SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2017, Vol. 60 092110:1–092110:17
doi: 10.1007/s11432-016-9072-3

Automated Android application permission recommendation

Lingfeng BAO[1], David LO[2], Xin XIA[1,3*] & Shanping LI[1]

[1] *College of Computer Science and Technology, Zhejiang University, Hangzhou* 310013*, China;*
[2] *School of Information Systems, Singapore Management University, Singapore* 188065*, Singapore;*
[3] *Department of Computer Science, University of British Columbia, Vancouver* B6T1Z4*, Canada*

Received January 2, 2017; accepted April 18, 2017; published online July 28, 2017

Abstract The number of Android applications has increased rapidly as Android is becoming the dominant platform in the smartphone market. Security and privacy are key factors for an Android application to be successful. Android provides a permission mechanism to ensure security and privacy. This permission mechanism requires that developers declare the sensitive resources required by their applications. On installation or during runtime, users are required to agree with the permission request. However, in practice, there are numerous popular permission misuses, despite Android introducing official documents stating how to use these permissions properly. Some data mining techniques (e.g., association rule mining) have been proposed to help better recommend permissions required by an API. In this paper, based on popular techniques used to build recommendation systems, we propose two novel approaches to improve the effectiveness of the prior work. The first approach utilizes a collaborative filtering technique, which is inspired by the intuition that apps that have similar features — inferred from their APIs — usually share similar permissions. The second approach recommends permissions based on a text mining technique that uses a naive Bayes multinomial classification algorithm to build a prediction model by analyzing descriptions of apps. To evaluate these two approaches, we use 936 Android apps from F-Droid, which is a repository of free and open source Android applications. We find that our proposed approaches yield a significant improvement in terms of precision, recall, F1-score, and MAP of the top-k results over the baseline approach.

Keywords Android, permission recommendation, association rule, collaborative filtering, text mining

Citation Bao L F, Lo D, Xia X, et al. Automated Android application permission recommendation. Sci China Inf Sci, 2017, 60(9): 092110, doi: 10.1007/s11432-016-9072-3

1 Introduction

Android is one of the most popular open source mobile platforms. In the second quarter of 2015, it dominated the smartphone market with a market share of 82.8%1). Thousands of developers are working to develop Android applications (also referred to as "apps"). It is reported that there were approximately 190000 apps in Google Play in the first quarter of 20162). Meanwhile, an increasing number of malicious apps, which are usually designed to steal sensitive data (e.g., private credentials and financial information), are produced by an increasing number of attackers.

* Corresponding author (email: xxia@zju.edu.cn)
1) Smartphone market share. http://www.idc.com/prodserv/smartphone-os-market-share.jspl.
2) Google Play. https://en.wikipedia.org/wiki/Google_Play.

To decrease the threats that Android apps pose to the security and privacy of their users, a unique permission mechanism is provided by Android to control the access of third party applications to sensitive resources, e.g., the contact list, camera, and network of users. Android app developers are required to write the required permissions explicitly into a config file named `AndroidManifest.xml`. Thus, Android app developers not only need to know how to use APIs to implement certain features of an application, but also the corresponding permissions. For instance, if an app wants to access the internet, the app developer not only needs to understand how to use network APIs such as "android.net.wifi.WifiManager" and "java.net.URL", but also needs to know the corresponding permissions, namely ACCESS_WIFI_STATE and ACCESS_NETWORK_STATE, which must be written to the AndroidManifest.xml file.

A better practice for Android developers to reduce the security risk is to minimize the number of permissions that their apps require; this is recommended by Android official[3)]. However, it is often difficult for Android app developers to determine which permissions are required. If an app requires a permission that is not declared, it will not work and will throw an exception. Thus, developers often require more permissions than the app needs [1]. Moreover, considerably more permissions are usually required by developers to ensure good user experience. Some researchers find that the official Android documentation for API classes and permissions is incomplete, which leads to overprivileged apps [1, 2]. Furthermore, more than 4000 classes and 151 system-level permissions exist in the Android library. Numerous misuses have been reported, even for the most popular permissions [3]. Hence, in this paper, we aim to investigate whether we can develop a recommendation system to help Android app developers determine suitable permissions.

Many tools have been proposed by researchers to recommend permissions by tracing APIs to specific permissions. For example, Stowaway uses static analysis to extract APIs used in apps, and dynamic analysis of the Android OS/stack to build a permission map [1]. A tool named PScout also uses static analysis of the Android OS to map permissions to APIs [2]. In a later work, based on PScout's methodology, Androguard[4)] can output likely APIs to permission mappings for a given app[5)]. However, these tools are based on program analysis, which makes numerous incorrect recommendations.

Recently, an approach using association rule mining, which is a popular data mining technique, was proposed by Karim et al. [4] to recommend the required permissions of an app. They conducted an experiment on 600 apps from F-Droid[6)] and found that their approach, named APMiner, performs better in terms of the F1-score than PScout and Androguard [4]. However, the average F1-score of APMiner is not sufficiently high (only approximately 55%) for it to be used in practice. Moreover, in the recommendation system field, there exist many other algorithms which may be applied to permission recommendation for Android apps. Hence, some alternative recommendation algorithms, which have been used successfully in building recommendation systems, are investigated in this paper. We aim to investigate whether our proposed approaches can outperform Karim et al.'s approach, which is based on association rule mining. To be consistent with the names of our proposed approaches, we refer to the best performing variant of APMiner as APRec$^{\mathrm{RULE}}$.

Our first proposed approach, which we refer to as APRec$^{\mathrm{CF}}$, is based on collaborative filtering, which has been adopted widely in many recommendation systems [5]. The intuition of using collaborative filtering is that apps that use similar APIs often support similar features; thus, they often use similar permissions. Hence, APRec$^{\mathrm{CF}}$ measures the similarity of two apps based on the APIs used by the apps. Given an app, APRec$^{\mathrm{CF}}$ first finds a list of the most similar apps to the target app, and then makes permission recommendations based on the used permissions of these similar apps.

Our second proposed approach to make permission recommendations, referred to as APRec$^{\mathrm{TEXT}}$, is based on text mining. Apps often have associated textual descriptions (e.g., descriptions on Google Play and readme files on Github) that describe the functionalities and features of the apps. These textual

3) https://developer.android.com/training/articles/security-tips.html#RequestingPermissions.

4) https://github.com/androguard/androguard.

5) Desnos A. Androguard: reverse engineering, malware and goodware analysis of android applications. http://code.google.com/p/androguard.

6) https://f-droid.org/.

Table 1 Example of an Android app database

Android app	Transaction	APIs	Permissions
A2DP Volume	T1	android.location.LocationManager	ACCESS_FINE_LOCATION
	T2	android.net.ConnectivityManager	ACCESS_NETWORK_STATE
	T3	android.net.wifi.WifiManager	ACCESS_WIFI_STATE
	T4	android.bluetooth.BluetoothAdapter, android.bluetooth.BluetoothDevice	BLUETOOTH
	T5	android.app.ActivityManager	KILL_BACKGROUND_PROCESSES
	T6	android.media.AudioManager	MODIFY_AUDIO_SETTINGS
	T7	android.app.ActivityManager	RESTART_PACKAGES
	T8	android.os.PowerManager	WAKE_LOCK
Alarm Clock	T9	android.net.ConnectivityManager	ACCESS_NETWORK_STATE
	T10	java.net.URL	INTERNET
	T11	java.lang.Runtime	READ_LOGS
	T12	android.telephony.TelephonyManager	READ_PHONE_STATE
	T13	android.media.MediaPlayer, android.os.PowerManager	WAKE_LOCK
	T14	android.provider.Settings	WRITE_SETTINGS

contents can be used to predict the required permissions. For example, the description of an app named Android-eye is "this is a simple flashlight app, free and without ads"; this indicates that this app needs a permission to allow it to access the camera (as it is a flashlight app). We build a text mining model using the naive Bayes multinomial classification algorithm, which is used to analyze the app descriptions.

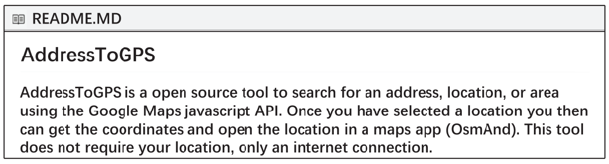
To evaluate these three permission recommendation approaches, an experiment is conducted on 936 open source Android apps from F-droid. These apps also have textual description processed by APRec$^{\text{TEXT}}$ (i.e., readme files) in the corresponding Github repositories. We use several metrics, including precision, recall, F1-score, and mean average precision (MAP), of the top-k recommendations to measure the effectiveness of these approaches. We find that APRec$^{\text{TEXT}}$ and APRec$^{\text{CF}}$ achieve better performance than APRec$^{\text{RULE}}$, but the performance difference between APRec$^{\text{TEXT}}$ and APRec$^{\text{CF}}$ in terms of F1-score@5 and F1-score@10 is very small.

The remainder of the paper is structured as follows. Section 2 describes the intuitions behind the three recommendation approaches studied in this work. Section 3 presents some preliminary concepts. Section 4 describes the details of the three permission recommendation approaches. Section 5 presents the experimental results. Section 6 briefly reviews the related work. Section 7 concludes the study and outlines potential future directions.

2 Motivation

The permission mechanism of Android requires developers to explicitly declare permission requirements if their apps require access to certain sensitive resources, e.g., camera, GPS, and Wi-Fi. Meanwhile, Android provides standard APIs to access these sensitive resources. Hence, developers must know the permissions required by different APIs. Unfortunately, the API documentation is not always helpful in determining the specific permissions required for a specific API. Some tools have been developed to help trace APIs to/from permissions automatically, such as PScout and Androguard. However, these tools are not accurate and many incorrect traceability links have been found.

Table 1 shows two example Android applications in F-Droid: A2DP Volume, namely a Bluetooth management app, and Alarm Clock, which is an alarm clock app. The column "Transaction" in the table indicates the transaction id. These transactions are generated by Androguard. A transaction consists of a set of APIs and a single permission that are traceable to each other. From Table 1, these two apps both have permission ACCESS_NETWORK_STATE and use android.net.ConnectivityManager API. In addition, both have permission WAKE_LOCK and use android.os.PowerManager API. These pieces of


README.MD

AddressToGPS

AddressToGPS is a open source tool to search for an address, location, or area using the Google Maps javascript API. Once you have selected a location you then can get the coordinates and open the location in a maps app (OsmAnd). This tool does not require your location, only an internet connection.

Figure 1 Example of a readme file of an app.

information could be used to help developers determine the permissions required if an app uses certain APIs. For instance, we can infer that another app, Podax, which is a podcast downloader and player, also requires permission ACCESS_NETWORK_STATE, as android.net.ConnectivityManager API is also used by this app. According to this observation, Karim et al. [4] used an association rule mining technique to recommend permissions to a new app, and their approach outperforms both PScout and Androguard.

Intuitively, Android apps with similar features or functionalities often require similar permissions. The APIs of apps can provide clues as to their features or functionalities. For example, an app in F-Droid, named wifiwarning, uses the same APIs (i.e., android.net.ConnectivityManager and android.net.wifi.WifiManager) as wifiwidget and we can infer that they share some features and functionalities. Then, we can infer that they are likely to require similar permissions, which is indeed the case (i.e., ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE). This hints at the use of another information retrieval technique, called collaborative filtering, which has been applied successfully in many real recommendation systems [6] to make permission recommendations.

Moreover, apps usually have associated textual descriptions that usually indicate what the apps are and what types of features they support. For example, Figure 1 is the readme file of app AddressToGPS. This readme file says that the app requires an internet connection, which implies that it needs INTERNET permission. It is often the case that many keywords that correspond to various resources often occur in readme files, e.g., "contacts", "network", and "SD card". This textual information could also be utilized for effective recommendation of permissions.

3 Preliminary

In this section, we describe several techniques used in our study: frequent itemset mining; association rule mining; collaborative filtering with three different similarity metrics: cosine similarity, Euclidean similarity, and Pearson correlation; and texting mining (i.e., classification) based on the naive Bayes multinomial algorithm. Frequent itemset mining is an essential part of association rule mining.

3.1 Frequent itemset mining

Frequent itemset mining [7] is a data mining technique widely used for affinity analysis (e.g., market basket analysis) to discover co-occurrence relationships among entities. It takes a transaction database (i.e., a multi-set of transactions) as input, where each transaction is a set of items, and outputs sets of items (itemsets) that appear frequently (i.e., each frequent itemset is a subset of many transactions) in the database. Given a set of N transactions, the frequency of an itemset I is defined as the number of transactions that contain all of the elements of I. The support of an itemset I is defined as

$$\mathrm{sup}(I) = \frac{\mathrm{freq}(I)}{N}.$$

An itemset is frequent if its support is no less than minsup, where minsup is a user-defined minimum support threshold.

Example 1. Consider the transaction database in Table 1 as an example. If minsup is 0.1, then L = {android.net.ConnectivityManager, ACCESS_NETWORK_STAT} is a frequent itemset in these transactions. L appears in two transactions (i.e., T2, T9); thus, freq(L) is 2. As the number of transactions in the database (N) is 14, sup(L) is 0.14, which is greater than minsup.

3.2 Association rule mining

In addition to frequent itemsets, another type of pattern, termed the association rule, can be extracted from a transaction database. For instance, from the database in Table 1, we can infer that "if an Android app uses API android.net.ConnectivityManager, the app is highly likely to require permission ACCESS_NETWORK_STAT" because all the transactions that contain API android.net.ConnectivityManager also contain permission ACCESS_NETWORK_STAT.

Frequent itemset mining is the first step of association rule mining. We can form association rules by enumerating all possible pairs of frequent itemsets, where one is a subset of another. Consider two frequent itemsets A and B, where A is a subset of B; then, the generated association rule R is of the form

$$A \Rightarrow B \backslash A.$$

Then, we use a metric referred to as support to measure the number of transactions to which the association rules can be applied. Obviously, the support of the association rule R is equal to the support of B, i.e.,

$$\mathrm{sup}(R) = \mathrm{sup}(B).$$

In addition, we use another metric confidence to measure the likelihood that a rule is true, and it can be computed as follows:

$$\mathrm{conf}(R) = \frac{\mathrm{sup}(B)}{\mathrm{sup}(A)}.$$

Example 2. In Table 1, if minsup is 0.1, itemsets A ={android.net.ConnectivityManager} and B = {android.net.ConnectivityManager, ACCESS_NETWORK_STAT} are frequent itemsets with support values of 0.14. Then, we can form an association rule R = android.net.ConnectivityManager $\Rightarrow$ ACCESS_NETWORK_STAT, where the support of R is the same as $\mathrm{sup}(B)$, which is 0.14, and the confidence of R is 1.0 as $\mathrm{sup}(A) = 0.14$.

3.3 Collaborative filtering

As one of the most successful approaches to building recommender systems, collaborative filtering utilizes information of a group of entities to make recommendations or predictions of a new entity. Collaborative filtering has been applied successfully in many real systems, such as environmental sensing, financial services, and electronic commerce [6].

A basic method to perform collaborative filtering is by finding the nearest neighbors of a target entity. The target entity is compared with all other entities and a list of the most similar entities is produced based on a distance metric. The similarities among the entities are used as a basis for making predictions about the entity.

In this study, collaborative filtering is chosen because, intuitively, apps with similar behaviors typically require similar permissions. In our setting, an entity of an Android app, which is represented by the set of APIs it uses, and the recommendation task is the recommendation of permissions that are likely to be required by the app.

3.4 Text mining based on naive Bayes multinomial

The intuition of the text mining model is that Android apps which have similar features and functionalities are often described in a similar way. The similar features and functionalities usually require the same permission. To build a text mining model that can recommend permissions for Android apps, we make use of a text classification technique. In this study, we leverage the naive Bayes multinomial [8], which is a fast and effective algorithm for text classification, to build a text mining model. Many other text mining algorithms, e.g., decision tree and SVM [9], have a long run-time on a raw text dataset, which has many features (every processed word is a feature).

Naive Bayes multinomial is a probabilistic learning method. The probability of a document D being in class C is computed as

$$P(C|D) \propto P(C) \times \prod_{j=0}^{v} P(t_j|C),$$

where $P(C)$ is the prior probability computed as the ratio of the number of documents belonging to class C and the total number of documents, and $P(t_j|C)$ is the conditional probability of term t_k occurring in a document of class C. The conditional probability $P(t_j|C)$ is computed as

$$P(t_j|C) = \frac{T_{C,t_j}}{\sum_{t\in v} T_{C,t}},$$

where T_{C,t_j} denotes the number of times term t_j appears in documents that belong to class C.

4 Approach

In this section, we first briefly describe the procedure of data collection and processing. Then, we introduce the details of three proposed permission recommendation approaches, i.e., APRec$^{\text{RULE}}$, APRec$^{\text{CF}}$, and APRec$^{\text{TEXT}}$.

4.1 Data collection and processing

The Android apps used in this study are sourced from F-Droid, which lists a number of free and open source Android applications. At the time of this study, there are 1993 apps on F-Droid[7]. The Android applications on F-Droid use different platforms to store their code and resource, e.g., Github, Bitbucket, and Google Code. This study only considers the applications whose source code is hosted on Github. This is because the majority of applications on Github have a readme file, and Github provides very convenient REST APIs for access to the readme files. Finally, we locate 936 Android apps with source code in Github and have readme files.

The permissions of an Android app are configured in the manifest file, i.e., `AndroidManifest.xml`, which is located at the root level of the app. Meanwhile, we can find the APIs of an app from its Java source code files, which are declared in import statements. First, we use a tool named srcML, which is a lightweight static analysis tool [10], to transform the source code into a single xml file. Then, it is very easy to extract permissions and APIs using an XML processing tool. Only the API class names in the Android software stack and Java standard libraries are considered (e.g., android.net.ConnectivityManager and java.lang.Runtime), and user-defined classes are ignored.

We use Androguard to obtain the likely mapping of permissions and each API class. We obtain the transactions that are used by APRec$^{\text{RULE}}$ from the output mappings. We transform the list of APIs that are used by an app into a feature vector, which is input to APRec$^{\text{CF}}$. We use the readme files as the input data of APRec$^{\text{TEXT}}$.

4.2 APRec$^{\text{RULE}}$ approach

The baseline approach used in this study refers to APRec$^{\text{RULE}}$, which is an implementation of the best performing variant of APMiner referred to as FilteredMiner in the original paper [4]. Figure 2 shows the process of APRec$^{\text{RULE}}$, which is based on an association rule mining technique, and achieves better performance than two prior state-of-the-art approaches: Androguard and PScout. APRec$^{\text{RULE}}$ use the transactions output from Androguard as the input. Each transaction contains a single permission and several APIs that are traceable to the permission.

There are two phases in APRec$^{\text{RULE}}$: model construction and recommendation phase. In the model construction phase, when given a training set of apps with known permissions, APRec$^{\text{RULE}}$ uses a set of transactions generated by Androguard as an input. To mine API-permission rules (i.e., APIs $\Longrightarrow$

7) https://f-droid.org/wiki/page/Repository_Maintenance.

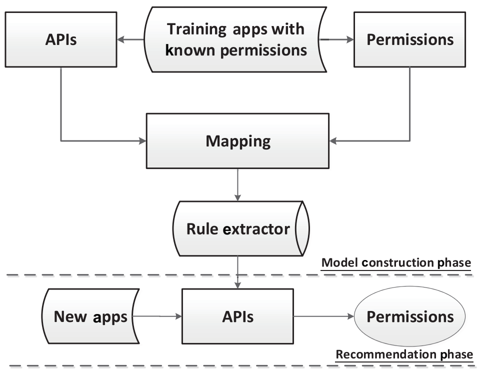


Figure 2 The framework of APRec$^{\text{RULE}}$.

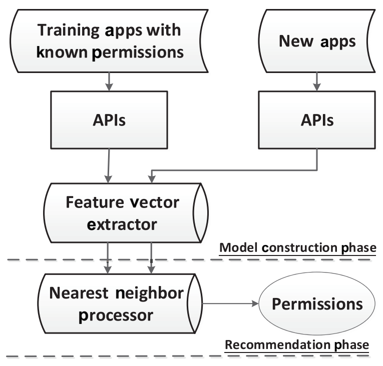


Figure 3 The framework of APRec$^{\text{CF}}$.

Permission), referred to as APRules, association rule mining is applied to the input transactions by a sub-component of APRec$^{\text{RULE}}$ named RuleExtractor. In the recommendation phase, given a new app with a set of APIs, currentAPIs, if a rule's precondition is a subset of currentAPIs, this rule matches currentAPIs. Based on the post-conditions of the matching rules, APRec$^{\text{RULE}}$ then recommends permissions.

A score is assigned by APRec$^{\text{RULE}}$ to assess the probability of a permission required by an app. The rule-based recommendation score for a permission P is the sum of confidence of any matching rule with as post-condition of P. This score is computed by the following formula:

$$\text{RecScore}^{\text{RULE}}(P) = \sum_{R \in \text{RMatched}(P)} \text{conf}(R).$$

In this equation, RMatched(P) is the set of rules of which the pre-condition is a superset of currentAPIs, and the post-condition is a permission P. If the set RMatched(P) is empty, the recommendation score of P is zero. Next, we normalize RecScore$^{\text{RULE}}$ to make the value range from 0 to 1. The permissions with the highest recommendation scores are deemed to be the most appropriate permissions based on the mined association rules.

4.3 APRec$^{\text{CF}}$ approach

Figure 3 presents the process of APRec$^{\text{CF}}$. APRec$^{\text{CF}}$ uses a collaborative filtering technique to make permission recommendations for Android apps. We use three metrics to calculate the similarity between two apps, i.e., cosine similarity, Euclidean distance, and Pearson correlation. Thus, we obtain three variants of APRec$^{\text{CF}}$, referred to as APRec$^{\text{CF}_{\text{cosine}}}$, APRec$^{\text{CF}_{\text{Euclidean}}}$, and APRec$^{\text{CF}_{\text{correlation}}}$, respectively. Given an Android app, APRec$^{\text{CF}}$ transforms the APIs used into a feature vector, and then uses the most similar apps from the training dataset to recommend permissions for the app.

APRec$^{\text{CF}}$ also works in two phases: model construction and recommendation phase. In the model construction phase, a sub-component named FeatureVectorExtractor transforms the APIs used in the apps into feature vectors. In the recommendation phase, a sub-component named NearestNeighborProcessor uses a nearest-neighbor-based collaborative filtering approach to recommend permissions for Android apps.

4.3.1 *FeatureVectorExtractor*

This sub-component transforms the APIs used by each app in a training set into a feature vector. FeatureVectorExtractor first finds all APIs used in the training set, denoted allAPIs. The APIs in allAPIs are sorted in alphabetical order of their names. Then, FeatureVectorExtractor assigns a unique index for each API, which is referred to as allAPIs$[i]$. The feature vector $V(A)$ of an app is defined as follows:

$$V(A) = (\text{ind}(\text{allAPIs}[0], A), \ldots, \text{ind}(\text{allAPIs}[|allAPIs|], A)),$$

where $\text{ind}(I, A) = 1$ if A uses API I, and $\text{ind}(I, A) = 0$, otherwise.

4.3.2 *NearestNeighborProcessor*

Given a new app, NearestNeighborProcessor first uses the same method as FeatureVectorExtractor to transform the APIs used in the app into a feature vector. Then, the similarities are calculated using the distance between this feature vector and the feature vectors of apps in the training set. In this study, three different metrics, i.e., cosine similarity, Euclidean similarity, and Pearson correlation similarity, are used to compute the distance.

The cosine similarity score of a new app A and an existing app B in the training set is calculated as follows:

$$S_{\text{cosine}}(A,B)=\frac{V(A)\cdot V(B)}{|V(A)||V(B)|}.$$

Here, $|V(i)|$ denotes the size of a vector $V(i)$ and $\cdot$ denotes the dot product, which is defined as the square root of the sum of the squares of its constituent elements. Since the term frequencies cannot be negative, cosine similarity ranges from 0 to 1.

The Euclidean similarity is calculated as follows:

$$S_{\text{Euclidean}}(A,B)=1\bigg/\sqrt{\sum_{i=1}^{n}(A_i-B_i)^2}\,.$$

Here, n is the size of vector. The Euclidean similarity score is the inverse of the Euclidean distance.

The Pearson correlation is calculated as follows:

$$\text{Correlation}(A,B)=\frac{\text{COV}(A,B)}{\sigma_A\sigma_B}.$$

Here, $\text{COV}(A,B)$ is the covariance between A and B, whereas σ_A and σ_B are the standard deviations of A and B, respectively. The value of $\text{COV}(A,B)$ ranges from -1 to 1. Then, the correlation similarity score is calculated as

$$S_{\text{correlation}}(A,B)=\frac{\text{Correlation}(A,B)+1}{2}.$$

Thus, the correlation similarity score is a normalized value in the range of $[0,1]$.

For all three similarity scores, the higher the similarity score, the more similar the app in the training set is to the new app. APRec$^{\text{CF}}$ ranks apps in the training data based on their similarity scores. Then, top-n apps with the highest similarity score as the nearest neighbors of the new app are selected from the training set. Next, given a permission P, APRec$^{\text{CF}}$ computes a collaborative filtering-based recommendation score for a permission P as follows:

$$\text{RecScore}^{\text{CF}}(P)=\sum_{B_i\in\text{Nearest}}S_x(A,B_i),\ \text{if } P\in B_i.$$

Here, Nearest is the nearest neighbor, $P\in B_i$ means that app B_i has permission P, and S_x is one of three similarity scores. Then, we normalize the recommendation score to force it into the range from 0 to 1. Finally, based on the collaborative filtering, the recommended permissions for the app are the permissions with the highest recommendation scores.

4.4 APRec$^{\text{TEXT}}$ approach

Figure 4 presents the process of APRec$^{\text{TEXT}}$. APRec$^{\text{TEXT}}$ leverages the naive Bayes multinomial, which is a fast and effective algorithm for text classification to build a text mining model. The textual descriptions of Android apps used in this study are their readme files in Github, which usually contain the apps' descriptions, features, functions, and licenses. We train a text classifier for each permission, and then combine all classifiers to recommend permissions for a new app.

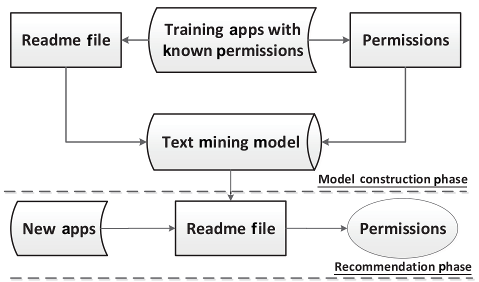


Figure 4 The framework of APRec$^{\text{TEXT}}$.

The approach APRec$^{\text{TEXT}}$ recommends permissions using readme files of apps in the training set to build a text mining model using the naive Bayes multinomial classification technique. In the model construction phase, we consider all n potential permissions for an app and denote the jth permission as P^j. APRec$^{\text{TEXT}}$ first duplicates the training dataset into a n binary dataset, one for each permission. Each app (A_i) in the jth training dataset contains two parts: the textual content in the readme file (Text$_i$) and a binary value R_i^j that indicates whether the app A_i has the permission P^j ($R_i^j = 1$) or not ($R_i^j = 0$). Following vector space modeling [11], we represent the text of the readme file of the ith app as a vector of term weights denoted $\text{Text}_i = \langle w_1, w_2, \ldots, w_3 \rangle$. The weight w_j denotes the number of times the jth term t_j appears in the textual content of the readme file of the ith app. Next, we construct n prediction models based on the n binary datasets by leveraging the naive Bayes multinomial technique. Further detail regarding the naive Bayes multinomial technique is given in Subsection 3.4.

In the recommendation phase, given a new app A_{new}, we first input its readme file into the n prediction models, and each prediction model outputs a recommendation score $\text{RecScore}^{\text{TEXT}}(A_{\text{new}}, P^i)$ that indicates the likelihood that A_{new} has permission P^i. The recommendation score $\text{RecScore}^{\text{TEXT}}(A_{\text{new}}, P^i)$ is computed as follows:

$$\text{RecScore}^{\text{TEXT}}(A_{\text{new}}, P^i) = P(P^i = 1) \times \prod_{j=0}^{v} P(t_j | P^i = 1).$$

Here, $P(P^i = 1)$ represents the prior probability that A_{new} has permission P^i, and $P(t_j|P^i = 1)$ represents the conditional probability that term t_j occurs in the readme file of A_{new}, which has permission P^i. Next, we rank the permissions based on these recommendation scores, and output the permissions with the highest scores.

5 Results

In this section, we first briefly describe the experimental setup and our evaluation metrics. Then, we present the three research questions of our study, and the corresponding experiment results. Finally, some threats to the validity are discussed.

5.1 Experiment setup

We use 936 Android apps from F-Droid as the dataset in this study. The average and standard deviation of the permissions and APIs of the apps in the dataset are 4.45 ± 2.57 and 104.01 ± 83.59, respectively. Then, the APIs that are not traceable to used permissions are excluded using Androguard. Thus, the apps have 8.31 ± 5.35 traceable APIs on average.

To provide a baseline model, we first implement the approach of Karim et al. We use a fast Apriori [12] algorithm implementation [13] for association rule mining. The minimum confidence value in association rule mining is set to 0.4, because Karim et al.'s approach achieves the best performance when the minimum confidence threshold is set to 0.4. The default numbers of the nearest neighbors for all three collaborative filtering approaches are set to 10.

The experimental environment is a 64-bit, Intel(R) Core(TM) i7-6500 2.50 GHz computer with 8 GB RAM running Windows 10.

5.2 Evaluation metrics

We use some evaluation metrics: precision@k, recall@k, F1-score@k and the mean average precision (MAP), which have been used in many studies (e.g., [14–16]), to evaluate the three permission recommendation approaches in this study. Consider that there are m Android apps in the testing dataset that require permission recommendation. For each app A_i, let the actual set of permissions of A_i be P_i^t, and N_i^k be the number of permissions that are correctly recommended in the top-k permissions P_i^k recommended by a permission recommendation system. Precision@k is the ratio of N_i^k to k, i.e.,

$$(\text{precision@}k)_i = \frac{N_i^k}{k},$$

whereas recall@k is the ratio of N_i^k to the actual number of permissions, i.e.,

$$(\text{recall@}k)_i = \frac{N_i^k}{|P_i^t|}.$$

Then, F1-score@k is a summary measure that combines both precision@k and recall@k, i.e.,

$$(F1\text{-score@}k)_i = 2 \times \frac{(\text{precision@}k)_i \times (\text{recall@}k)_i}{(\text{precision@}k)_i + (\text{recall@}k)_i}.$$

Finally, for m apps in a testing dataset, we calculate the average precision@k, recall@k, and F1-score@k.

As a permission recommendation system returns a ranked list of permissions, it is desirable to also consider the order in which the returned permissions are presented. Hence, we also use MAP, which is one of the most popular measures to evaluate ranked retrieval results as an evaluation metric. MAP is known to be a stable [17] and highly informative [18] measure. In this study, average precision (AP) is the average of the precisions computed at the point of each of the permissions that are recommended correctly in the ranked permission list. It is computed as follows:

$$\text{AP} = \frac{\sum_{k=1}^{n}(P(k) \times \text{rel}(k))}{\text{the number of actual permissions}},$$

where, k is the rank in the sequence of recommended permissions, n is the number of recommended permissions, $P(k)$ is the precision at cut-off k in the list, and rel(k) is an indicator function that is equal to one if the item at rank k is a permission used by the target app, and zero otherwise. Then, for the m apps in the testing dataset, MAP is calculated as follows:

$$\text{MAP} = \frac{\sum_{i=1}^{m}\text{AP}_i}{m}.$$

5.3 Research questions

RQ1: How effective are the two permission recommendation systems based on text mining and collaborative filtering? How much improvement can these two approaches achieve over the baseline approach?

Motivation. The better the performance of APRec$^{\text{TEXT}}$ and APRec$^{\text{CF}}$, the greater the benefit to their users. Thus, in this research question, we aim to investigate whether our proposed approaches of APRec$^{\text{TEXT}}$ and APRec$^{\text{CF}}$ can outperform the baseline approach APRec$^{\text{RULE}}$.

Approach. To answer RQ1, we use 10-fold cross validation to estimate the results of these three permission recommendation approaches. To evaluate their performance, we compute the top-k precision, recall, F1-score ($k = (1, 2, \ldots, 10)$), and MAP.

To check whether the differences between the results of our proposed approaches and the baseline APRec$^{\text{RULE}}$ are significant, we apply the Wilcoxon signed-rang test [19] at 95% significance level on 10-fold metrics.

In addition, we compute Cliff's delta (δ) [20], which is a non-parametric effect size measure that quantifies the differences between the results of our proposed approaches and the baseline APRec$^{\text{RULE}}$. Table 2 defines the different Cliff's delta values and their corresponding interpretations.

Table 2 Cliff's delta and the effectiveness level [20]

Cliff's delta ($\|\delta\|$)	Effectiveness level
$\|\delta\| < 0.147$	Negligible
$0.147 \leqslant \|\delta\| < 0.33$	Small
$0.33 \leqslant \|\delta\| < 0.474$	Medium
$0.474 \leqslant \|\delta\|$	Large

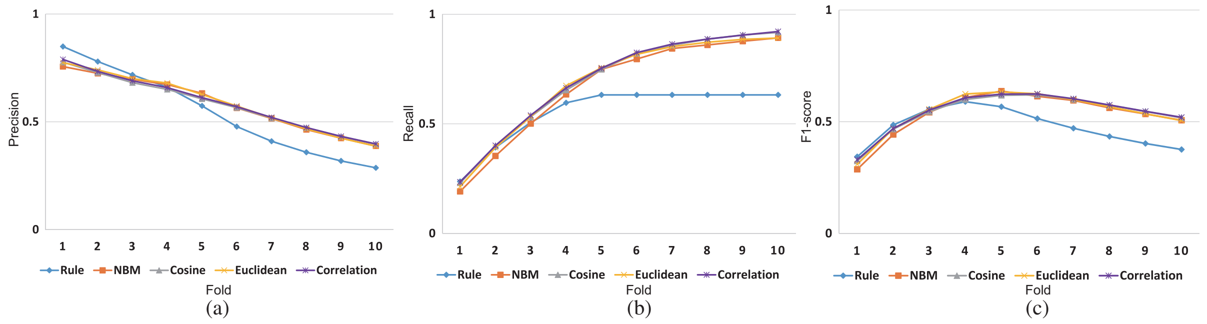


Figure 5 (Color online) The results of (a) top (1–10) precision, (b) recall, and (c) F1-score.

Table 3 Results of precision@k, recall@k, F1-score@k (k=5, 10) and MAP

	Precision@5	Recall@5	F1-score@5	Precision@10	Recall@10	F1-score@10	MAP
APRec$^{\rm RULE}$	0.5739	0.6322	0.5671	0.2869	0.6322	0.3759	0.6275
APRec$^{\rm TEXT}$	0.6322	0.7487	0.6371	0.3881	0.8925	0.5070	0.7474
APRec$^{\rm CF_{cosine}}$	0.6064	0.7487	0.6183	0.3960	0.9172	0.5176	0.7651
APRec$^{\rm CF_{Euclidean}}$	0.6265	0.7560	0.6336	0.3870	0.8916	0.5054	0.7439
APRec$^{\rm CF_{correlation}}$	0.6121	0.7540	0.6233	0.3976	0.9215	0.5198	0.7693

Results. Figure 5(a)–(c) presents the results of top-k precision, recall, and F1-score. In these figures, 'Rule', 'NBM', 'Cosine', 'Euclidean' and 'Correlation' represent APRec$^{\rm RULE}$, APRec$^{\rm TEXT}$, APRec$^{\rm CF_{cosine}}$, APRec$^{\rm CF_{Euclidean}}$, and APRec$^{\rm CF_{correlation}}$, respectively. We find that as the number of recommended permissions (i.e., top-k) increases, the precisions of all approaches decrease, but the recalls increase. This might be because when k is small, the permissions with high recommendation scores are more likely to be correct, which leads to high precision. However, more permissions will be wrongly recommended as k increases. In contrast, for recall, the greater the number of permissions recommended, the higher the recall.

Further, we find that the precision of baseline APRec$^{\rm RULE}$ is higher than that of other approaches when k is small (from 1 to 4), but the differences are very small. Moreover, Figure 5(b) and (c) shows that when k is small, the recalls and F1-scores of APRec$^{\rm RULE}$ are almost the same as those of other approaches. However, the precisions, recalls, and F1-scores of APRec$^{\rm RULE}$ are all smaller than those of other approaches when k is greater than five. The precisions of APRec$^{\rm RULE}$ decrease much faster than those of the others, but its recalls show almost no increase when k is greater than five. Its F1-score, which is a harmonic mean of precision and recall, also decreases very rapidly. However, the differences in the precision results, recalls, and F1-scores for the other approaches are very small when k varies from one to ten. In summary, we find that our proposed recommendation approaches, i.e., APRec$^{\rm TEXT}$ and APRec$^{\rm CF}$, achieve better performance than the baseline approach APRec$^{\rm RULE}$.

Table 3 presents the detailed top-k (k=5, 10) precision, recall, F1-score, and MAP for all approaches. We find that all metrics of APRec$^{\rm RULE}$ are smaller than those of other approaches. The precision@5 and F1-score@5 of APRec$^{\rm TEXT}$ are the largest and they outperform those of APRec$^{\rm RULE}$ by ∼6% and 7%, respectively. The recall@5 of APRec$^{\rm CF_{Euclidean}}$ is the largest, which is an improvement of ∼12% over that of APRec$^{\rm RULE}$. The improvements of all newly proposed approaches over APRec$^{\rm RULE}$ of the top-10 metrics are larger than those of the top-5 metrics. These improvements in precision@10, recall@10, and F1-score@10 are greater than 10%, 25%, and 13%, respectively. The MAP of APRec$^{\rm CF_{correlation}}$ is the

Table 4 P-value and Cliff delta (δ) for APRec$^{\text{TEXT}}$ and APRec$^{\text{CF}}$ with the baseline APRec$^{\text{RULE}}$

	Precision@5		Recall@5		F1-score@5		Precision@10		Recall@10		F1-score@10		MAP	
	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ
APRec$^{\text{TEXT}}$	<0.0001	0.78	<0.0001	0.89	<0.0001	0.85	<0.0001	0.9	<0.0001	0.9	<0.0001	0.9	<0.0001	0.9
APRec$^{\text{CF}_{\text{cosine}}}$	6.43E$-$03	0.49	<0.0001	0.89	<0.0001	0.8	<0.0001	0.9	<0.0001	0.9	<0.0001	0.9	<0.0001	0.9
APRec$^{\text{CF}_{\text{Euclidean}}}$	2.95E$-$03	0.73	<0.0001	0.9	<0.0001	0.85	<0.0001	0.9	<0.0001	0.9	<0.0001	0.89	<0.0001	0.89
APRec$^{\text{CF}_{\text{correlation}}}$	<0.0001	0.6	<0.0001	0.89	<0.0001	0.76	2.95E$-$03	0.9	<0.0001	0.9	<0.0001	0.9	<0.0001	0.9

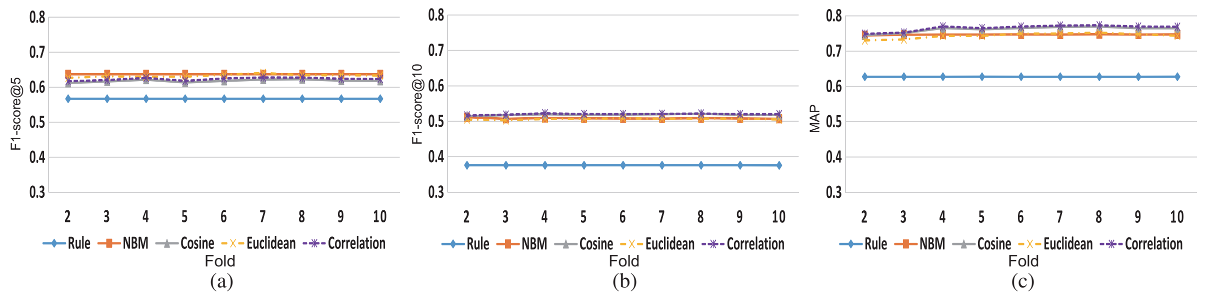


Figure 6 (Color online) (a) F1-score@5, (b) F1-score@10, and (c) MAP for N-Fold cross-validation.

largest, which is an improvement of ∼14% over that of APRec$^{\text{RULE}}$. Moreover, the differences between the results of the four newly proposed approaches are minor.

Table 4 presents the p-values and Cliff's delta values for APRec$^{\text{TEXT}}$, APRec$^{\text{CF}_{\text{cosie}}}$, APRec$^{\text{CF}_{\text{Euclidean}}}$, and APRec$^{\text{CF}_{\text{correlation}}}$ with the baseline APRec$^{\text{RULE}}$ in terms of precision@k, recall@k, F1-score@k (k=5, 10), and MAP. We find that all the p-values are less than 0.05 and all the Cliff's delta values are at a large effectiveness level. This indicates that the improvement of each approach over the baseline APRec$^{\text{RULE}}$ is significant.

RQ2: How does the size of training data affect the results of the permission recommendation approaches?

Motivation. We aim to investigate whether different sizes of training data affect the performance of the permission recommendation approaches investigated in this study.

Approach. We run a n-fold cross-validation to evaluate the performance of each approach, where n ranges from 2 to 10. As we reduce the value of n, we reduce the amount of training data. We evaluate the results in terms of F1-score@5, F1-score@10, and MAP.

Results. Figure 6(a)–(c) presents the F1-score@5, F1-score@10, and MAP of these permission recommendation approaches for different n-fold cross-validations, respectively. We find that, for all approaches, the F1-score@5, F1-score@10, and MAP change very little when the size of the training dataset is varied. Hence, the permission recommendation approaches proposed in our study perform very well across a wide range of training data sizes.

RQ3: How does the number of the nearest neighbors affect the results of the permission recommendation approaches using collaborative filtering?

Motivation. By default, the number of the nearest neighbors used in our collaborate filtering based approaches is ten. In this research question, we aim to investigate whether different numbers of the nearest neighbors could affect the performance of these approaches.

Approach. To answer this research question, the number of the nearest neighbors is varied (1, 5, 10, 15, and 20). For each collaborative filtering approach, we run a 10-fold cross-validation and evaluate the results in terms of F1-score@5, F1-score@10, and MAP.

Results. Figure 7(a)–(c) presents the F1-score@5, F1-score@10, and MAP, respectively, of the permission recommendation approaches based on collaborative filtering using different numbers of the nearest neighbors. We notice that when the number of the nearest neighbors increases from 1 to 5, the improvement in F1-score@5, F1-score@10, and MAP is relatively high for all three approaches; however, when the number of the nearest neighbors continues to increase, the improvement is not substantial.

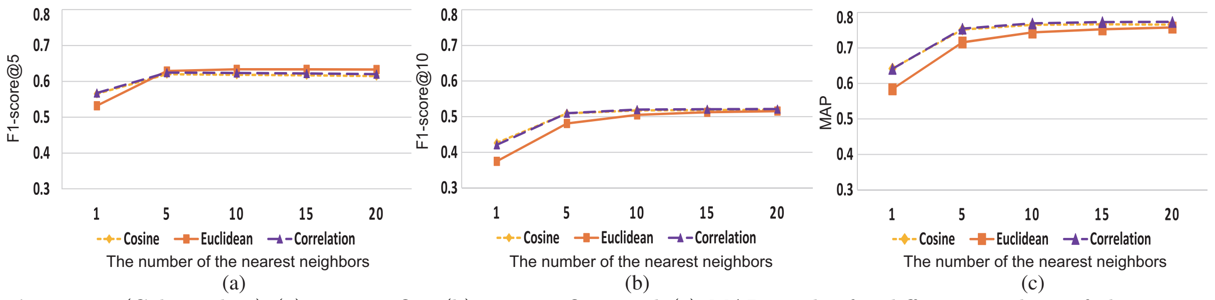


Figure 7 (Color online) (a) F1-score@5, (b) F1-score@10, and (c) MAP results for different numbers of the nearest neighbors.

Table 5 Average time of one round in the 10-fold cross-validation for all approaches

Approach	Time (s)
APRec$^{\mathrm{RULE}}$	12.4
APRec$^{\mathrm{TEXT}}$	210.1
APRec$^{\mathrm{CF}_{\mathrm{cosine}}}$	208.5
APRec$^{\mathrm{CF}_{\mathrm{Euclidean}}}$	202.3
APRec$^{\mathrm{CF}_{\mathrm{correlation}}}$	283.6

We can see that, even if the number of the nearest neighbors is set to one, the results of F1-score@5, F1-score@10, and MAP are similar to those of the baseline APRec$^{\mathrm{RULE}}$. Meanwhile, the number of the nearest neighbors does not affect the performance of collaborative filtering approaches considerably when it varies from 5 to 20. Thus, it is acceptable to set the default value to ten in RQ1.

RQ4: How much time does it take for each permission recommendation approach to run?

Motivation. The permission recommendation systems evaluated in this work require substantial computational time to build models and recommend permissions for an app. For the association rule approach, extracting rules from training data is the most time-consuming step. For the collaborative filtering approaches, considerable time is required to determine the nearest neighbors. For the text mining approach, multiple naive Bayes classifiers take a considerable time to be built. Thus, in this research question, we aim to investigate the time efficiency of these permission recommendation approaches.

Approach. For each approach, we run a 10-fold cross-validation and report the average time of each round.

Results. Table 5 gives the average time of one round of the 10-fold cross-validation for all permission recommendation approaches investigated in this study. We can see that APRec$^{\mathrm{RULE}}$ runs considerably faster than other approaches, and only requires 12 s/round. This is because there are not many items in the transactions in our dataset. The slowest approach is APRec$^{\mathrm{CF}_{\mathrm{correlation}}}$; this takes approximately one minute longer than the other three approaches. Although the duration of these proposed approaches is longer than that of the baseline APRec$^{\mathrm{RULE}}$, we believe it is acceptable. For APRec$^{\mathrm{TEXT}}$, the naive Bayes classifiers need to be built only once, and it takes much less time to recommend permissions in the recommendation phase. Moreover, for APRec$^{\mathrm{CF}}$, it can recommend permissions to an app within seconds – nearly 100 apps are tested in each round of the 10-fold cross-validation in our dataset.

5.4 Discussion

In this paper, we propose two approaches: APRec$^{\mathrm{CF}}$ and APRec$^{\mathrm{TEXT}}$, to recommend permissions to developers. Our experiment shows that both achieve good performance. However, there exist several limitations for these two approaches.

For APRec$^{\mathrm{CF}}$, we recommend permissions based on the training data. As the amount of training data increases, an app will require more time to find similar apps and recommend permissions based on the similar apps. Furthermore, the quality of recommendation is dependent on the training data.

APRec$^{\mathrm{TEXT}}$ might not detect some permissions for a given app as its text description may not explain the hidden internal workings of the app and only provides a high-level overview of the functionalities performed. Besides, certain permissions might not be directly related to app functionality. For example, it is quite difficult to capture the following permissions based on the app's textual description: BROADCAST_STICKY, which allows an app to broadcast sticky intents, and READ_SYNC_STATS, which allows apps to read the sync stats. These two permissions are not related directly to the app functionalities, which are difficult to infer from the textual descriptions of the app. To mitigate this, given an app, APRec$^{\text{TEXT}}$ finds other apps with similar text descriptions and recommends permissions based on those used in the similar apps. For example, Physics Drop[8] and Flow Free[9] are two puzzle game apps with similar text descriptions. They require WAKE_LOCK permission, which is not apparent from their descriptions. Based on the similarity between these two apps, APRec$^{\text{TEXT}}$ regards them as similar apps and recommend one's permissions to the other. However, not all apps with similar textual descriptions use the same permissions. Thus, APRec$^{\text{TEXT}}$ might make an inaccurate permission recommendation. To improve APRecTEXT further, it will be interesting to investigate new approaches that can better understand the semantics of text descriptions of apps. Related to this research direction, recently, Qu et al. [21] found text patterns of 11 Android permissions by using NLP techniques. Qu et al.'s text patterns can potentially be combined with APRec$^{\text{TEXT}}$ to help improve the accuracy of its recommendations.

5.5 Threats to validity

Threats to internal validity refer to errors in our code and experiment bias. We double-check our code; however, some errors may not have been noticed. Another threat to internal validity is that the declared permissions in the apps may be incorrect. We randomly choose 20 apps in our dataset and find that the declared permissions in these apps are indeed required. Thus, we believe the usage of permissions for most of apps in our dataset is correct.

Threats to external validity refers to the generalizability of the dataset used in our study. In our study, we use 936 open source Android apps from F-Droid. In the future, more open source Android apps, even closed source apps, will be considered to reduce this threat to validity. To extract the API usage of closed source apps (e.g., those distributed on Google Play), we require some existing tools (e.g., Androguard) to reverse engineer them. Another threat to external validity is that we do not cover all permissions. Only 45 of the 151 system-defined permissions in Android are used in our dataset. Besides, the customized permissions are not considered.

Threats to construct validity refer to the suitability of our evaluation measures. We use top-k precision, recall, and F1-score, and MAP, which have also been used in prior studies, to evaluate the effectiveness of various software engineering studies [14–16]. Thus, we believe there is little threat to construct validity.

6 Related work

The work of Karim et al. [4], which uses the association rule mining technique to recommend permissions, gives us hint to use other algorithms to make permission recommendation. Karim et al. extract rules based on the co-occurrence of Android APIs and permissions. To the best of our knowledge, there are no other works that utilize recommendation system algorithms to predict the required permissions for an app. In this study, we use two other recommendation system algorithms: collaborative filtering and text mining, to make permission recommendation. We use the approach based on the association rule as the baseline, and find that our approaches outperform the baseline approach.

There are many different approaches proposed by researchers to identify the mapping between APIs and permissions. Vidas et al. [22] propose an approach that scans the Android documentation to extract a permission specification. However, the incompleteness of the Android documentation affects the usage of their work in practice. Several tools extract the mappings between APIs and permissions by static analysis, such as Stowaway [1], PScout [2], and Androguard. Of these tools, PScout has performs better than Stowaway. It has been applied on four versions of Android and helps to determine that ∼22%

8) https://play.google.com/store/apps/details?id=com.dreamed.physicsdrop.
9) https://play.google.com/store/apps/details?id=com.bigduckgames.flow.

of the non-system permissions are unnecessary. Androguard, a reverse engineering tool, embeds the methodology of PScout and enables API to make permission recommendations for Android apps. In this study, we use Androguard to extract the mappings between APIs and permissions. Notice that a lot of mappings recovered by these program analysis approaches are not correct. We do not compare our proposed approaches with these tools in this study; this is because the study of Karim et al. has shown that their approach based on association rule mining outperforms these program analysis tools. Qu et al. [21] investigated the relationships between app descriptions and permissions (referred to as description-to-permission fidelity) by leveraging natural language processing (NLP) techniques. Their study investigated 11 permissions and highlighted text patterns that correspond to each of these permissions. Their text patterns can be used to aid permission recommendations, but the number of permissions investigated in their study is too small. Our proposed approach APRec$^{\text{TEXT}}$ leverages text classification techniques to recommend permissions for apps. Thus, the number of permissions, which is based on the training data, can be considerably larger than that of the approach of Qu et al. However, as APRec$^{\text{TEXT}}$ does not understand the semantic of app descriptions, it might make an inaccurate permission recommendation. In the future, to improve the performance of APRec$^{\text{TEXT}}$, we plan to use NLP techniques to better understand app textual descriptions.

Many researchers have investigated the misuse of permissions in Android. An internet survey with 308 Android users and a laboratory study with 25 users were conducted by Felt et al. [23] to understand whether Android users pay attention to, understand, and act on permission information during installation. Only 3% of internet survey respondents answered all three permission comprehension questions correctly, and very few users (17%) care about permission information during installation. Stevens et al. [3] found that the popularity of a permission is strongly associated with its misuse through a study conducted on the apps from the Android market and questions regarding security permission use on StackOverflow. The above work provides the motivation for our work. If we can recommend permissions for both developers and users effectively, the phenomenon of permission misuse might decrease.

As permission misuse could be indicative of stealthy and malicious behavior in Android apps, we could also use the permission recommendation approaches in this study to detect malicious behavior. Researchers have proposed many approaches to identify malicious behavior for Android apps [24–29]. For instance, Bläsing et al. [26] use static and dynamic analysis to detect suspicious Android apps. Zhou et al. [27] propose a permission-based behavioral footprinting scheme to detect Android malware. AsDroid [24] analyze user interface and program behavior contradiction to identify stealthy behavior.

7 Conclusion and future work

In this paper, we propose two permission recommendation approaches, inspired by the work of Karim et al. [4]. Karim et al. propose an approach based on association rule mining to recommend permissions for Android apps. Our two proposed approaches are based on the collaborative filtering technique and text mining (in particular, text classification). For the collaborative filtering approach, we also investigate three different similarity metrics (i.e., cosine similarity, Euclidean similarity, and Pearson similarity). To evaluate the effectiveness of these approaches, we conduct an experiment on 936 Android apps from F-Droid and compare our proposed approaches with the association rule approach proposed by Karim et al. The experimental results show that our two proposed approaches perform better than the baseline approach in terms of top-k precision, recall, F1-score, and MAP.

In the future, we will consider the use of more Android apps, including closed source apps (e.g., apps in Google Play,) to evaluate our proposed permission recommendation approaches. In addition, we aim to improve the effectiveness of the proposed approach by using more sophisticated machine-learning techniques, such as learning to rank and deep learning.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61602403, 61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China (Grant No. 2015BAH17F01).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Felt A P, Chin E, Hanna S, et al. Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. New York: ACM, 2011. 627–638

2 Au K W Y, Zhou Y F, Huang Z, et al. Pscout: analyzing the android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. New York: ACM, 2012. 217–228

3 Stevens R, Ganz J, Filkov V, et al. Asking for (and about) permissions used by android apps. In: Proceedings of the 10th Working Conference on Mining Software Repositories. Piscataway: IEEE Press, 2013. 31–40

4 Karim M Y, Kagdi H, Di Penta M. Mining android apps to recommend permissions. In: Proceedings of the 23th IEEE/ACM International Conference on Software Analysis, Evolution, and Reengineering. Piscataway: IEEE Press, 2016. 427–437

5 Ricci F, Rokach L, Shapira B. Introduction to Recommender Systems Handbook. Berlin: Springer, 2011

6 Su X Y, Khoshgoftaar T M. A survey of collaborative filtering techniques. Adv Artif Intell, 2009, 2009: 1–19

7 Agrawal R, Srikant R. Mining sequential patterns. In: Proceedings of the 11th International Conference on Data Engineering. Piscataway: IEEE Press, 1995. 3–14

8 McCallum A, Nigam K. A comparison of event models for naive bayes text classification. In: Proceedings of AAAI-98 Workshop on Learning for Text Categorization, Madison, 1998. 41–48

9 Han J W, Kamber M, Pei J. Data Mining: Concepts and Techniques. Waltham: Elsevier, 2011. 744

10 Collard M L, Kagdi H H, Maletic J I. An xml-based lightweight C++ fact extractor. In: Proceedings of the 11th IEEE International Workshop on Program Comprehension. Piscataway: IEEE Press, 2003. 134–143

11 Baeza-Yates R, Ribeiro-Neto B. Modern Information Retrieval. New York: ACM, 1999

12 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc., 1994. 487–499

13 Bodon F. A fast apriori implementation. In: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03). Piscataway: IEEE Press, 2003

14 Rao S, Kak A. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In: Proceedings of the 8th Working Conference on Mining Software Repositories. Piscataway: IEEE Press, 2011. 43–52

15 Xia X, Lo D, Wang X, et al. Cross-language bug localization. In: Proceedings of the 22nd International Conference on Program Comprehension. New York: ACM, 2014. 275–278

16 Zhou J, Zhang H, Lo D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: Proceedings of the 34th International Conference Software Engineering (ICSE). Piscataway: IEEE Press, 2012. 14–24

17 Buckley C, Voorhees E M. Evaluating evaluation measure stability. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM, 2000. 33–40

18 Aslam J A, Yilmaz E, Pavlu V. The maximum entropy method for analyzing retrieval measures. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM, 2005. 27–34

19 Wilcoxon F. Individual comparisons by ranking methods. Biometrics Bull, 1945, 1: 80–83

20 Cliff N. Ordinal Methods for Behavioral Data Analysis. London: Psychology Press, 2014

21 Qu Z, Rastogi V, Zhang X, et al. Autocog: measuring the description-to-permission fidelity in Android applications. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2014. 1354–1365

22 Vidas T, Christin N, Cranor L. Curbing android permission creep. In: Proceedings of IEEE Web 2.0 Security and Privacy Workshop, Oakland, 2011. 91–96

23 Felt A P, Ha E, Egelman S, et al. Android permissions: user attention, comprehension, and behavior. In: Proceedings of the 8th Symposium on Usable Privacy and Security. New York: ACM, 2012. 3

24 Huang J, Zhang X, Tan L, et al. Asdroid: detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In: Proceedings of the 36th International Conference on Software Engineering. New York: ACM, 2014. 1036–1046

25 Schmidt A D, Schmidt H G, Batyuk L, et al. Smartphone malware evolution revisited: Android next target? In: Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE). New York: ACM, 2009. 1–7

26 Bläsing T, Batyuk L, Schmidt A D, et al. An android application sandbox system for suspicious software detection. In: Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE). Piscataway: IEEE Press, 2010. 55–62

27 Zhou Y J, Wang Z, Zhou W, et al. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, 2012. 25: 50–52

28 Chan P P, Hui L C, Yiu S M. Droidchecker: analyzing android applications for capability leak. In: Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks. New York: ACM, 2012. 125–136

29 Wu D J, Mao C H, Wei T E, et al. Droidmat: Android malware detection through manifest and API calls tracing. In: Proceedings of the 7th Asia Joint Conference on Information Security (Asia JCIS). Piscataway: IEEE Press, 2012. 62–69