Mining and Recommending Software Features across Multiple Web Repositories

Yue Yu, Huaimin Wang, Gang Yin, Bo Liu
National Laboratory for Parallel and Distributed Processing
School of Computer Science, National University of Defense Technology, Changsha, 410073, China
yuyue_whu@foxmail.com, whm_w@163.com, jack_nudt@163.com

ABSTRACT

The "*Internetware*" paradigm is fundamentally changing the traditional way of software development. More and more software projects are developed, maintained and shared on the Internet. However, a large quantity of heterogeneous software resources have not been organized in a reasonable and efficient way. Software feature is an ideal material to characterize software resources. The effectiveness of feature-related tasks will be greatly improved, if a multi-grained feature repository is available. In this paper, we propose a novel approach for organizing, analyzing and recommending software features. Firstly, we construct a *Hierarchical rEpository of Software feAture* (HESA). Then, we mine the hidden affinities among the features and recommend relevant and high-quality features to stakeholders based on HESA. Finally, we conduct a user study to evaluate our approach quantitatively. The results show that HESA can organize software features in a more reasonable way compared to the traditional and the state-of-the-art approaches. The result of feature recommendation is effective and interesting.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Mining Software Repository; H.3.3 [**Information Storage and retrieval**]: Feature Model, Clustering, Query formulation

General Terms

Algorithms, Human Factors

Keywords

Mining Software Repository, Domain Analysis, Feature Ontology, Recommender System

1. INTRODUCTION

The Internet is undergoing a tremendous change towards the globalized computing environment. With the vision of "Internet as computer"[21][23], more and more software projects are developed, maintained and diffused through the Internet computing environment. The Internet-based software repositories, such as Sourceforge.net[1], Freecode.com[2], Ohloh.com[3] and Softpedia.com[4], have hosted large amounts of software projects, which are fundamentally changing the traditional paradigms of software development. Around the repositories, manifold reusable software resources[13] have been accumulated, including code bases, execution traces, historical code changes, mailing lists, bug databases, software descriptions, social tags, user evaluations and so on.

However, all of these valuable resources have not been reorganized in a reasonable and efficient way to assist in the activities of software development[30]. For example, a large proportion of projects in the above repositories have not been categorized or marked with some effective tags. In Sourceforge.net, there are 39.8% software projects have no category label and in Ohloh.com 61.68% projects have not tagged by users. Table 1 (according to data in mid-2011) presents the details about our statistical results. Considering the large-scale, heterogeneous and multi-grained software resources, it is a great challenge for stakeholders to retrieve the suitable one.

Table 1: Labels in open source communities

Repository	total projects	unique labels	ratio (#label=0)	ratio (#label=0,1)
SourceForge	298,402	363	39.80%	77.00%
Ohloh	417,344	102,298	61.68%	69.89%
Freecode	43,864	6,432	8.61%	20.60%

As a kind of visible attributes which capture and identify commonalities and differences in a software domain, *Feature*[3][15] is an ideal material to represent the software resources. For example, when a company wants to develop a new commercial software product about Video-Player, domain analysts might evaluate user comments to pick out outstanding competing products, analyze and extract the reusable feature assets, combine the related function points and design a novel feature model. Based on the feature model, developers match the features with corresponding software resources including code fragments, components and mature open source software.

However, classic feature analysis techniques, such as Feature Oriented Domain Analysis (FODA)[14] and Domain Analysis and Reuse Environment (DARE)[9], are heavily relied on the experience of domain experts and plenty of

[1]http://sourceforge.net
[2]http://freecode.com
[3]http://www.ohloh.net
[4]http://www.softpedia.com

Internetware '13 October 23-24 2013, Changsha, China

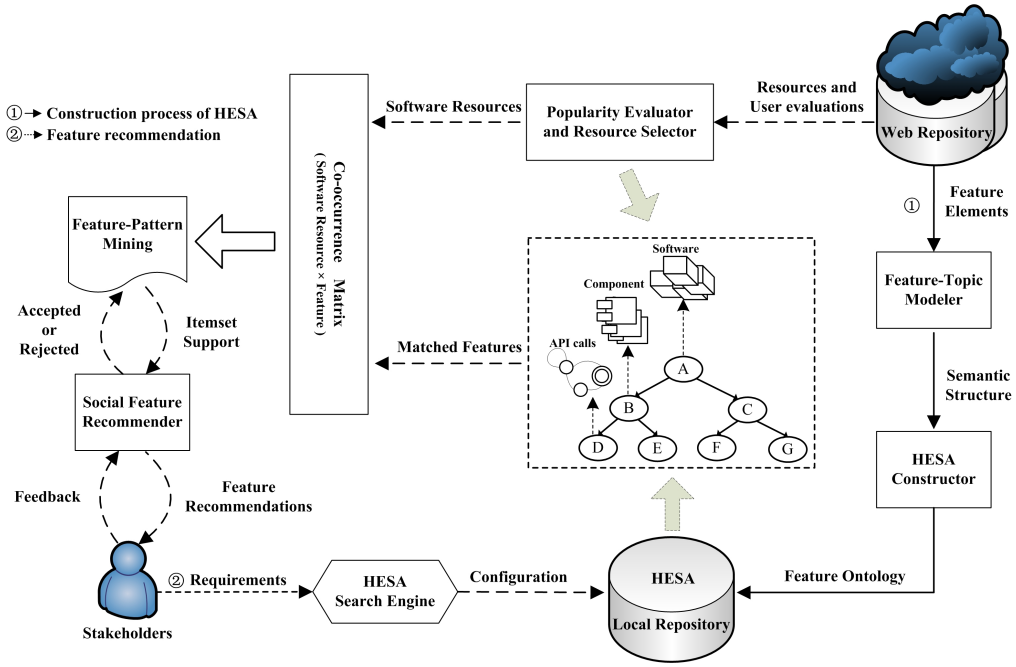


Figure 1: The overview of HESA construction and feature recommendation

market survey data. Hence, the feature analysis is a labor-intensive and error-prone process. Nowadays, in order to promote software products to users, stakeholders write some marketing-like summaries, release notes and feature descriptions on the profile pages via natural language, as shown in Figure 2. The massive number of software profiles can be treated as a kind of repository contains a wealth of information about social software features.

In this paper, we propose a novel approach for organizing, analyzing and recommending software features to reduce the costs of domain analysis. First of all, mining the hidden semantic structure, we construct a *Hierarchical rEpository of Software feAture* (HESA) using an improved agglomerative hierarchical clustering algorithm. The features are organized as a kind of hierarchical structure in HESA. From top down, the semantic granularity is finer and finer accompanying with the increasing number of features, which can satisfy the requirements of multi-grained reuse environments. Then, utilizing the search engine for HESA, features in the specific layers are retrieved and tied together to mine the affinities among them. Finally, we design a novel strategy to evaluate the popularity and quality of software products and circularly recommend features to stakeholders.

The rest of this paper is or organized as follows. Section 2 introduces the overview of our work. Section 3 describes the construction of HESA in detail. We mine the hidden relationship of software features and present the novel approach of feature recommendation in Section 4. Experiments and analysis can be found in Section 5. Finally, we review related work in Section 6 and draw conclusions in Section 7.

2. OVERVIEW

The objective of this paper is to recommend a set of the most relevant and high-quality software features to stakeholders. We have collected a massive number of social software features to build a *Hierarchical rEpository of Software feAture* (HESA). At the beginning, stakeholders can just input limited information for the initial idea about an innovative software product. When stakeholders accept some recommended features or provide more requirements, our system would refine the recommendations in the next stage.

Before describing the specific details of the underlying algorithms, an architectural overview of approach will be provided as below. There are actually two processes concerning the application of our method, i.e., the construction process of HESA and the process of feature recommendation to stakeholders. As depicted in Figure 1, the construction process consists of three primary modules and the input data are software profiles collected and updated continuously by a web crawler. There is a wealth of information on a software profile page as shown in Figure 2. Our system can automatically extract social feature elements (blue boxes), softsware categories (green box) and the user evaluations (red boxes).

The *Feature-Topic Modeler* is responsible for mining the semantic structures hidden in social feature elements. Then, the *HESA Constructor* will merge the synonymic elements and build the feature ontology based on the semantic structures. After all the raw data under our category are disposed, the construction process of HESA is finished.

In terms of user evaluations, the *Popularity Evaluator and Resource Selector* can classify the software resources into different groups such as popular and high-quality group and unpopular and low-quality group. When stakeholders input their requirements of a specific domain, the *HESA Search Engine* can retrieve the features in the corresponding granularity. Combining the feature and the different group of software resources, the system can mine the *Feature-Pattern* based on the resource-by-feature matrix. The last module, *Feature Recommender*, outputs a set of features to stakeholders and gather their feedback. Then, according to the feedback, it will circulate the mining process and recommend more relevant features to stakeholders.

3. THE CONSTRUCTION OF HESA

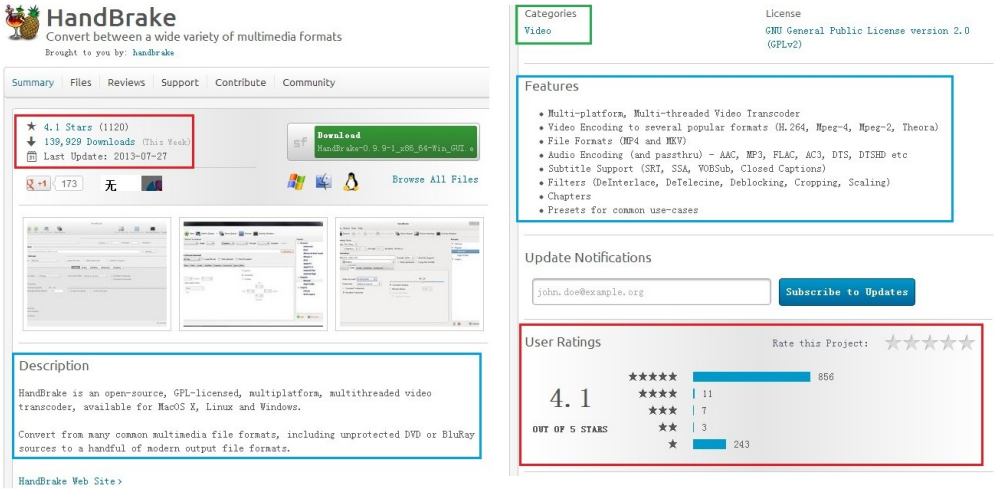


Figure 2: An example of the software profile

In this section, we present the key algorithms of the HESA construction. You can find more details about our stages, algorithms and data analysis in our previous work[30].

3.1 Social Feature Elements in Profiles

Social feature element is a kind of raw descriptions written by different users of web repositories, which can indicate a functional characteristic or concept of a software product. In this paper, all the social feature elements are extracted from Softpedia.com, Freecode.com and Sourceforge.net.

Due to the open, dynamic and uncontrollable natures of the Internet[22], different people describe the functions in terms of their personal understanding. We face two main challenges of the unstructured social feature elements.

Hybrid Semantic level: The problem of hybrid semantic level is that different social feature elements describe a common theme in different semantic level, such as the following descriptions:

(1) *"Internationalized GUI"*;

(2) *"Various language packs are available"*;

(3) *"Multi-language supported: including English, Simplified Chinese, Traditional Chinese, Japanese, Korean, German, French, Spanish, Italian, Russian etc"*;

The first two sentence describes the theme of multilingual setting in a general level. However, the last sentence present more details including what kind of languages would be supported. On the one hand, the massive number of social feature elements in different semantic-level are good materials for the construction of flexible granularity ontology. On the other hand, it is a great challenge for the traditional methods to cluster and reorganize these social feature elements.

Synonymic Element: The problem of synonymic element happens when two features are used to describe some common or very similar functional attributes.

Some social feature elements are almost the same with each other, such as the three elements below:

(1) *"Simple user interface"*;

(2) *"User-friendly interface"*;

(3) *"Easy and friendly user interface"*;

Another typical problem is that each pair only shares few core words, such as the following:

(1) *"Ability to update that does not require downloading full package"*;

(2) *"Incremental database updates and often to include information about latest threats"*;

(3) *"Incremental updating system minimizes the size of regular update files"*;

These three elements present the common attribute about incremental updating database, but only the word "***update***" is shared by the two sentences. Thus, social feature elements should be merged together by an effective method.

3.2 Feature-Topic Model

According to the observation in our previous work[30], the probabilistic topic model can be used to mine the semantic structures hidden in the massive number of social feature elements.

In a specific category, such as Video-Player, all the social feature elements in the corpus can be represented as $\mathbf{F}_m = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$, where f_i denotes the ith elements in the corpus. Assuming that K latent topics $\mathbf{T}_k = \{t_1, t_2, \ldots, t_j, \ldots, t_k\}$ are implicit in the social feature elements, where t_j denotes the jth topic. Although a social feature element can be bound up with several topics, it may put more emphasis on some topics than the others. The topic degree within f_i can be represented as a K-dimensional vector $\boldsymbol{v}_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,j}, \ldots, p_{i,k})$, where $p_{i,j}$ is a topic weight describing the extent to which the topic t_j appears in f_i. When $p_{i,j} = 0$, f_i is irrelevant to t_j. Thus, the $\boldsymbol{v}_i, i \in [1, m]$, represented by $\mathbf{V}_m$, can be used to indicate the semantic structure implied in social feature elements. If the $\mathbf{V}_m$ can be obtained, the thematic similarity measure would be induced for each pair of social feature elements and the synonymic elements would be merged together. Because topic models answer what themes or topics a document relates to and quantify how strong such relations are, it is a effective way to learn $\mathbf{V}_m$.

In this paper, we choose Latent Dirichlet Allocation (LDA) [6] to learn $\mathbf{V}_m$, because it has been shown to be more effective for a variety of software engineering purposes[28][4] than other topic models. A social feature element f_i can be viewed as a document which is preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. According to category, we apply LDA to process the documents using the MALLET tool[19] which is an implementation of the Gibbs sampling algorithm[10].

3.3 iAHC: improved Agglomerative Hierarchical Clustering

To support multi-grained reuse environment, the semantic similar social feature elements should be merged and reorganized as a flexible hierarchical structure defined as feature-ontology. In this paper, we present the iAHC algorithm (**Algorithm 1**) integrated with the LDA.

Algorithm 1 improved Agglomerative Hierarchical Clustering

Require:
$\mathbf{F}_m = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$;
feature-topic distribution $\mathbf{V}_m$;
Ensure:
The construction of feature-ongtology;
1: $M \leftarrow D$
2: $featureSet \leftarrow \emptyset$
3: **repeat**
4: $\langle c_i, c_j \rangle = \textbf{findTwoClosestClusters}(M)$
5: merge c_i and c_j as c
6: delete c_i and c_j from M
7: add c to M
8: $centroid = \textbf{calculateCentroid}(c)$
9: **for** $c_i \in c$ **do**
10: $value_s = \textbf{Similarity}(c_i, centroid)$
11: $degree_t = \textbf{calculateTopicDegree}(c_i)$
12: $score_m = \kappa \cdot value_s + \lambda \cdot degree_t$
13: add $score_m$ to $MedoidScore$
14: **end for**
15: $medoid_C = \textbf{findMaximumScores}(MedoidScore)$
16: $score_F = \textbf{Similarity}(medoid_C)$
17: $feature_C = \textbf{mergeMedoid}(medoid_C, score_F)$
18: $\textbf{saveFeaturetoHESA}(M, feature_C)$
19: **until** $|M| = 1$

Initially, every social feature elements is a distinct cluster. Line 4-7 finds the closest two clusters c_i and c_j in the current cluster set M, and merge them into a new cluster c and update M. The proximity used to measure the distance between every two clusters, defined as below:

$$proximity(c_i, c_j) = \frac{\sum_{\substack{fi \in c_i \\ fj \in c_j}} similarity(fi, fj)}{|c_i| \times |c_j|} \quad (1)$$

Where $c_i, c_j \subseteq \mathbf{F}_m$, $similarity(f_i, f_j)$ used to calculate the divergence between any two data point. Based on LDA, the divergence can be understood as the thematic space coordinate distance between the two elements. There are several ways to calculate the divergence between any two *feature-topic* distributions, such as *cosine* similarity, *Jenson-Shannon* divergence and *Kullback-Leibler* divergence. Taking cosine similarity as an example, the Equation is shown as below:

$$\begin{aligned} similarity(f_i, f_j) &= \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{||\boldsymbol{v}_i|| \; ||\boldsymbol{v}_j||} \\ &= \frac{\sum_{r=1}^{k} p_{ir} \times p_{jr}}{\sqrt{\sum_{r=1}^{k} p_{ir}^2} \times \sqrt{\sum_{r=1}^{k} p_{jr}^2}} \end{aligned} \quad (2)$$

Where k is the topic number and p is the probability value of $\boldsymbol{v}$.

Line 8-14 pick out a set of social feature elements from the new cluster, defined as *medoid*, which can be used to represent the theme of c. Two metrics, *similarity value* and *topic degree*, are used to determine the medoid. Firstly, to get the $value_s$, we calculate the similarity between $c_i \in c$ and the *centroid* of c through Equation 2, where the vector $\boldsymbol{v}_{\bar{c}}$ of centroid is calculated by $\boldsymbol{v}_{\bar{c}} = \frac{\sum_{i=1}^{|c|} \boldsymbol{v}_i}{|c|}$. Then, the Equation 3 is used to calculate the $degree_t$ based on the following two important observations of feature-topic distribution V_m in our datasets.

$$degree_t = x_{max} + \frac{1}{e^{\sqrt{\frac{\sum_{r=1}^{\hat{k}} (x_{max} - p_{ir})^2}{\hat{k}}}}} \quad (3)$$

Where x_{max} is the maximum value of $\boldsymbol{v}_i$, and $\hat{k}$ is the frequency when $\boldsymbol{v}_i$ not equal to zero, and p_{ir} is any value that not equals to zero in $\boldsymbol{v}_i$.

Observation 1 The most probable topic reflects the most prominent theme that the document (social feature element) is about.

Observation 2 The more widely and evenly distributed its topics are, the higher-level the document (social feature element) is.

In brief, Equation 3 can ensure the social feature element in the coarsest granularity have the highest score $degree_t \in (0, 2]$, where $x_{max} \in (0, 1]$ can reflect the emphasis topic and the formula $\frac{1}{e^{\sqrt{\sum_{r=1}^{\hat{k}} (x_{max} - w_{ir})^2}}} \in (0, 1]$ can reflect the semantic generality.

The $score_m$ is used to measure the *medoid* calculated as the Equation of line 12, where κ and λ is the empirical coefficients.

Finally, the *medoid* with the highest $score_m$ would be selected. Measuring the similarity for the each pair of elements in $medoid_C$ (line 15), the $feature_C$ (line 17) can be formed by merging distinguished social feature elements whose similarity score below a threshold (set to 0.38). Each iteration in the repeat clause saves the M and $feature_C$ to HESA. On the termination of the algorithm, a feature-ontology for the category is constructed.

After all the feature-ontologies under different categories are generated, the construction process of HESA is finished.

4. FEATURE RECOMMENDATION

In this section, mining the hidden relationship between software features, we present a novel strategy of feature recommendation. Firstly, we classify software resources into different groups. Then, combining the feature with the popular and high-quality resources, we use association rules to mine the feature-patterns. Finally, analyzing the hidden associations between the features, we can recommend relevant features to stakeholders.

4.1 Software Resource Evaluation

When users try out a kind of software resource from web repositories, they may give evaluations according to their experience. Taking software projects in Sourceforge.net, Softpedia.com, Ohloh.com and Freecode.com as an example, we have extracted a large number of user's evaluations from the software profiles including rating, vote, downloads, and subscriptions as shown in table 2.

Because the core services of the web repository are different, there are different types of user evaluation to measure the popularity and quality. Some giant forges like Sourceforge.net host a massive amount of software source code,

Table 2: User evaluation in software profiles

Project Name	Repository	Rating (#Raters)	Vote (#Voters)	#Downloads	#Subscriptions
Hard Disk Sentinel	Softpedia	3.7 Stars (42)		19,347	
Mozilla Firefox	Sourceforge	4.0 Stars (60)		332	
Mozilla Firefox	Ohloh	4.4 Stars (3,492)			12,227
Chromium	Ohloh	4.5 Stars (538)			1,883
PostgreSQL	Freecode		145 Score (153)		755
phpMyAdmin	Freecode		202 Score (213)		956

so the code *#Downloads* is adopted to quantify the project popularity. However, the core service of some repositories is providing social directories such as Ohloh.com. Thus, the *#Subscriptions* is used to measure the project popularity.

In addition, the *Rating* and *Vote* can reflect the quality of the software project. Most of the repositories use the Rating from 1 to 5 stars to measure the project quality. Hence, we convert the *Vote* into a five-point scale by the following formula, where ℓ is the reliability of the web repository.

$$Rating = \ell \cdot \frac{Vote}{\#Voters} \times 5 \quad (4)$$

For example, if ℓ sets to 1, the *Vote* of *PostgreSQL* in Freecode.com can convert to 4.74 points *Rating* by the formula: $1 \times \frac{145}{153} \times 5 \approx 4.74$.

Moreover, a project may have several different ratings when it is hosted across multiple repositories. In this case, to lower the users' personal bias, we choose the rating based on most users' records. As depicted in Table 2, our system automatically select 4.4 Stars in Ohloh.com to represent the quality of *Mozilla Firefox* [5][6], because it is rated by more users (3,492) than the rating in Sourceforge.net (60).

$$Ranking_{pq} = \alpha \cdot \frac{\#Downloads}{\mathbf{MAX}(\#Downloads)} \times \beta \cdot Rating \quad (5)$$

Then, synthesizing the popularity and the quality, we use the Equation 5 to rank the software projects, where α and β ($\alpha \cdot \beta = 1$) are the empirical coefficients used to balance the popularity with the quality. The *#Downloads* can be replaced by the *#Subscriptions* and the $Ranking_{pq} \in (1, 5]$.

In this paper, we pick up the software projects over a $Ranking_{pq}$ threshold. Actually, the software resources can be divided into any number of groups, such as popular group, high-quality group, unpopular group and low-quality group. Combining with the features and different groups of software resources, we can mine the hidden feature pattern and recommend relevant features more accurately. Furthermore, the software resources are manifold, such as software components, code fragments, bug reports and so on. Based on the feature location technique, we can recommend these resources to stakeholders. We really expect to do this research in future work.

4.2 Feature Pattern Mining

We merge the software projects and the software features into a resource-by-feature matrix and a series of implicit feature rules can be discovered. As illustrated in Table 3, the features about *Video-Codec* and *Customizable Option* are essential for a new products in the Video-Play domain, because the most of competing software implement these functionalities. The No.35 may be the latest and novel feature in this domain, because only a few of high-quality products own this functionality. In this paper, we define the *Feature-Pattern* to represent the relationship between software features and

[5]http://www.ohloh.net/p/firefox

[6]http://sourceforge.net/projects/firefox.mirror/

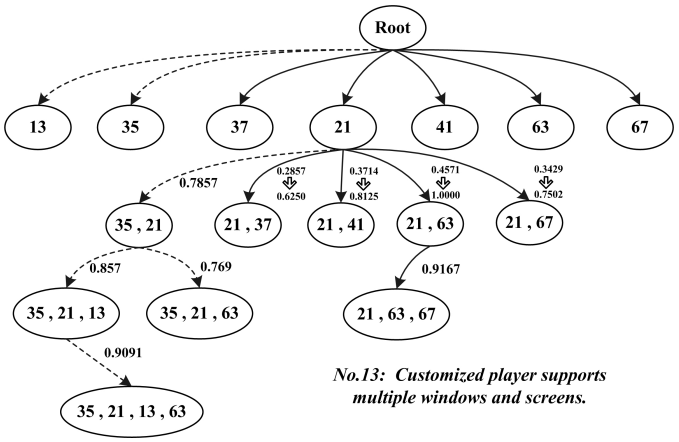


Figure 3: Part of the feature-pattern in Video-Player domain

generate the feature-pattern based on the association rule learning[11][29].

Let $F = f_1, f_2, \ldots, f_m$ be a set of item in the resource-by-feature matrix. An association rule is an implication of the form $\{X \Rightarrow Y\}$, where $X \subset F$, $Y \subset F$ and $X \cap Y = \emptyset$. The $\sigma(X)$, called support count, is used to get the is the number of transactions that contain the X itemset. Therefore, *support* can indicate the frequency and the association degree of the itemsets, defined as below:

$$support(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{|F|} \quad (6)$$

The feature-pattern is a kind of directed graph, which can be represented as $G_{FP} = \langle V, E \rangle$, where V is a set of feature itemsets and E is a sub set of $E(V)$, $E(V) = \{(u, v) | u, v \in V\}$. The element e of E is the a pair of vertices and the $confidence$ is the weight assigned to the arcs. As shown in Equation 7, $confidence$ indicate the percentage of transactions containing a given itemset that also contain the other specific itemset.

$$confidence(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} = \frac{support(X \cup Y)}{support(X)} \quad (7)$$

For example, in Table 3, the $support(No.21 \Rightarrow No.41) = 1/4$ and the $confidence(No.21 \Rightarrow No.41) = 1/3$. In Figure 3, the edge $e = \{(21), (21, 67)\}$ represent the association rule $\{No.21 \Rightarrow No.67\}$ and the weight 0.3429 is the $confidence$ of this rule.

Firstly, we use Apriori algorithm [11] to generate the frequent itemsets (*support* satisfies a threshold 0.35) as the initial value of V. Then, we add the infrequent itemsets except the item within frequent itemsets (*support* below a threshold 0.05) to the V. Finally, we connect the nodes together based on the association rules and allocate the weights of the edges in terms of their $confidence$ values.

Figure 3 illustrates a part of feature-pattern in the Video-Player domain. The solid lines show the connections of the frequent itemsets. We discover that the subgraphs about

some fundamental features always have the homogeneous weights and they are much flatter than the subgraphs of common features. In a specific domain, the fundamental features is implemented by the most of software such as the feature No.21 about video-codec in the Video-Player domain, so it have the high *support* and widely co-occur with other features, which leads the subgraph to become flat. If a connection contains a element of the infrequent itemsets, we use the dotted lines to represent it. We find that the infrequent features may be some innovative functionalities, such as the feature No.13 about displaying video in customized multiple screens. Thus, the connections between infrequent and fundamental features are significant to the feature recommendation.

Table 3: An example of resource-by-feature matrix

Software \ Feature	No.21[1]	No.35[2]	No.37[3]	No.41[4]	No.63[5]	No.67[6]
nDVD	0	0	1	1	1	0
FLV Player	1	0	1	1	1	0
Aviosoft DTV Player	1	1	0	0	1	1
Mac Blu-ray Player	1	0	0	1	1	1

[1] Support multi-formats of video files such as FLV, MPEG4, DIVX, HD-MOV, M2TS, MKA, 3GPP and so on.
[2] Smart stretch lets video smart fit on all monitor with different aspect ratio, avoid video loss or distortion.
[3] Title repeat, chapter repeat, AB repeat function that lets you set your favorite scenes for instant repeat.
[4] Fast Forwards and Backwards with Customizable Speeds.
[5] You can easily configure every option of the Player by using a nice preferences dialog.
[6] Brightness, contrast, hue, saturation and gamma settings.

4.3 Feature Recommendation Algorithm

When a user input his requirements, the system convert the requirements to a set of features which is the same as the preprocessing of raw social feature elements. If it can be matched with the any one of the itemsets in the feature-pattern, the system begin the process of feature recommendation. **Algorithm 2** present a breadth-first recommendation algorithm, where the input are user requirements, the minimum $confidence$ and the feature-pattern. The output $recSet$ is a set of relevant feautres for recommendation.

We firstly initialize the $recSet$ with the matched itemset (Line 1-2). If the matched itemset equals a fundamental feature or feature set, we should update the feature-pattern G_{FP}. Line 4-5 find the maximum weight $maxconf_{tmp}$ within the edges $E(reqItem)$ and update the weight w_i by using blew formula:

$$weight_{new} = \frac{w_i}{maxconf_{tmp}} \in (0, 1] \quad (8)$$

For example, as shown in Figure 3, when we matched the feature No.21, we can find 0.4571 is the $maxconf_{tmp}$ of $E(reqItem)$ and update the weight(0.3714) of {(21), (21,41)} to $0.8125 = \frac{0.3714}{0.4571}$. Therefore, although user set the $minconf$ over 0.75, the feature No.41 also can be recommended.

In addtion, Line 6 associate the $E(reqItem)$ with the infrequent itemsets, which make some interesting features can be recommended to users. Then, we merge the itemsets into the $recSet$, whose the $confidence$ values are over the $minconf$ and use the breadth-first search strategy to find all relevant features (Line 8-12).

For example, supposing we begin with {21}, we would update the weights of {(21), (21,37)}, {(21), (21,41)}, {(21), (21,63)} and {(21), (21,67)} and the subgraph of the feature No.35 is added. Then, if we set the $minconf$ to 0.8, we would retrive (21,41), (21,63), (21,63,67) step by step. Finally, we merge them together and the recommendation set is (21,41,63,67). Similarly, if we set the $minconf$ to 0.75, the feature No.35, No.63, No.67, No.13 would be selected out one by one.

Feature recommadtion is a iterative process. Once users choose the recommended features or give their feedback, our system will refine the recommendations in the next stage.

Algorithm 2 a breadth-first recommendation algorithm

Require:
req the feature set of you need;
$minconf$ the threshold of the $confident$; G_{FP} the feature pattern graph;
Ensure:
$recSet$ a set of features recommended to users;
1: $reqItem \leftarrow$ **matchItemsets**(req)
2: $recSet \leftarrow reqItem$
3: **if** $reqItem = fundSet$ **then**
4: $maxconf_{tmp} \leftarrow$ **getMaxWeight**$(E(reqItem))$
5: **updateConfidences**$(E(reqItem), maxconf_{tmp})$
6: **addConnections**$(req, infreqSet)$
7: **end if**
8: $Tset \leftarrow$ **getInitialRecSet**$(G_{FP}(reqItems), minconf)$
9: **repeat**
10: $recSet \leftarrow recSet \cup$ **searchNextDepthSet**$(Tset)$
11: **until** G_{FP} END
12: **return** $recSet$

5. EMPIRICAL EVALUATION

In this section, we present our dataset and experiment setting, research questions and answers, and describe some threats to validity.

5.1 Dataset and Experimental Setting

Dataset: We have collected 187,711, 432,004 and 45,021 projects' profiles from Softpedia.com, Sourceforge.net and Freecode.com respectively. Compared with the other two repositories, the quantity of projects from Freecode.com is relatively small. Thus, we just adopt projects in Softpedia.com and Sourceforge.net for the experiment.

The social feature elements have been classified into 385 categories and we randomly choose the data of 6 unique categories to evaluate our method including Antivirus, Audio-Player, Browser, File-manger, Email and Video-Player. Furthermore, the social feature elements are preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form. To ensure the quality of data, we omit the preprocessed data with less than 6 words. In each category, we choose hundreds of candidate projects to find the popular and high-quality software in the two repositories. Table 4 presents the details about our experiment dataset.

Parameter Setting: As shown in Table 4, for LDA, the number of topics K was empirically set as different value, and the hyper-parameters α and β were set with $\alpha = 50/K$ and $\beta = 0.01$ respectively, and the iteration of Gibbs Sampling was set as 1000. In addition, the coefficients κ and λ of **Algorithm 1** were set as $\kappa = 0.7$ and $\lambda = 0.3$. We treat the popularity is as important as the quality for a software project, so we set $\alpha = 1$ and $\beta = 1$ (Equation 5). The $Ranking_{pq}$ threshold is set as 3.0.

Table 4: Preprocessed experiment datasets

Category	#Feature_sp	#Feature_sf	#Project_sp	#Project_sf	#Topic
Antivirus	2919	1105	667	435	40
Audio-Player	3714	1283	379	530	60
Browser	3010	831	344	177	40
File-Manager	2270	970	330	177	40
Email	8511	1050	823	204	80
Video-Player	3318	2697	379	530	60

5.2 Research Questions

To demonstrate the effectiveness of the approach in this paper, we are interested in the following research questions:

RQ1 How accurate is the clustering result of iAHC? Are the hierarchical structures of HESA reasonable?

RQ2 What the popular and high-quality projects look like? What rules can be mined based on the feature-pattern?

RQ3 Are the recommendations from our approach are reasonable and effective?

5.3 Cross-Validation Design of the User Study

The cross-validation limits potential threats to validity such as fatigue, bias towards tasks, and bias due to unrelated factor. We randomly divided the 30 students from computer school of NUDT into three groups to evaluate the questions. Each group randomly picks up 2 categories and finishes the evaluations in one day, and then we summarize the result.

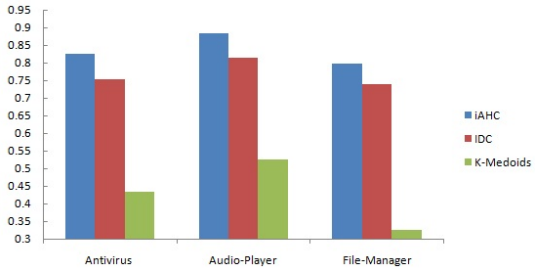
RQ1: Clustering Result and HESA Structure. We choose the K-Medoids (tf-idf), a traditional and widely used clustering algorithm, and the Incremental Diffusive Clustering (IDC), the state-of-the-art technique proposed in paper [8], as the baseline. Especially, the IDC use the feature descriptions from Softpedia.com which is the same as our dataset. We also use the modified version of Can's metric[8] to compute the ideal number of clusters. Then, we retrieve the corresponding number of clusters from HESA for comparison. Precision is a percent of the reasonable elements in a cluster. Figure 4 shows the average value and standard deviation of the judgments given by different groups under the Antivirus, Audio-Player and File-Manger categories.

We can see that our approach achieves the highest precision in all three categories and relatively low deviations. The precisions and deviations are comparatively stable across different categories, which shows the probability that our approach is more generalizable in different domains. We plan to conduct more quantitative experiments in future work.

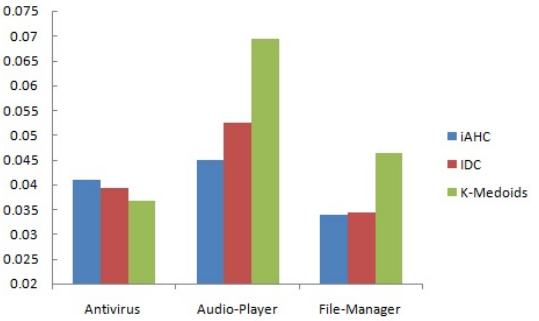
According to the six categories, participants randomly choose 30 clusters in different layers from HESA using the search engine respectively. Each participant is randomly assigned 10 layers and asked to provide a 3-point Likert score for each cluster to indicate whether they agree if the feature is the most representative of all terms. Score 3 means "very reasonable", Score 2 means "reasonable but also have better one", Score 1 means "unreasonable".

Table 5: Evaluation of HESA structure

Category	Score-3	Score-2	Score-1	Likert
Antivirus	33.3%	50.0%	16.7%	2.17
Audio-Player	39.1%	46.3%	14.6%	2.25
Browser	36.8%	41.9%	21.3%	2.16
File-Manager	32.7%	52.4%	14.9%	2.18
Email	36.4%	52.8%	10.8%	2.26
Video-Player	47.2%	41.5%	11.3%	2.36
Average	37.58%	47.48%	14.93%	2.23



(a) the average value



(b) the standard deviation

Figure 4: The clustering results

Table 5 shows that 37.58% features are reasonable, 47.48% partially reasonable and only 14.93% unreasonable. The mean of Likert score is 2.23, which means that the feature selected out by our approach is reasonably meaningful.

RQ2: Project Evaluation and Feature-pattern. As shown in Table 6, we use our evaluation method to select out a fraction of project synthesizing the software popularity and the quality. We can get a reasonable number of candidate features. Specially, to illustrate the importance of project evaluation for feature pattern mining, we lower the threshold (1.0) in File-Manager category. Hence, we get a unnormal number of itemsets which exert a negative impact on feature-pattern mining and feature recommendation.

Figure 3 shows an example of feature-pattern under the Video-Player, where the feature No.21 (about video-codec) and No.63 (about user interface) is the fundamental features. Similarly, we find the feature about virus database updating is one of essential functionalities for an antivirus software.

RQ3: Feature Recommendation. We show some meaningful and interesting case in Table 7. In the Email category, if you input "High level filter system for spam", our system can recommend 5 relevant features. The first three features are directly related to the input, but the third feature about privacy protection is very easy to leave out. The last two features represent the fundamental abilities under the Email category. From these examples, we can demonstrate our ap-

Table 6: Project Evaluation and Feature-pattern

Category	#Project_total	#Project_ranking	Ratio	#|Itemsets|	#Feature_fund
Antivirus	1,082	63	5.82%	75	7
Audio-Player	909	65	7.15%	68	13
Browser	521	57	10.94%	64	8
*File-Manager	447	192	42.95%	187	46
Email	1,027	39	3.80%	53	4
Video-Player	523	91	17.40%	68	5

Table 7: Feature recommandations

Category	Input	Recommendations
Email	-High level filter system for spam.	-Automatically start spam process when Windows starts. -Attachment and keyword filtering. -Privacy guaranteed-your emails never leave your network. -Full support for international characters. -Automatic import of local address book.
Video-Player	-Support multi-formats of video.	-Smart stretch lets video smart fit on all monitor with different aspect ratio, avoid video loss or distortion. -Customized player supports multiple windows and screens. -Play anything including movie, video, audio, music and photo. -Video desktop lets you view video in true background mode like wallpaper.
Audio-Player	-Radio streaming. -Easy to use and friendly User Interface.	-Free Lossless Audio Codec. -Playlists for each day of week or date. -Flexible XML based skinning engine, Create your own skins, or choose one of the available skins. -Contextual Help System.

proach is highly significant of the software development.

5.4 Threats to validity

First, the participants manually judge the clustering results and their ratings could be influenced by fatigue, prior knowledge and the other external factors. These threats were minimized by randomly distributing participants to the various groups and dividing the tasks into multiple parts. Second, the clustering error may exert a negative influence on the recommendation. Third, due to our limited datasets, parameters used in our approach, the evaluation is not comprehensive enough.

6. RELATED WORK

Recently, mining software repository has been brought into focus and many outstanding studies have emerged to support various aspects of software development[12]. However, to the best of our knowledge, fewer previous works have been done for mining software feature and especially construction of feature-ontology to manage software resources. In this section, we review some previous works about feature analysis, ontology learning and recommender system.

In feature analysis area, most approaches extract feature related descriptions from software engineering requirements and then use the clustering algorithm to identify associations and common domain entities[2][9][25]. Mathieu Acher et al.[1] introduced a semi-automated method for easing the transition from product descriptions expressed in a tabular format to feature models. Niu et al.[24] propose an on-demand clustering framework that provided semi-automatic support for analyzing functional requirements in a product line. A decision support platform is proposed in paper [5] to build the feature model by employing natural language processing techniques, external ontology and MediaWiki system. However, the quantity of the existing documents is so limited that the brilliance of data mining techniques cannot be fully exploited. To address this limitation, paper [8] and [20] proposed the Incremental Diffusive Clustering to discover features from a large number of software profiles in Softpedia.com. Based on the features, a recommendations system is build by using association rule mining and the k-Nearest-Neighbor machine learning strategy. Compared with these studies, the clustering algorithm presented in this paper is more effective by mining the semantic structures from social feature elements and especially focus on the construction of feature-ontology.

Ontology learning from text aims at extracting ontological concepts and relation from plain text or Web pages. Paper [18] developed an ontology learning framework using hierarchical cluster and association rule for ontology extraction, merging, and management. Jie Tang et al.[27] proposed a generative probabilistic model to mine the semantic structure between tags and their annotated documents, and then create an ontology based on it. Xiang Li et al.[16] enhance an agglomerative hierarchical clustering framework by integrating it with a topic model to capture thematic correlations among tags. In this paper, to support multi-grained reuse, emphases of the feature-ontology's construction is on the measure of similarity and granularity instead of generality.

As an indispensable type of information filtering technique, recommender systems have attracted a lot of attention in the past decade[17]. In the field of collaborative filtering, two traditional types of methods are widely studied: neighborhood-based approaches and model-based approaches. Based on the co-occurrence matrix, neighborhood-based[7] methods mainly focus on finding the similar items for recommendations. However, model-based approaches train a compect model to explain the data firstly, and then predict the hidden values. Recently, the low-dimensional matrix approximation methods[26] are used widely in dealing with large scale datasets. In future work, we will design some comparative studies by using different methods and it is possible to help us mining much more feature patterns.

7. CONCLUSION AND FUTURE WORK

The continuing growth of open source ecosystems creates ongoing opportunities for mining reusable knowledge. In this paper, we have explored the idea of mining large scale repositories and constructed the *Hierarchical rEpository of Software feAture* (HESA) to support software reuse. Then,

we generate the co-occurrence matrix by the features and the software projects across multiple web repositories. Finally, our approach induced the feature-pattern and circularly recommend the most relevant features to stakeholders.

In the future, we plan to improve the performance of our method and aggregate richer software resources from software repositories. For example, Stack Overflow community[7] which is a programming question and answer websites, contains abundant knowledge of software development. In addition, we will design several representative applications based on HESA, such as software resource recommender system, to support the reuse of multi-grained resources.

8. ACKNOWLEDGEMENT

This research is supported by the National High Technology Research and Development Program of China (Grant No. 2012AA011201) and the Postgraduate Innovation Fund of University of Defense Technology (Grant No.B130607).

9. REFERENCES

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *VaMoS*, pages 45–54, 2012.

[2] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *SPLC*, pages 67–76, 2008.

[3] S. Apel and C. Kastner. An overview of feature-oriented software development. pages 49–84, 2009.

[4] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *ICSE (1)*, pages 95–104, 2010.

[5] E. Bagheri, F. Ensan, and D. Gasevic. Decision support for the software product line domain engineering lifecycle. pages 335–377, 2012.

[6] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[8] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *ICSE*, pages 181–190, 2011.

[9] W. B. Frakes, R. P. DÍłaz, and C. J. Fox. Dare: Domain analysis and reuse environment. pages 125–141, 1998.

[10] T. Griffiths. Gibbs sampling in the generative model of Latent Dirichlet Allocation. Technical report, Stanford University, 2002.

[11] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

[12] A. E. Hassan. The road ahead for mining software repositories. 2008.

[13] A. E. Hassan and T. Xie. Mining software engineering data. In *ICSE (2)*, pages 503–504, 2010.

[14] K.C.Kang, S.G.Cohen, J.A.Hess, W.E.Novak, and A.S.Peterson. Feature-oriented domain analysis (foda) feasibility study. technical report. 1990.

[15] K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In *ICSR*, pages 62–77, 2002.

[16] X. Li, H. Wang, G. Yin, T. Wang, C. Yang, Y. Yu, and D. Tang. Inducing taxonomy from tags: An agglomerative hierarchical clustering framework. In *Advanced Data Mining and Applications*, volume 7713, pages 64–77. Springer Berlin Heidelberg, 2012.

[17] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[18] A. Maedche and S. Staab. Learning ontologies for the semantic web. In *SemWeb*, 2001.

[19] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[20] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher. Recommending source code for use in rapid software prototypes. In *ICSE*, pages 848–858, 2012.

[21] H. Mei, G. Huang, and T. Xie. Internetware: A software paradigm for internet computing. *Computer*, 45(6):26–31, June 2012.

[22] H. Mei, G. Huang, H. Zhao, and W. Jiao. A software architecture centric engineering approach for internetware. *Science in China Series F: Information Sciences*, 49(6):702–730, 2006.

[23] H. Mei and X. Liu. Internetware: An emerging software paradigm for internet computing. *J. Comput. Sci. Technol.*, 26(4):588–599, 2011.

[24] N. Niu and S. M. Easterbrook. On-demand cluster analysis for product line functional requirements. In *SPLC*, pages 87–96, 2008.

[25] S. Park, M. Kim, and V. Sugumaran. A scenario, goal and feature-oriented domain analysis approach for developing software product lines. pages 296–308, 2004.

[26] J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[27] J. Tang, H. fung Leung, Q. Luo, D. Chen, and J. Gong. Towards ontology learning from folksonomies. In *IJCAI*, pages 2089–2094, 2009.

[28] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *MSR*, pages 163–166, 2009.

[29] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[30] Y. Yu, H. Wang, G. Yin, X. Li, and C. Yang. Hesa: The construction and evaluation of hierarchical software feature repository. In *SEKE*, pages 624–631, 2013.

[7] http://stackoverflow.com