combine θ_d and θ_{d_i} by tuning parameter λ to generate a document distribution θ;
  for each sentence $s \in d$ **do**
    choose a topic $z \sim$ Multinomial(θ);
    generate words $w \sim$ Multinomial(ϕ_z);
    generate the category $c \sim$ Multinomial(ϕ'_z);
  end for
end for

We aim to find an efficient way to compute the joint likelihood of the observed variables with hyperparameters:

$$P(\mathbf{w}, \mathbf{z}|\alpha, \beta, \delta, \lambda, \mathbf{l}) = P(\mathbf{w}|\mathbf{z}, \beta)P(\mathbf{c}|\mathbf{z}, \delta)P(\mathbf{z}|\alpha.\lambda, \mathbf{l}). \tag{1}$$

The probability of generating the words is given as follows:

$$P(\mathbf{w}|\mathbf{z}, \beta) = \int \prod_{d=1}^{D} \prod_{s=1}^{S_d} \prod_{i=1}^{W_{ds}} P(w_{dsi}|\phi_{z_{ds}})^{N_{dsw_{dsi}}} \prod_{z=1}^{K} P(\phi_z|\beta) d\Phi. \tag{2}$$

The probability of generating the categories is given as follows:

$$P(\mathbf{c}|\mathbf{z}, \beta) = \int \prod_{d=1}^{D} \prod_{i=1}^{S_d} P(c_{di}|{\phi'}_{z_{di}}) \prod_{z=1}^{K} P({\phi'}_z|\beta) d\Phi. \tag{3}$$

After combining the formula terms, we apply Bayes rule and fold terms into the proportionality constant, the conditional probability of the kth topic for the ith sentence is defined as follows:

$$P(z_i = k|\mathbf{z}_{-\mathbf{i}}, \mathbf{w}, \mathbf{l}, \alpha, \beta, \lambda) \propto$$
$$\frac{(1-\lambda)C_{dk}^{DK} + \lambda C_{lk}^{DK} + \alpha_k}{\sum_{k'=1}^{K}((1-\lambda)C_{dk'}^{DK} + \lambda C_{lk'}^{DK} + \alpha_{k'})} \frac{C_{kc}^{KC} + \delta_c}{\sum_{c'=1}^{C}(C_{kc}^{KC} + \delta_{c'})} \tag{4}$$
$$\frac{\Gamma(\sum_{w=1}^{W}(C_{kw}^{KW} + \beta_w))}{\Gamma(\sum_{w=1}^{W}(C_{kw}^{KW} + \beta_w + N_{iw}))} \prod_{w=1}^{W} \frac{\Gamma(C_{kw}^{KW} + \beta_w + N_{iw})}{\Gamma(C_{kw}^{KW} + \beta_w)}$$

where C_{lk}^{DK} is the number of sentences that are assigned topic k in document l, which is a randomly sampled document that is linked by the document d.
After processing the app developers by the proposed model, the ith developer's profile is represented by a search topic vector $(\theta_{i1}, \theta_{i2}, ..., \theta_{in})$, where θ_{ik} is a real number that indicates the ith user's endorsement for the kth search topic. The value of θ_{ik} is computed as follows:

$$\theta_{ik} = \frac{C_{dk}^{DK} + \alpha_k}{\sum_{k'=1}^{K}(C_{dk'}^{DK} + \alpha_{k'})}. \tag{5}$$

Table 1. Notations Used in the ADI Framework

Parameters	**Meaning**	**Parameters**	**Meaning**
D	the number of documents	λ	a parameter controlling the influence of the linked document
K	the number of topics	z_i	the topic of word i
z	a topic	$\mathbf{z_{-i}}$	the topic assignments for all words except word i
w	a word	$\mathbf{w}$	word list representation of the corpus
θ	multinomial distribution over topics	C_{kc}^{KC}	the number of times that c is assigned topic k
ϕ	multinomial distribution over words	C_{lk}^{DK}	the number of words assigned to topic k in the linked document l
ϕ'	multinomial distribution over categories	δ	Dirichlet prior vector for ϕ'
α	Dirichlet prior vector for θ	C_{dk}^{DK}	the number of sentences that are assigned topic k in document d
β	Dirichlet prior vector for ϕ	C_{kw}^{KW}	the number of times that w is assigned topic k
$\mathbf{l}$	link list		

We utilize the generative model to mine an individual developer's expertise of a specific field. Compared with the category information, the topics whose amount can be determined by the users represent the developer characteristics with finer granularity. Note that the topic amount K can be customized and thus, it strikes a good balance between efficiency and granularity. Note that the method proposed here is potentially scalable to very large datasets. For example, the Gibbs sampling is scaled to run very large sized datasets in [18].

5 App Developer Reputation Model

In this section, we introduce *App Developer Reputation* (ADR) model to evaluate app developers' reputation. Before elaborating how ADR works, we define app developers' reputation as the overall trustworthiness and proficiency in developing apps. ADR is model based on RankSVM[12] and works as follows. It first ranks each developer using a series of extracted features, i.e., popularity, rating and so forth. Then reputation of each developer can be calculated according to the generated ranking.

5.1 App Developer Reputation Generation

The generation process of ADR is carried out in three steps. We first input some pairwise training instances into RankSVM and get a generated rank model. Next, we utilize this rank model to rank a set of developers. Finally, the reputation of the developer i is given by:

$$R_i = 1 - \frac{rank_i}{N}, \tag{6}$$

where $rank_i$ is the ranking of developer i, N is the total number of developers.

5.2 Extracted Features

There are four features, popularity, rating, web site quality, and good ratio, used in RankSVM. We now present the construction of these features from multiple information sources.

App Popularity The popularity of an app is evaluated by the number of times that the app has been downloaded[5]. To compute a developer's *app popularity*, we first compute the popularity of an app as follows:

$$p_j = log(N_j). \tag{7}$$

Where N_j denotes the downloaded number of app j. Then the *app popularity* feature of a developer i is defined as follows:

$$Pop_i = \frac{\sum_{j \in A(i)} p_j}{|A(i)|}, \tag{8}$$

where $A(i)$ is the collection of apps developed by the developer i.

Bayes Average Rating The more ratings are given to an app, the more reliable the average of ratings is to reflect the app quality. Otherwise, app rating may be a misleading indicator. Here we use *Bayes average rating* to represent this intuition:

$$Br_j = \frac{N_{av} \cdot r_{av} + r_j \cdot N_j}{N_j + N_{av}}, \tag{9}$$

where N_{av} and r_{av} are respectively the average number of rating from users and average rating over all apps, r_j and N_j respectively denote the raw rating and the number of rating for app j.

If number of rating for an app is much smaller than the average number of rating over all apps, *Bayes average rating* is close to the average rating over all apps. Otherwise, *Bayes average rating* approximately equals app's raw rating. This property makes *Bayes average rating* a more reliable indicator.

Considering a developer may produce more than one apps, we use the average *Bayes average rating* of all the apps developed by the developer as the rating feature. Let $A(i)$ contain all the apps developed by developer i, rating feature is computed as follows:

$$\overline{Br_i} = \frac{1}{|A(i)|} \sum_{j \in A(i)} Br_j. \tag{10}$$

Web Site Quality Good developers usually have their own web sites where they post information about their app products. In this case, these web sites can be an important auxiliary information for evaluating app developer reputation. Here we define *web site quality* feature of a developer as the content relevance between the developers' web sites and app development or app products. The process of extracting *web site quality* feature is done in two steps. First, we collect a corpus of words related to app development and app products. Then we compute cosine similarity in vector space model [21] between developers' web sites and the corpus collected in first step. This cosine similarity is considered as *web site quality*. For developers not having their own web sites, we simply set their *web site quality* to 0.

Good Ratio In app marketplace such as Google Play, every app is open for app users to comment. Therefore, we can obtain a general user opinion of apps by mining the user reviews. Here we present *good ratio* feature, which is the proportion of positive reviews among all reviews. Considering that app user reviews are very short texts similar to tweets, we adapt a two-step SVM classifier model proposed by [3] to conduct sentiment analysis on app user reviews. The first step aims to distinguish subjective reviews from non-subjective reviews through a subjectivity classifier. Then we further classify the subjective reviews into positive and negative reviews, namely, polarity detection. The features used in these two SVM are word *meta features* in app user reviews.

Meta Features For a word in app user reviews, we map it to its part-of-speech using a part-of-speech dictionary[3]. In addition to POS tags, we also map the word to its prior subjectivity and polarity. The prior polarity is switched from positive to negative or from negative to positive when a negative expression, e.g., "don't", "never" precedes the word.

[3] The POS dictionary is available at: http://wordlist.sourceforge.net/pos-readme

6 Applications

In this section, we show the use of app developer profiles in facilitating app search and app development outsourcing.

6.1 Facilitate App Search

In this application, rather than exploring the background rank schema in existing app search engines, we turn to rank aggregation which deals with problem of combining the result lists returned by multiple search engines. The rank aggregation method we utilize here is Borda Count[19], which is a rank based aggregation method.

We apply app developer profiles in facilitating app search in the following way. We first rank apps by their developers' reputation and get a ranking list, which we call reputation ranking list here. Then we utilize Border's method to aggregate the ranking list returned by existing app search engine and the reputation ranking list. The aggregated ranking list is considered as the ranking result after applying app developer profiles in app search. The experiment in Section 7.3 shows this strategy improves the search quality of existing app search engines.

6.2 App Development Outsourcing

We consider the following scenario in app development outsourcing. Given a detailed description of desired app, which may contain the functionalities and the interface design, we would like to identify and recommend proper app developers to users. Let us call this app development outsourcing problem. Candidate Model and the Document Model proposed in [2] can be applied to app development outsourcing after some adaptation and are used as two baselines in our experiments. We now explain the details of the adaptation of the two models as follows: in the app development outsourcing scenario, each app description is considered as a document and the description of wanted app is referred to as query.

However, candidate model and document model merely take relevance between the given query and existing app descriptions into consideration, which can only recommend developers who have developed similar apps with the wanted one but can't guarantee that the recommended developers are all proficient. To tackle this defect, we propose a BM25 and reputation based recommendation (*BMr-BR*) methods to recommend excellent and experienced developers.

BMr-BR *BMr-BR* not only considers the relevance between the given query and existing app descriptions but also takes into app developer reputation. In this way, the recommended developers can be guaranteed to be experienced and proficient. The ranking function to recommend developrs in *BMr-BR* is as follows:

$$R \cdot \sum_i score(T_i, q), \tag{11}$$

where T_i is the key words set of Expertise i generated by ADE, q is the given query, R is developer's reputation.

The value $score(T_i, q)$ is defined as:

$$score(T_i, q) = \sum_{j=1}^{n} IDF(t_j)\alpha_i \frac{n(t_j, T_i)(k_1 + 1)}{n(t_j, T_i) + k_1(1 - b + b\frac{|T_i|}{avgdl})}, \tag{12}$$

where t_j is a term in q, α_i is developer's proficiency in expertise i, $n(t_j, T_i)$ denotes the frequency of t_j in T_i, k_1 and b are free parameters. Usually, $k_1 \in [1.2, 2.0]$ and $b = 0.75$ according to [15].

7 Experiments

7.1 Experiment Setting Up

In this paper, we select the official Android marketplace, Google Play, as the core information source of apps and developers.

We collected a total of 533,740 apps, which accounts for 82% of the whole apps on Google Play. From the webpage of an app on Google Play, we can obtain detailed information about this app such as the description, rating, download number and user reviews. Besides, the *URLs* of developer's web sites (if any) can be also accessed from Goog Play. We use these *URLs* to collect app developer's web sites.

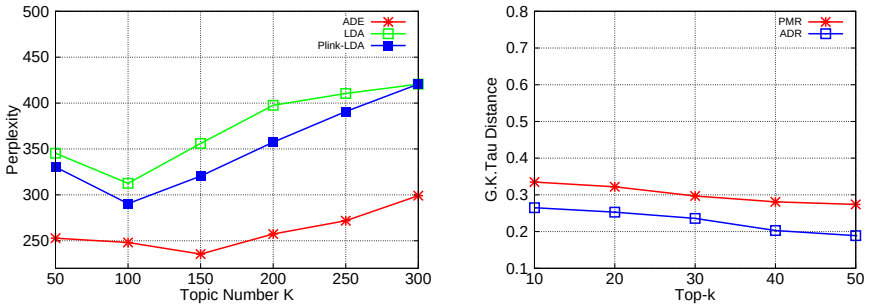
7.2 Evaluation of ADE and ADR

Evaluation of ADE 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 [6], 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 [23] for further details.

Currently, very few probabilistic topic models are proposed to analyze app developers, thus it is hard to find counterparts for the proposed one. Thus, we select Latent Dirichlet Allocation (LDA) [4] and Pink-LDA as baselines, since they are general enough to be applied in the task. We use a held-out dataset to evaluate the proposed model's capability of predicting unseen data. Perplexity is a standard measure of evaluating the generalization performance of a probabilistic model [20]. It is monotonically decreasing in the likelihood of the held-out data. Therefore, a lower perplexity indicates better generalization performance. Specifically, perplexity is calculated according to the following equation:

$$Perplexity_{held-out}(\mathcal{M}) = (\prod_{d=1}^{D}\prod_{i=1}^{N_d} p(w_i|\mathcal{M}))^{\frac{-1}{\sum_{d=1}^{D}(N_d)}}, \tag{13}$$

where $\mathcal{M}$ is the model learned from the training process. The result of perplexity comparison is presented in Figure 3(a), from which we observe that the proposed models demonstrate much better capability in predicting unseen data comparing with the LDA baselines. For example, when the number of search topics set to 300, the perplexity of LDA is 420.65, the perplexity of PLink-LDA is 419.23 and that of the proposed topic model is 299.12. The result verifies that our proposed model provides better fit for the underlying structure of the information of each app developer. Thus, the proposed model has better performance to derive different facets of the expertness for app developers.



(a) Perplexity Comparison (b) ADR Evaluation

Fig. 3. ADE and ADR Evaluations

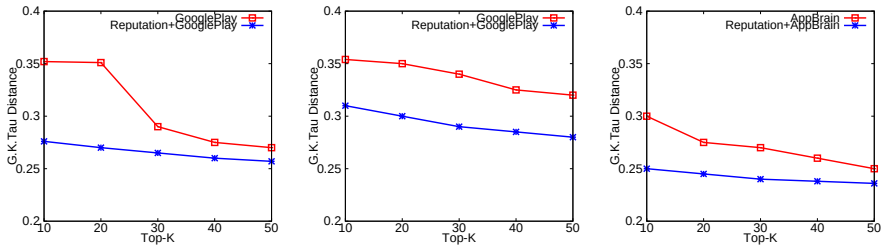
Evaluation of ADR To evaluate ADR, we manually label 400 pairwise comparison among about 800 developers which are used to train RankSVM. The comparison between two developers is performed according to their web sites and the average rating, average installations, user reviews of their apps. We first consider the average rating and average installations. If two app developers are very close in above two aspects, we then refer to their web sites and reviews of their apps. Apart from these pairwise comparison, we also rank 50 developers manually according to 10 volunteers' judgements as the ground truth to evaluate effectiveness of ADR and baselines.

We propose an intuitive baseline (PMR) which ranks developers according to the arithmetic product of popularity and average rating. The evaluation metric we use here is the well-known generalized Kendall's Tau distance[16].

The experiment result is showed in Figure 3(b) where ADR is about 0.1 lower than PMR in terms of Kendall's Tau Distance, which indicates that ADR gives much better reputation ranking.

7.3 Evaluation of Applications

Evaluation of Facilitating App Search We conduct some experiments to show that app developer profiles make app search more effective. We first prepare the *pseudo ground truth* by rank aggregation. We use Borda Count[19] to



(a) googleplay (b) appgravity (c) appbrain

Fig. 4. Facilitating App Search Evaluation

aggregate the app ranking lists of three commercial search engines, i.e. Google Play, appgravity, appbrain, for each query. Apps are ranked by Borda scores in the aggregated ranking list. Then we use the aggregated ranking list as the *pseudo ground truth* of the corresponding query.

We also employ the generalized Kendall's Tau distance to evaluate the correlation between the *pseudo ground truth* and the ranking list generated by methods under study. The larger the generalized Kendall's Tau distance is, the less correlation between two ranking lists.

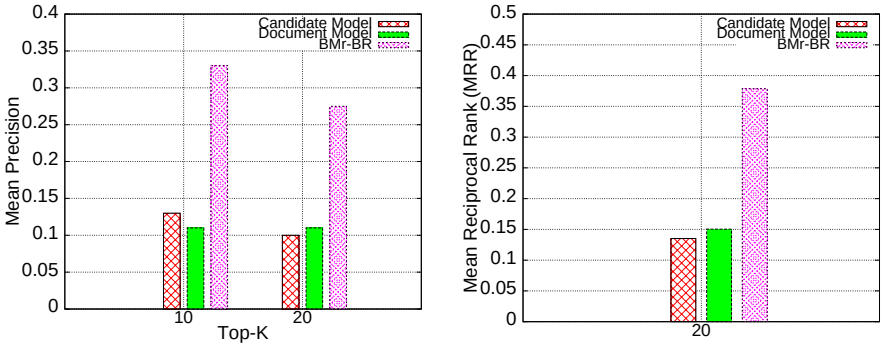
From Figure 4(a) to 4(c), we see that all ranking results that aggregate app developer reputation ranking list have smaller generalized Kendall's Tau distance than these not taking app developer reputation into consideration. After combined with app developer reputation ranking, the generalized Kendall's Tau distance of Google Play, appgravity, appbrain all decrease in studied top-k ranking results, which show app developer profiles effectively improves app search quality.

Evaluation of App Development Outsourcing We implement *BMr-BR* and two baselines, candidate model and document model, in this section. 5000 developers are selected as the candidates set, in which some big famous studios are excluded. We first fabricate 10 queries which are descriptions of ten wanted apps. Then BMr-BR, candidate model and document model are applied to recommend developers for these 10 queries. In order to evaluate the recommended results, we manually check the top-10 and top-20 recommended developers for each query. The check standard is that, if a developer has developed a similar[4] apps to the given query we consider it is a hit developer. Besides, a hit developer is more proper if its apps that are similar to the given query have higher rating as well as more good user reviews.

When implementing *BMr-BR*, we set the topic amount K=50 and the average length of expertise key words sets is 802. To evaluate the performance of these three methods, we compute the mean precision at top-10 and top-20 as well as Mean Reciprocal Rank (MRR) at top-20. As Figure 5(a) shows, *BMr-BR* has the highest mean precision both at top-10 and top-20, which are respectively 0.33

[4] "similar" means the two app have similar functions, game rules or undertake similar tasks.

and 0.275. Besides, *BMr-BR* also outperforms candidate model and document model in MRR at top-20 respectively by 0.24 and 0.22.



(a) Mean Precision (b) Mean Reciprocal Rank (MRR)

Fig. 5. App Development Outsourcing Evaluation

8 Conclusion

In this paper, we present a new ADI framework which is founded on two underlying models of ADE and ADR to profile app developers. Within the framework, these two models take into account multiple information sources, such as app descriptions, ratings and user reviews, in order to effectively and comprehensively profile app developers. The ADE model uses app's description and category to generate app developer's functionality-based expertise. The ADR model utilizes information from apps, users and developers to evaluate the reputation of an app developer. The extensive experiments show that the two models are effective. In addition, we demonstrate the use of app developer profiles by two applications, facilitating app search and app development outsourcing. All the empirical results show that app developer profiling is extremely useful for both the users and developers. Our modeling approach paves the way to establish more sophisticated profiling, for example taking into account social information in a mobile platform to extend our ADI framework.

References

1. M. Alazab, V. Monsamy, L. Batten, P. Lantz, and R. Tian. Analysis of malicious and benign android applications. In *ICDCSW*, 2012.
2. K. Balog, L. Azzopardi, and M. De Rijke. Formal models for expert finding in enterprise corpora. In *SIGIR*. ACM, 2006.
3. L. Barbosa and J. Feng. Robust sentiment detection on twitter from biased and noisy data. In *ACL*, 2010.
4. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 2003.

5. P. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, pages 311–320. ACM, 2012.
6. T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proc. of the National Academy of Sciences of the United States of America*, 2004.
7. V. Ha-Thuc, Y. Mejova, C. Harris, and P. Srinivasan. A relevance-based topic model for news event tracking. In *SIGIR*, 2009.
8. D. Jiang, K. Leung, and W. Ng. Context-aware search personalization with concept preference. In *CIKM*, 2011.
9. D. Jiang and W. Ng. Mining web search topics with diverse spatiotemporal patterns. In *SIGIR*.
10. D. Jiang, J. Vosecky, K. W.-T. Leung, and W. Ng. G-wstd: A framework for geographic web search topic discovery. In *CIKM*, 2012.
11. D. Jiang, J. Vosecky, K. W.-T. Leung, and W. Ng. Panorama: a semantic-aware application search framework. In *EDBT*, 2013.
12. T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, 2002.
13. D. Kang, D. Jiang, J. Pei, Z. Liao, X. Sun, and H. J. Choi. Multidimensional mining of large-scale search logs: a topic-concept cube approach. In *WSDM*, 2011.
14. C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008.
15. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.
16. J. Mazurek. Evaluation of ranking similarity in ordinal ranking problems. *Acta academica karviniensia*.
17. Q. Mei, C. Liu, H. Su, and C. X. Zhai. A probabilistic approach to spatiotemporal theme pattern mining on weblogs. In *WWW*, 2006.
18. D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *The Journal of Machine Learning Research*, 10:1801–1828, 2009.
19. M. Renda and U. Straccia. Web metasearch: rank vs. score based rank aggregation methods. In *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003.
20. M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *Proceedings of the UAI conference*, 2004.
21. G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.
22. J. Vosecky, K. Leung, and W. Ng. Searching for quality microblog posts: Filtering and ranking based on content analysis and implicit links. In *DASFAA*, 2012.
23. H. M. Wallach. Structured topic models for language. *Unpublished doctoral dissertation, Univ.of Cambridge*, 2008.
24. X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Malicious android applications in the enterprise: What do they do and how do we fix it? In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 251–254. IEEE, 2012.
25. Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proc. of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012.