Quality Classifiers for Open Source Software Repositories

George Tsatsaronis, Maria Halkidi, and Emmanouel A. Giakoumakis

Abstract Open Source Software (OSS) often relies on large repositories, like SourceForge, for initial incubation. The OSS repositories offer a large variety of meta-data providing interesting information about projects and their success. In this paper we propose a data mining approach for training classifiers on the OSS meta-data provided by such data repositories. The classifiers learn to predict the successful continuation of an OSS project. The 'successfulness' of projects is defined in terms of the classifier confidence with which it predicts that they could be ported in popular OSS projects (such as FreeBSD, Gentoo Portage).

1 Introduction

Initial open source software (OSS) projects rely on large repositories for hosting and distribution until they become independent. A huge amount of project meta-data is collected and maintained in such software repositories providing useful information about projects and their success. In this paper we propose a data mining approach that processes the meta-data contained in such OSS repositories. The proposed approach aims at the construction of a classifier that is trained on the meta-data of existing projects and predicts the successful continuation of any given OSS. The *successfulness* of a project is defined with regard to the confidence level of the classifier which predicts that this project will be ported in widely used OSS projects (e.g. FreeBSD). We argue that the classifier decision, along with its confidence level, can be incorporated into known models of software success, like the model of DeLone

George Tsatsaronis
Department of Informatics, Athens University of Economics and Business, 76, Patission Str., Athens, e-mail: gbt@aueb.gr

Maria Halkidi
Department of Technology Education and Digital Systems, University of Piraeus, e-mail: mhalk@unipi.gr

Emmanouel A. Giakoumakis
Department of Informatics, Athens University of Economics and Business, 76, Patission Str., Athens, e-mail: mgia@aueb.gr

and McLean [4], or its reexamination and expansion for OSS software, by Crowston et al. [3]. We have experimentally evaluated the proposed mining approach in the SourceForge and the FreshMeat OSS project meta-data released from the Floss project. We also evaluated the importance of the underlying features using Information Gain and Chi-Square. The results of this study report high F-Measures for the classifiers based on the most important FLOSS features.

The proposed approach consists of two main steps: a) data collection and preprocessing, b) training classifiers. The meta-data in OSS repositories are usually provided in formats that are not suitable for mining. Thus, one of the most important elements in the proposed approach is the *pre-processing procedure*. Techniques such as *parsing, crawling* and *feature selection* are used to collect data from the FLOSS project, which contains crawled projects from important OSS repositories, like SourceForge and FreshMeat. We also discuss methods that can be used to train classifiers on the stored project data.

The rest of the paper is organized as follows. Section 2 discusses related work in mining OSS projects and related models for measuring information systems' success. Section 3 presents the data mining approach for extracting knowledge from OSS repositories. Section 4 provides experimental results and analysis. Section 5 concludes and gives further insight to possible future work.

2 Related Work

Data Mining in Software Engineering. Data mining is widely used for supporting industrial scale software maintenance, debugging and testing. An approach that exploits classification methods to analyze logical bugs is proposed in [7]. This work treats program executions as software behavior graphs and develops a method to integrate closed graph mining and SVM classification in order to isolate suspicious regions of non-crashing bugs. A semi-automated strategy for classifying software failures is presented in [9]. This approach is based on the idea that if m failures are observed over some period during which the software is executed, it is likely that these failures are due to a substantially smaller number of distinct defects. A predictive model for software maintenance using data and text mining techniques is proposed in [10]. To construct the model, they use data collected from more than 100.000 open source software projects lying in the SourceForge portal. Using SAS Enterprise Miner and SAS Text Miner, they focused on collecting values for variables concerning maintenance costs and effort from OSS projects, like Mean Time to Recover (MTTR) an error. They clustered the remaining projects based on their descriptions, in order to discover the most important categories of OSS projects lying in the SourceForge database. Finally, they used the SAS Enterprise Miner to train classifiers on the MTTR class variable. The reported results highlight interesting correlations between the class variable and the number of downloads, the use of mail messages and the project age. There is also a number of other works [12, 1, 5, 9, 7] that use classification methods in software engineering, assisting

with its main tasks (development, debugging, maintenance, testing).

Models of OSS Success. The most popular model for measuring information systems' (IS) success is the one proposed by DeLone and McLean [4]. They actually introduce six interrelated factors of success: 1) system quality, 2) information quality, 3) use, 4) user satisfaction, 5) individual impact, and 6) organizational impact. Based on this approach, Seddon [11] reexamined the factors that can measure success and concluded that the related factors are system quality, information quality, perceived usefulness, user satisfaction and IS use. Based on those approaches, a number of measures that can be used to assess the success in FLOSS is presented by Crowston et al. in [3]. These measures are defined based on the results of a statistical analysis applied to a subject of project data in FLOSS. Specifically the empirical study was based on a subset of SourceForge projects. In this paper we propose another such measure, that can be added to the *use* and the *user satisfaction* factors of the proposed models of success.

3 Software Success Classification Approach

We introduce a classification approach that adopts data mining techniques in order to extract useful information from OSS repositories and further analyze it to predict softwares' successful continuation. Based on that, our approach aims at constructing a metric that assesses software success and which can be considered as an implementation of the factors *use* and *user satisfaction* to the model of DeLone and McLean [4] for measuring IS success. An expansion of this model has been proposed by Crowston et al. [3] for OSS software, according to which the data lying in FLOSS from SourceFourge are mapped to potential measures of OSS success. The following metric can be incorporated to the *System use* process phase, as this is described in [3]. We have developed the proposed metric taking into account the FLOSS data for both SourceForge and FreshMeat repositories.

3.1 OSS Porting Classification Metric

Without loss of generality we consider that OSS software ported to widely used open source operating systems is widely accepted as useful and significant by the majority of users. Then we claim that a project is considered to be *successful*, from the point of view of popularity and user satisfaction, if it is selected to be ported in two of the most popular open source operating systems, namely FreeBSD and Gentoo. Based on this, we construct classifiers, and we use the confidence of their classification output as a metric of OSS successfulness. The main modules of our approach can be summarized as follows: (a) *Data collection and pre-processing*. The OSS repositories maintain huge amount of OSS meta-data that provide useful information about the hosted projects. This module refers to methods used to collect data from OSS repositories and properly pre-process them for data mining.(b) *Software classification*. This module provides the methods to train classifiers for

predicting projects' course over time, trained on the collected OSS meta-data. The classifiers are built to predict the successful continuation of an OSS based on a specific set of project features. Based on the trained classifiers, new project releases (or unclassified projects) can be classified with regard to their successfulness (as this has been defined above).

Data Collection and Preprocessing Techniques

A large amount of data are collected and maintained in OSS repositories. These data contain useful information about projects that we aim to analyze so as to extract interesting knowledge and make inferences about future course of projects. However the data are provided in various formats that in most of the cases are not suitable for data mining. Thus, a pre-processing procedure is needed before data mining is applied to the available data. Specifically, techniques such as *crawling* and *feature selection* are used to collect project data from various OSS portals and select those that contain interesting information for further analysis. *Crawling* usually refers to the process of browsing the World Wide Web in a methodological and automated manner. An extensive analysis of a crawling mechanism is provided by Chakrabarti in Chapter 2 of [2]. We used a smaller crawling mechanism to browse the Web pages of the two open source operating systems (FreeBSD, Gentoo) and collect further information about the projects existing in SourceForge and FreshMeat for later processing. Many of the programming techniques used are described in [8]. For the processing of the SourceForge and the FreshMeat data, we used the FLOSS data and index. *Feature selection* is the technique of selecting a subset of relevant features for creating robust learning models. In our approach, feature selection techniques assist in a two-fold manner. Firstly, they assist the mining procedure with further analyzing the project data crawled. Thus, they provide an image of all the features being used. Secondly they assist in selecting the features that are more useful with regards to the final classification, which in our case is to predict whether an OSS software is successful enough to be ported in FreeBSD and Gentoo Portage. As feature selection criteria, we have adopted Information Gain (IG) and Chi-Squared (x^2).

According to the IG criterion, the expected reduction of uncertainty in guessing the class variable, once the feature value is known, needs to be measured. This is expressed through measuring the expected reduction in entropy, once the feature value is known, given by equation 3. While for the case of discrete value features equations 1-2 apply, in the case of the continuous value features, the domain of each feature variable X is divided into many subintervals of a given equal length and each X_i is true iff X belongs to the corresponding interval. Let S be the set of s data objects. Considering a set of m classes C_i, the expected information needed to classify a given object is defined as follows:

$$I(s_1,\ldots,s_m) = -\sum_{i=1}^{m} p_i log_2(p_i) \quad (1)$$

where s_i is the number of data objects in S labeled c_i and p_i is the probability that an object belongs to class c_i, $p_i = s_i/s$. Let attribute A have v distinct values $\{\alpha_1, \alpha_2, \ldots, \alpha_v\}$. The attribute A can be used to partition S in v subsets $\{S_i\}_{i=1}^{v}$,

where S_i contains those objects in S that have value α_i for A. The entropy or expected information based on the partitioning of S into subsets by A is given by:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + \ldots + s_{mj}}{s} - \sum_{i=1}^{m} p_{ij} log_2(p_{ij}), \ldots, s_{mj}) \quad (2)$$

where s_{ij} is the number of data objects of class C_i in a subset S_j, and $p_{ij} = \frac{s_{ij}}{|S_j|}$ is the probability that an object in S_j belongs to class C_i. The encoding information that would be gained by branching on A is:

$$Gain(A) = I(s_{1j}, \ldots, s_{mj}) - E(A) \quad (3)$$

Chi-squared (x^2) is often used to measure the association between two features in a contingency table. In the case of a binary classification problem it can be used to measure the association between an input feature and the class variable, as shown in the following equation:

$$x^2 = \sum_{i=1}^{r} \left(\frac{(n_{iP} - \mu_{iP})^2}{\mu_{iP}} + \frac{(n_{iN} - \mu_{iN})^2}{\mu_{iN}}\right) \quad (4)$$

where r are all the possible values of the examined feature, N and P are the two classes (Negative and Positive respectively), n_{iP} and n_{iN} are the number of instances belonging to class P or N respectively and have the value i of the examined feature, and $\mu_{ij} = \frac{n_{*j} n_{i*}}{n}$ with n being the number of instances, $i = 1, \ldots, r$ and $j = P, N$. In this work we use both IG and chi-square to measure the significance of the stored features contained in the OSS projects' meta-data, as crawled by the FLOSS project.

Software Classification

In order to train classifiers that can predict the degree of successfulness of an OSS lying in FreshMeat or SourceForge, the most important factor is to define the projects that belong to the classes of *successful* and *unsuccessful* projects. This can be the training set of projects that can be used for training the classifiers. The approach we adopt is to define as *successful* the OSS projects that have been ported in both FreeBSD and Gentoo Portage. This heuristic criterion satisfies part of the *System use* process phase of the model of successfulness defined in [3], namely *Interest*, *Number of Users* and *User Satisfaction*. We concluded to that criterion after having crawled the FreeBSD, Gentoo Portage and the Debian Popularity Contest, aiming to find their common set of projects with the considered FLOSS projects. FreeBSD Ports[1] is a package management system for the FreeBSD operating system and it contains all projects that are ported to FreeBSD. Portage[2] is also a package management system but it contains projects ported to Gentoo Linux. Furthermore, Debian Popularity Contest[3] is a project which attempts to map the usage of Debian packages. For the Gentoo Portage, we have used the latest snapshot[4] in order to determine the projects that are ported into it. From the Debian Popularity Contest we have collected for each package of Debian the number of users who installed

[1] `http://www.freebsd.org/`

[2] `http://www.gentoo.org/`

[3] `http://popcon.debian.org/`

[4] `http://gd.tuwien.ac.at/opsys/linux/gentoo/snapshots/`

it, the number of users who use it frequently, the number of users who do not use it frequently, the number of users who upgraded to the latest version of the software and the number of users that did not publish enough information. After having carefully analyzed all these data and examined several criteria that can be used as determining *successful* and *unsuccessful* projects, we concluded that the aforementioned criterion (porting of an OSS project in both FreeBSD and Gentoo Portage) expressed better the *System Use* as discussed earlier, and also provided better experimental results that other criteria examined.

Having determined the *successfulness* criterion, it is straightforward to construct classifiers taking into account the features offered by SourceForge and FreshMeat (an analysis of the features follows in the next section). Classification is a two step process:

1. *Training step.* A classification model is built describing a predefined set of classes (i.e *Successful*, *UnSuccessful*). The model is trained by analyzing a set of data whose classification (class labels) is known (training data set).
2. *Classification step*. First the predictive accuracy of the trained model (classifier) is estimated. If it is acceptable the classifier is used to classify project instances for which the class label is unknown.

Since each repository (FreshMeat, SourceForge) maintains different attributes for the hosted projects, a classifier per repository must be developed. Based on our approach any of the widely used classification algorithms can be used to analyze the training set of projects and construct the software classification model. In our current work, we have adopted *Naive Bayesian, Decision trees* and *Support Vector Machines* classification methods to train three different classifiers based on the available set of project data. For each of these classifiers, the successfulness metric is their confidence level of the classified instance. For the SVM, it is the distance of the considered project feature vector, from the hyperplane separating negative from positive instances. For the NB (Naive Bayesian) classifier, it is the probability that the considered project feature vector belongs to the *successful* class. Finally, for the decision trees (i.e. the C4.5 classifier) it is the frequency of occurrence of the training examples that are in the *successful* class, following the branch of the tree with attribute values of the examined instance. The experimental study in section 4 shows the performance of all three used classifiers for both FreshMeat and SourceForge.

3.2 Features Description

In this work we have used project data that have been collected and are available from the FLOSSMole project [5]. Specifically we work with the releases of SourceForge and FreshMeat project data indexed by the FLOSS repository. Below we present the main attributes (features) of the project data in the FLOSS repository that are used for the classification process. Also we indicate which portal (FreshMeat - FM or SourceForge - SF) maintains each attribute for the projects it hosts.

[5] `http://flossmole.sourceforge.net/`

1. *Project License (FM and SF):* Project's license type (such as GPL, LGP, BSD License, Freeware, Shareware etc)
2. *Vitality Score* (FM) which is defined as $VitalityScore = \frac{V*T_0}{T_n}$
   where V is the number of the project's versions, T_0 is the time elapsed since first project upload (usually counted in days), and T_n is the time elapsed since latest version upload.
3. *Popularity Score (FM):* Represents project's popularity based on:
   - Users' visits in project's URL, i.e. URL hits(further referred to as a). They refer to the visits in project's original web site, and not at FreshMeat's site for the project.
   - Users' visits in FreshMeat's project site (b).
   - Users' subscriptions (c).

   Then, the popularity score is measured as: $PopularityScore = \sqrt{(a+b)\cdot(c+1)}$
4. *Rating (FM):* Any subscribed user can rate a project. Given 20 or more user ratings, the project engages a ratings based ranking. Rating is measured as a means of a weighted ranking (WR), which is computed as follows:
   $$WR = (v(v+m))R + (m(v+m))\cdot C \tag{5}$$
   where: R = average (mean) rating for the project from users = (Rating)
   v = number of votes for the project = (votes)
   m = minimum votes required (currently 20)
   C = the mean vote across the whole report
5. *Subscriptions (FM):* Number of subscribed users in a project.
6. *Developers* (FM and SF): Number of developers per project and other information about them.
7. *Target audience / End Users* (SF): The target group of the resulting software.
8. *Executing operating system* (SF): The operating system in which the resulting project software can execute.
9. *DBMS Environment/Technology* (SF): For the projects using a DBMS, this is the used DBMS environment, or technology in general.
10. *Programming Language* (SF): The programming language in which the project software is written.
11. *Number of downloads* (SF).
12. *Interface* (SF): The interface of the project (Library, Command Line, Web, GUI etc).
13. *Natural language* (SF): The natural language of the project (English, France, etc).
14. *Topic* (SF): The topic of the project (Databases, Network Administration).
15. *Registration Date* (FM and SF): The date that the project was inserted into the portal.
16. *Days since Registration Date* (FM and SF): Days elapsed since the registration date.

17. *Project status* (SF): The status of the project (such as Beta, Planning, Production/Stable, Alpha, Pre-Alpha etc)
18. *Number of project donators* (SF).
19. *Rank* (SF): A ranking of the projects produced by SourceForge, considering downloads and days since registration date.

4 Experimental Evaluation

Experimental Setup. The projects that we considered for our experiments numbered 112915 for SourceForge and 41908 for FreshMeat. For both portals, we found the projects which are available from FreeBSD Ports and Gentoo Portage in order to label them as *successful*. For our experiments, we used 10-fold cross validation on three classifiers (decision trees, Naive Bayes, and SVM). For learning decision trees, we used the J48 algorithm that is WEKA's [13] implementation of the C4.5 (an extension of the basic ID3 algorithm) decision tree learner. We also used WEKA for the Naive Bayesian classifier. For support vector machines (SVMs) we used Joachim's SVM^{light} [6]. For each of the two data sets (FreshMeat and SourceForge), we trained all three classifiers on all of the features described. For the features evaluation IG and chi-square was used. For the evaluation of the classifiers, we measured precision, recall and F-Measure. The goal of this experiment is to prove the value of the OSS portals' meta-data, and consequently the value of the proposed success metric that is based on training classifiers, as an additional factor of OSS success.

Features Analysis. Feature selection requires analysis of all considered features. We have computed IG and chi-square values of all features using 10-fold cross validation, based on the used criterion. The measurements for the IG and the chi-square criteria are shown in Table 1 for the FreshMeat and the SourceForge repositories. The results are shown in decreasing order of importance based on the IG measure. From the results obtained we note primarily that the produced feature rankings based

	FreshMeat	
	Information Gain	Chi-Square
Popularity	**0.324**	**1793.254**
Subscriptions	**0.319**	**1756.487**
Vitality	**0.238**	**1781.661**
#Rating	**0.219**	**1253.699**
Rating	**0.189**	**111.144**
Days	**0.107**	**620.316**
Developers	**0.05**	**269.701**
License	**0.033**	**187.66**

	SourceForge	
	Information Gain	Chi-Square
Downloads	**0.199**	**759.95**
Rank	**0.153**	**595.337**
OS	**0.145**	**558.826**
Language	**0.138**	**533.17**
Days	**0.119**	**469.172**
Status	**0.095**	**381.716**
Interface	**0.078**	**317.054**
Developers	**0.075**	**298.099**
Users	**0.072**	**276.279**
License	**0.03**	**112.715**
DBMS	**0.01**	**5.548**
Donors	**0**	**0**

Table 1 Information Gain and Chi-Square for the FreshMeat and SourceForge Features.

on the two measures (IG and chi-square) are exactly the same in the case of SourceForge, while in the case of FreshMeat the only discrepancy is that the third feature according to IG is ranked second according to chi-square. This shows that the selected criterion of successfulness (porting of an OSS to FreeBSD and Portage) produces a stable ranking of features for both IG and chi-square. Regarding the features' importance, in the case of the FreshMeat data, the top 5 features proved to be *popularity*, *subscriptions*, *vitality*, *number of ratings* and *ratings*, while in the case of the SourceForge data, these are *downloads*, *rank*, *OS*, *language* and *days*. The IG drops dramatically for the rest features. At this point we must note that the top ranked features according to both measures include the features we were expecting to rank high, based on previously proposed models [3]. The feature ranking can be used to decrease the classifiers' model size. In the next section we show that the learned models can be reduced to only considering the top 5 features for each repository, without important decrease in performance.

Classifiers Evaluation. In order to measure the classifier's performance without introducing subset selection or feature selection bias, we have used 10-fold cross validation. The results from the 10 folds are averaged to produce a single estimation. We use *F-measure* to estimate the quality of each classifier. *F-measure* is defined as the harmonic mean between a classifier's precision and recall. All three measures were computed as follows:

$$Recall = \frac{TruePos}{TruePos + FalseN}, Precision = \frac{TruePos}{TruePos + FalsePos} \tag{6}$$

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{7}$$

where *TruePos* is the number of the actual *successful* projects classified as *successful*, *FalseN* is the number of the actual *successful* projects classified as *unsuccessful* and *FalsePos* is the number of the actual *unsuccessful* projects classified as *successful*. Table 2 shows Precision, Recall and F-Measure values for the 10-fold cross validation execution of the J.48, Naive Bayes and SVM classifiers in the FreshMeat and the SourceForge data sets respectively. SVMs managed overall the top F-Measure compared to J.48 and the NB classifiers. For the FreshMeat data set, the classifiers reached an F-Measure of around 75%, with a precision reaching 88% for the SVMs. For the SourceForge data set, the classifiers reached an F-Measure of the same level with an overall smaller precision from the FreshMeat data set. In general, the classifiers performance for the SourceForge data set is smaller than in

	FreshMeat			SourceForge		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
SVM All Features	**0.88**	**0.57**	**0.69**	**0.67**	**0.75**	**0.7**
SVM Top-5 Features	**0.62**	**0.96**	**0.75**	**0.66**	**0.73**	**0.69**
C4.5 All Features	**0.79**	**0.81**	**0.79**	**0.77**	**0.71**	**0.73**
C4.5 Top-5 Features	**0.77**	**0.78**	**0.77**	**0.75**	**0.7**	**0.72**
NB All Features	**0.76**	**0.83**	**0.79**	**0.81**	**0.78**	**0.79**
NB Top-5 Features	**0.74**	**0.84**	**0.78**	**0.79**	**0.76**	**0.77**

Table 2 Precision (P), Recall (R) and FMeasure (F1) for SVM, C4.5 and NB in the FreshMeat and SourceForge data sets.

FreshMeat, depicting that FreshMeat's features are more descriptive for the used criterion of porting. This is also verified from the IG and chi-square feature values in Table 1. Overall, the proposed metric can predict whether a project will be ported into FreeBSD and Gentoo Portage, with high F-Measure.

5 Conclusions and Future Work

In this paper we propose a new Open Source Software (OSS) successfulness metric, that is based on the development of classifiers which predict the porting of an OSS into the FreeBSD and Gentoo Portage open source operating systems. We have evaluated the proposed metric by measuring the performance of Support Vector Machines (SVM), Decision Trees (C4.5) and Naive Bayes classifiers constructed on the features contained for all projects in FreshMeat and SourceForge. We also conducted an analysis of the features' importance and we experimentally show that the classifiers obtain similar performance if the top-5 features are kept, instead of all, which also include the most important features according to previous related work. As a future work we aim at combining heuristic criteria and/or manually annotated projects to enrich the training procedure with instances of better quality. We also aim at stacking classifiers for the purpose of boosting the classifier's performance.

Acknowledgments. We wish to thank D. Drosos for his assistance with the experimental study. This work was supported by EU Commision, FP6, contract No IST-5-033331 (SQO-OSS).

References

1. Bowring, J., Rehg, J., M.J., H.: Acive learning for automatic classification of software behavior. ISSTA (2004)
2. Chakrabarti, S.: Mining the Web. Morgan Kaufmann (2003)
3. Crowston, W., Hoison, J., Annabi, H.: Information systems success in free and open source software development: Theory and measures. Software Process Improvement and Practice, Vol. 11, pp. 123–148 (2006)
4. DeLone, W., McLean, E.: Information systems success: The quest for the dependent variable. Information Systems Research, Vol. 3(1), pp. 60–95 (2006)
5. Francis, P., Leon, D., Minch, M., Podguraki, A.: Tree-based method for classifying software failures. In: International Symposium on Software Reliability Engineering (2004)
6. Joachims, T.: Making large-scale SVM learning practical. Advances in Kernel methods - support vector learning. B. Scholkopf, C. Burges and A. Smola (ed.), MIT-Press. (1999)
7. Liu, C., Yan, X., Yu, H., Han, J., Yu, P.: Identifying reasons for software changes using historic databases. In: SDM (2006)
8. Loton, T.: Web Content Mining with Java. Wiley (2002)
9. Podgurski, A., Masri, W., McCleese, Y., Minch, M., Sun, J., Wang, B., Masri, W.: Automated support for classifying software failure reports. In: ICSE (2003)
10. Raza, U., Tretter, M.J.: Predicting software outcomes using data mining and text mining. In: SAS Global Forum (2007)
11. Seddon, P.: A respecification and extension of the delone and mclean model of is success. Information Systems Research, Vol. 8(3), pp. 240–253 (1997)
12. Williams, C., Hollingsworth, J.: Automating mining of source code repositories to improve bug finding techniques. IEEE Transactions on Software Engineering 31(6):466–480. (2005)
13. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann (2005)