SCIENCE CHINA
Information Sciences

• **RESEARCH PAPER** •

Automated Android App Permission Recommendation

Lingfeng Bao[1], David Lo[2], Xin Xia[1,3]* & Shanping Li[1]

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

Abstract As Android is one of the most popular open source mobile platforms, ensuring security and privacy of Android applications is very important. Android provides a permission mechanism which requires developers to declare sensitive resources their applications need, and users need to agree with this request when they install these applications (Android API Level 22 or lower) or during runtime (Android API Level 23 or higher). Although Android provides very good official documents to explain how to properly use permissions, unfortunately misuses even for the most popular permissions have been reported.
Recently, Karim *et al.* propose an association rule mining based approach to better infer permissions that an API needs. In this work, to improve the effectiveness of the prior work, we propose two approaches which are based on popular techniques used to build recommendation systems. One approach is based on collaborative filtering technique, which is designed based on the intuition that apps that have similar features – inferred from the APIs that they use – usually share similar permissions. Another approach utilizes a text mining technique, which analyzes descriptions of apps and builds a model using the naive Bayes multinomial classification algorithm, to recommend permissions. We evaluate these approaches on 936 Android apps from F-Droid, which is a repository of free and open source Android applications. The experimental results show that our proposed approaches achieve significant improvement in terms of the precision, recall, F1-score and MAP of the top-k results over Karim *et al.*'s approach.

Keywords Android, Permission Recommendation, Association Rule, Collaborative Filtering, Text Mining

Citation Lingfeng Bao, David Lo, Xin Xia and Shanping Li. Automated Android App Permissions Recommendation. Sci China Inf Sci, for review

1 Introduction

Android has become a very popular platform that dominated the smartphone market with a market share of 82.8% in the second quarter of 2015 [1]. More and more Android applications (also referred to as "apps") are produced by thousands of developers. In the first quarter of 2016, there are about 1,900,000 apps in *Google Play* [2]. Meanwhile, the huge number of Android apps attracts more attackers to develop malicious apps, which are often designed to steal sensitive data, such as private credentials and financial information.

* Corresponding author (email: xxia@zju.edu.cn)

In order to decrease the threats that Android apps pose to the privacy and security of their users, Android provides a unique permission mechanism to control access of third party applications to sensitive resources, such as the user's contact list, camera, network, etc. Android requires app developers to write the needed permissions explicitly in a config file named `AndroidManifest.xml`. Hence, 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 example, if an app requires internet access, the developer not only needs to know the network APIs, such as "android.net.ConnectivityManager" and "java.net.Socket", but also the corresponding permissions namely ACCESS_NETWORK_STATE and INTERNET which need to be written to the AndroidManifest.xml file.

To reduce security risk, Android official document[1)] mentions that it is better for developers to minimize the number of permissions that their apps request. However, often it is not easy for Android app developers to decide which permissions are needed. Felt *et al.* report that developers usually require more permissions than the app need [3]. This is because if an app uses services whose permission is not declared, the app will throw an exception. So, in order to ensure good user experience, developers will include more permissions that they need. The penitential cause of the overprivileged apps is that the official Android documentation for API classes and permissions is incomplete [3, 4]. Furthermore, there are 151 system-level permissions available and over 4,000 classes in Android library. Stevens *et al.* have also shown that there exist many misuses even for the most popular permissions [5]. Hence, there is a need for a recommendation system that can help developers decide suitable permissions for their apps.

To address this need, some researchers have proposed some tools to recommend permissions by tracing APIs to specific permissions. *Stowaway* extracts APIs used in apps through static analysis and builds a permission map through dynamic analysis of the Android OS/stack [3]. In a later work, Au *et al.* propose *PScout* which maps permissions to APIs based on the static analysis of the Android OS [4]. *Androguard*[2)] builds upon *PScout*'s methodology to also output likely API to permission mappings for a given app [6]. Unfortunately, these program analysis approaches are not perfect and many wrong recommendations are made.

Recently Karim *et al.* process likely API to permission mappings output by *Androguard* using association rule mining, a popular data mining technique, to recommend the required permissions of an app [7]. An experiment is conducted on 600 apps from F-Droid [3)] and the results show that their proposed approach named *APMiner* outperforms *PScout* and *Androguard* [7]. Although their approach outperforms tools which are based on static analysis, the averaged F1-score of their approach is around 55%, which may not be high enough for it to be used in practice. Moreover, there are many other algorithms proposed in the recommendation system area which could have been applied to recommend permissions for Android apps. Hence, in this paper, we want to investigate some approaches which have been successfully used in building recommendation systems and compare their effectiveness with Karim *et al.*'s approach which is based on association rule mining. We refer to the best performing variant of *APMiner* as $APRec^{RULE}$ to make this name be consistent with the names of our proposed approaches in this paper.

Our first proposed approach, which we refer to as $APRec^{CF}$, is based on collaborative filtering which has been widely adopted in many recommendation systems [8]. The intuition of using collaborative filtering is that apps which use similar APIs, usually support similar features, so the required permissions are usually similar too. Hence, $APRec^{CF}$ first finds a list of most similar apps to a target app, and then recommend permissions based on the used permissions of these similar apps. We measure similarity of two apps based on the APIs used by the apps.

Our second proposed approach, we refer to as $APRec^{TEXT}$, is based on Text mining to make permission recommendation. Apps often come with textual descriptions (e.g., descriptions on *Google Play*, readme files on *Github*) that describes the functionalities and features of apps. These textual 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

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

to allow it to access the camera (since it is a *flashlight* app). We build the text mining model using the naive Bayes multinomial classification algorithm which is used to analyze app descriptions.

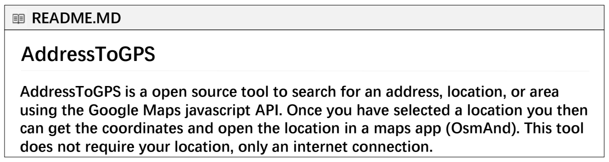
We evaluate these three permission recommendation approaches on 936 open source Android apps from F-droid which have corresponding *Github* repositories. The textual contents processed by $APRec^{TEXT}$ are readme files of apps extracted from their *Github* pages. We measure the effectiveness of these approaches in terms of precision, recall, F1-score and Mean Average Precision (MAP) of the top-k recommendations. The experiment results show that $APRec^{TEXT}$ and $APRec^{CF}$ outperforms $APRec^{RULE}$ and the performance difference between $APRec^{TEXT}$ and $APRec^{CF}$ in term of F1-score@5 and F1-score@10 is small.

The remainder of the paper is organized as follows. We elaborate the intuitions behind the three recommendation approaches that we study in this work in Section 2. Some preliminary concepts are presented in Section 3. Next, we describe the details of the three permission recommendation approaches in Section 4. Our experiment results are presented in Section 5. Related work is briefly reviewed in Section 6. Finally, we conclude and outline potential future directions in Section 7.

2 Motivation

The permission mechanism of Android requires developers to explicitly declare permission requirements if their apps want to access certain sensitive resources, e.g. camera, GPS, Wifi, etc. Meanwhile, Android provides standard APIs to access these sensitive resources. Hence, developers must know what permissions different APIs require. Unfortunately, the API documentation is not always very helpful to determine the specific permissions needed for a specific API. Some tools have been developed to help trace APIs from/to permissions automatically, such as *PScout* and *Androguard*. However, these tools are not perfect and many wrong traceability links are also inferred.

Table 1 shows two example Android applications in F-Droid: *A2DP Volume* which is a bluetooth management app, and *Alarm Clock* which is an alarm clock app. The column "Transaction" in the table indicates a transaction id. These transactions are generated by *Androguard*. A transaction consists of a set of APIs and a single permission which can be traceable to each other. From Table 1, these two apps both have permission ACCESS_NETWORK_STATE and use *android.net.ConnectivityManager* API. Both of them also have permission WAKE_LOCK and use *android.os.PowerManager* API. These pieces of information could be used to help developers determine what permissions are needed 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, since *android.net.ConnectivityManager* API is also used by this app. According to this observation, Karim *et al.* [7] used an association rule mining technique to recommend permissions to a new app and their approach outperforms *PScout* and *Androguard*.


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

Intuitively, Android apps which have similar features or functionalities often need similar permissions. To infer features or functionalities of apps, their used APIs can provide a clue. 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. We can then infer that they are likely to require similar permissions, which is indeed the case (i.e. ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE). This gives us a hint to use another information retrieval technique called collaborative filtering which has been applied successfully in many real recommendation systems [9] to make permission recommendation.

Moreover, apps usually come with textual descriptions which usually tell users what the apps are and what kinds 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 the INTERNET permission. It is often the case that many keywords which correspond to various resources often occur in readme files, e.g. "contacts", "network", "SD card", etc. These textual information could also be utilized to effectively recommend permissions.

3 Preliminary

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

3.1 Frequent Itemset Mining

Frequent itemset mining [10] is a data mining technique widely used for affinity analysis (e.g., market basket analysis) to discover co-occurrence relationships among entities. It takes as input a transaction database (i.e., a multi-set of transactions), where each transaction is a set of items, and outputs sets of items (aka. 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:

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

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 2 transactions (i.e., T2, T9) , so $freq(L)$ is 2. Since 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 kind of pattern named *association rule* can be extracted from a transaction database. For instance, from the database in Table 1, we can infer: "if an Android app uses API *android.net.ConnectivityManager*, the app is very likely to need permission AC-

CESS_NETWORK_STAT" because all of 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, 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 how many transactions the association rules can apply to. Obviously, the support of the association rule R is equal to the support of B, i.e.

$$sup(R) = sup(B).$$

. We also use another metric *confidence* to measure the likelihood that a rule is true, and it can be computed as follows:

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

<u>Example 2.</u> In Table 1, if *minsup* is 0.1, itemsets $A = \{android.net.ConnectivityManager\}$ and $B = \{$*android.net.ConnectivityManager*, ACCESS_NETWORK_STAT$\}$ are frequent itemsets whose support values are both 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 $sup(B)$ which is 0.14, and the confidence of R is 1.0 since $sup(A) = 0.14$.

3.3 Collaborative Filtering

As one of the most successful approaches to building recommender systems, collaborative filtering utilizes information collected about 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, electronic commerce etc [9].

A basic method to perform collaborative filtering is by finding the nearest neighbors of a target entity. A target entity is compared with all other entities and a list of most similar entities based on a distance metric is produced. 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 need similar permissions. In our setting, an entity of an Android app which is represented by a set of APIs used by it, 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 naive Bayes multinomial [11] which is a fast and effective algorithm for text classification to build a text mining model. For many other text mining algorithms, e.g., decision tree and SVM [12], they take a long time to be run on a raw text dataset which has a large number of 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:

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

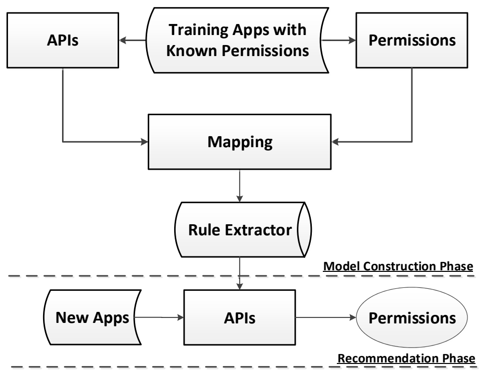


Figure 2 The framework of $APRec^{RULE}$

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 which belong to class C.

4 Approach

In this section, we first describe how to extract the data including APIs, permissions and readme files from Android apps. We then present the details of three proposed recommendation approaches which are based on different data mining techniques, i.e., $APRec^{RULE}$, $APRec^{CF}$ and $APRec^{TEXT}$.

4.1 Data Collection and Processing

We use Android applications from F-Droid which is a catalogue of free and open source applications for the Android platform as data for this study. In total, we find 1,993 apps on F-Droid[4)]. The Android applications on F-Droid are hosted in different platforms, such as *Github*, *Bitbucket*, *Google Code*, etc. In this study, we only consider applications whose source code is hosted on *Github*, since most of projects on *Github* provide a readme file and we can get the readme file easily by very convenient REST APIs provided by *Github*. Among the 1,993 apps, 936 of them put their source code in *Github* and have readme files.

Given an Android app, its permissions are specified in the manifest file, i.e. `AndroidManifest.xml`, which is located at the root level of the app, while its APIs can be found in its Java source code files, declared in import statements. To extract APIs from Java source code, we first transform the source code into a single xml file using *srcML* which is a lightweight static analysis tool [13]. Then we can extract permissions and APIs using an XML processing tool very easily. We only consider the API class names in the Android software stack and Java standard libraries, such as *android.content.ContentResolver* and *java.net.URL*, and ignore user-defined classes.

We use *Androguard* to obtain likely mapping of each API classes and permissions. This forms the transactions that are used by $APRec^{RULE}$. The list of APIs that are used by an app is transformed into a feature vector and input to $APRec^{CF}$. The readme files are the input data of $APRec^{TEXT}$.

4.2 $APRec^{RULE}$ Approach

Figure 2 presents the process of $APRec^{RULE}$. $APRec^{RULE}$ is an implementation of the best performing variant of *APMiner* referred to as *FilteredMiner*' in the original paper [7]. *APMiner* utilizes association

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

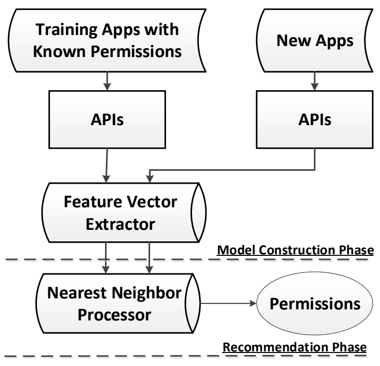


Figure 3 The framework of $APRec^{CF}$

rule mining technique and outperforms the prior state-of-the-art approaches: *Androguard* and *PScout*. Hence, we use $APRec^{RULE}$ as a baseline to compare the two proposed approaches. Each input transaction of $APRec^{RULE}$ contains a single permission and several APIs which can be traceable to the permission. These transactions are the output of *Androguard*.

$APRec^{RULE}$ works in two phases: model construction and recommendation phase. In the model construction phase, $APRec^{RULE}$ takes as an input a set of transactions that is generated by running Androguard on a training set of apps with known permissions. A sub-component of $APRec^{RULE}$ named *RuleExtractor* employs association rule mining to mine API-permission rules (e.g., $APIs \Longrightarrow Permission$), referred to as $APRules$. In the recommendation phase, given a set of APIs of a new app $currentAPIs$, a rule matches $currentAPIs$ if its precondition is a subset of $currentAPIs$. $APRec^{RULE}$ then recommends permissions, based on the post-conditions of the matching rules.

We assign a score to assess the probability of a permission for which an app requires. The rule-based recommendation score for a permission P is the sum of confidence of any matching rule whose post-condition is P. This score is computed by the following formula:

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

In the above equation, $RMatched(P)$ is the set of rules whose pre-conditions is a superset of $currentAPIs$ and whose post-condition is a permission P. If the set $RMatched(P)$ is empty, the recommendation score of P is 0. Next, we will normalize $RecScore^{RULE}$ to make the value ranges 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 $APRec^{CF}$ Approach

Figure 3 presents the process of $APRec^{CF}$. $APRec^{CF}$ utilizes a collaborative filtering technique to recommend permissions for Android apps. Three metrics are used to calculate the similarity between two apps, including *cosine similarity*, *Euclidean distance* and *Pearson correlation*. Thus, there are three variants of $APRec^{CF}$, denoted as $APRec^{CF_{cosine}}$, $APRec^{CF_{euclidean}}$ and $APRec^{CF_{correlation}}$, respectively. For each Android app, $APRec^{CF}$ uses the used APIs to form a feature vector, then get the nearest neighbor apps from the training dataset to perform permission recommendation.

$APRec^{CF}$ recommends permissions based on those that are used by similar Android apps, following a nearest-neighbor-based collaborative filtering approach. We measure the similarity of two apps based on their set of commonly used APIs. $APRec^{CF}$ works in two phases: model construction and recommendation phase. It consists of the $FeatureVectorExtractor$ component and the $NearestNeighborProcessor$ component. The first component is called in the model construction phase, while the latter is called in the recommendation phase.

4.3.1 *FeatureVectorExtractor*

This component converts the list of APIs used by each app in a training set into a feature vector. Let *allAPIs* be the set of all APIs arranged in alphabetical order of their names. Each API can then be assigned a unique index in *allAPIs* and referred to as $allAPIs[i]$. The feature vector of app A, denoted as $V(A)$, is defined as follows:

$$V(A) = (ind(allAPIs[0], A), ..., ind(allAPIs[|allAPIs|], A)),$$

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

4.3.2 *NearestNeighborProcessor*

Given a new app, *NearestNeighborProcessor* first converts the list of APIs used in the app into a feature vector in the same manner as is done by the *FeatureVectorExtractor*. It then calculates the distance between this feature vector and the feature vectors of apps in the training set. In this study, we try three different similarity metrics to compute the distance: *cosine similarity*, *Euclidean similarity* and *Pearson correlation similarity.*

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

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

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

The *Euclidean similarity* is calculated as follows:

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

In the above equation, n is the size of vector. The Euclidean similarity score is the inverse of the Euclidean distance.

The *Pearson correlation* is calculated as follows:

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

In the above equation, $COV(A, B)$ is the covariance between A and B while σ_A and σ_B is the standard deviation of A and B, respectively. The value of $COV(A, B)$ ranges from -1 to 1. Then the correlation similarity score is calculated as:

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

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 is, the more similar an app in the training set is to the new app. We rank apps in the training data based on their similarity scores. Then, we pick top-n apps with the highest similarity score as the nearest neighbors of the new app. The next step is to compute a recommendation score for each permission. Given a permission P, the collaborative filtering-based recommendation score of an app A is calculated as follows:

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

In the above equation, $Nearest$ is the nearest neighbors, $P \in B_i$ means app B_i has permission P and S_x is one of three similarity scores. Then, we normalize the recommendation score. The permissions

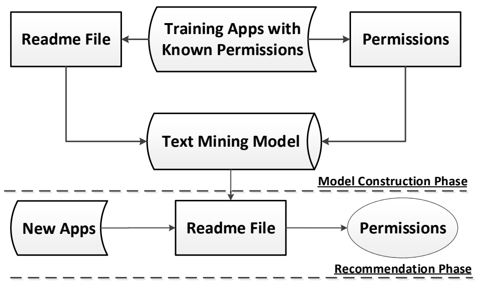


Figure 4 The framework of $APRec^{TEXT}$

with the highest recommendation scores are the most appropriate permissions based on the collaborative filtering.

4.4 $APRec^{TEXT}$ Approach

Figure 4 presents the process of $APRec^{TEXT}$. $APRec^{TEXT}$ leverages 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*. The readme files in *Github* usually contain the apps' descriptions, features, functions, licenses, etc. We train a text classifier for each permission then combine all classifiers to recommend permissions for a new app.

The approach $APRec^{TEXT}$ 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 there all n potential permissions for an app and denote the j^{th} permission as P^j. $APRec^{TEXT}$ first duplicates the training dataset into n binary dataset, one for each permission. Each app (A_i) in the j^{th} training dataset contains two parts: the textual content in readme file ($Text_i$) and a binary value R_i^j which 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 [14], we represent the text of the readme file of the i^{th} app as a vector of term weights denoted by $Text_i = \langle w_1, w_2 ... w_3 \rangle$. The weight w_j denotes the number of times the j^{th} term t_j appears in the textual content of the readme file of the i^{th} app. Next, we construct n prediction models based on the n binary datasets by leveraging the naive Bayes multinomial technique. More details of the naive Bayes multinomial technique is given in Section 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 would output a recommendation score $RecScore^{TEXT}(A_{new}, P^i)$ that indicates the likelihood that A_{new} has permission P^i. The recommendation score $RecScore^{TEXT}(A_{new}, P^i)$ is computed as follows:

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

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

5 Results

In this section, we describe the experiment setup, followed by our evaluation metrics. We then present our research questions and the results of our experiments. Finally, we discuss some threats to validity.

5.1 Experiment Setup

The dataset in this evaluation is 936 Android apps from F-Droid. The apps in the dataset have 4.45 ± 2.57 (mean±standard deviation) permissions and 104.01 ± 83.59 APIs in average. We also use *Androguard* to filter the APIs which are not traceable to the used permissions. Thus, the apps have 8.31 ± 5.35 traceable APIs in average.

To provide a baseline model, we first implement the approach of Karim *et al.*. We use a fast Apriori [15] algorithm implementation [16] for association rule mining. We set miniconf value to 0.4 since Karim *et al.* has shown that their approach has the best performance when the minimum confidence threshold is set to 0.4. For the collaborative filtering approaches $APRec^{CF_{cosine}}$, $APRec^{CF_{euclidean}}$ and $APRec^{CF_{correlation}}$, the default numbers of nearest neighbors of them are all set to 10.

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

5.2 Evaluation Metrics

To evaluate the three permission recommendation approaches in this study, we use several well-known evaluation metric: precision@k, recall@k, F1-score@k and the Mean Average Precision (MAP) which have been used as yardsticks in many studies, e.g., [17–19]. Consider m Android apps in the testing dataset that should receive 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. The precision@k is the ratio of N_i^k over k, i.e.,

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

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

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

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

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

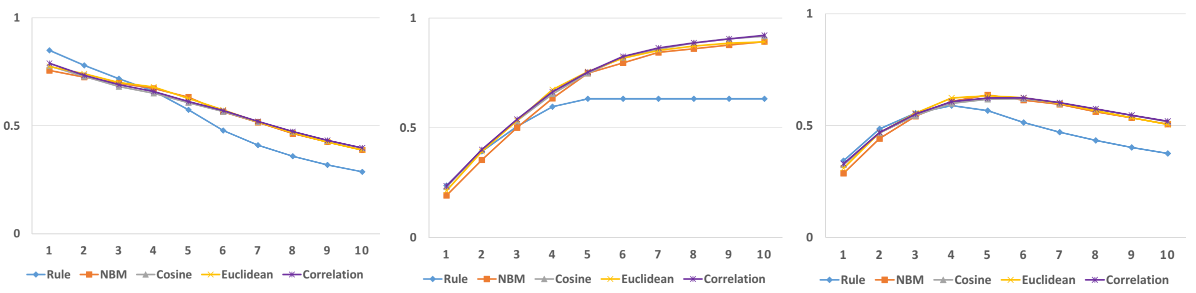
Finally, for m apps in a testing dataset, we calculate the average precision@k, recall@k and F1-score@k.

Since 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 Mean Average Precision (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 [20] and highly informative [21] measure. In this study, Average Precision (AP) is the average of precisions computed at the point of each of the permissions that are correctly recommended in the ranked permission list. It is computed as follows:

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

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 which is equal to 1 if the item at rank k is a permission that the target app uses, and 0 otherwise. Then, for the m apps in the testing dataset, MAP is calculated as follows:

$$MAP = \frac{\sum_{i=1}^{m} AP_i}{m}.$$



(a) Precision   (b) Recall   (c) F1-score

Figure 5 The results of top (1-10) precision, recall and F1-score

Table 2 Cliff's delta and the effectiveness level [23]

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

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 performance $APRec^{TEXT}$ and $APRec^{CF}$ have, the more benefit they would give to their users. Thus, in this research question, we evaluate the effectiveness of $APRec^{TEXT}$ and $APRec^{CF}$ and compare them with the baseline approach $APRec^{RULE}$.

Approach. To answer RQ1, we use 10 fold cross validation to compute the top-k precision, recall and F1-score ($k = (1, 2, ..., 10)$) and MAP to evaluate the performance of each permission recommendation approach.

To check if the differences between the results of the baseline $APRec^{RULE}$ and other approaches are significant, we apply the Wilcoxon signed-rank test [22] at 95% significance level on 10 paired data which represents the results of 10 fold cross validation of compared approaches.

We also use Cliff's delta (δ) [23], which is a non-parametric effect size measure that quantifies the amount of difference between the baseline $APRec^{RULE}$ and other approaches. The delta value ranges from -1 to 1, where $\delta = 1$ or -1 represents the absence of overlap between two approaches (i.e., all values of one group are higher than the values of the other group, and vice versa), while $\delta = 0$ indicates the two approaches are completely overlapping. Table 2 describes the meaning of different Cliff's delta values and their corresponding interpretations.

Results. The results of top-k precision, recall and F1-score are presented in Figure 5(a), 5(b) and 5(c), respectively. In these figures, 'Rule', 'NBM', 'Cosine', 'Euclidean' and 'Correlation' represent $APRec^{RULE}$, $APRec^{TEXT}$, $APRec^{CF_{cosine}}$, $APRec^{CF_{euclidean}}$ and $APRec^{CF_{correlation}}$, respectively. From these figures, we can see that the precisions of all approaches decrease when the number of recommended permissions (i.e. top-k) increases while the recalls increase when the number of recommended permissions increases. This results make sense. For precision, the permissions with high recommendation scores are more likely to be correct, so the precision is high if k is small. But when k increases, more permissions are wrongly recommended. For recall, the more permissions are recommended, the higher the recall is.

From Figure 5(a), we can see that when k is small (from 1 to 4), the precision of baseline $APRec^{RULE}$ is higher than other approaches, but the difference is very small. But the recalls and F1-scores of

Table 3 The 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**
$APRec^{RULE}$	0.5739	0.6322	0.5671	0.2869	0.6322	0.3759	0.6275
$APRec^{TEXT}$	0.6322	0.7487	0.6371	0.3881	0.8925	0.5070	0.7474
$APRec^{CF_{cosine}}$	0.6064	0.7487	0.6183	0.3960	0.9172	0.5176	0.7651
$APRec^{CF_{euclidean}}$	0.6265	0.7560	0.6336	0.3870	0.8916	0.5054	0.7439
$APRec^{CF_{correlation}}$	0.6121	0.7540	0.6233	0.3976	0.9215	0.5198	0.7693

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

	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
$APRec^{TEXT}$	< 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
$APRec^{CF_{cosine}}$	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
$APRec^{CF_{euclidean}}$	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
$APRec^{CF_{correlation}}$	< 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

$APRec^{RULE}$ is almost the same as other approaches when k is small, see Figure 5(b) and 5(c). However, when k is bigger than 5, the precisions, recalls and F1-scores of $APRec^{RULE}$ are all smaller than other approaches. The precisions of $APRec^{RULE}$ decrease more rapidly than others and its recalls almost do not increase when k exceeds 5. As F1-score is a harmonic mean of precision and recall, so the F1-score of $APRec^{RULE}$ also decreases rapidly. But there are small difference on the results of precision, recall and F1-score for the other approaches when k is varied from 1 to 10. In summary, we can see the permission recommendation systems proposed in this paper, i.e. $APRec^{TEXT}$ and $APRec^{CF}$ outperform the baseline approach $APRec^{RULE}$.

We present the detailed top-k (k=5,10) precision, recall, F1-score and MAP for all approaches in Table 3. From this table, we can see all metrics of $APRec^{RULE}$ are smaller than those of other approaches. The precision@5 and F1-score@5 of $APRec^{TEXT}$ are the largest and they outperform those of $APRec^{RULE}$ by ∼6% and 7% respectively. The recall@5 of $APRec^{CF_{euclidean}}$ is the largest which improves over that of $APRec^{RULE}$ by ∼12%. For all newly proposed approaches, their improvements over $APRec^{RULE}$ in terms of top-10 metrics are larger than those in terms of top-5 metrics. The improvements on precision@10, recall@10 and F1-score@10 are more than 10%, 25%, and 13% respectively. The MAP of $APRec^{CF_{correlation}}$ is the largest which improves over that of $APRec^{RULE}$ by ∼14%. Moreover, the differences among the results of the 4 newly proposed approaches are minor.

Table 4 represents the p-values and cliff's delta values for $APRec^{TEXT}$, $APRec^{CF_{cosie}}$, $APRec^{CF_{euclidean}}$ and $APRec^{CF_{correlation}}$ with the baseline $APRec^{RULE}$ in terms of precision@k, recall@k, F1-score@k (k=5, 10) and MAP. In this table, all of the p-values are less than 0.05 and all of the Cliff's delta values are in large effectiveness level which means the improvement of each approach over the baseline $APRec^{RULE}$ is significant.

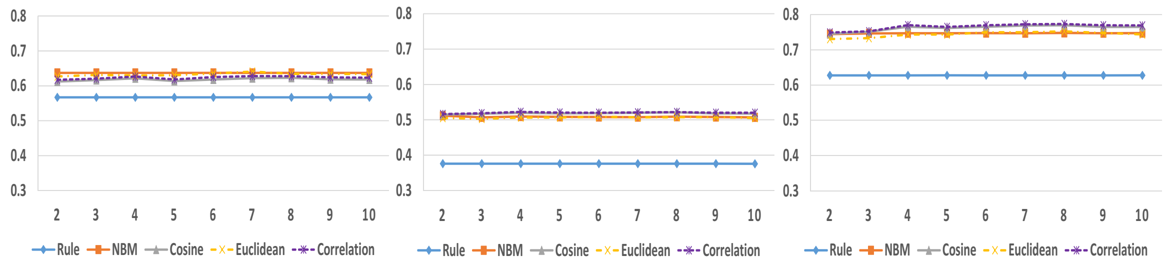
RQ2: How does the size of training data affect the results of the permission recommendation approaches?

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

Approach. We run 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.

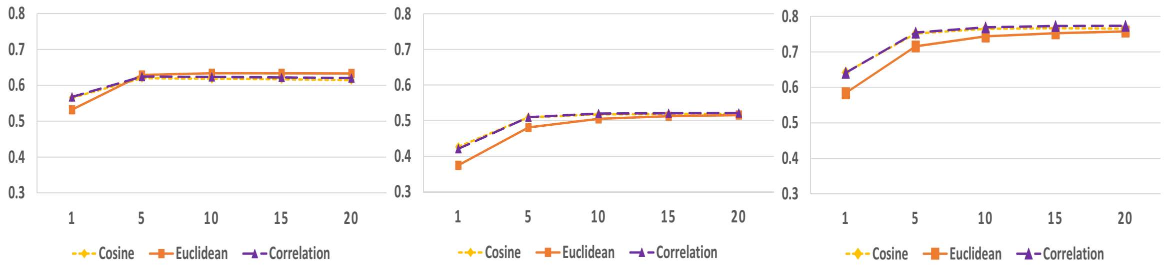
Results. Figure 6(a), 6(b) and 6(c) present the F1-score@5, F1-score@10 and MAP of these permission recommendation approaches for different n fold cross validation, respectively. We notice that both the F1-score@5, F1-score@10 and MAP change only very little when we vary the size of the training dataset for all approaches in this study. Hence, we find that the permission recommendation approaches evaluated in our study perform well across a wide range of training data sizes.

RQ3: How does the number of nearest neighbors affect the results of the permission rec-



(a) F1-score@5 (b) F1-score@10 (c) MAP

Figure 6 F1-score@5, F1-score@10 and MAP for N-Fold cross validation



(a) F1-score@5 (b) F1-score@10 (c) MAP

Figure 7 F1-score@5, F1-score@10 and MAP results for different numbers of nearest neighbors

ommendation approaches using collaborative filtering?

Motivation. By default, the number of nearest neighbors used in our collaborate filtering based approaches is 10. In this research question, we want to investigate whether different numbers of nearest neighbors could affect the performance of these approaches.

Approach. To answer this research question, the different numbers of nearest neighbors are varied to be 1, 5, 10, 15, and 20. For each collaborative filtering approach, we run 10 fold cross validation and evaluate the results in terms of F1-score@5, F1-score@10 and MAP.

Results. Figure 7(a), 7(b) and 7(c) present the F1-score@5, F1-score@10 and MAP respectively of the permission recommendation approaches based on collaborative filtering using different numbers of nearest neighbors. We notice that when the number of nearest neighbors increases from 1 to 5, the improvement on F1-score@5, F1-score@10 and MAP is relatively high for all of three approaches, but when the number of nearest neighbors continues to increase, the improvement is not very big. We can see that even if the number of nearest neighbors is set to 1, the results of F1-score@5, F1-score@10 and MAP are close to those of the baseline $APRec^{RULE}$. Meanwhile, the number of nearest neighbors do not affect the performance of collaborative filtering approaches very much when it varies from 5 to 20. So, it is acceptable to set the default value to 10 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, it takes much time to get the nearest neighbors. For the text mining approach, multiple naive Bayes classifiers need long time to be built. Thus, in this research question, we want to investigate the time efficiency of these permission recommendation approaches.

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

Results. Table 5 shows the average time of one round in 10 fold cross validation for all permission recommendation approaches investigated in this study. We can see that $APRec^{RULE}$ runs much faster

Table 5 The average time of one round in 10 fold cross validation for all approaches

Approach	**Time (seconds)**
$APRec^{RULE}$	12.4
$APRec^{TEXT}$	210.1
$APRec^{CF_{cosine}}$	208.5
$APRec^{CF_{euclidean}}$	202.3
$APRec^{CF_{correlation}}$	283.6

than other approaches, and only requires 12 seconds/round. This is because the number of items in the transactions in our dataset is not many. The slowest approach is $APRec^{CF_{correlation}}$. It takes about 1 more minute than the other three approaches. Although the time of these proposed approaches is longer than that of the baseline $APRec^{RULE}$ but we believe it is still acceptable. For $APRec^{TEXT}$, only one time the naive Bayes classifiers needs to be built and it takes much less time to recommend permissions in the recommendation phase. And for $APRec^{CF}$, it can recommend permissions to an app within seconds – nearly 100 apps is tested in each round of 10 fold cross validation in our dataset.

5.4 Discussion

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

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

For $APRec^{TEXT}$, it might miss detecting some permissions for a given app since its text description may not explain the hidden internals of the app and only provides a high level overview of functionalities performed. Besides, certain permissions might not be directly related to app functionality. For example, it will be quite difficult to capture the following permissions based on 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 directly related to app functionalities, which are difficult to be inferred from app textual descriptions. To mitigate the above mentioned weakness, given an app, $APRec^{TEXT}$ finds other apps with similar text description and recommends permissions based on those used in the similar apps. For example, *Physics Drop*[5] and *Flow Free* [6] are two puzzle game apps with similar text description. They require WAKE_LOCK permission which is not apparent from their descriptions. Based on similarity of these two apps, $APRec^{TEXT}$ can regard them as similar apps and recommend one's permissions to the other. However, not all apps with similar textual description use the same permissions. Thus, $APRec^{TEXT}$ might make wrong permission recommendation. To improve APRecTEXT further, it will be interesting to investigate new approaches that can better understand semantics of text descriptions of apps. Related to this direction, recently Qu *et al.* found text patterns of 11 Android permissions by using NLP techniques [24]. Qu et al.s text patterns can potentially be combined with $APRec^{TEXT}$ to help improve the accuracy of its recommendations.

5.5 Threats to Validity

One of threats to internal validity relates to errors in our code and experiment bias. We have double-checked our code, still there could be errors that we did not notice. 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.

5) https://play.google.com/store/apps/details?id=com.dreamed.physicsdrop

6) https://play.google.com/store/apps/details?id=com.bigduckgames.flow

One of threats to external validity is the dataset used in our study. We have analyzed 936 open source Android apps from F-Droid. In the future, we plan to consider more open source Android apps even closed source apps to reduce this threat to validity. Closed source apps, such as those distributed on GOOGLE PLAY, require reverse engineering which can be performed by existing tools (e.g., *Androguard*) to extract the API usage. Another threat to external validity is that not all permissions are covered in out study. Out of the 151 system-defined permission in Android, 45 permissions are used in our dataset. We also do not consider the customized permissions.

Threats to construct validity refers to the suitability of our evaluation measures. We use top-k precision, recall and F1-score, and MAP which are also used by many prior automated software engineering studies [17–19].

6 Related Work

Our work is inspired by the work of Karim *et al.* [7], which uses association rule mining technique to recommend permissions. The extracted rules in their study are based on the co-occurrence of Android APIs and permissions. To our best knowledge, there are no other studies that use recommendation system algorithms to predict possible required permissions for an app. In our study, we propose two approaches based on two other recommendation system techniques: collaborative filtering and text mining. We also compare the proposed approaches with the approach based on association rule mining and find that our approaches achieve better performance than that of the association rule mining approach.

Many researchers have proposed different approaches to identify the mappings between APIs and permissions. Vidas *et al.* [25] extract a permission specification by scanning the Android documentation. But the usefulness of their work is affected by the incompleteness of the Android documentation. Some tools (e.g. *Stowaway* [3], *PScout* [4], and *Androguard* [6]) rely on static analysis to extract the mappings between APIs and permissions. The results of *PScout* is more complete and accurate than those of *Stowaway*. *PScout* have been applied on four versions of Android and help figure out that about 22% of the non-system permissions are unnecessary. *Androguard* is a reverse engineering tool which embeds PScouts methodology and enables API to recommend permissions to a given app. In our study, *Androguard* is used to extract the possible mappings between APIs and permisions. Note that these program analysis approaches are not perfect and many mappings that are recovered are not correct. Furthermore, the approach based on association rule mining has been proved to have better performance than these tools which only rely on program analysis in the study of Karim *et al.* [7]. Thus, we do not compare our proposed approaches with these tools. Qu *et al.* investigated relationships between app descriptions and permissions (referred to as *description-to-permission fidelity*) by leveraging natural language processing (NLP) techniques [24]. Their study investigated 11 permissions and highlighted text patterns that correspond to each of the permissions. Their text patterns can be used to help make permission recommendation, but the number of permissions investigated in their study is too small. Our proposed approach $APRec^{TEXT}$ leverages text classification techniques to recommend permissions for apps. So, the number of permissions which is based on the training data can be much larger than that of the approach of Qu *et al.*. But as $APRec^{TEXT}$ do not understand the semantic of app descriptions, it might make wrong permission recommendation. In the further, to improve the performance of $APRec^{TEXT}$, we plan to use NLP techniques to better understand app textual descriptions.

The misuse of permissions in Android has been investigated by researchers. To understand whether Android users pay attention to, understand, and act on permission information during installation, Felt *et al.* [26] conduct an Internet survey with 308 Android users and a laboratory study with 25 users. They find that only 17% of users pay attention to permissions during installation and 3% of Internet survey respondents could correctly answer all three permission comprehension questions. Stevens *et al.* [5] conduct a study on permission misuses by considering both the apps from Android market and questions about security permission use on StackOverflow. They find the popularity of a permission is strongly associated with its misuse. The above works provide the motivation of our research. The phenomenon of permission misuses exists in Android. Hence, permission recommendation approaches could be very helpful to both developers and end-users.

The permission recommendation approaches in this study could be used to detect malicious behavior, since the permission misuses could be indicative to stealthy and malicious behavior in Android apps. There are many approaches proposed by researchers to help identify malicious behavior in Android apps [27–32]. For example, *AsDroid* [27] could identifying stealthy behavior by analyzing user interface and program behavior contradiction. Blasing *et al.* [29] propose a tool which is based on static and dynamic analysis to detect suspicious Android apps. Zhou *et al* [30]. propose a permission based behavioral footprinting scheme to detect Android malware.

7 Conclusion and Future Work

Android provides permission mechanism to help protect the privacy and security of Android app users. Unfortunately, there still exist many misuses of permissions [5, 26] which may be caused by the incompleteness of Android documentation. An approach based on association rule mining, proposed by Karim *et al.*, has been reported to be useful to recommend appropriate permissions for an app [7]. This gives us a hint that other algorithms proposed in the recommend system area may also be applied to recommend permissions to apps. Hence, in this paper we propose another two approaches to recommend permissions for a new app. One approach is based on collaborative filtering technique and another is based on text mining (in particular text classification). We also try three different similarity metrics (i.e., *cosine similarity*, *Euclidean similarity* and *Pearson similarity*) for the collaborative filtering approach. We evaluate these approaches on 936 Android Apps from F-Droid and compare our proposed approaches with the association rule approach proposed by Karim *et al.* Results show that our proposed approaches outperform the baseline approach in terms of top-k precision, recall and F1-score and MAP.

In the future, we plan to apply these permission recommendation approaches on more Android apps, such as the apps in *Google Play*. We also plan to develop more sophisticated machine learning techniques to improve the effectiveness of the proposed approach further.

Acknowledgements This research was supported by NSFC Program (No. 61602403 and 61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01).

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

References

1 Smartphone market share. http://www.idc.com/prodserv/smartphone-os-market-share.jspl
2 Google play. https://en.wikipedia.org/wiki/Google_Play
3 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
4 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
5 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
6 Desnos A. Androguard: Reverse engineering, malware and goodware analysis of android applications. http://code.google.com/p/androguard
7 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
8 Ricci F, Rokach L, Shapira B. Introduction to recommender systems handbook. Berlin: Springer, 2011
9 Su X, Khoshgoftaar T M. A survey of collaborative filtering techniques. Advances in artificial intelligence, 2009: 4
10 Agrawal R, Srikant R. Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering. Piscataway: IEEE Press, 1995, 3–14
11 McCallum A, Nigam K, et al . A comparison of event models for naive bayes text classification. In: Proceedings of AAAI-98 workshop on learning for text categorization. 1998, 41–48
12 Han J, Kamber M, Pei J. Data mining: concepts and techniques. Elsevier, 2011

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

14 Baeza-Yates R, Ribeiro-Neto B, et al. Modern information retrieval. volume 463. New York: ACM, 1999

15 Agrawal R, Srikant R, et al. 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

16 Bodon F. A fast apriori implementation. In: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03). Vol 90. Piscataway: IEEE Press, 2003

17 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

18 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

19 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 34th International Conference Software Engineering (ICSE). Piscataway: IEEE Press, 2012, 14–24

20 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

21 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

22 Wilcoxon F. Individual comparisons by ranking methods. Biometrics bulletin. Washington: International Biometric Society, 1945, 1(6): 80–83

23 Cliff N. Ordinal methods for behavioral data analysis. London: Psychology Press, 2014

24 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

25 Vidas T, Christin N, Cranor L. Curbing android permission creep. In: Proceedings of the Web. 2011, 91–96

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

27 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

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

29 Bläsing T, Batyuk L, Schmidt A D, et al. An android application sandbox system for suspicious software detection. In: Proceedings of 5th international conference on Malicious and unwanted software (MALWARE) Piscataway: IEEE Press, 2010, 55–62

30 Zhou Y, Wang Z, Zhou W, et al. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In: NDSS. Vol. 25. 2012, 50–52

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

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