Accurate Library Recommendation Using Combining Collaborative Filtering and Topic Model for Mobile Development

Xiaqiong ZHAO†, Shanping LI†, Huan YU†, Nonmembers, Ye WANG†§, Member, and Weiwei QIU†, Nonmember

SUMMARY Background: The applying of third-party libraries is an integral part of many applications. But the libraries choosing is time-consuming even for experienced developers. The automated recommendation system for libraries recommendation is widely researched to help developers to choose libraries. Aim: from software engineering aspect, our research aims to give developers a reliable recommended list of third-party libraries at the early phase of software development lifecycle to help them build their development environment faster; and from technical aspect, our research aims to build a generalizable recommendation system framework which combines collaborative filtering and topic modeling techniques, in order to improve the performance of libraries recommendation significantly. Our works on this research: 1) we design a hybrid methodology to combine collaborative filtering and LDA text mining technology; 2) we build a recommendation system framework successfully based on the above hybrid methodology; 3) we make a well-designed experiment to validate the methodology and framework which use the data of 1,013 mobile application projects; 4) we do the evaluation for the result of the experiment. Conclusions: 1) hybrid methodology with collaborative filtering and LDA can improve the performance of libraries recommendation significantly; 2) based on the hybrid methodology, the framework works very well on the libraries recommendation for helping developers’ libraries choosing. Further research is necessary to improve the performance of the libraries recommendation including: 1) use more accurate NLP technologies improve the correlation analysis; 2) try other similarity calculation methodology for collaborative filtering to rise the accuracy; 3) on this research, we just bring the time-series approach to the framework and make an experiment as comparative trial, the result shows that the performance improves continuously, so in further research we plan to use time-series data-mining as the basic methodology to update the framework.

key words: library recommendation, text mining, latent dirichlet allocation, software engineering

1. Introduction

Software development continues to grow at a rapid speed globally, and it becomes the most popular bridge between the people and the business. Although the rapid increasing and expanding of software market has brought huge benefits for users and clients, competition in the software development is very tough for all the teams which provide these softwares and the related services. At that background, developers use more and more third-party libraries to reduce development time of softwares, publish the newest features fast, increase the reliability, improve the productivity of development teams, and avoid reinventing the wheel and time waste.

However, the choosing of third-party libraries is a kind of time-consuming task even for the experienced developers, so the automated recommendation system for libraries recommendation is widely researched for the performance improvement of recommendation algorithm. In this paper, to bridge the gap between the large amount of available third-party libraries and the requirement of third-party libraries, we propose an approach to automatically recommend third-party libraries based on collaborative filtering and topic modeling techniques to improve the performance of recommendation based on the previous researches. That’s the motivation of our work.

In recent years, many related researches have been published on the domain of third-party libraries recommendation [5], [6], [29]. Association rule mining [1] is the technique which has been widely used on those works [5], [6], [29] currently. At the same time, the recommendation accuracy of association rule mining is highly influenced by the probability of the co-occurrence of third-party libraries. Thung et al. proposed an approach to recommend an entire third-party library based on the libraries by using in a certain number of projects commonly [29]. This methodology has an scenario assumption that development team knows some required libraries and only needs to find other relative libraries, they may have few knowledge on third-party libraries in practice. However, the above assumption doesn’t appear frequently in real world software project. First, there is no obvious evidence to show that the development team has knowledge of all kinds of third-party libraries; second, even if the development team has partial knowledge the level of the knowledge would be various; third, many third-party libraries have not only one usage or feature. So the motivation of a recommendation system with more general assumption is urgent for developers.

To solve this problem, the text mining technology gives our work a light. In practice, taking Fig. 1 as an example it shows the description information of a mobile application as an example of software. The description provides the most useful information for both users and developers which includes the intent of application such as the introduction, the functions, the running environment, the version information, the main update of this version, and so on. At the early phases of development lifecycle developers can get the

 DOI: 10.1587/transinf.2018EDP7227
description from the Business Analyst (BA) which is the text description of the software that the developers will work on. At the same time, taking Fig. 2 as an example of third-party library, as a third-party library there is also these similar information to describe itself for the developers. That gives us the text description of the third-party library. So mining these textual description to build the relationship between these softwares and third-party libraries is obviously an rational way to improve the performance of libraries recommendation.

So in this paper, we propose an AppLibRec approach method with automated text mining which combines Latent Dirichlet Allocation (LDA) and collaborative filtering to recommend a list of third-party libraries for software development. Latent Dirichlet allocation (LDA) is a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model that each item of a collection is modeled as a finite mixture over an underlying set of topics. And collaborative filtering is a filtering process for information or patterns using multiple agents techniques. In the collaborative filtering, we use software as entity, and the target entity is compared with other entities in the dataset to product a most similar entities list based on a distance metric, and then we make recommendation for the

target entity based on the similarity entities. Specifically, for more detailed AppLibRec performs two kinds of analysis: README file based analysis (RM-based) by using LDA, and the libraries based analysis (Lib-based) by using collaborative filtering. The hybrid methodology combining these two methodologies improves the overall performance of the recommendation significantly.

Though AppLibRec can be applied to any type of software development from both the methodology aspect and the framework aspect, in the experiment we apply our experiment of the hybrid methodology to mobile applications as subjects. We conducted an experiment on a big set of real world mobile applications on github, and we filtered those applications by a criteria in order to simulate the real world scenario better. Our proposed recommendation experiment setting focus on the mobile applications development in terms of: 1) The library recommendation problem in mobile application development is more necessary to the developers than general development, because the development relies on simulator heavily so the test is much harder to execute. 2) Mobile application is a very important software domain, and continues to grow in popularity globally, and they become one of the most popular interface between the people and the internet. 3) Mobile application is more suitable for libraries recommendation, because all the mobile Apps are independent, are packed to be an deployable package, and are similar on structure and architecture. We perform the experiment on a set of 1,013 mobile Apps.

At the result analysis, to make a comparison for our research, we choose LibRec approach which was proposed by Thung et al. on 2013 as a baseline. LibRec approach combines association rule mining and a nearest-neighbor-based collaborative filtering approach to recommend libraries for projects on GitHub. The results show that our approach improve the precision@5 and recall@5 of LibRec by 38% and 35% respectively.

The main contributions of this paper are:

- We propose a hybrid methodology that combines topic model and collaborative filtering to recommend third-party libraries for mobile applications.
- We implement the above hybrid approach into a framework based on k-nearest neighbors for collaborative filtering and LDA for text mining.
- Experiment on 1,013 applications show the effectiveness of our AppLibRec approach, and our approach outperforms the state-of-the-art approaches significantly.
- We analyze the result of the experiment by answering several research questions from different aspects.

The remainder of this paper is organized as follows: Sect. 7 reviews the related work. Section 2 presents the base algorithms. Section 3 presents the recommendation framework and our approach. Section 4 formally defines details of the experiment. Section 5 shows experimental
results and the discussion. Section 8 concludes the paper and mentions future work.

2. Preliminaries

In this section, we define library recommendation problem at first. And then we describe the basic preliminaries of our research. These preliminaries which works as the basic methodology set of our research are including libraries representation, Pearson Correlation Coefficient metric (PCC) similarity calculation algorithm[10], [21], text pre-processing, Latent Dirichlet Allocation model (LDA)[2], and collaborative filtering [27].

2.1 Problem Definition

Before introducing the methodologies and preliminaries, we should define the problem domain of library recommendation problem. 

First, we define 3 basic proper names and their concept as following:

1. The set of all available libraries in our dataset is called “LibsRepos” in this paper
2. The set of libraries which is currently used in the mobile application is called “UsedLibs” in this paper
3. The set of libraries that is to be recommended to developers is called “RecLibs” in this paper.

After the above definition, the goal of our approach of Accurate Library Recommendation is abstracted into finding “AppLibRecs” which satisfies the following conditions:

1. RecLibs ⊆ LibsRepos
2. RecLibs ∩ UsedLibs = ∅

2.2 Libraries Representation and PCC

The libraries usage of a mobile application can be represented by a vector of feature values, and one feature component of the vector represents a library invoked by this mobile application. Thus, a set of libraries can be denoted as \( \{p_1, p_2, p_3, \ldots, p_n\} \), where \( p_i, i \in \{1, 2, \ldots, n\} \). If there are two different mobile applications: project \( u \) and project \( a \), there should be \( u = (p_1, p_2, p_3, \ldots, p_n) \) and \( a = (q_1, q_2, q_3, \ldots, q_n) \).

Pearson Correlation Coefficient (PCC) can be employed for the similarity calculation. The similarity between \( u \) and \( a \) can be calculated as following function:

\[
Sim(u, a) = \frac{\sum_{i \in I_u \cap I_a} (p_{u,i} - \overline{p}_u)(p_{a,i} - \overline{p}_a)}{\sqrt{\sum_{i \in I_u \cap I_a} (p_{u,i} - \overline{p}_u)^2} \sqrt{\sum_{i \in I_u \cap I_a} (p_{a,i} - \overline{p}_a)^2}}
\]  

(1)

In the above function, \( I_u \cap I_a \) is the set of libraries which are invoked by both project \( u \) and project \( a \). The \( p_{u,i} \) is the library invoked by project \( u \). If application \( u \) uses the \( i \)th library, then \( p_{u,i} = 1 \); otherwise if application \( u \) does not use the \( i \)th library, then \( p_{u,i} = 0 \). The \( \overline{p}_u \) denotes the average quantity of libraries that are invoked by project \( u \).

2.3 Text Pre-Processing

Text pre-processing is an important and integral part of text mining. The pre-processing converts a piece of text into a common representation by a certain text mining algorithm, and it also removes the noise of corpus.

The typical process of text pre-processing is firstly to tokenize the corpus into word tokens. And normally the tools for this step could be found in many natural language processing (NLP) packages (e.g., nltk, provided by the university of Pennsylvania1). Secondly, we can also use some other tools for parsing the words to remove morphological affixes.

**Tokenization** Tokenization is employed to breaking a document into word-tokens which could be the words or punctuation in a string format. Delimiters, for example the space or a kind of special punctuation, can be used to segment and demarcate one token from another. As the output of tokenization, a document is converted to a set of word tokens.

**Stemming** In NLP domain, stemming is the process of reducing inflected words to their word stems. It removes morphological affixes from words. For instance, words “stemms”, “stemmed”, “stemmer”, and “stemming” have a common stem which is “stem”. This stemming conversion makes us be able to identify different forms of a word, and aggregate those word forms together. The multiple forms of one word would be treated as different words without stemming, which is not ideal in many cases. It strips a suffix of the word to convert a word to its stem by using several rule based heuristics. There are many software engineering studies for the stemming technologies [9], [24].

2.4 Latent Dirichlet Allocation Model

The goal of Latent Dirichlet Allocation (LDA) is to find short descriptions of the members in a set so that it enable efficient processing of large collections while preserving the essential statistical relationships which are useful for basic tasks such as classification, novelty detection, summarization, and similarity and relevance judgments [2]. LDA is a generative topic model which aims to mining the hidden structure of the document collection. Each document is assumed to a distribution over \( k \) topics, and the discrete topic distributions are drawn from a symmetric Dirichlet distribution. Figure 3 demonstrates the graphical representation of LDA.

A piece of document can be represented as random mixtures over \( k \) latent topics whose proportions are drawn from a Dirichlet prior.

In LDA, word, document, and corpus are 3 basic concepts. Word is the basic unit of discrete data; Document is

1http://www.nltk.org/
a sequence of \( N \) words denoted by \( w = (w_1, w_2, \ldots, w_n) \), in that the word \( w_n \) means the \( n \)th word in the document sequence; Corpus is a collection of \( M \) documents denoted by \( D = \{ w_1, w_2, \ldots, w_m \} \).

The generative process of LDA for documents \( w \) in a corpus \( D \) can be summarized as follows:

1) Choose \( N \sim \text{Poisson}(\xi) \)
2) Choose \( \theta \sim \text{Dir}(\alpha) \)
3) For each of the \( N \) words \( w_n \):
   (a) Choose a topic \( z_n \sim \text{Multinomial}(\theta) \)
   (b) Choose a word \( w_n \) from \( p(w_n \mid z_n, \beta) \), a multinomial probability conditioned on the topic \( z_n \).

Finally, after that generative process, LDA produce two distributions: document-topic distribution (DT) and topic-word distribution (TW).

In our setting, a document is the README file of a mobile application, and the usage of LDA is to measure the distance among the README files. This distance presents the similarity between mobile applications based on the document-topic distribution. In other words, we can calculate the similarity between any 2 given documents by comparing their document-topic distributions.

In our work, cosine similarity is employed to measuring the distance between two document-topic distributions, which is a topic vector that contains the probability of each topic to be presented in the document.

### 2.5 Collaborative Filtering

Collaborative filtering is the process of recommendation for information or patterns by involving collaboration among multiple agents [27]. This methodology has been widely used in all kinds of recommendation systems [14], [25].

The most famous implementation of collaborative filtering in the context of web application is from Amazon.com which recommends new commodity by item-based collaborative filtering to its client. Finding the nearest neighbors of the target entity is a typical method to perform collaborative filtering. The target entity is compared with other entities in the dataset to produce a list of most similar entities based on a distance metric (e.g., PCC), and then making recommendation for the target entity based on the interest of similarity entities.

In the newest sense, collaborative filtering is a method of making automatic predictions (filtering) about the interest to an object of a user by collecting preferences information of the object from many users (collaborating). And in our research we use mobile Apps instead of users in original Collaborative Filtering, and use third-party Libs instead of users’ interest objects in original Collaborative Filtering. So that collaborative filtering can be narrow down to the method of automatic predictions (filtering) about the using possibility to a library of an App by collecting the usage of the Libs from many other Apps in our research.

### 3. Proposed Approach

In this section, based on the above problem definition and preliminaries, we introduce the approach of our work by providing a description of our overall framework in Fig. 4 and Sect. 3.1. And then we introduce our AppLibRec method based on the framework to solve the third-party libraries recommendation problem by 3 sub-sections:

1. In Sect. 3.2, we describe the RM-based component which uses LDA model.
2. In Sect. 3.3, we present the Lib-based component which uses collaborative filtering.
3. We aggregate the two components presented in Sect. 3.4.

#### 3.1 Overall Framework

Figure 4 shows the overall framework of our approach. We name it as AppLibRec. It contains two major components, and each of them has its pre-process component.

1. **RM-based** analysis Component which is used for modelling the description of mobile applications by using LDA.
2. **Lib-based** analysis Component which is used for finding similar applications and recommending libraries by investigating the set of libraries by using collaborative filtering method.

**README files based (i.e., RM-based)** analysis takes textual description of new mobile applications as input. This requires textual descriptions of training set of applications whose libraries are known, and the testing set of applications whose libraries are hidden by us to validate the method. **README file based (i.e., RM-based)** analysis give a list of recommended libraries along with their recommendation scores as its output. **RM-based** uses LDA model which takes textual descriptions as input to compute the similarity between these textual descriptions, and then finds the applications by k-nearest neighbors. The RM-based analysis finally recommends relevant libraries based on the libraries used by the k-nearest neighbors algorithm.

Third-party libraries based (i.e., Lib-based) analysis takes two sets as input including: 1) a set of libraries that
a new given mobile application has used, and 2) a set of libraries that training set of applications have used. Lib-based analysis give the outputs as a list of recommended libraries along with their recommendation scores for a given application. Lib-based analysis recommends libraries by investigating the set of libraries that are used by similar applications. Lib-based analysis uses k-nearest neighbors based collaborative filtering method. And we use PCC for the calculation of the similarity between an App to others.

Finally, the aggregator combines and merges these two lists of recommendation libraries into a new list along with their final scores. The libraries on the top of the list with highest scores are final result which are recommended by the framework.

3.2 **RM-Based Analysis**

**RM-based** uses LDA model which takes textual descriptions as input to get the document-topic distribution, and the cosine similarity is employed to computing the distance among those textual descriptions. It finally recommends relevant libraries based on the libraries used by the similarity applications, these similarity applications are picked by the distance calculation algorithm of textual descriptions. These are two major steps what we need to do to implement our **RM-based** analysis:

- **Training LDA Model.** We make Mapping Component on Fig. 4 to converts the textual descriptions of training set of applications to topic vectors.
- **Find Similar Textual Description.** We make RM-based Analysis Component on Fig. 4 to calculate the distance between this topic vector and the topic vectors of training set of applications by using LDA model created above.

### 3.2.1 Training LDA Model

This step is to train the LDA Model and convert the textual descriptions of training set of applications to topic vectors. In natural language processing (NLP), a topic represents a distribution of words, and a document is a distribution of topics. We get the 1,013 mobile applications with all the information we need from github, and organized them to the Mobile application Description Documents Data Source which is shown on Fig. 4.

Using LDA model, this component can convert a textual description to a topic vector which is the document-topic distribution.

Section 2.4 provides the description of LDA, and in our paper, we define the following parameters for LDA implementation:

- We define the description of a mobile application is a document with name of \( m \).
- Every document \( m \) has its topic proportion vector with name of \( \theta_m \).
- \( k \) is the number of all topics of training set of applications, so \( \theta_m \) contains \( k \) elements.
- The \( T[i] \) is the proportion of \( i \)th topic in document \( m \), and \( i \) follows \( i \in \{1, 2, \ldots, k\} \).

After the text pre-processing (i.e., tokenization and stemming), we get a training textual description corpus. LDA takes the corpus and a number of parameters as input which are described in Sect. 5. For each document \( m \), LDA would generate a topic proportion vector \( \theta_m \) which contains \( k \) elements. Each element presents a topic with its value in ranges from 0 to 1. The value corresponds the proportion of the term (i.e., word) in \( m \) belonging to the topic in \( m \).

Finally, the value of \( \theta_m \) can be defined as the following:

\[
\theta_m = (T[1], T[2], \ldots, T[k]) \tag{2}
\]

### 3.2.2 Find Similar Textual Description

By given a new application, we first convert the textual description of the application into a topic vector by using the LDA model which is trained in the above Training LDA Model step. Then we should calculate the distance between this topic vector and the topic vectors of training set of applications which have extracted in the first step. Cosine similarity is employed to measuring the value of the distance.

\[
\text{Cosine}(New, Training) = \frac{\theta_{New} \cdot \theta_{Training}}{|\theta_{New}| |\theta_{Training}|} \tag{3}
\]

In the above function, the \( \cdot \) denotes the dot product; and the \( |\theta| \) presents the size of a topic vector.

We rank the training set of applications based in the cosine similarity values. The higher cosine similarity value
is, the more similar a training application is to the new application. We select the top \( k \) applications which have the highest cosine similarity scores as the \( k \)-nearest neighbors for the new project.

Finally, we collect all of the libraries that are used by \( k \)-nearest neighbors and calculate the score for each library in these libraries. The calculation algorithm of the library \( L \) score is represented as follows:

\[
Score_{RM-based}(L) = \frac{Count(L)}{k}
\]  

(4)

In the above function, the \( Count(L) \) is the number of nearest neighbors that have used the library \( L \); and the \( k \) is the number of nearest neighbors. The value of \( Score_{RM-based}(L) \) ranges from 0 to 1. The libraries that have the highest scores is the one which we are going to recommend to developers via \( RM-based \) Analysis.

### 3.3 Lib-Based Analysis

**Lib-based** recommends libraries by investigating the set of libraries that are used by similar applications, using \( k \)-nearest neighbors based collaborative filtering method. We employ the Pearson Correlation Coefficient (PCC) to measure the similarity of two applications based on these third-party libraries. These are two steps to implement our **Lib-based** analysis:

- **Feature Vector Extractor.** We make **Feature Vector Extraction Component** on Fig. 4 to convert the set of libraries to feature vectors.
- **Find Similar Applications.** We make **Lib-based Analysis Component** on Fig. 4 to calculates the distance between lib’s feature vector and other feature vectors which have extracted by using collaborative filtering.

#### 3.3.1 Feature Vector Extractor

It is the pre-process of **Lib-based** Analysis. In this step, we convert the set of libraries to feature vectors by Feature Vector Extractor component. Totally, we get the libraries used by the above 1,013 mobile applications, and get the information we need from github as well, and organized them to the Mobile application Libs Data Source which is shown on Fig. 4.

We denote the set of all libraries as a list with index which are arranged in alphabetical order of their name as \( F \). The index of each library is a unique number \( F[i] \), where \( i \in \{1, 2, \ldots, n\} \) and the \( n \) is the number of all distinct libraries.

The feature vector of application \( a \), is defined as follows:

\[
Vector(a) = (index(F[0], a), index(F[1], a), \ldots, index(F[n], a))
\]  

(5)

In the above function, \( F[i] = 1 \) if application \( a \) uses the \( i \)th library, or \( F[i] = 0 \) otherwise; \( i \in \{1, 2, \ldots, n\} \); and \( n \) is the number of all distinct libraries.

#### 3.3.2 Find Similar Applications

At the pre-process introduced above, given a new application we first convert the set of libraries that the application has used into a feature vector which was done in the Feature Vector Extractor step.

After that we implement the real **Lib-based** Analysis based on Collaborative Filtering (CF) by calculating the distance between the new application’s feature vector and each feature vector of all training set of applications which have extracted in the above step (Feature Vector Extractor).

In the newest sense, collaborative filtering is a method of making automatic predictions (filtering) about the interest to an object of a user by collecting preferences information of the object from many users (collaborating). And in our research we use mobile Apps as users, and use third-party Libs as objects. So collaborative filtering can be narrow down to the method of automatic predictions (filtering) about the using possibility to a library of an App by collecting the usage of the Lib from many other Apps.

The Collaborative Filtering based **Lib-based** analysis is including the following 3 steps:

1. **Similarity**

Our Lib-based analysis component on Fig. 4 recommends libraries by investigating the set of libraries that are used by similar mobile Apps, we use a nearest neighbor by collaborative filtering approach. To find the nearest neighbor we uses Pearson correlation coefficient (PCC) as the metric of similarity to compute this distance [10], [21].

The similarity could be calculated as PCC values by using the following function:

\[
Sim(New, Training) = \frac{\sum_{i \in I_T \cap I_N} (F_{N,i} - \bar{F}_N)(F_{T,i} - \bar{F}_T)}{\sqrt{\sum_{i \in I_T \cap I_N} (F_{N,i} - \bar{F}_N)^2} \sqrt{\sum_{i \in I_T \cap I_N} (F_{T,i} - \bar{F}_T)^2}}
\]  

(6)

In the above function, \( I_N \cap I_T \) is the set of libraries invoked by both project \( New \) and project \( Training \); \( F_{N,i} \) is the library extracted by project \( New \); \( F_{N,i} = 1 \) if application \( New \) uses the \( i \)th library, or \( F_{N,i} = 0 \) otherwise; and \( \bar{F}_N \) denotes the average quantity of libraries invoked by project \( New \).

2. **Time-series Behavior**

The Apps could be updated by the version evolution, time-series changes for the Apps would be reflected in the recommendation result significantly.

For improving the performance of tradition collaborative filtering method, we propose a dynamic App-Lib-Time (ALT) three-dimensional model based on rolling time windows approach.

The basic methodology is to add a time dimension to the original App-Lib collaborative filtering method shown on Fig. 5, and bring Ebbinghaus forgetting method to the
time dimension.

By improved by time dimension, the similarity function based on PCC values would be updated to the following function:

$$S_{\text{im}}(t + T) = \frac{\sum_{k=1}^{t-1} T_k \cdot S_{\text{im}}^{k+1} + T_s \cdot S_{\text{im}}^{s}}{\sum_{k=1}^{T_k}}$$  \hspace{1cm} (7)$$

We use the classic Ebbinghaus forgetting method to update the $T_i$ with the following function and the curve shown on Fig. 6.

$$T_i = e^{-\lambda(1-\frac{i}{n})}$$  \hspace{1cm} (8)$$

3. Apps Ranking

Then we use the PCC values above to rank the training set of applications which has 1,013 Apps.

The higher the PCC value of a mobile application is, the more similar the mobile application in the training set is to the new application. We select the top $n$ applications from the training set of applications which have the highest PCC scores as the $n$-nearest neighbors for the new project, in order to get a Top $n$ Apps Set.

4. Library Scoring.

Finally, we collect all of the libraries that applications on the above Top $n$ Apps Set have by using Maven information, and we are going to calculate the score for each library in these libraries.

We calculate the library $L$ score as the following function:

$$\text{Score}_{\text{Lib-based}}(L) = \frac{\text{Count}(L)}{n}$$  \hspace{1cm} (9)$$

In the above function, $\text{Count}(L)$ is the number of nearest neighbors that have used the library $L$; $n$ is the number of nearest neighbors; and the $\text{Score}_{\text{Lib-based}}(L)$ ranges from 0 to 1. The libraries that have the highest scores are to be recommended for developers.

3.4 Aggregator Components

In this section, we are to get the recommendation score by combining $\text{Score}_{\text{RM-based}}$ and $\text{Score}_{\text{Lib-based}}$, denoted as $\text{Score}_{\text{AppLibRec}}$. For each library, the recommendation score is defined as follows:

$$\text{Score}_{\text{AppLibRec}} = \alpha \times \text{Score}_{\text{RM-based}} + \beta \times \text{Score}_{\text{Lib-based}}$$  \hspace{1cm} (10)$$

In the above function, $\alpha$ and $\beta$ represent the weights of $\text{RM-based}$ and $\text{Lib-based}$ respectively. We set $\alpha + \beta = 1$, so the value of $\text{Score}_{\text{AppLibRec}}$ will range from 0 to 1. The top $n$ libraries which have the highest score are the third-party libraries which we are going to recommend to developers for this application.

4. Experiment Setup

Though, our methodology and framework can be applied to any types of software development, in the experiment we apply our method to mobile Apps as subjects. We conducted an experiment on a big set of real world mobile applications on github, and we filtered those applications by the following steps with their criteria:

4.1 App Identification

In this sub section, we introduce the process to find the mobile applications set for our following experience.

We first search the mobile application in github by the topic search functionality of github. We choose the "android-application" as our search topic with 3,648 mobile applications in the search result, because the topic of "android" contains not only the applications, but also many framework, libraries, tools, monitors, and so on. The screenshot Fig. 7 shows some details of the search result. We define this as $\text{Set-A}$. After that, we filter those applications by the following criteria:

1. The application contains README file

Our $\text{RM-based}$ approach requires textual description of mobile application in measuring the similarity among applications. So we have to select the applications which have the README file. From these README files, we extract the effective description information about application by text pre-processing.

2. The application uses third party libraries

Our $\text{Lib-based}$ approach requires a set of libraries which have been used in application by analysing the source
code files as the training set. So we have to select the applications which are using third party libraries. Of course, third party libraries are so wildly used that this filter doesn’t drop many applications from the Set-A of search result, but by manually review this filtering step, we drop all the empty projects from Set-A from the result of this step.

3. The application is not fork from other application

This is intended to filter out the duplicated clone applications, because they are not independent Apps, and the forked applications are likely to use the same libraries with the original ones, so they are not the typical sample data for our following experiment.

After filtering applications by using the above 3 filters, we gain the final data set for our experiment with 1,013 applications, including popular projects such as AdAway (303.0 kLOC) which is an open source ad blocker for Android using the hosts file, NewsBlur (419.4 kLOC) which is a personal news reader, AnySoftKeyboard (186.0 kLOC) which is an on-screen keyboard with multiple languages support. We name the data set with 1,013 mobile applications with Set-B.

4.2 Library Identification

The mobile Apps are developed based on the various frameworks and technologies, so after we gain the 1,013 mobile applications with Set-B, we have to extract the libraries from the Set-B by using the following 2 steps.

4.2.1 From Maven integration

We try to find the maven integration of the project. If the maven integration is existing and available, then we can locate the pom.xml file of this project and extract the third party libraries information to identify the libraries, that’s the fastest and easiest way to find the third party libraries usage. Then we name the set of libraries as Set1.

4.2.2 From Source Code

If we cannot find the maven integration information as the first step, we choose to identify the third party libraries from the source code by analyzing the import statements in their source code files. A typical Android mobile application usually uses several third party libraries which are used by developers via importing in their own java class files. The Fig. 8 show an example of the “import” statements of java source code file. Then, we model the libraries data as a forest W. Every tree in the forest W represents a third party library, the non-leaf nodes in the tree represent potential libraries since they might be sub-packages of libraries, while leaf nodes in the tree represent APIs in the sub-packages of libraries. After that, we compare these potential libraries with the libraries we identified in the Maven repository (Set1). If a potential library can be found in Set1, we denote it as a true library. And we record the potential libraries which cannot be matched in Set1 into another set which we name it as Set2.

An Illustrative Example. Figure 9 illustrates the corresponding third party libraries forest extracted from Fig. 8. We extract the four potential libraries {org.junit, net.evendanan.chauffeur.lib, net.evendanan. frankenrobot, org.robolectric}. These packages which are extracted as recommendatory libraries do not have leaf “sibling” nodes in our experimental dataset, i.e., all the “sibling” nodes of these sub-packages are non-leaf nodes if they have “sibling” nodes. And all of them can be found in Set1, thus they are true libraries.

4.2.3 Dropping Libs

For each testing application, we drop half of the libraries (ClassA), and the remaining libraries (ClassB) are taken as inputs to our Lib-based component. We do this to
simulate the situation that a developer knows some of the
needed libraries which is normal, and needs help to find
other relevant libraries. If they do know about needed
libraries, we can use only our RM-based component which
takes as input textual description to recommend libraries for
developers. The method of selecting dropped libraries is
based on clustering algorithm as the following:

1. First step. We list all the third party libraries used by
the 1,013 mobile applications with Set-B. There is 685
independent libs.

2. Second step. We use all the 685 libs above as nodes,
and labeled each node with the number of usage in
Set-B.

3. Third step. We use K-means clustering algorithm and
set K with 2. So that this step can cluster our libs in
Set-B into 2 class by the usage of these libs. The usage
of libs in one class (ClassA) is more than in the other
class (ClassB) by significance testing.

4. Fourth step. We drop all the libs in the ClassA,
and the remaining ClassB is what we need.

4.3 Evaluation Metrics and Procedure

In our experiment, the Precision and Recall metrics are em-
ployed as measurement criteria of the recommendation ac-
curacy. Precision and recall are defined as [19]:

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{11}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{12}
\]

In above functions, \(TP\) is True Positive; \(FP\) is False
Positive; and \(FN\) is False Negative.

Suppose that there are \(m\) applications, for each appli-
cation \(a_i\), let ground truth be the set of libraries \(GT_i\). We
recommend the set of top-k libraries \(P_k\) for \(a_i\) according
to our approach. In the set of all recommendations \(P_k\) (for
all projects), that includes at least one library in the ground
truth. The Precision@k and Recall@k for the \(m\) applications
are defined as:

\[
\text{Precision@k} = \frac{1}{m} \sum_{i=1}^{m} \frac{|P_k \cap GT_i|}{|P_i|} \tag{13}
\]

\[
\text{Recall@k} = \frac{1}{m} \sum_{i=1}^{m} \frac{|P_k \cap GT_i|}{|GT_i|} \tag{14}
\]

The ten-fold cross validation is employed to measuring
the accuracy of our method. We use scikit-learn\(^1\) which is
a Python module for machine learning to perform ten-fold
cross validation.

Our approach AppLibRec combines RM-based compo-
nent and Lib-based component, that takes a number of pa-
rameters. The RM-based component uses LDA that accepts
five parameters. We set the maximum number of iterations
to 1500 and the hyperparameters \(\alpha\) and \(\beta\) to 50/k and 0.01,
where \(k\) is the number of topics.

We use percentages of distinct terms in out training
data rather than a fix number to set the number of topic.
We vary the number of topics for 1% to 11% of the num-
ber of distinct words in our training data which is proposed
by [33], and we set \(k = 1\%\) by default. As the RM-based
component would find its k-nearest neighbors, we set the
number of neighbors to 30 by default.

We use Python LDA package\(^{11}\) as the LDA implemen-
tation, which uses collapsed Gibbs sampling.

The Lib-based component uses collaboration filtering,
and we set the number of neighbors to 30 by default. The
other parameters in Eq. (10) to their default values i.e., \(\alpha =
0.3\) and \(\beta = 0.7\).

For the state-of-the-art approach LibRec [29], there
are five parameters and we set the parameters as follows:
\(\text{minsup} = 0.1\), \(\text{minconf} = 0.8\), \(n\) (the number of neighbors)
= 20, \(\alpha = 0.5\), and \(\beta = 0.5\).

4.4 Time Series

For evaluating the performance improvement by time series
analysis and the dynamic ALT methodology as a support
of our AppLibRec, we set up the independent experiment
environment for ALT methodology as the following:

\(^1\)http://scikit-learn.org/stable/modules/cross_validation.html#k-
fold

\(^{11}\)https://pypi.python.org/pypi/lda
Setting Time window
We separate all the mobile applications to 10 independent folder averagely and randomly, in order to simulate the time sequential creation of new mobile application projects. Each folder has 101 Apps, so that we set 10 time windows. And we use recall and precision as metrics to compare with the original AppLibRec which is used as a benchmark.

In that experiment, we use the recommended libraries \(S_{t+1}\) as the testing data to evaluate the current one \(S_t\). So that we get 9 tests.

Setting Forgetting factor
Because we get all the Apps on one-time, so the setting forgetting factor is as simple as to compare the recall and precision on the last time window (the 9th time window), and pick up the forgetting factor with highest recall and precision.

5. Research and Evaluation
We are interested to answer the following research questions as the evaluation and discussion of our research:

RQ1 How effective is our proposed AppLibRec? How much improvement could our proposed approach gain?

Our AppLibRec combines \textit{RM-based} component which use LDA model and \textit{Lib-based} component which employ the collaboration filtering approach. In our Lib-based component, PCC is employed as metric to measure the distance between two applications. The state-of-the-art work introduced by [29] is a combination of association rule mining (i.e., \textit{Rule}) and collaboration filtering which uses cosine similarity as the metric to compute this distance. To answer this question, we compare our approach with the following baselines:

1. **Association Rule Mining (Rule):** This method recommends libraries by mining library usage patterns expressed as association rules.

2. **Collaboration Filtering with Cosine Similarity (CF.cosine):** This method recommends libraries by investigating the set of libraries that are used by similar applications, using a nearest neighbor based collaborative filtering approach. It uses cosine similarity as the metric to compute this distance.

3. **LibRec:** Thung et al. proposed LibRec which combines association rule mining and collaborative filtering with cosine similarity to recommend libraries for projects on GitHub.

4. **LDA:** LDA is commonly used latent factor based model for text similarity. Our \textit{RM-based} component recommends libraries by finding similar applications, using LDA model to find the similar applications.

5. **Collaboration Filtering with PCC (CF.pcc):** Our \textit{Lib-based} component recommends libraries by investigating the set of libraries that are used by similar applications, also using a nearest neighbor based collaborative filtering approach. It uses PCC as the metric to compute this distance.

AppLibRec from Thung et al.’s study [29] is the comparison benchmark and baseline of our work AppLibRec, and the Table 1 and Table 2 present the precision@k and recall@k (\(k = 1, 3, 5, 7, 10, 15, \) and \(20\) of AppLibRec compared with the benchmark approaches and the improvements of AppLibRec. The improvement of our approach over the benchmark approaches are substantial.

RQ2 What is the performance of the \textit{RM-based} component and \textit{Lib-based} component?
To answer this research question, we investigate the performance of two component of AppLibRec separately. The result is presented in Table 1 and Table 2. Table 1 shows that AppLibRec outperforms the \textit{RM-based} (i.e., LDA) component above 10% on precision, at the same time Table 1 also shows that AppLibRec outperforms the \textit{Lib-based} (i.e., CF.pcc) component about 1.5% on average precision.

From Table 2, AppLibRec outperforms \textit{RM-based} component above 10% on average recall, and at the same time AppLibRec outperforms \textit{Lib-based} component from 0.90% to 2.80% on average recall.

The results show that it is beneficial to combine the \textit{RM-based} and \textit{Lib-based} components, as it improves accuracy significantly.

RQ3 What is the effect of varying the number of topics to the performance of AppLibRec?
We next investigate the effect of varying the number of topics in LDA. We vary the number of topics for 1% to 11% of the number of distinct terms in our training data [33].

The result can be seen that the performance of AppLibRec over the various numbers of topics only varies slightly, so our AppLibRec is stable to different number of topics that in a reasonable range.

RQ4 What is the effect of varying the number of neighbors to the performance of AppLibRec?
We finally investigate the sensitivity of our approach to the number of nearest neighbors of two components, respectively. In reality, it is hard to know the best number, and thus it is best if our method is robust on a particular range. We vary the number of nearest neighbors of two components from 5 to 40 and show the accuracy of AppLibRec to investigate the sensitivity of AppLibRec on this parameter. As shown in Fig. 11, it presents the precision of AppLibRec for different numbers of nearest neighbour of \textit{RM-based}, respectively. From Fig. 12, it presents the precision of \textit{Lib-based} for different numbers of nearest neighbour of \textit{Lib-based} respectively. We find that the precision of our approach is relatively stable over different numbers of nearest neighbors.
Table 1  Performance comparison by p (precision)

<table>
<thead>
<tr>
<th></th>
<th>AppLibRec</th>
<th>LibRec</th>
<th>Rule</th>
<th>CF cosine</th>
<th>LDA</th>
<th>CF_pcc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p</td>
<td>r</td>
<td>improve</td>
<td>p</td>
<td>improve</td>
<td>p</td>
</tr>
<tr>
<td>@1</td>
<td>0.569</td>
<td>0.501</td>
<td>13.57%</td>
<td>0.071</td>
<td>701.41%</td>
<td>0.491</td>
</tr>
<tr>
<td>@3</td>
<td>0.386</td>
<td>0.278</td>
<td>38.85%</td>
<td>0.068</td>
<td>467.65%</td>
<td>0.276</td>
</tr>
<tr>
<td>@5</td>
<td>0.307</td>
<td>0.223</td>
<td>37.67%</td>
<td>0.065</td>
<td>372.31%</td>
<td>0.214</td>
</tr>
<tr>
<td>@7</td>
<td>0.256</td>
<td>0.186</td>
<td>37.63%</td>
<td>0.061</td>
<td>319.67%</td>
<td>0.181</td>
</tr>
<tr>
<td>@10</td>
<td>0.204</td>
<td>0.153</td>
<td>33.33%</td>
<td>0.057</td>
<td>257.89%</td>
<td>0.146</td>
</tr>
<tr>
<td>@15</td>
<td>0.153</td>
<td>0.114</td>
<td>34.21%</td>
<td>0.052</td>
<td>194.23%</td>
<td>0.109</td>
</tr>
<tr>
<td>@20</td>
<td>0.124</td>
<td>0.088</td>
<td>40.91%</td>
<td>0.048</td>
<td>158.33%</td>
<td>0.087</td>
</tr>
</tbody>
</table>

Table 2  Performance comparison by r (recall)

<table>
<thead>
<tr>
<th></th>
<th>AppLibRec</th>
<th>LibRec</th>
<th>Rule</th>
<th>CF cosine</th>
<th>LDA</th>
<th>CF_pcc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r</td>
<td>r</td>
<td>improve</td>
<td>r</td>
<td>improve</td>
<td>r</td>
</tr>
<tr>
<td>@1</td>
<td>0.147</td>
<td>0.128</td>
<td>14.84%</td>
<td>0.026</td>
<td>465.38%</td>
<td>0.120</td>
</tr>
<tr>
<td>@3</td>
<td>0.270</td>
<td>0.201</td>
<td>34.33%</td>
<td>0.031</td>
<td>770.97%</td>
<td>0.196</td>
</tr>
<tr>
<td>@5</td>
<td>0.337</td>
<td>0.249</td>
<td>35.34%</td>
<td>0.037</td>
<td>810.81%</td>
<td>0.244</td>
</tr>
<tr>
<td>@7</td>
<td>0.382</td>
<td>0.231</td>
<td>31.27%</td>
<td>0.040</td>
<td>855.00%</td>
<td>0.281</td>
</tr>
<tr>
<td>@10</td>
<td>0.429</td>
<td>0.291</td>
<td>30.79%</td>
<td>0.045</td>
<td>897.67%</td>
<td>0.323</td>
</tr>
<tr>
<td>@15</td>
<td>0.478</td>
<td>0.328</td>
<td>31.68%</td>
<td>0.045</td>
<td>962.22%</td>
<td>0.358</td>
</tr>
<tr>
<td>@20</td>
<td>0.508</td>
<td>0.363</td>
<td>31.61%</td>
<td>0.052</td>
<td>876.92%</td>
<td>0.378</td>
</tr>
</tbody>
</table>

Fig. 10  Recall@5, Recall@10, Precision@5, and Precision@10 for different numbers of topics of RM-based (1–11% of the number of distinct words in the training data)

Fig. 11  Recall@5, Recall@10, Precision@5, and Precision@10 for different numbers of neighbors of RM-based (from 5 to 40)

Fig. 12  Recall@5, Recall@10, Precision@5, and Precision@10 for different numbers of neighbors of Lib-based (from 5 to 40)

RQ5 As a state-of-art approach what kind of scenario the AppLibRec doesn’t work very well?

As a discussion question, we focus the real-world software development and want to know what kind of scenario the AppLibRec doesn’t work very well as a state-of-art approach. An to answer this question, we made a simple survey which is designed as following:

1. Survey Design. We design a very simple survey with just one question of “What kind of scenario do you think the AppLibRec doesn’t work very well?” which 2 attachments. One attachment is all the FP (false positive) and FN (false negative) tests in the experiment result, the other attachment is the function specification of the AppLibRec.

2. Informant Selection. We select 50 experienced Java engineers with more than 5 Java experience from
outsourcing company. The reason to pick up Informant from outsourcing company is that in outsourcing company developers contribute on numbers of projects one by one, so they have far more chances to choose libs for a project than the other kind of companies such as products company.

Finally, we collect all the 50 surveys from these engineers, and we analyzed all the surveys and classify the answers on them manually. The result show that:

1. **Version compatibility.** 68% engineers believe that sometimes the difference between versions is big and version compatibility is critical. At that time, the recommendation of third party lib is not enough, the engineers also need help for choosing right version.

2. **Non-functional feature of libs.** 62% engineers believe that they need more reliable description of libs when they make choosing such as the performance, security, testability, maintainability, reliability, and so on, and all these characters need data and detail information to support their choosing. But AppLibRec is coarse-grained for them.

3. **Feature integration level of library.** 36% engineers believe that there are 2 patterns of choosing libs. Someone tend to use one lib for one kind of usage which the lib work very well, at the same time the others tend to use big and integrated lib with multi-features for many usage. In real world project engineers cannot say which tendency is better than the other, but AppLibRec obviously work as the former.

**RQ6 How the time series improve the performance of AppLibRec?**

As the setting described in the last section, we draw a line chat shown on Fig. 13 to elaborate the selection of forgetting factor.

Just as shown on Fig. 13, the time series based Collaborative Filtering (CF) has higher recall rate and precision rate when we use 3 as the forgetting factor in Ebbinghaus forgetting function, so we use 3 as the forgetting factor, so as the following discussion.

Just as shown on Fig. 14, the time series based Collaborative Filtering (CF) has very good performance and the its precision and recall is improving significantly, smoothly and continuously. On the 9 cycles iteration, the performance of time series based Collaborative Filtering (CF) has been improved to a very high level.

After analysis. We believe that the time series based Collaborative Filtering (CF) could improve our framework by the following 3 aspects:

1. When the environment is lack of data to train the model very well at the beginning.

2. When the environment has too much data to do the full scale update whenever there is new Apps and Libs or few Apps and Libs changed.

3. When the third party libraries are evolved faster and faster, so that there should be a forgetting factor to avoid the outdated history information impact our framework very much.

4. The performance of time series based Collaborative Filtering (CF) is improved very significantly after several iterations (time windows), so we believe that the model is more reliable than the original model who use full scale data to build.

**6. Threats to Validity**

At this section, we discuss the potential threats of our paper.

1. **Experiment**

   Threats to internal validity refers to error in our experiment. We have checked our dataset and experiment seriously, and the main of our experiment steps are automated without subjective factors. Besides, all the experiment result can be reproduced, so there are limited errors that we don’t notice.

2. **Generalizability**

   Threats to external validity refers to the generalizability of our result. In this paper, there are 1,013 mobile
applications with Set-B used in our experiment. All of these projects are based on android with android-application search topic on github, and all of them are marked as Java projects.

However, our proposed recommendation system can be applied to general software. There is no specific recommendation mechanism and restrictions for mobile Apps. Such kind of Generalizability becomes an advantage of the proposals.

Our proposed recommendation experiment setting focus the mobile App development in terms of: 1) The library recommendation problem in mobile App development is more necessary to the developers than general development, because the development relies on simulator heavily so the test is much harder to execute. 2) Mobile Apps continue to grow in popularity at a rapid pace globally, and they become one of the most popular interface between the people and the internet. 3) Mobile Apps is more suitable for libraries recommendation, because all the mobile Apps are independent, are packed to be an deployable package, and are similar on structure and architecture aspect.

In this paper, we the methodology and framework can be used with both mobile development and general development without significant modification.

3. Evaluation

Threats to construct validity refers to the suitability of experiment measures, We employ both Precision and Recall to evaluate the effectiveness of our approach. These measures have been used by many previous studies [12], [18], [31].

7. Related Work

Thung et al.’s study [29] is most related to ours. We got the idea from their work which proposed LibRec which combines association rule mining and a nearest-neighbor-based collaborative filtering approach to recommend libraries for projects on GitHub. Our work is different from Thung et al.’s study in several ways:

We and Thung et al. focus the different aspect of software development: With the programming language independence, we focus on library recommendation for mobile application; and with the platform independence, Thung et al.’s study focuses on library recommendation for Java projects on GitHub.

We leverage a different information source namely textual description of apps, and after using topic models to extract the topic distributions of apps, our approach could recommend libraries from apps which has similar topic distributions.

Although both of the two approaches use collaborative filtering, our approach uses Pearson Correlation Coefficient (PCC) to measure the similarity of two apps based on their commonly used libraries, while LibRec uses cosine similarity to measure the similarity of two apps. Experiments show that PCC achieves better performance than cosine similarity.

Recommender systems are widely utilized in software engineering. Many previous studies have proposed approaches to recommend various code elements (e.g., method calls, blocks of code), using various information sources (e.g., source code, commit logs) by various heuristics. Mandelin et al. [16] propose the problem of jungloid mining, jungloid mining code fragments that satisfy the query which describes the input and output types. Robbes et al. [22] improve code auto-completion by analyzing recorded program history. Serval tools provide real-time code clone detection [13], [15]. Zimmermann et al. [35] use association rule mining to infer that if changes are made to a set of program elements, then another set of program elements need to also be changed. McCarley et al. [17] investigate the history of methods that the developers have used before to recommend methods to a developer in a group of developers. Chan et al. [4] and Thung et al. [30] recommend API methods through natural-language queries. Those studies form a number of methods that convert the source object type to the destination object type.

Compared with these code-level recommendation systems, there are several previous that works at a different level of granularity i.e., library-level, and recommends analogical third party libraries to the developers. Thung et al. [29] extract the co-occurrence libraries which are always commonly used together on the historical third party library. Teyton et al. [28] recommend libraries that can replace an existing library in a software project by analyzing the evolution of projects’ dependencies on third party libraries. Chen et al. [5] mine analogous libraries from the crowdsourced knowledge in domain-specific Q&A sites by association rule mining and phrase chunking methods, recommend libraries for the same programming Language and comparable libraries across different programming languages.

Latent Dirichlet Allocation [2] is a well-known unsupervised topic modeling technique, which learns topics from a corpus of sentences derived from README text of mobile application. The origin of the topic model is latent semantic index Semantic Indexing (LSI) [8]. Based on the LSI model, Hofmann et al. [11] proposed a probabilistic Latent Semantic Indexing (pLSI) that is a actual topic model. Blei et al. [2] propose the Latent Dirichlet Allocation (LDA) model based on the pLSI model, LDA is a complete probabilistic generation model.

Inspired by these studies, our approach not only relies on the information about the third party libraries that other (e.g., historical) mobile apps used, but also relies on the textual descriptions of mobile applications. Also, we focus a different problem, i.e., recommending third-party libraries for mobile apps.

8. Conclusion and Future Work

The use of third-party libraries allows the developers to reuse the reliable and mature component, reduce the development cost, focus on the key business logic of their ap-
lications, and improves the stability of applications. But the large number of third-party libraries which also always evolve iteratively and rapidly makes it hard for developers to find available and appropriate third-party libraries, they may not become aware of these libraries as they are released. That makes the the choosing of third-party libraries time-consuming, risky, and with difficulty for most developers even they are experienced and skillful.

Many previous studies have proposed some approaches to recommend third-party libraries which are based on the libraries that an application has used. Amongst them, association rule mining is widely used. However, these applications may not contain the sufficient co-occurrence of third-party libraries. So that issue limited the recommendation performance of the previous researches.

The main thinking of our research is to take the textual descriptions of applications into account and take other description information of applications into consideration.

In this paper, we propose a new method AppLibRec to automatically recommend third-party libraries for the developers. AppLibRec combines Latent Dirichlet Allocation and collaborative filtering to automatically recommend libraries for developers. AppLibRec performs two kinds of analysis: 1) README file based analysis (RM-based) by LDA, and 2) libraries based analysis (Lib-based) by collaborative filtering. Indeed, our approach could be deployed first to recommend the sub packages of third-party libraries which can feed to the existing approaches to recommend particular methods or an entire third-party libraries.

We investigate the evaluate the performance of our approach (AppLibRec) by comparing with the state-of-the-art approaches by a well-designed experiment. The result of this experiment shows that our approach outperforms the state-of-the-art.

In the future, we plan to improve the effectiveness of AppLibRec further by employing advanced NLP techniques, for example, integrate the Gaussian LDA method proposed by Das et al. [7] or employ other text mining solutions (e.g., [20], [26], [32]) to develop a better approach that can increase the precision rate and recall rate in semantic sparse description file of application. We also plan to include more mobile applications to further validation our results.

In the future, we plan to improve the effectiveness of AppLibRec further by: 1) use more accurate NLP technologies improve the correlation analysis such as integrate the Gaussian LDA method proposed by Das et al. [7], other text mining solutions (e.g., [20], [26], [32]), and so on; 2) try other similarity calculation methodology for collaborative filtering to increase the precision rate and recall rate in semantic sparse description file of application; 3) on this research, we just bring the time-series approach to the framework and make an experiment as comparative trial, the result shows that the performance improves continuously, so in further research we plan to use time-series data-mining as the basic methodology to update the framework; 4) generalize and evaluate the usage of AppLibRec to help the software development not only mobile applications but also general software development.

References


Xiaoqiong Zhao is a Ph.D candidate in College of Computer Science and Technology, Zhejiang University, China. His research interests include empirical software engineering.

Shanping Li received his Ph.D degree from the College of Computer Science and Technology, Zhejiang University in 1993. He is currently a professor in the College of Computer Science and Technology, Zhejiang University. His research interests include Software Engineering, Distributed Computing, and the Linux Operating System.

Huan Yu is a Ph.D candidate in the College of Computer Science and Technology, Zhejiang University, Hangzhou. Her research interests include mining software repository, blockchain and software repositories mining.

Ye Wang received the Ph.D. degree in Computer Science from Zhejiang University in 2013. She is a lecturer in School of Computer and Information Engineering, Zhejiang Gongshang University. Her research interests include software engineering and service computing.

Weiwei Qiu received the Ph.D. degree in Computer Science from Zhejiang University in 2015. Her research interests include software engineering, software reliability, cloud computing and services computing.