A Risk and Similarity Aware Application Recommender System

Xiaoyuan Liang, Jie Tian, Xiaoning Ding and Guiling Wang

Department of Computer Science, New Jersey Institute of Technology, Newark, New Jersey, USA

As mobile devices, especially smartphones, become more and more popular, the number of mobile applications increases dramatically. Though mobile applications provide users convenience and entertainment, they have potential threat to violate users' privacy and security. In order to decrease the risk of violation, we propose a risk and similarity aware application recommender system, which recommends high quality applications to users. The system estimates applications' risk based on the requested permissions and calculates the similarity between applications based on the ratings and the number of ratings. It recommends applications with the lowest risk and highest similarity based on a user's current applications. The evaluation shows that the system works efficiently in recommending low-risk and high-similarity applications.

Keywords: computer science, security

1. Introduction

Mobile devices, such as smartphones, iPad, and tablets, now become increasingly popular. A large number of mobile applications have been developed to fully utilize these smart devices. For example, there are over 1.3 million Android applications and over 1.2 million iOS applications by year 2014 [1, 2]. These applications have uneven quality. Some have even been identified as malwares (e.g., DroidDream and DroidKungFu) [3]. With such a big number of applications, when a user searches for an application to fit her needs, usually she will find multiple applications on the market. It is important to avoid selecting the applications with low quality or high risks. Low-quality applications lead to bad user experience, and unsafe applications open doors to security and privacy intrusions [4]. However, it would be a challenge to screen the applications, since the number of applications may be large and their actual qualities vary widely.

The potential risks of bad user experience and security/privacy breaches make many users reluctant to install mobile applications. Statistics have shown that 54 % of smartphone users decide not to install a cell phone application when they discover that their personal information needs to be shared [5]. Studies also show that nowadays about 38 % of users are more concerned about phones than laptops in terms of security and over 50 % are more concerned about privacy [6]. Thus, it is an urgent issue to automatically and reliably evaluate applications and help users make decisions on which applications should be selected.

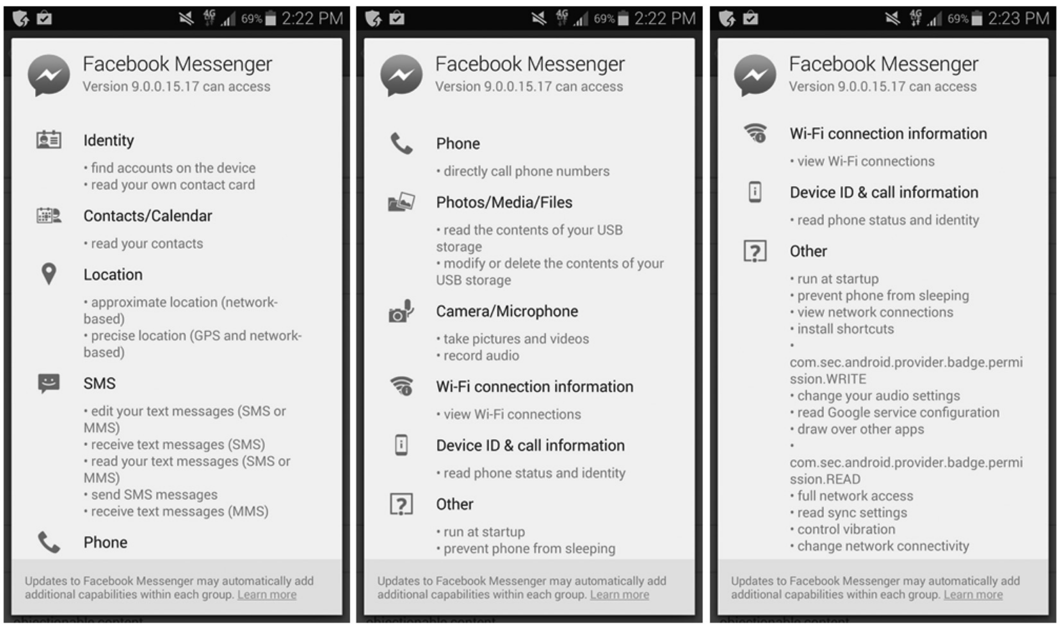
Permission warning, malware detecting, and using recommender systems are existing methods that can help users evaluate applications to keep safe ones and avoid risky ones. The first two methods apply to the applications which are about to be installed or are already installed on mobile devices. With *permission warning*, users are informed of the permissions applications may request before they install the applications. However, since it requires users to pay attention to and understand the permissions and permission warnings, it can be a big burden for most users. One reason is that most users have limited knowledge of the meaning of each permission [4, 7, 8]. The other reason is that applications may request a large number of permissions and/or incur frequent permission warnings. Figure 1 shows a list of permissions requested by Facebook Messenger as an example [9]. For most users, it would be overwhelming to pay attention to every permission warning. Impatient users may choose to ignore some permission warnings, making them vulnerable to malwares. *Malware detecting* uses software detectors to identify behavior breakdowns of the installed applications [10]. However, malware detectors may fail to detect new malwares that have not been included in their libraries [11]. At the same time, it takes time for malware detectors to identify malwares on mobile devices. *Recommender systems* have been designed to evaluate and recommend mobile applications [12, 13]. However, most existing recommender systems consider only one metric, either potential risks or user preference. They cannot provide users with the most suitable applications which are also safe and can preserve user privacy.

In this paper, we propose a Risk and Similarity Aware Application recommender system, named RSAA. It evaluates applications based on their potential risks and their similarity with the existing applications already installed on the mobile device of the user. Based on the evaluation, it recommends a number of applications that carry the lowest risks and are most similar to the applications already owned by the user. Specifically, the system first estimates the potential risk of every application based on the permissions it requests. It adopts the Inverse Document Frequency (IDF) method used in text mining area to estimate the importance of a permission in each application category. Different permissions indicate different risks. Thus, they are given different weights in the estimation. Based on the estimation, RSAA filters out the applications with high risks. Then, RSAA estimates the similarity between applications in the same category based on their ratings and rating numbers. Finally, RSAA considers both the potential risks and the similarity, and recommends a number of applications with the largest similarity in the low risk application category.

The remainder of the paper is organized as follows. Section 2 introduces the background information. The system design is presented in Section 3. Section 4 analyzes the data and evaluates the system. We discuss the deficiency of permission systems and the privacy expectations of mobile users in Section 5. The related work is discussed in Section 6. Finally, Section 7 concludes this paper and introduces our future work.

2. Background

In this section, we introduce the life cycles of mobile applications from development to installation, to better understand the quality and security/privacy issues with mobile applications. Then, we introduce permission-based security


69% 2:22 PM

Facebook Messenger
Version 9.0.0.15.17 can access

Identity
• find accounts on the device
• read your own contact card

Contacts/Calendar
• read your contacts

Location
• approximate location (network-based)
• precise location (GPS and network-based)

SMS
• edit your text messages (SMS or MMS)
• receive text messages (SMS)
• read your text messages (SMS or MMS)
• send SMS messages
• receive text messages (MMS)

Phone

Updates to Facebook Messenger may automatically add additional capabilities within each group. Learn more

69% 2:22 PM

Facebook Messenger
Version 9.0.0.15.17 can access

Phone
• directly call phone numbers

Photos/Media/Files
• read the contents of your USB storage
• modify or delete the contents of your USB storage

Camera/Microphone
• take pictures and videos
• record audio

Wi-Fi connection information
• view Wi-Fi connections

Device ID & call information
• read phone status and identity

Other
• run at startup
• prevent phone from sleeping

Updates to Facebook Messenger may automatically add additional capabilities within each group. Learn more

69% 2:23 PM

Facebook Messenger
Version 9.0.0.15.17 can access

Wi-Fi connection information
• view Wi-Fi connections

Device ID & call information
• read phone status and identity

Other
• run at startup
• prevent phone from sleeping
• view network connections
• install shortcuts
• com.sec.android.provider.badge.permission.WRITE
• change your audio settings
• read Google service configuration
• draw over other apps
• com.sec.android.provider.badge.permission.READ
• full network access
• read sync settings
• control vibration
• change network connectivity

Updates to Facebook Messenger may automatically add additional capabilities within each group. Learn more

Figure 1. Screenshot of the permissions requested by Facebook Messenger.

measures widely used in mobile devices, using Android permission system as an example. RSAA is built upon permissions.

2.1. Mobile Application Life Cycle

Mobile applications are developed and then released on application markets by millions of developers, most of whom are individual developers or small teams. There are several steps before an application is released. Taking Android application development as an example, Google divides the whole development process into the steps shown in Figure 2 [14]. In the first two steps, the application is being built by its developers. Then, it is tested on both virtual devices and hardware devices, signed by the developers, and connected to remote servers for production. At the last step, it is put on application markets for users to download. For Android applications, the official market is Google Play. There are also a number of third-party application markets, such as 1Mobile and AppsLib. Compared to official application markets, third-party markets may have special advantages, such as free applications, in order to attract more users. Developers may publish their applications in third-party markets to reach more users or certain users (e.g., those speaking a specific language) [15]. However, third-party markets may repackage applications from official markets for some unsafe purposes [16].

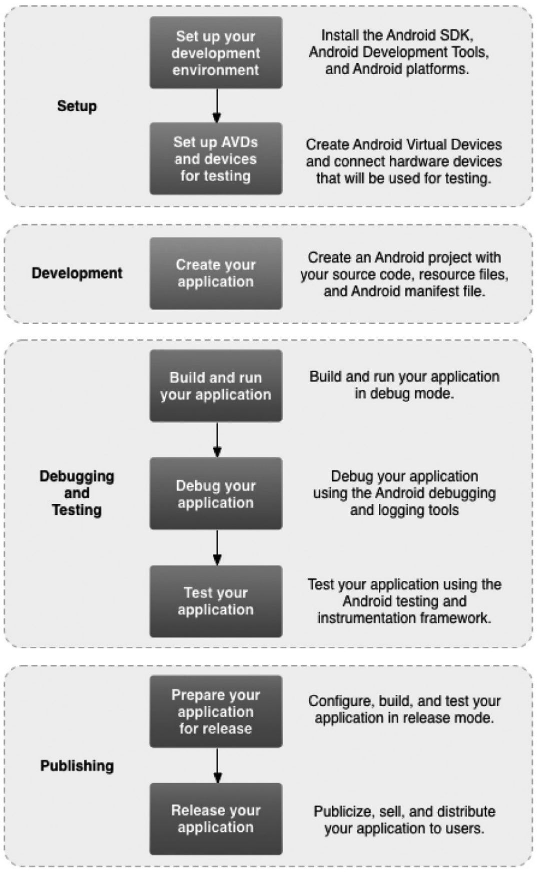

Setup
Set up your development environment
Install the Android SDK, Android Development Tools, and Android platforms.
Set up AVDs and devices for testing
Create Android Virtual Devices and connect hardware devices that will be used for testing.
Development
Create your application
Create an Android project with your source code, resource files, and Android manifest file.
Debugging and Testing
Build and run your application
Build and run your application in debug mode.
Debug your application
Debug your application using the Android debugging and logging tools
Test your application
Test your application using the Android testing and instrumentation framework.
Publishing
Prepare your application for release
Configure, build, and test your application in release mode.
Release your application
Publicize, sell, and distribute your application to users.

Figure 2. Life cycle of Android applications.

When a user looks for an application to meet a specific need, with her mobile device, she may search an application market with related keywords, or receive some recommendations [6]. Usually, multiple applications will be provided to choose after the search. The user may look at the ratings and reviews before she chooses one to install. When the installation is about to start, the system lists the permissions that the application requests. The user must decide whether the permissions should be granted or not. The actual installation starts when the system receives the confirmation from the user.

2.2. Android Permission System

In most mobile operating systems, applications are isolated from each other. Personal data and resources can only be accessed by applications that have been granted with the related permissions. For example, Android provides 135 permissions for developers [17]. If one application wants to send a short message, it must request the *SEND_SMS* permission in advance.

Permissions can be specified in advance, before installation or on demand when a specific resource or data is needed. For an Android application, all its permissions are saved in an xml file. When a user installs the application, these permissions will be displayed on a confirmation page. Only after the user confirms the permissions, can the application be installed. An iOS application requests permissions on demand, just before it accesses the related data/resources.

Permission systems protect critical data and code that could be misused to distort or damage the user experience. Usually, permissions are classified into multiple levels, depending on their risks and how critical the related data/resources are. For example, based on official Android documents, there are four protection levels on Android systems [18]:

- **Normal** for lower-risk permissions that allow the accesses to isolated application-level features. Normal permissions have minimal risks to other parts of the system.
- **Dangerous** for higher-risk permissions that allow the accesses to private user data or grant the control over the device that can negatively impact the user.
- **Signature** for the permissions that are only granted to the applications with certain signatures.
- **SignatureOrSystem** for the permissions that are only granted to pre-installed applications and applications with certain signatures.

Though the permissions at *Signature* level and *SignatureOrSystem* level may allow accesses to critial data and resources, pre-installed applications and applications with required signatures are usually deemed safe, and these permissions are granted without the awareness of users. For this reason, users are more concerned about the permissions at the *Normal* and *Dangerous* protection levels. The levels of Android permissions are hard-coded in Android systems [19].

3. System Design

3.1. Overview

The objective of our study is to recommend high quality and safe applications to users based on their personal needs. To achieve this goal, a Risk and Similarity Awareness Application recommender system, called RSAA, is proposed. It recommends a number of applications which are similar to the applications a user has installed and have the lowest risk in the same category as the already installed applications. There are two challenges in implementing the system. One is how to evaluate the risks of applications. The other is how to recommend applications based on both the risk and the similarity. We have designed two algorithms to deal with these challenges.

Risk Prediction Algorithm: The purpose of this algorithm is to estimate the risk level of each application and to filter out the applications with high risks. The algorithm estimates each application's potential risk based on the permissions it requests. With the application risk and the number of requested permissions, the system clusters applications into three levels — *low risk*, *medium risk*, and *high risk*. High-risk applications are filtered out, and applications at the other two levels are kept for recommendation.

Recommendation Algorithm: With the recommendation algorithm, applications with the highest similarity at the medium and low risk levels are recommended to users. The algorithm first estimates the similarity between each installed application and each one left on the market based on their ratings and the number of ratings. Then, it recommends a certain number of applications based on the application risk and the similarity.

These two algorithms are described in detail in the following two subsections.

3.2. Risk Prediction Algorithm

The risk prediction algorithm evaluates the risk level of an application based on the permissions the application requests. Two factors are considered in the evaluation: (1) whether an application requests a particular permission that is not necessary to use in the applications with similar functionality. If the application does, the request is considered to be suspicious, and the application is likely to be problematic. (2) If a particular permission is granted, what is the potential risk associated with the permission. The algorithm assumes that applications have been categorized based on their functionalities. With the two factors, it evaluates the risks of the applications within each category.

Before we present the algorithm, we introduce the following notations.

- Let c_k denote a **category**. The size of the category is $|c_k|$ (i.e., the total number of applications in category c_k).
- Let a_i denote an **application** in category c_k $(1 \leq i \leq |c_k|)$.
- Let p_j denote a **permission**. Suppose there are m different permissions in a system (e.g., 135 for Android). Then, j is from 1 to m.

- We use $p_{i,j}$ to represent whether permission p_j is requested by application a_i. If application a_i requests permission p_j, $p_{i,j}$ is 1. Otherwise, $p_{i,j}$ is 0.

The algorithm first estimates to what degree a permission is essential for the applications in each category. For this purpose, the algorithm adopts the Inverse Document Frequency (IDF) concept widely used in text mining areas [20]. For a permission p_j, it calculates an inverse frequency F_{p_j} as follows:

$$F_{p_j} = \lg \frac{|c_k|}{\sum_{i=1}^{|c_k|} p_{i,j}}. \qquad (1)$$

For a permission p_j, the smaller the value F_{p_j} is, the more likely the applications in the category need the permission. If all the applications in the category request permission p_j, F_{p_j} takes the smallest value 0. This indicates that permission p_j may be indispensable for the applications in this category. Thus, later on, when RSAA evaluates the risk of an application in this category, the request for this permission (p_j) is not considered to be suspicious.

Then, the algorithm takes into account the potential risks associated with the permissions by giving a weight to each permission. Currently, the algorithm classifies permissions into two groups, *safe* and *dangerous*. It assigns a higher weight to the applications in the dangerous group and a lower weight to the applications in the safe group. The weights are predefined in advance and will be discussed in detail in the next section.

Suppose the weight of permission p_j is w_j. With the weight and the inverse frequency of p_j, the algorithm calculates the risk R_{p_j} of permission p_j in category c_k as follows:

$$R_{p_j} = w_j \times F_{p_j}. \qquad (2)$$

Please note that the algorithm calculates the inverse frequency and the weighted risk of a permission based on the requests for the permission made by the applications in the same category. Thus, the inverse frequency and the risk are specific to the category. They may have other values in another category.

With the risk of each permission, the algorithm defines a weighted risk R_{a_i} for application a_i, which is calculated by adding up the risks of all the permissions requested by application a_i.

$$R_{a_i} = \sum_{j=1}^{m} R_{p_j} \times p_{i,j}. \qquad (3)$$

Finally, the algorithm clusters applications into different risk levels – high risk, medium risk, and low risk. The applications with medium risks or low risks are kept for recommendation and those with high risks are filtered out. The algorithm adopts k-medoids algorithm to cluster applications [21]. K-medoids is similar to k-means. It first selects K initial cluster centers. Then, it iteratively assigns instances (i.e., applications in our case) to clusters until there are no further changes with the assignment. During the process, the cluster centers may change. With different k-means, when choosing a new cluster center, k-medoids chooses one of the feature values to be the new center, instead of the virtual mean of the feature values. K-medoids algorithm proves to have a better performance in handling noise and outliers [22].

The clustering of applications in RSAA is based on three features – the number of safe permissions, the number of dangerous permissions, and the weighted risk R_{a_i}. However, these features have different scales. For example, the number of safe permissions can be larger than the weighted risks by one or more orders of magnitudes. When estimating the distance between two applications, directly using their feature values may lead to undesirable effects since the features with large scales will dominate the clustering. Thus, the feature values of the applications must be normalized before clustering.

For each feature, we employ Z-score to normalize the feature values of the applications using the following equation:

$$Z = \frac{X - \mu}{\sigma}. \qquad (4)$$

In the equation, X is the feature value before normalization, and Z is the feature value after normalization. μ and σ are, respectively, the mean value and standard variance of the feature values of all the applications.

3.3. Recommendation Algorithm

Collaborative filtering (CF) algorithms are widely adopted in recommender systems [23, 24]. We employ one kind of the CF algorithms, the item-based collaborative filtering (CF) algorithm, to classify and recommend applications at low-risk or medium-risk levels [25]. The recommendation algorithm first calculates the similarity scores between the installed applications and the application on the market in the corresponding categories, and then classifies the applications on the market based on the scores. We employ *cosine* similarity to calculate similarity scores. The similarity score between applications a_i and a_j is calculated as follows:

$$similarity(a_i, a_j) = \frac{\vec{a_i} \cdot \vec{a_j}}{\|\vec{a_i}\| \, \|\vec{a_j}\|}. \qquad (5)$$

In this equation, $\vec{a_i}$ and $\vec{a_j}$ are two vectors respectively for application a_i and a_j. Each vector includes two attribute values of the corresponding application. One attribute value is the average rating of the application. The other is the number of ratings received by the application.

The system calculates a similarity score for every pair of applications, with one application already installed on the mobile device and the other application on the market in the corresponding category. The similarity scores of all the application pairs form a similarity matrix. With the similarity matrix, the system chooses a number of applications that have the highest similarity scores with the installed applications in the user's device. Then it sorts the selected applications based on their weighted risks, and presents the sorted list of applications to the user. If a user wants to check the top L applications, the system picks $2L$ applications first based on their similarity scores. Then, among these $2L$ applications, it selects L applications with the lowest weighted risks.

4. Evaluation

In this section, we evaluate the proposed system. We also explain some implementation details.

4.1. Data Analysis

To evaluate the system, we use real applications provided by Frank *et al.* [26], which were downloaded from Google Play. After filtering out duplicated applications, we get 183127 applications in total. These applications are classified into 30 categories, including Productivity, News & Magazines and Entertainment. Figure 3 shows the top 20 largest categories and the proportions of the applications in them. The Entertainment category has the most applications, about 12 %, followed by Personalization, 10 %, and Tools, 8 %. In our evaluation, the applications in these top 3 largest categories are selected to test the proposed system.

In each category there are free applications and paid applications. Their percentages are shown in Table 1 for the selected three categories. In

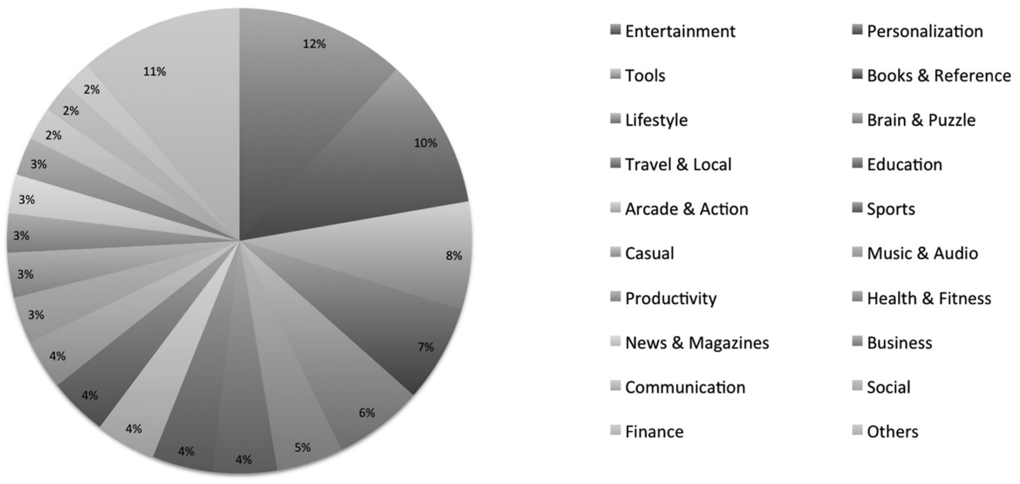


Figure 3. Application categories.

	Entertainment	Tools	Personalization
Free	69.85 %	74.23 %	26.74 %
Paid	30.15 %	25.77 %	73.26 %
Total	100.00 %	100.00 %	100.00 %

Table 1. Percentages of free applications and paid applications in the selected categories.

the first two categories, there are more free applications than paid applications. The ratio between free applications and paid ones in Entertainment is about 2 : 1 while the ratio in Tools is around 3 : 1. In Personalization category, the number of paid applications is approximately 3 times as the free ones. It indicates that the applications in Personalization are more likely to be paid ones than those in the other two categories.

Apart from the costs of applications, we have also analyzed the permissions requested by the applications. We want to know the number of safe permissions and the number of dangerous permissions requested for the applications in each category. On Android systems, permissions are classified into four protection levels, which have been introduced in Subsection 2.2. In our system, permissions at the *normal* protection level on Android systems are classified as safe permissions, and permissions at other Android protection levels are considered as dangerous ones. Since application descriptions only list detailed permissions (as illustrated in Figure 1) and do not report their protection levels, we translated the requested permissions into their protection levels, and then determined for each requested permission whether it is "safe" or "dangerous".

For the free applications and paid applications in

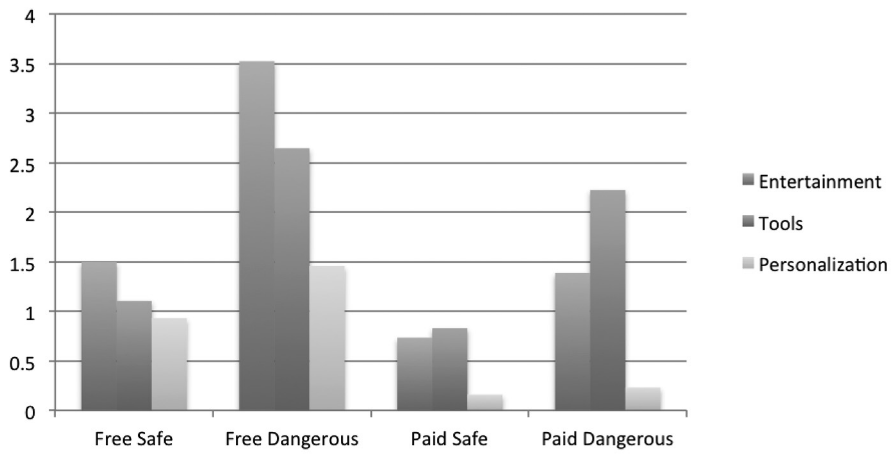


Figure 4. Average number of requested permissions.

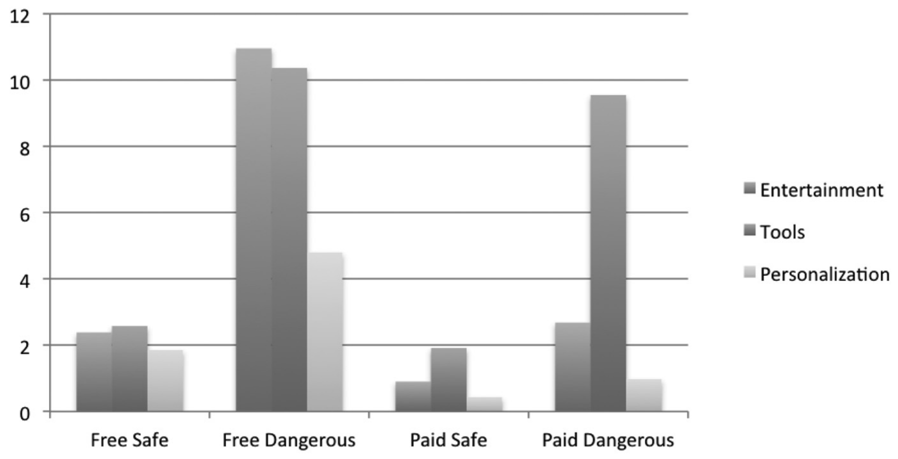


Figure 5. Variance of requested permission numbers.

each category, Figure 4 shows the average number of safe permissions and the average number of dangerous permissions requested by these applications, and Figure 5 presents the variance of safe permission numbers and the variance of dangerous permission numbers requested by the applications. As shown in Figure 4, on average, free applications request more permissions than paid applications, indicating that paid applications generally have higher quality than free ones. This also indicates that some permissions may not be necessary for free applications.

The figures also show that both free and paid applications request more dangerous permissions than safe permissions. Free applications in Entertainment category tend to request the most permissions (both safe ones and dangerous ones). In paid applications, the applications in Tools category tend to request more permissions than those in other categories (both safe ones and dangerous ones). At the same time, the number of permissions requested by these applications show larger variances than other paid applications.

4.2. Risk Evaluation

For each category, we employ the k-medoids method to cluster free and paid applications separately. The applications in the category are clustered into three levels, high risk, low risk and medium risk.

When estimating the potential risks, we use the following features to cluster applications:

- *Safe Permission Number*: the number of safe permissions that are requested by each application.
- *Dangerous Permission Number*: the number of dangerous permissions that are requested by each application.
- *Weighted Risk*: the weighted risk of each application calculated in Eq. 3.

Different weights are assigned to dangerous permissions and safe permissions based on the numbers of safe permissions and the number of dangerous permissions. The weight for normal permissions is 1. The weight for dangerous permissions is the value of the number of safe permissions divided by that of dangerous permissions.

The results of the clustering are shown in Table 2. We can see that in each category, for both free and paid applications, high risk applications account for much smaller percentages than the applications with low risks. The free applications in the high risk cluster of Personalization category account for a higher percentage than those in the other two categories, but paid applications in the high risk cluster of Personalization category account for a much lower percentage than those in the other two categories. This indicates that, for Personalization applications, users can get much higher quality applications if they download the paid ones.

Taking Personalization applications as samples, we have also studied the mean values of the features in every risk level. The mean values (before normalization) are shown in Table 3.

This table clearly shows that applications at the high risk level tend to request more permissions (both dangerous ones and safe ones) and tend to obtain higher weighted risks. In addition, for each feature, the differences between different risk levels in paid applications are larger than those with free applications. Take the number

Category		**High Risk**	**Medium Risk**	**Low Risk**
Entertainment	Free	10.16 %	19.98 %	69.86 %
	Paid	16.13 %	37.18 %	46.69 %
Tools	Free	5.97 %	30.39 %	63.64 %
	Paid	6.30 %	32.39 %	63.11 %
Personalization	Free	17.40 %	24.88 %	57.72 %
	Paid	3.83 %	22.59 %	73.58 %

Table 2. Percentages of applications in different risk clusters.

Type		High Risk	Medium Risk	Low Risk
Free	Number of Dangerous Permissions	5.23	3.22	0.13
	Number of Safe Permissions	1.97	1.34	0.03
	Weighted Risk	0.10	0.06	0.01
Paid	Number of Dangerous Permissions	12.98	5.26	0.36
	Number of Safe Permissions	4.53	1.88	0.10
	Weighted Risk	0.96	0.27	0.02

Table 3. Mean feature values of Personalization applications.

of dangerous permissions requested by applications as an example. At the high risk level, the number with paid applications is much larger than that with free applications. However, at the medium and low risk levels, the numbers with paid applications are close to those with free applications.

After obtaining the risk levels of applications, we filter out those at the high risk level. This is necessary to ensure the high quality of applications. The percentages of free applications and paid applications that are filtered out in each category are shown in Table 2. For example, 10.16 % of free applications in Entertainment category are filtered out.

4.3. Recommender System

We evaluate the recommender system with simulation. In the simulation, 10 applications are randomly chosen and installed on a mobile device. Then, with these installed applications, we use three different methods to choose applications to recommend, and compare the quality and the risks of the applications recommended by these methods. Among these three methods, RSAA is the proposed method, and the rating first recommendation method and the random recommendation method are used as baselines. The rating first method recommends applications based only on their ratings. The random method picks applications randomly. To evaluate the quality and risks of the recommended applications, we select two metrics, maximum weighted risk and minimum similarity. *Maximum weighted risk* is the largest weighted risk value in a recommendation list. *Minimum similarity* is the minimum similarity value in a recommendation list.

In each experiment, we use a recommendation method to generate a list of recommended applications. In the evaluation, we vary the length of the list *L* between 50 and 200, and use each method to generate a list for each *L* value. We repeat the experiments 100 times. Every time, we re-select 10 random applications and use them as the applications installed on the mobile device. Then we rerun the tests to generate lists of recommended application for these newly selected applications. We average the maximum weighted risks over the lists generated for different *L* values and across the repeated experiments. For brevity, we still refer to the average value as maximum weighted risk. Similarly, we also average the minimum similarities and refer to the average value as minimum similarity for brevity.

Figure 6 compares the maximum weighted risks for the three methods. This figure shows that the applications recommended by RSAA have the lowest maximum weighted risks, which are close to 0. For RSAA, the maximum weighted

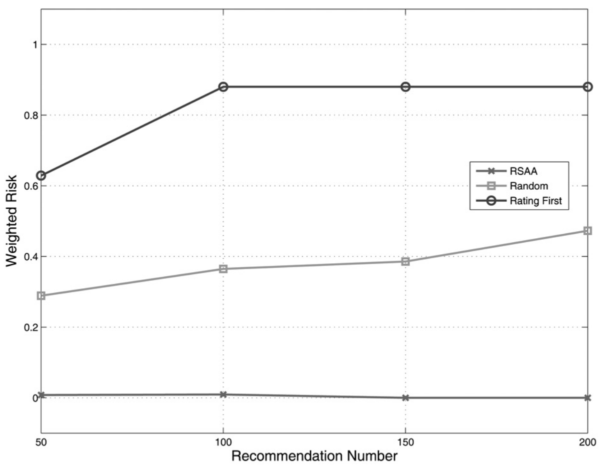


Figure 6. Maximum weighted risks of the applications recommended by three methods.

risk decreases when the number of recommended applications increases, because applications with lower risks can be chosen based on the similarity metric, which replaces the high risk applications on the final recommendation list. This also indicates that there are a large number of low risky applications available to users. The figure also shows that applications recommended by rating first method have high risks. The maximum weighted risks of the rating first method is always above 0.6.

We have also compared the minimum similarities for the three methods with the number of recommended applications L varied from 50 to 200. The results are shown in Figure 7. The figure shows that the rating first method can keep the similarity high for different L values. RSAA can also keep the similarity over 99.5 %, achieving similar performance as the rating first method. When the number of recommended applications increases, the minimum similarity decreases slightly. The random method shows the worst performance.

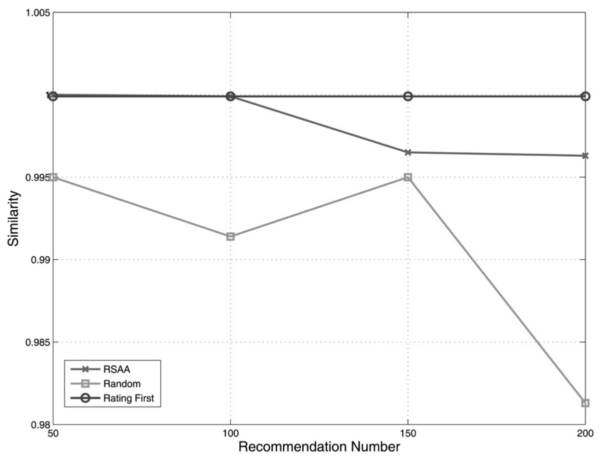


Figure 7. Minimum similarities of the applications recommended by three methods.

5. Discussion

In this section, we discuss the potential problems in recommender systems. The first one is about the deficiency on permission systems. The second one is about users' expectations on security and privacy of mobile devices.

5.1. Deficiency on Permission Systems

The permission system is widely used in mobile systems, like Android and iOS. The initial purpose for this system is to isolate personal data and resources from applications. However, most users do not pay attention to the permissions, which gives an opportunity for malware.

Google provides developer documentation for Android developers, but there is limited information of permissions. Felt *et al.* even find that there are 6 errors in the Android permission documentation [27]. In addition, they list several reasons that permissions are requested more than an application really needs. The first one is permission name error, such as ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE. Developers usually get confused about the above two. So they often request them together. The second is deputies. An application can send an Intent to another deputy application, asking it to perform an operation. If the deputy does a permission-needed operation, it needs the permission while the sender does not need it. Felt *et al.* also find that developers tend to request unnecessary permissions because applications would not receive automatic updates if the updated ones need more permissions [28].

From the above reasons, we can see that the permission system is not efficient or effective, which makes developers confused and tends to request unnecessary permissions. Thus, a mechanism is needed to penalize those applications that request too many permissions, especially dangerous permissions. It not only ensures that users get high quality applications, but also pushes developers to develop more secure applications.

5.2. Users' Expectations on Security and Privacy

Users' expectations on security and privacy show that we need to focus on improving the security and protecting the privacy. From the survey done by Erika et al., the top four factors that users worry about regarding the privacy in smartphones are as follows [6]:

1. physical phone loss
2. physical damage
3. data loss and (lack of) backup

4. trusting applications

We can see that most people are not aware of the importance of applications' security and privacy. In fact, personal information loss caused by applications happens frequently. For example, 13 GB of Snapchat content was recently leaked due to the use of third party Snapchat applications [29] and Whisper was said to track users and share information with one of the U.S. departments [30]. There are more applications unreported and still stealing users' information.

For users, it is very easy to install low quality applications in mobile devices, since it is hard for most of them to identify them. When users are installing a new application, they only pay attention on the functions it provides. For example, if a user wants to download a video application, then she will certainly choose the one with many video resources and smooth watching experience, ignoring whether it has potential risk or not.

In addition, the mobile application system is not good enough. It does not filter out the malware in time and does not highlight these potentially dangerous information. Also, the provided information is confusing to many users. It is found that only 9 % of users could answer the meaning of permission READ_CONTACTS broadly correctly [7].

Thus, we need to find new ways of developing the mobile application system to provide users with more information as well as to identify the potential risks by the system instead of users.

6. Related Work

Mobile Privacy and Security

Several works focus on understanding the users' feeling about the mobile privacy and security. Lin *et al.* do a survey on users' expectations about what an application does and does not do to build a users' mental model of privacy. They also design a new permission screen that displays the permissions requested by an application [4]. This increases users' privacy awareness and is easier to comprehend than the original permission screen. Felt *et al.* analyze users' preference and comprehension of different permissions via Internet and lab surveys [7]. Chin *et al.* measure users' confidence in smartphone security and privacy via interviews and surveys and compare it with that in computers [6]. These works provide us with intuitive feeling about users' expectations and give a guide for the improvement.

Some other works mainly pay attention to how to detect mobile malware from the normal ones. Zhou *et al.* present a system, named DroidRanger, to detect malicious applications on both official and unofficial markets, which includes footprint-based detection and heuristics-based detection [31]. Zhou *et al.* develop a system, called DroidMoss, to detect repackaged smartphone applications in third-party markets [16]. Enck *et al.* analyze the Trojans applications and set malware rules only with the permissions [32]. A follow-up work reports a series of systematic findings in Android application security from the study of 1100 free Android applications [33]. Frank *et al.* build a probabilistic model to mine permission request patterns from Android and Facebook applications [26]. These methods provide some important information on detecting malware. But it is still hard for users to detect all of them since there are no certain patterns in them.

Application Recommender System

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [34, 35, 36]. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read [23].

Research on mobile application recommender systems focus on suggesting mobile applications to users with one or more metrics. Following are some related application recommender systems. Sanz *et al.* implement a system to automatically classify Android applications using machine learning techniques [37]. Enrique *et al.* propose a recommender system using user-based CF algorithms, which monitors users' interaction [12]. However, they only take the users' preference into consideration and do not consider the applications' quality. A mobile application recommender system with different security levels is presented in [13]. But it does not provide a differentiated service and permissions are treated as the same. In this paper, we take both application security and user preference into consideration.

7. Conclusion and Future Work

In this paper, we propose a risk and similarity aware application recommender system to provide differentiated services for different users. The system is to recommend applications with the lowest risk and highest similarity based on user's current applications.

In the future, we plan to do the following work. More data, including training data and testing data, will be added to better evaluate the system. A user data collector will be developed to get users' application usage information. Using the usage information, we plan to refer a user's application usage habit. The usage habit information, combined with a user's other information, such as age and occupation, will be leveraged to provide more personalized recommendation service.

References

[1] *iTunes App Store Now Has 1.2 Million Apps, Has Seen 75 Billion Downloads To Date*, `http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/`, 2014.

[2] *Number of Android applications*, `http://www.appbrain.com/stats/number-of-android-apps`, 2014.

[3] *Android malicious apps*, `http://www.bullguard.com/bullguard-security-center/mobile-security/mobile-threats/android-malicious-apps.aspx`

[4] J. LIN, S. AMINI, J. I. HONG, N. SADEH, J. LINDQVIST, J. ZHANG, Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 2012.

[5] J. L. BOYLES, A. SMITH, M. MADDEN, Privacy and data management on mobile devices. *Pew Internet & American Life Project*, 2012.

[6] E. CHIN, A. P. FELT, V. SEKAR, D. WAGNER, Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, 2012.

[7] A. P. FELT, E. HA, S. EGELMAN, A. HANEY, E. CHIN, D. WAGNER, Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, 2012.

[8] P. G. KELLEY, S. CONSOLVO, L. F. CRANOR, J. JUNG, N. SADEH, D. WETHERALL, A conundrum of permissions: installing applications on an android smartphone. In *Financial Cryptography and Data Security*, 2012.

[9] *The Truth About the Facebook Messenger App and Your Privacy*, `http://crambler.com/truth-about-facebook-messenger-app-privacy/`, 2014.08.

[10] Y. ZHOU, X. JIANG, Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012.

[11] T. PETSAS, G. VOYATZIS, E. ATHANASOPOULOS, M. POLYCHRONAKIS, S. IOANNIDIS, Rage against the virtual machine: hindering dynamic analysis of Android malware. In *Proceedings of the Seventh European Workshop on System Security*, 2014.

[12] E. COSTA-MONTENEGRO, A. B. BARRAGÁNS-MARTÍNEZ, M. REY-LÓPEZ, Which App? A recommender system of applications in markets: Implementation of the service for monitoring usersò interaction. *Expert systems with applications*, 2012.

[13] H. ZHU, H. XIONG, Y. GE, E. CHEN, Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[14] *Introduction | Android Developers*, `http://developer.android.com/tools/workflow/index. html`

[15] *Understanding 3rd Party Android App Stores*, `http://www.airpush.com/understanding-3rd-party-android-app-stores/`, 2014.05.

[16] W. ZHOU, Y. ZHOU, X. JIANG, P. NING, Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, 2012.

[17] *Manifest.permission | Android Developers*, `http://developer.android.com/reference/android/Manifest.permission.html`

[18] *⟨Permissions⟩ | Android Developers*, `http://developer.android.com/guide/topics/manifest/permission-element.html`

[19] *Android Permissions – Protection Levels*, `http://rupertrawnsley.blogspot.de/2011/11/android-permissions-protection-levels.html`, 2011.11

[20] *tf-idf*, `http://en.wikipedia.org/wiki/Tf-idf`, 2014.08

[21] L. KAUFMAN, P. ROUSSEEUW, *Clustering by means of medoids*. North-Holland, 1987.

[22] P. BERKHIN, A survey of clustering data mining techniques. In *Grouping multidimensional data*, Springer, 2006, pp. 25–71.

[23] F. RICCI, L. ROKACH, B. SHAPIRA, *Introduction to recommender systems handbook*, 2011.

[24] B. SARWAR, G. KARYPIS, J. KONSTAN, J. RIEDL, Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, 2001.

[25] G. D. LINDEN, J. JACOBI, A. JENNIFER, E. A. BENSON, Collaborative recommendations using item-to-item similarity mappings, 2001, US Patent 6,266,649. [Online]. Available: `http://www.google.com/patents/US6266649`

[26] M. FRANK, B. DONG, A. P. FELT, D. SONG, Mining Permission Request Patterns from Android and Facebook Applications. In *ICDM*, 2012.

[27] A. P. FELT, E. CHIN, S. HANNA, D. SONG, D. WAGNER, Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[28] A. P. FELT, K. GREENWOOD, D. WAGNER, The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, 2011.

[29] *200,000 Snapchat leaks and it is all your fault*, `http://houstonianonline.com/2014/10/16/200000-snapchat-leaks-and-it-is-all-your-fault/`, 2014.10.

[30] *Whisper App Accused Of Violating Privacy*, `http://www.valuewalk.com/2014/10/whisper-app-accused-of-violating-privacy/`, 2014.10.

[31] Y. ZHOU, Z. WANG, W. ZHOU, X. JIANG, Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *NDSS*, 2012.

[32] W. ENCK, M. ONGTANG, P. MCDANIEL, On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, 2009.

[33] W. ENCK, D. OCTEAU, P. MCDANIEL, S. CHAUDHURI, A Study of Android Application Security. In *USENIX security symposium*, **2**, 2011.

[34] R. BURKE, Hybrid web recommender systems. In *The adaptive web*, 2007.

[35] T. MAHMOOD, F. RICCI, Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, 2009.

[36] P. RESNICK, H. R. VARIAN, Recommender systems. *Communications of the ACM*, **40**, 1997.

[37] B. SANZ, I. SANTOS, C. LAORDEN, X. UGARTE-PEDRERO, P. G. BRINGAS, On the automatic categorisation of android applications. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, 2012.

Received: November, 2014
Revised: June, 2015
Accepted: August, 2015

Contact addresses:

Xiaoyuan Liang
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
e-mail: `xl367@njit.edu`

Jie Tian
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
e-mail: `jt66@njit.edu`

Xiaoning Ding
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
e-mail: `xiaoning.ding@njit.edu`

Guiling Wang
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ, USA
e-mail: `gwang@njit.edu`

XIAOYUAN LIANG started the Ph.D. studies in Computer Science Department at the New Jersey Institute of Technology in 2013. He received his Bachelor Degree in Information Security from the Department of Computer Science and Engineering at Harbin Institute of Technology, China, in 2013. His research interest includes wireless networks, intelligent transportation and mobile applications.

JIE TIAN received the B.S. degree in computer science from the Tianjin University, Tianjin, China, in 2005, the M.S. degree in computer science from the Nankai University, Tianjin, China, in 2008 and the Ph.D. degree from the Department of Computer Science at New Jersey Institute of Technology, USA, in 2015. He joined Audible Inc. as a software development engineer in 2015. His research interest includes wireless networks, ad hoc/sensor network and mobile computing.

XIAONING DING is an Assistant Professor of computer science at the New Jersey Institute of Technology. His interests are in the area of experimental computer systems, such as distributed systems, cloud computing, storage systems, and databsase systems. He earned his Ph.D. degree in computer science and engineering from the Ohio State University in 2010. He may be reached at `xiaoning.ding@njit.edu`.

DR. GUILING WANG received the B.S. degree in software from the Nankai University, Tianjin, China, and the Ph.D. degree in computer science and engineering with a minor in statistics from the Pennsylvania State University, State College, PA, USA, in 2006. After that, she joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA. She is currently an Associate Professor with tenure at NJIT. Her research area includes mobile computing, intelligent transportation and wireless sensor networks.