Research on mining the online community: a case of Open Source Software community

Luo Yan
Sydney Institute of Language and Commerce
Shanghai University
20 Chengzhong RD,Shanghai
China
luoyan@shu.edu.cn

Abstract: - The development of Open Source Software (OSS) projects is a process of collective innovation in the environment of online community. The paper addresses the challenge of efficiently mining data from OSS web repositories and building models to study OSS community features. Data collection for OSS community study is nontrivial since most OSS projects are developed by distributed developers using web tools. We design a mining process which combines web mining and database mining together to identify, extract, filter and analyze data. We address and analyze the difficulty of mining OSS community data. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms to collect and analyze online community data.

Key-Words: - Online community, Data mining, Open source software

1 Introduction

Online communities can be defined as a social relationship aggregation, facilitated by Internet-based technology, in which users communicate and build personal relationships [1]. They allow the creation of weak links among geographically dispersed individual who regularly participates in the community. Examples of online communities can be found on fields like education [2] and [3], software development [4] or consumer behavior [5].

Most OSS projects are developed in a distributed and decentralized environment. The number of developers of OSS projects ranges from several to thousands of people, who are usually not physically located in the same place. They rarely meet face to face. For example, the Linux development team consists of more than 3, 000 developers from over 90 countries [6]. The development of OSS projects relies on the online communities, which are built on relationships among members, being their final objective sharing knowledge and improving the underlying project.

OSS communities contain abundant information about projects and their developers. Although these OSS communities can be treated as attractive data sources for researchers, difficulties exist in collecting data. They may not have access to the backend databases, in which case, tools are needed to extract, download, parse, and port web data. The researchers must eliminate irrelevant or unnecessary items in the extracted data. Furthermore, overlapping data or incomplete data also exists. Some users may use multiple IDs or an ID may be used by multiple users. The researchers also need to deal with missing and dirty data. For example, dumped and defunct projects must be distinguished from normal projects [7]. It is imperative for researchers to improve data quality and accuracy.

There are two ways to collect data from OSS communities. In most studies, the collecting process is based on retrieving web pages or files (usually by a web crawler), parsing the downloaded pages, and analyzing data for different research goals. Robles et al. [8] design a data collecting system, GlueTheos, which includes modules to download raw data from CVS repositories, analyze the source code and store data as XML files or in a SQL database. Some researchers focus on gathering, sharing, and storing OSS development data for academic research [9, 10]. These systems use a crawler to collect OSS project and developer information from OSS communities and provide ready-to-use databases for researchers. Jensen et al. [11] explore mining techniques for discovering OSS development processes by analyzing text, links, usage, and update patterns from OSS communities. Crob et al. [12] discuss selecting and extracting OSS communities' server log files and use data mining techniques to analyze those log files.

Collecting data by web crawling has several disadvantages in the process of mining OSS communities. First, a web crawler can hardly be used for different OSS communities' web sites because those sites have different web structures or a site may change its structure. Second, because crawling is always time- and resource- consuming, downloading data by a web crawler may affect the performance of the downloaded site and may result in a denial of service for regular users. Third, historic data are often unavailable to web crawlers because an OSS community's web site usually only contains current information. The other way to collect data is to get a copy of a backend database dump directly from the OSS communities' web sites. Although direct access to the data dump can avoid those disadvantages of web crawling or spidering, this approach still poses several problems. The lack of documentation about data dumps may require a huge effort to understand schemas and tables. Unlike data listed directly on web pages, names and meaning of table fields may be confusing and obscured and the interpretation may not be easy. Moreover, data processing and cleaning are complicated due to the fact that there are noisier and duplicated data stored in databases.

This paper addresses the challenge of efficiently mining data from OSS communities. Most previous studies focus on manually creating a web crawler to collect data from OSS communities' web sites based on specific research needs. We design a mining process which combines web mining and database mining to identify, extract, filter and analyze data. Furthermore, OSS communities' database schemas and tables are described to help researchers understand data dump structures. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms to collect and analyze OSS communities' repository data.

The rest of this chapter is organized as follows: Section 2 describes OSS communities' data sources and data features. Section 3 presents our mining process and gives an example of using a web crawler to collect data. Section 4 focuses on design and implementation of data collection from OSS community. Conclusions and future work are given in Section 5.

2 Data Sources

We collect our data from SourceForge - the world's largest Open Source software development and collaboration community. With over 324, 000 projects and over 3.4 million registered users as of March 2013, SourceForge.net, sponsored by VA Software, offers a centralized place for OSS developers to control and manage OSS development by providing project web servers, trackers, mailing lists, discussion boards, and software releases, etc. Users can download and use those projects for free under an Open Source license. This community provides highly detailed information about projects and developers, including project characteristics, developers' activities, and "top ranked" developers. By mining this OSS community, we can explore developers' behaviors and projects' growth.

2.1 Project Information

SourceForge provides a summary for each project, including its name, registered data, classification, list of developers, and activity data. According to the features and properties of a project, SourceForge groups each project into different categories, called a software map. Those categories have multi-level subcategories. A project can be classified into different subcategories in the same top category. The top level category includes database environment, development status (planning, pre-alpha, alpha, beta, production/stable, mature, inactive), intended audience, license, operating system, programming Language, topic, translations (natural language the project is based on), and user interface. Fig.1 shows the first level subcategories of SourceForge projects topics.

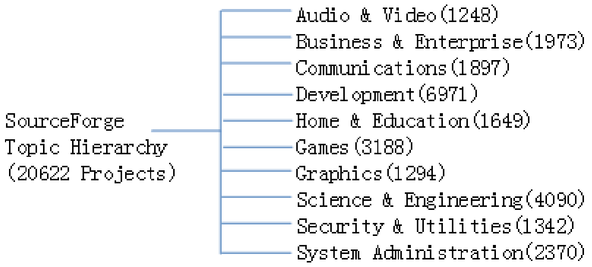


Fig.1. SourceForge topics hierarchy. This graph only shows the first level of the topics hierarchy tree.

The tracker system is a tool provided by SourceForge.net that allows developers to develop, collaborate, and support software. By using trackers, developers and users can report and manage bug reports, patch submissions, support requests, feature requests. Each project may have multiple trackers. Each tracker keeps total and open counts, summary of a request, open date, update date, current status (open, close), submitter ID, and assignee ID.

Some projects may maintain forums for different types of purposes. Forums provide a convenient communication for developers and users to raise, discuss, and answer questions. SourceForge records each message, as well as its topic starter, total replies, and follow-ups.

For each project, you can find historical data of its activities, such as its rank, total page views, total downloads, and posts on forums. The history of file releases is also available, which provides information about the file release time, version, size, and package names.

2.2 Member and User Information

A project has a list of administrators and developers. They are registered members of SourceForge. Each project has one or multiple project administrators, who are often the initiators of that project. These project administrators usually control the source code change and maintain project membership. Besides project administrators, each project may have developers who work on that project but have no control privileges. These developers must be registered users on SourceForge and granted membership by project administrators. Because both project administrators and developers are registered members, SourceForge keeps records of their user IDs, user names, real names, contact information, and projects they participate on.

There are a large number of registered users on SourceForge who do not belong to any projects. These users may or may not have activities on some projects. We can track their information by checking a project's trackers and forums.

However, SourceForge is also visited by a large number of other unregistered end-users, who may download software, report bugs, and post messages on forums. Since they are not registered members of SourceForge, we have no way to know their quantities and identifications although they may be very important for the development of a project.

2.3 Data Limitation

Although we can get abundant data from OSS community hosting web sites, the actual data collection process has been proven to be challenging. An OSS community web site can be accessed by millions of developers and users. Each person has her own natural language, input format, and other preferences. Moreover, some developers decide to move their project to an OSS community after the project is developed for some time, while some developers may decide to stop the development of their project or move it out of an OSS community and build their own project site. All these actions taken by developers or users can lead to difficulties in the data collection process. In summary, data collected from OSS communities may include the following dirty or noisy data:

1. Missing data: Data collected may be incomplete. For example, since SourceForge started in November 1999, for those projects which started earlier, we could not get their history data earlier than that date.

2. Outliers: Abnormal values may be contained in the collected data. Some projects are dumped into SourceForge. These projects should be identified because they will cause inaccuracy in research studies if we treat them the same way as other projects.

3. Anonymous data: Some data may have no identification which poses difficulty in analyzing them. For example, users are allowed to use a user ID called “nobody” to post messages on SourceForge. We have no way to know how many different users are behind those “nobody” activities. When we include this data in data analysis, the way we treat it (delete all “nobody”, treat each one as a different individual, or treat all as one individual) will lead to huge difference in results.

3 Data mining

3.1 Mining Process

In order to explore the OSS community data, we need to identify, extract, filter and analyze data resources. Our OSS community mining process combines web retrieval, statistics, and data mining techniques together to provide information for our research. The mining process, as shown in Fig 2, consists of four phases: identification and extraction, cleansing and preprocessing, analysis, and report.

The first phase is to select appropriate content from the data sources. The OSS community data sources may be web sites or data dumps. Usually, a web site contains many web pages, such as documentation, newsgroups, forums, mailing lists, etc. A data dump stores a large number of tables.

Either web pages or database tables include relevant and irrelevant information to our research purpose. This phase decides which web pages or database tables should be extracted. Different tools and techniques will be used depending on the data source. If data is extracted from a web site, a crawler/spider will be used to automatically extract specific fragments of a web document. Extracted data from a data dump requires interpretation and exploration of database tables. Database management and connection tools, i.e. SQL and JDBC, are used in our data collection from data dumps.

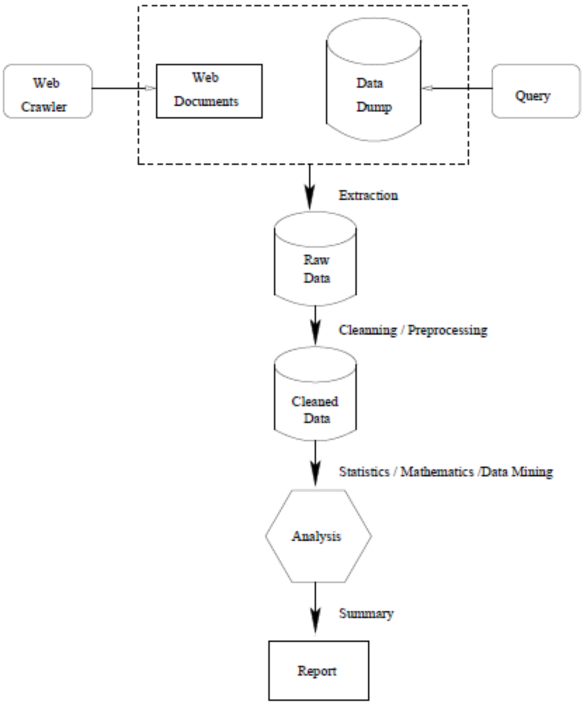


Fig.2. Mining process. Data are extracted from web pages or data dumps and stored into back-end database; raw data in the database are cleaned and preprocessed for later statistics, mathematics or data mining analysis; scientific report is based on data analysis.

The second phase converts the raw data into cleaned data abstractions necessary for analysis. The purpose of data cleansing and preprocessing is to improve data quality and increase mining accuracy. The main tasks of data cleansing consist of handling missing values, identifying outliers, filtering out noisy data and correcting inconsistent data. Data cleansing eliminates irrelevant or unnecessary items in the analyzed data. A web site can be accessed by millions of users. Different users may use different format when creating data. Furthermore, overlapping data and incomplete data also exist. By Data cleansing, errors and inconsistencies will be detected and removed to improve the quality of data. Data preprocessing converts data to a format suitable for later analysis, which may include integrating data from multiple sources into a coherent store; scaling values of data to a range; reducing a huge data set to a smaller representative subset.

In the analysis phase, advanced techniques are utilized to discover patterns. Statistics can be applied to calculate measures such as totals and means, etc. Data mining functions can be performed to find dependencies and relationships. Simulation models can be built to predict future development. This phase gives analytical data which needs further interpretation.

The report phase includes interpreting and validating the potential information from patterns offered by analysis. The objective of this task is to gain knowledge from the information provided by former tasks. According to their knowledge and experience, different researchers may get different or even opposite explanations for the same analytical data.

3.2 OSS Community Mining

In the first stage of our OSS community study, web mining is our main method used to collect data. Because web data are semi-structured or even unstructured, which cannot be manipulated by traditional database techniques, it is imperative to extract web data to port them into databases for further handling. The purpose of our OSS community mining is to extract a specific portion of web documents useful for a research objective. Our mining process takes web documents as input, identifies a core fragment, and transforms that fragment into a structured and unambiguous format [13].

3.2.1 Web Crawler Implementation

In OSS community mining, a web crawler is developed to help extract useful information from OSS community resources. A web crawler is a program which automatically traverses web sites, downloads documents and follows links to other pages [14]. It keeps a copy of all visited pages for later use. Most web search engines use web crawlers to create entries for indexing. They can also be used in other possible applications such as page validation, structural analysis and visualization, update notification, mirroring and personal web assistants/agents etc. [15].

We designed a web crawler shown in Fig.3, which is implemented using Perl because Perl provides useful modules to help web retrieval such as HTTP communication, HTML parsing, etc. A crawler can also be implemented in Java, Ruby, and Python, etc.

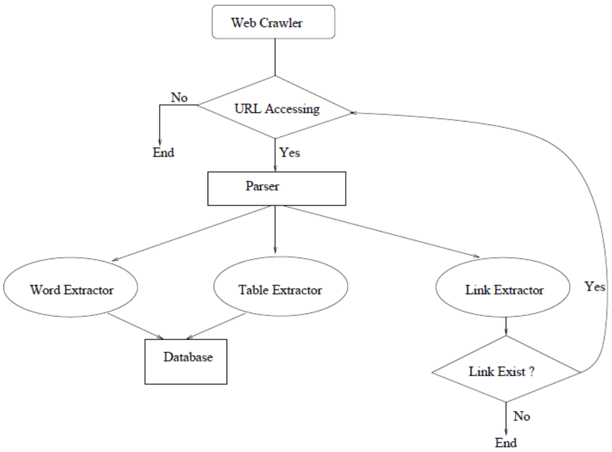


Fig.3. A web crawler. The crawler starts with a URL, identifies other links on the HTML page, visits those existing links, extracts and parses web pages. The parser includes a word extractor, a table extractor and a link extractor.

Our crawler starts with a URL, identifies other links on the HTML page, and visits those links, extracts information from web pages and stores information into databases. Thus, the crawler consists of a URL access method, a web page parser with some extractors, and a back-end database. The access function of a crawler should prevent repeatedly accessing the same web address and should identify dead links. The parser recognizes start tags, end tags, text and comments. The database provides storage for extracted web information.

The key component of our web crawler is the parser, which includes a word extractor, a table extractor and a link extractor. The word extractor is used to extract word information. It should provide a string checking function. Tables are used commonly in web pages to visually align information. A table extractor identifies the location of the data in a table. A link extractor retrieves links contained in a web page. There are two types of links- absolute links and relative links. An absolute link gives the full address of a web page, while a relative link needs to be converted to a full address by adding a prefix.

The code of web crawler as follows:

```
#!/usr/local/bin/perl
use lib qw( ..);
use lib "/afs/nd.edu/user9/jxu1/research/webmining/jin/\
lib/HTML-TableExtract-1.08/lib";
use HTML::TableExtract;
use LWP::Simple;
use Data::Dumper;
my $str = "http://sourceforge.net/project/stats/?group_id=";

for (my $s = @ARGV[0]; $s <= @ARGV[1]; $s++){
    $file = "result".$s."k.txt";
   $m = ">>".$file;
   print $m . "\n";
my $lo = $s."000";
my $hi = $s."999";
for (my $i = $lo; $i <= $hi; $i++){
    print $i. "\n";
    my $te = new HTML::TableExtract( depth=>1, count=>3,
    gridmap=>0);
    $url = $str . $i;
    #print $url . "\n";
#my $content = system ("wget". "-o " " .$url);
my $content = get($url);
my $random = int (rand 3);
my $cmd = "sleep ".$random;
print "before ".$cmd . "\n";
system($cmd);
print "end\n";
if(defined $content){
    #print $content;
}
else {
    # print "Error: Get failed";
}
#print ("content". "\n");
$te->parse($content);
#print ("parse" . "\n");
foreach my $ts ($te->table_states)
{
    foreach $row ($ts->rows)
    {
        open (outfile, $m );
        print outfile $i;
        print outfile "|";
        @a = @$row;
        $length = @a;
        if ($a[0] !~ /Lifespan/){
        for (my $j = 0; $j < @a; $j++){
        #print $a[$j];
        print outfile $a[$j];
        print outfile "|";
    }
    print outfile "\n";
}
        close (outfile);
        }
    }
}
}
```

All project home pages in SourceForge have a similar top-level design. Many of these pages are dynamically generated from a database. The web crawler uses LWP, the libwww-Perl library, to fetch each project's homepage. CPAN has a generic HTML parser to recognize start tags, end tags, text and comments, etc. Because both statistical and member information are stored in tables, the web crawler uses an existing Perl Module called HTML::Table Extract and string comparisons provided by Perl to extract information. Link extractors are used if there is more than one page of members.

3.2.2 Web Raw Data Collection

After informing SourceForge of our plans and receiving permission, we gathered data monthly at SourceForge. Data is collected at the community, project, and developer level, characterizing the entire OSS phenomenon, across multiple numbers of projects, investigating behaviors and mechanisms at work at the project and developer levels. The primary data required for this research are stored in two tables- developers and project statistics. The developers table, shown in Table 1, has 2 fields: project ID and developer ID. The project statistics table, shown in Table 2, consists of records with 9 fields: project ID, lifespan, rank, page views, downloads, bugs, support, patches and CVS. Because projects can have many developers and developers can be on many projects, neither field is a unique primary key. Thus the composite key composed of both attributes serves as a primary key. Each project in SourceForge is given a unique ID when registered with SourceForge.

Table 1. MEMBERSHIP FOR EACH PROJECT

Project ID	Login ID
1	agarza
1	alurkar
1	burley
1	chorizo
2	atmoshere
2	burley
2	bdsabian

Table 2. PROJECT STATISTICS

Project ID	lifespan	rank	Page views	downloads	bugs	support	patches	All trackers	tasks	cvs
1	1355 days	31	12,163,712	71,478	4,160	46,811	277	52,732	44	0
2	1355 days	226	4,399,238	662,961	0	53	0	62	0	14,979
3	1355 days	301	1,656,020	1,064,236	364	35	15	421	0	12,236
7	1355 days	3322	50,257	20,091	0	0	0	12	0	0
8	1355 days	2849	6,541,480	0	17	1	1	26	0	13,896

4 Data analysis

We use several data mining algorithms to analyze the web raw data we retrieved from SourceForge by the crawler. Among them, we use Naive Bayes, Classification and Regression Tree (CART) for classification, A Priori for association rules mining, K-means and Orthogonal-Cluster for clustering.

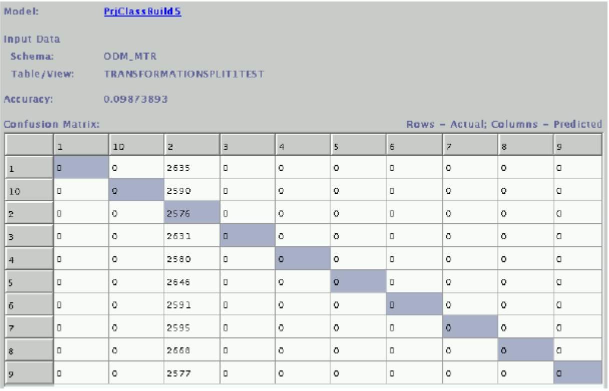
4.1 Classification

The Naive Bayes algorithm makes predictions using Bayes' Theorem. Naïve Bayes assumes that each attribute is independent of others, which is not the case in our study. For example, the “downloads” feature is closely related to the “cvs” feature; the “rank" feature is closely related to other features, since it is calculated from other features.

CART builds classification and regression trees for predicting continuous dependent variables (regression) and categorical predictor variables (classification). We are only interested in the classification type of problems since the software we are using only handles this type of problems. Oracle's implementation of CART is called Adaptive Bayes Network (ABN). ABN predicts binary and multi-class targets. Thus discretizing the target attribute is desirable.

Classification includes the following steps: build models, test models, compute lift, and apply models. The table “projects_build_binned” which stores the transformed data, contains 65,000 records. We randomly partition the table into three parts based on the 6:3:1 criteria. Each part is for building, testing and computing lift respectively. We call these parts model-build-data, model-test-data and lift-compute-data.

In our case, we try to predict “downloads” from other features. As stated previously, the “downloads” feature is binned into ten equal buckets. We predict which buckets “downloads” resides based on the values of other features. As expected, the Naive Bayes algorithm is not suitable for predicting “downloads”, since it is related to other features, such as “cvs”. Fig.3. shows that the accuracy of Naive Bayes is less than 10%.



Model: PrjClassBuild5

Input Data
Schema: ODM_MTR
Table/View: TRANSFORMATIONSPLIT1TEST

Accuracy: 0.09873893

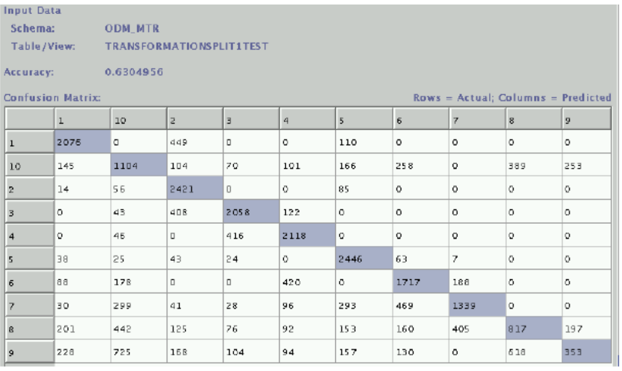
Confusion Matrix: Rows – Actual; Columns – Predicted

	1	10	2	3	4	5	6	7	8	9
1	0	0	2635	0	0	0	0	0	0	0
10	0	0	2590	0	0	0	0	0	0	0
2	0	0	2576	0	0	0	0	0	0	0
3	0	0	2631	0	0	0	0	0	0	0
4	0	0	2580	0	0	0	0	0	0	0
5	0	0	2646	0	0	0	0	0	0	0
6	0	0	2591	0	0	0	0	0	0	0
7	0	0	2595	0	0	0	0	0	0	0
8	0	0	2668	0	0	0	0	0	0	0
9	0	0	2577	0	0	0	0	0	0	0

Fig.3. Naive Bayes prediction for downloads. The “downloads” is predicted based on other project's statistics. Rows contain actual data. Columns contain predicted data. The accuracy of Naive Bayes is less than 10%.

While Naive Bayes performs badly on predicting “downloads”, the ABN algorithms can predict “downloads" quite accurately. As shown in Fig.4, the accuracy is about 63%. At first sight, the accuracy 63% is not attractive, but it is a good prediction since we could only get 10% of correct predictions without classification. The lift computation confirms that the resulting classification model is quite good, as shown in Fig.5. These figures show that all records whose “downloads" feature is 1 in just the first 20% of records. Fig.6. shows the rules built by the ABN classification model. As we see from the figure, “downloads" is closely related to “cvs".

We conclude that the ABN algorithm is suitable for predicting “downloads" in our study. The following table compares the two algorithms, namely, ABN and Naive Bayes. From the Table 3, we see that ABN takes much longer time to build a classification model, but the resulting model is much more accurate.



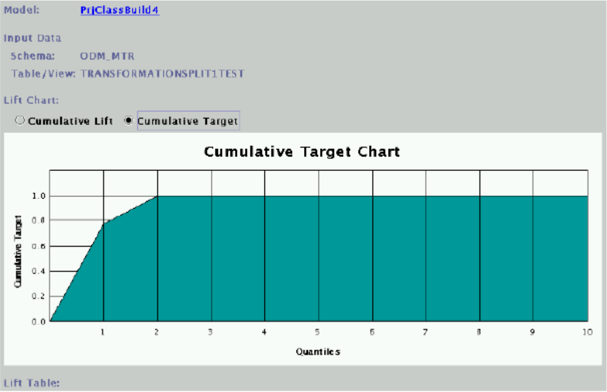
Input Data
Schema: ODM_MTR
Table/View: TRANSFORMATIONSPLIT1TEST

Accuracy: 0.6304956

Confusion Matrix: Rows = Actual; Columns = Predicted

	1	10	2	3	4	5	6	7	8	9
1	2076	0	449	0	0	110	0	0	0	0
10	145	1104	104	70	101	166	258	0	389	253
2	14	56	2421	0	0	85	0	0	0	0
3	0	43	408	2058	122	0	0	0	0	0
4	0	46	0	416	2118	0	0	0	0	0
5	38	25	43	24	0	2446	63	7	0	0
6	88	178	0	0	420	0	1717	188	0	0
7	30	299	41	28	96	293	469	1339	0	0
8	201	442	125	76	92	153	160	405	817	197
9	228	725	168	104	94	157	130	0	618	353

Fig.4. ABN prediction for downloads. The “download" is predicted based on other project's statistics. Rows contain actual data. Columns contain predicted data. The accuracy of ABN is 63%.



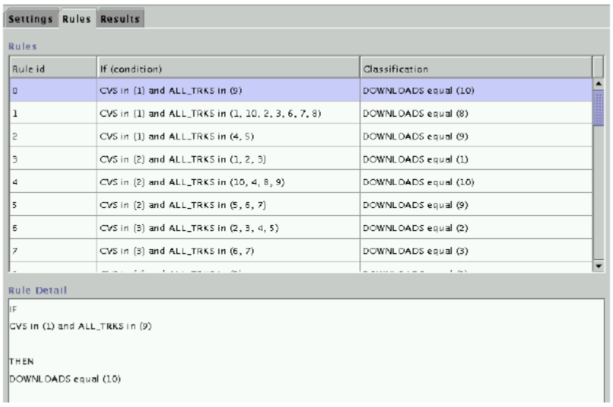
Model: PrjClassBuild4

Input Data
Schema: ODM_MTR
Table/View: TRANSFORMATIONSPLIT1TEST

Lift Chart:
○ Cumulative Lift ● Cumulative Target

Lift Table:

Fig.5. The Lift chart for ABN prediction. By using ABN algorithm, the cumulative target of “downloads" is 1 in just the first 20% of records, which indicates the prediction is accurate.



Rules

Rule id	If (condition)	Classification
0	CVS in (1) and ALL_TRKS in (9)	DOWNLOADS equal (10)
1	CVS in (1) and ALL_TRKS in (1, 10, 2, 3, 6, 7, 8)	DOWNLOADS equal (8)
2	CVS in (1) and ALL_TRKS in (4, 5)	DOWNLOADS equal (9)
3	CVS in (2) and ALL_TRKS in (1, 2, 3)	DOWNLOADS equal (1)
4	CVS in (2) and ALL_TRKS in (10, 4, 8, 9)	DOWNLOADS equal (10)
5	CVS in (2) and ALL_TRKS in (5, 6, 7)	DOWNLOADS equal (9)
6	CVS in (3) and ALL_TRKS in (2, 3, 4, 5)	DOWNLOADS equal (2)
7	CVS in (3) and ALL_TRKS in (6, 7)	DOWNLOADS equal (3)

Rule Detail

IF
CVS in (1) and ALL_TRKS in (9)

THEN
DOWNLOADS equal (10)

Fig.6. The rules built by ABN classification. The rule computation contains if-condition field and classification field. “Downloads” is closely related to “cvs”.

Table 3. COMPARISON OF ABN AND NAIVE BAYES.

Name	Build Time	accuracy
ABN	0:19:56	63%
Naive Bayes	0:0:30	9%

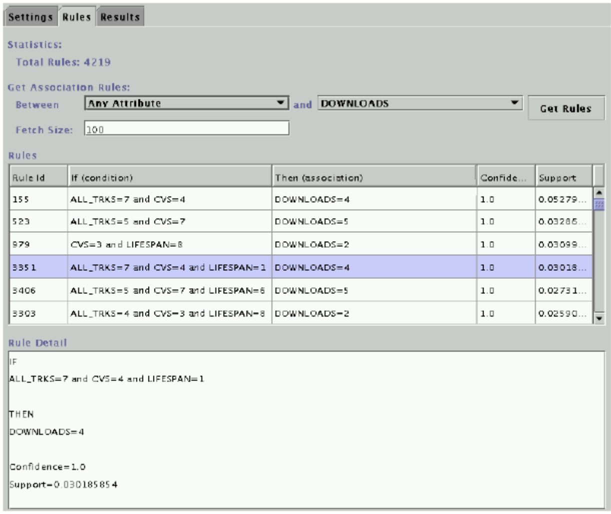
4.2 Association Rules

Association rule algorithm is applied to find the correlations between features of projects in OSS community. The association rules mining problem can be decomposed into two subproblems:

1.Find all combinations of items, called frequent itemsets, whose support is greater than the minimum support.

2.Use the frequent itemsets to generate the association rules. For example, if AB and A are frequent itemsets, then the rule A→B holds if the ratio of support(AB) to support(A) is greater than the minimum confidence.

One well known association rule mining algorithm is called A Priori. Oracle implements this algorithm using PL/SQL. We use the algorithm to find correlations between features of projects. The algorithm takes two input data- the minimum support and the minimum confidence. We choose 0.01 for minimum support and 0.5 for minimum confidence. Fig.7. shows the results of the association rules mining. From the figure, we see that the feature “all_trks", “cvs" and “downloads" are “associated". More rules can be seen from the above figure. Not all of the rules discovered are of interest.



Statistics:
Total Rules: 4219

Get Association Rules:
Between: Any Attribute and DOWNLOADS
Fetch Size: 100

Rules

Rule Id	If (condition)	Then (association)	Confide...	Support
155	ALL_TRKS=7 and CVS=4	DOWNLOADS=4	1.0	0.05279...
523	ALL_TRKS=5 and CVS=7	DOWNLOADS=5	1.0	0.03286...
979	CVS=3 and LIFESPAN=8	DOWNLOADS=2	1.0	0.03099...
3351	ALL_TRKS=7 and CVS=4 and LIFESPAN=1	DOWNLOADS=4	1.0	0.03018...
3406	ALL_TRKS=5 and CVS=7 and LIFESPAN=6	DOWNLOADS=5	1.0	0.02731...
3303	ALL_TRKS=4 and CVS=3 and LIFESPAN=8	DOWNLOADS=2	1.0	0.02590...

Rule Detail

IF
ALL_TRKS=7 and CVS=4 and LIFESPAN=1

THEN
DOWNLOADS=4

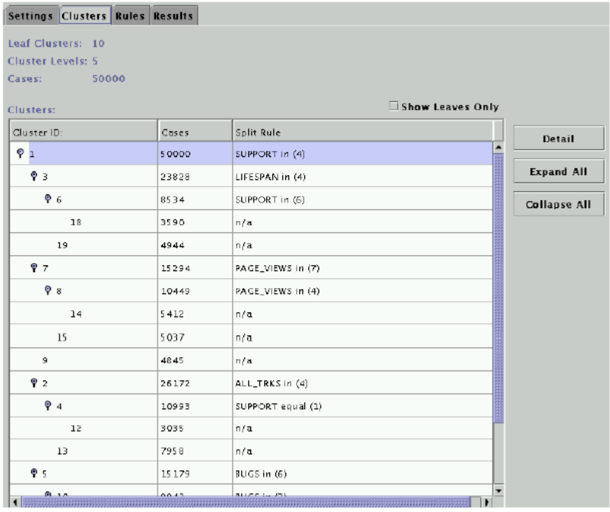
Confidence=1.0
Support=0.030185854

Fig.7. The rules built by A Priori. Features “all_trks”, “cvs” and “downloads” are “associated” because the confidence and support are greater than their thresholds.

4.3 Clustering

Clustering is a data analysis which is used to find a collection of similar patterns [16]. We wish to put projects with similar features together to form clusters. Two algorithms can be used to accomplish this: k-means and o-cluster.

The k-means algorithm is a distance-based clustering algorithm, which partitions the data into predefined number of clusters. The o-cluster algorithm is a hierarchical and grid-based algorithm. The resulting clusters define dense areas in the attribute space. The dimension of the attribute space is the number of attributes involved in the clustering algorithm. We apply the two clustering algorithms to projects in this case study. Fig.9 and Fig.10 show the resulting clusters and rules that define the clusters.



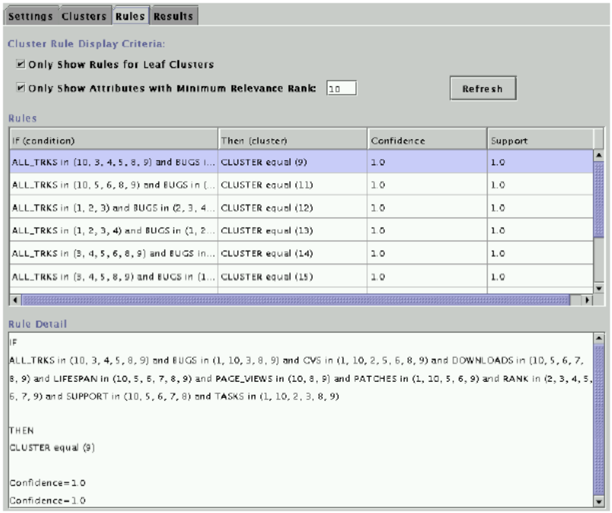
Leaf Clusters: 10
Cluster Levels: 5
Cases: 50000

Clusters:

☐ Show Leaves Only

Cluster ID:	Cases	Split Rule
1	50000	SUPPORT in (4)
3	23828	LIFESPAN in (4)
6	8534	SUPPORT in (6)
18	3590	n/a
19	4944	n/a
7	15294	PAGE_VIEWS in (7)
8	10449	PAGE_VIEWS in (4)
14	5412	n/a
15	5037	n/a
9	4845	n/a
2	26172	ALL_TRKS in (4)
4	10993	SUPPORT equal (1)
12	3035	n/a
13	7958	n/a
5	15179	BUGS in (6)

Fig.8. The clusters. There are total of 50, 000 projects. They are divided into 5 groups, each of which contains subgroups. For example, cluster 6, which contains 8534 projects, can be divided into group 18 and group 19.



Cluster Rule Display Criteria:

☑ Only Show Rules for Leaf Clusters

☑ Only Show Attributes with Minimum Relevance Rank: 10

Rules

IF (condition)	Then (cluster)	Confidence	Support
ALL_TRKS in (10, 3, 4, 5, 8, 9) and BUGS i...	CLUSTER equal (9)	1.0	1.0
ALL_TRKS in (10, 5, 6, 8, 9) and BUGS in (...	CLUSTER equal (11)	1.0	1.0
ALL_TRKS in (1, 2, 3) and BUGS in (2, 3, 4...	CLUSTER equal (12)	1.0	1.0
ALL_TRKS in (1, 2, 3, 4) and BUGS in (1, 2...	CLUSTER equal (13)	1.0	1.0
ALL_TRKS in (3, 4, 5, 6, 8, 9) and BUGS in...	CLUSTER equal (14)	1.0	1.0
ALL_TRKS in (3, 4, 5, 8, 9) and BUGS in (1...	CLUSTER equal (15)	1.0	1.0

Rule Detail

```
IF
ALL_TRKS in (10, 3, 4, 5, 8, 9) and BUGS in (1, 10, 3, 8, 9) and CVS in (1, 10, 2, 5, 6, 8, 9) and DOWNLOADS in (10, 5, 6, 7,
8, 9) and LIFESPAN in (10, 5, 6, 7, 8, 9) and PAGE_VIEWS in (10, 8, 9) and PATCHES in (1, 10, 5, 6, 9) and RANK in (2, 3, 4, 5,
6, 7, 9) and SUPPORT in (10, 5, 6, 7, 8) and TASKS in (1, 10, 2, 3, 8, 9)

THEN
CLUSTER equal (9)

Confidence=1.0
Confidence=1.0
```

Fig.9. The rules that define clusters. Statistical features of a project are divided into 10 categories. Clusters are formed according to categories in each statistical feature.

5 Conclusion

The success of Open Source Software (OSS) has attracted increased interest in many research areas. Unlike proprietary closed software, OSS projects are developed in a distributed and decentralized way. The OSS community is largely composed of part-time developers. These developers have developed a substantial number of outstanding technical achievements.

In this paper, we address the challenge of efficiently mining data from OSS web repositories and building models to study OSS community features. Data collection for OSS community study is nontrivial since most OSS projects are developed by distributed developers using web tools. Most previous studies focus on manually creating a web crawler to collect data from OSS community. This method is usually implemented by creating a web crawler based on specific research goals. We design a mining process which combines web mining and database mining together to identify, extract, filter and analyze data. We address and analyze the difficulty of mining OSS community data. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms to collect and analyze online community data.

Acknowledgement

This research work is supported by Innovation Program of Shanghai Municipal Education Commission (serial No. 12YS018) and Humanity and Social Science Youth Foundation of Ministry of Education, China (No. 12YJC630136)

References:

[1] H. Rheingold. The Virtual Community: Homesteading on the Electronic Frontier. Addison-Wesley, Reading, MA (1993)

[2] F. Barrero, S.L. Toral, S. Gallardo. EDSPLab: remote laboratory for experiments on DSP applications. Internet Research, 18 (1) (2008), pp. 79–92.

[3] S.L. Toral, F. Barrero, M.R. Martínez-Torres, S. Gallardo, J. Lillo. Implementation of a web-based educational tool for digital signal processing teaching using the technological acceptance model. IEEE Transactions on Education, 48 (4) (2005), pp. 632–641.

[4] F. Barcellini, F. Détienne, J.-M. Burkhardt, W. Sack. A socio-cognitive analysis of online design discussions in an open source software community. Interacting with Computers, 20 (1) (2008), pp. 141–165.

[5] R.-A. Shang, Y.-C. Chen, H.-J. Liao. The value of participation in virtual consumer communities on brand loyalty. Internet Research, 16 (4) (2006), pp. 398–418.

[6] J. Y. Moon and L. Sproull. Essence of distributed work: The case of the linux kernel. First Monday, v. 5(11), 2000.

[7] J. Howison and K. Crowston. The perils and pitfalls of mining sourceforge. In Proceedings of Mining Software Repositories Workshop, Internat ional Conference on Software Enginnering (ICSE 2004), Edinburgh, Scotland, 2004.

[8] G. Robles and R. Gonzalez-Barahona, J.and Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In Proceedings of the Mining Software Repositories Workshop. 26th International Conference on Software Engineering, Edinburgh, Scotland, 2004.

[9] J. Howison, M. S. Conklin, and K. Crowston. Ossmole: A collaborative repository for oss research data and analysiss. In Proceedings of 1st International Conference on Open Source Software, Genova, Italy, 2005.

[10] D. Weiss. A large crawl and quantitative analysis of open source projects hosted on sourceforge. Research report ra-001/05, Institute of Computing Science, Poznan university of Technology, Poland, 2005.

[11] C. Jensen and W. Scacchi. Data mining for software process discovery in open source

software development communities. In Proc. Workshop on Mining Software Repositories, Edinburgh, Scotland, 2004.

[12] H. L. Grob, F. Bensberg, and F. Kaderali. Controlling open source intermediaries- a web log mining approach. In Proceedings of the 26th Int. Conf. Information Technology Interfaces ITI 2004, Sagreb, Kroatien, 2004.

[13] L. Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[14] M. Koster. The web robots pages. http://info.webcrawler.com/mak/projects/robots/robots.html, 1999.

[15] F. Crimmins. Web crawler review. http://dev.funnelback.com/crawlerreview.html, 2001.

[16] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. ACM Computing Surveys, 31(3):264-323, 1999.