DATA COLLECTION AND ANALYSIS OF GitHub REPOSITORIES AND USERS

Fragkiskos Chatziasimidis
Gnomon Informatics SA
21 Antonis Tritsis Str.
57001 Thessaloniki, Greece
+30-2310804150
f.chatziasimidis@gmail.com

Ioannis Stamelos
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki,Greece
+30-2310991910
stamelos@csd.auth.gr

Abstract
In this paper, we present the collection and mining of GitHub data, aiming to understand GitHub user behavior and project success factors. We collected information about approximately 100K projects and 10K GitHub users//owners of these projects, via GitHub API. Subsequently, we statistically analyzed such data, discretized values of features via k-means algorithm, and finally we applied apriori algorithm via weka in order to find out association rules. Having assumed that project success could be measured by the cardinality of downloads we kept only the rules which had as right par a download cardinality higher than a threshold of 1000 downloads. The results provide intersting insight in the GitHub ecosystem and seven success rules for GitHub projects.

Keywords
GitHub, open source, GitHub API, association rules, k-means, discretization, project success

Introduction
Open source software (*OSS*) is nowadays an extremely popular type of software among the community of programmers and computer users and it seems that this trend will intensify in the years to come. So far, OSS communities have produced scores of successful projects such us Linux, Apache server, Mozilla Firefox, LibreOffice etc.As a consequence, OSS ecosystems (consisting of OSS projects and communities) have been actively reasearched in the past years. Among other things it is important to understand how exactly the OSS paradigm works and why it works well. The present paper analyzes the OSS projects of *GitHub* portal and mines success rules for such projects.

GitHub is an OSS forge founded in 2008 with the purpose to simplify code sharing. GitHub is growing fastly month by month and nowadays is probably the largest repository for OSS projects. In 2008 GitHub scored 41157 users and 38423 projects and in 2012 these numbers increased to 2763000 users and 4614306 projects. Scuh figures indicate the growth rate of GitHub and demonstrate that OSS people prefer GitHub very often. One major reason which made GitHub so successful is the fact that project hosting is based on the popular *git* system.

Related Work
There have been several studies in the domain of OSS data collection and analysis. For example, Gousios et al. in [1] describe a detailed process for data collection using GitHub rest API and the exact schema of the data which were collected. Another interesting study in the sector of OSS data mining is that of Chawla, Arunasalam, and Davis [2]. In this study authors collected data from Sourcforge and then used Association Rules Network(ARN) to discover relationships among project features. In another related study, Gao, Huang and Madey retrieved the entire Sourceforge database, used the non-negative matrix factorization (NMF) method to select independent significant features and applied data mining techniques (clustering and summarization) to find rules and relative features[3]. Yet another study was presented by Raja and Tretter and is a combination of Logistic Regression (LR), Decision Trees (DT) and Neural Networks aiming to identify the factors that explain the success of OSS[4]. The most recent and most related study to ours is is the one by Emanuel, Wardoyo, Istiyanto, and Mustofa[5]. In this paper authors collected a dataset of OSS projects from Sourceforge and then tried to discover success rules with Datamining 3-Itemset Association Rule. Previously Bibi, Stamelos and Angelis combined Association Rules(AR) and Classification and Regression Trees (CART) in order to identify logical associations between project attributes and the required effort for the development of a project[6]. In addition, Stamelos and Bibi used association rules on a closed source project dataset in order to estimate Analogy-based project cost[7]. Compared to previous works, the contribution of this paper is twofold: (1) we attempt to discover rules that combine both project features and user characteristics and (2) we provide and discuss such results on the relatively new and thriving ecosystem of GitHub.

Collected Data
For the collection of GitHub data we used GitHub rest API. This API is designed to return all information about a user, including repositories and their characteristics. Thus, we created a java application which takes a given username and stores in a PostgreSQL database all information related to this user. We did not use the tool described in [1], because it retrieves data in an event driven way, ignoring inactive projects and inactive users. In our work we needed random users and projects in order to have a global view of GitHub ecosystem.

For practical performance reasons we had to work on a random sample of the GitHub user base. We used the Google big query service[1] through which we retrieved data relative to GitHub. We chose to work with usernames of users who appeared to be active spanning all years between 2008 and today. For all of these usernames we ran the collecting process for 15 days. The major performance problem during the collection of the data was the fact that GitHub API has a restriction of 4000 requests per hour, including the authentication requests. This problem reduced the collection rate and forced our collector application to stop until request limit was nullified after a time interval.

For every user, the information we retrieved was *name*, *email*, *blog*, *created date*, *url*, *location*, *number of public repositories*, *number of private repositories*, *number of public gists*, *number of private gists*, the boolean variable *isHirable*, *number of collaborators*, *number of followees, number of followers* and

1 https://developers.google.com/bigquery/

date of user last action. *Gist* refers to a limited amount of code that completes a specific operation. Feature *isHirable* denotes the user's intention to be hired or not.

Also for every repository we retrieved *programming language*, *description*, *created date*, *number of forks*, *homepage url*, *name*, *owner*, *git url*, *number of open issues*, boolean *is fork* (which denotes whether repository is fork of another repository), boolean *has downloads*, *number of downloads*, boolean *has issues*, boolean *has wiki*, boolean *is private* and *number of hooks*. The term *fork* denotes a repository that is an extension of another repository. The two repositories are considered to be different. The term *hook* denotes a change in the functionality of a software project.

Not all of the GitHub API informaton is useful for quantitative analysis. For example username, email or urls don't provide any readily useful information for data analysis. Therefore, we chose to keep for every project 13 features, to be used for subsequent analysis. The features stored are: repository main language, number of forks, number of open issues, has wiki, number of hooks, repository age (extracted from created date), country of the repository user, number of pubic repositories of repository user, is hirable (with reference to repository user), number of repository user collaborators, number of repository user followers, and number of the repository user followees.

Data Collection Process

Collection process was implemented through a custom java application taking as input a list of usernames and storing in a database all GitHub user information. The pseudo-code of collection process is:

```
start_time_minutes ← getCurrent timestamp
size_of_users_list ← 10153
sum_of_requests ← 0
while(true){
while(size_of_users_list>0 and
sum_of_requests<4000 )
{
  take_information_related_to_current_username
  store_information_to_database
   sum_of_requests ++
 while(size_of_repo_list>0 and
sum_of_requests<4000 )
  {
    take_information_of_user's_repository
    store_information_to_db
    sum_of_requests ++
   }
}
wait(60-(current_time_minutes – start_time_minutes))
start_time_minutes ← getCurrent timestamp
}
```

This java application ran for 15 days, from 1st of April 2013 to 15h April 2013, and produced a dataset of 192732 repositories and 10153 users.

Data Analysis

Our goal was to mine the collected data and produce association rules describing interesting data relationships. We chose to use the *apriori* algorithm, available through *weka tool.* Apriori algorithm [9] is a light algorithm for extracting association rules from big databases. This algorithm works by identifying the frequent individual items in the database and extending them to larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules.

Apriori alogorithm works with discrete values, while most of GitHub features are given in continuous values. To resolve this problem we chose to use k-means algorithm to discretize feature values. As mentioned in [8], a common unsupervised way to discretize continuous values is to run k-means algorithm and discover centrals in the range of values. Next, each value may be put into the cluster with the nearest central. K-means is an iterative algorithm that takes as input the number of clusters and computes iteratively new centrals until it converges in a stable permutation of the centers. In this way continuous values are replaced by the discrete values of the centers.

As mentioned above, data collection produced two csv files, one for the users and one for the repositories. Both files were imported into spss and analyzed. First, descriptive statistics for both user and repository data were computed revealing the main characteristics of the datasets. Next, we applied k-means algorithm to the the 13 features that have been selected to participate in the association rules extraction process, and we found five centers (discrete values) for each feature. After that we recomputed the values of each feature and we created arff files which contained the discretized 13 features. Eventually apriori algorithm was run and producing the association rules. Apriori algorithm requires that the user set the *support* and *confidence* thresholds. We remind the reader that support refers to the frequency of the item set in the left side of the rule and confidence refers to the frequency of the right side of the rule in the set of rows in witch the itemset of the left side of the same rule is found.

One significant characteristic of our dataset was distribution of the values of *number of downloads*. As frequenlty observed in OSS ecosystems, a very small percentage of projects has a very large amount of downloads and a large percentage of projects has few or even zero downloads. In other words, the values of this feature follows a zipf distribution. For example by looking at the descriptive statistics we observe that the ratio of projects having more than 1000 downloads are 246/192735=0.001, so if we had set support threshold equal to 0.8 we would have never found rules that would contain the item downloads>1000 because this item's support was just 0.001. To cope with this problem, we run the algorithm with no support filtering but with confidence 0.75 and kept rules that detect rare itemsets but we secured that if certain itemsets are found in the left site of the rule, the right side would certainly contain the item downloads>1000. In this way we managed to study the most popular (in terms of downloads) and therefore most successful OSS projects in our GitHub subset.

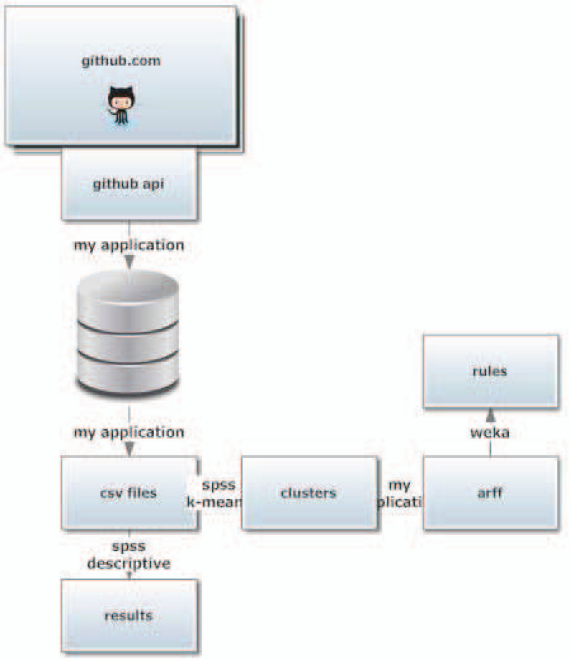

Figure 9. Collection and analyze process

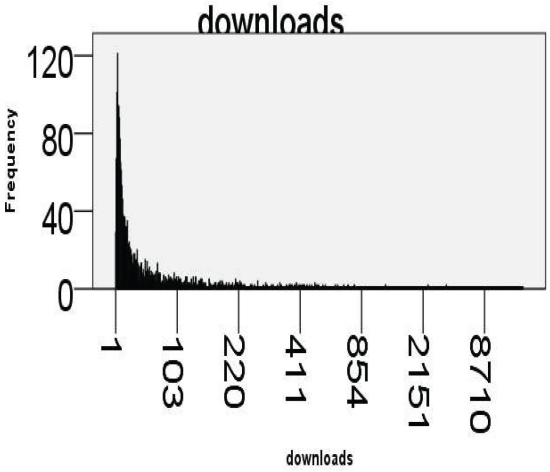

Figure 1.Downloads per project

Results

Figures 1 to 8 and Table 1 provide information that describes the dataset. A zipf distribution is observed in most cases.
Figure 1 shows the zipf distribution of downloads. It is evident that a large amount of projects has just one download and a very small percentage has a lot of downloads. Both the diagram and the statistics table are extracted from the subset of the projects that has at least one download(2642/192735).In figure 2 we can see the distribution of open issues within the projects. Both the diagram and the statistics table data extracted of the subset of the projects that has at least 1 open issue (2642/192735).In figure 3 presented the distribution of forks for every project. Both the diagram and the statistics table data extracted of the subset of the projects that has at least 1 fork (31088/192735).In figure 4 we can see the distribution of languages within the projects. Figure 5 illustrates the percentages of users and the amount of their public repositories. As we can see most of the users has from 10 to 40 repositories. In figure 6 we can observe that the majority of the users have from 2-5 collaborators, so we can safely say that most of the projects has small development teams. Finally Fig. 7 and Fig.8 shows the number of followers and the number of followees. A small percentage of users has a very large amount of followers and followees but the majority has a small amount or even one follower or followee.

Table1: Descriptive statistics of the features of projects

Descriptive statistics				
	Mean	Variance	Minimum	Maximum
Downloads	5025	14994816042,07	1	5548233
Open Issues	5,34	312,308	1	1233
forks	5,94	967,81	1	2204
Public Repositories	19,13	2163,16	0	3582
Collaborators	2,39	39,83	0	347
Followees	14,74	17788,20	0	12789
Followers	25,65	22438,51	0	6846

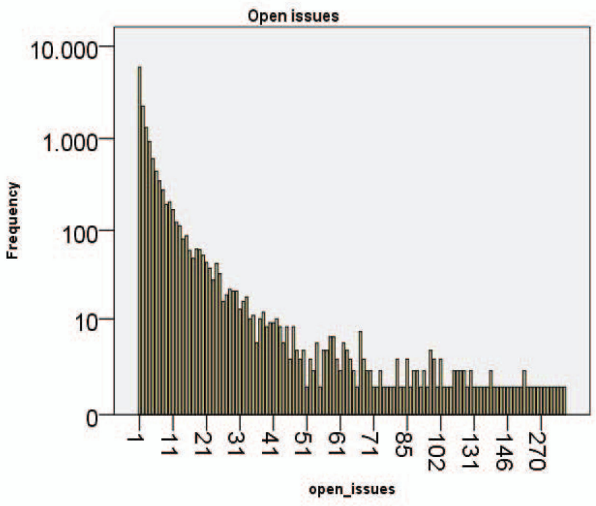

figure2.Open Issues

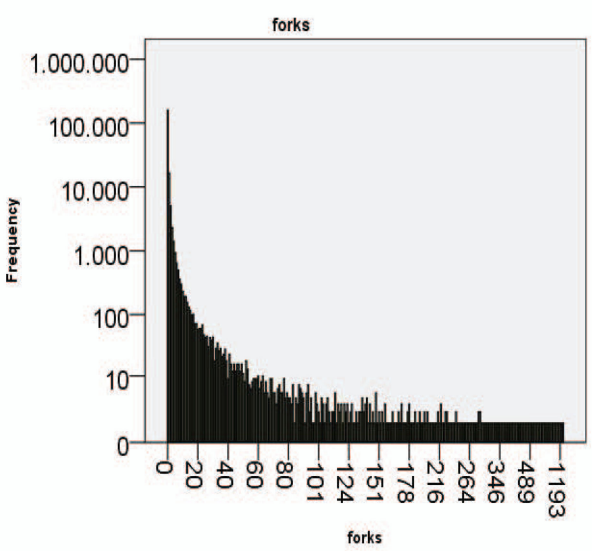

figure3.forks

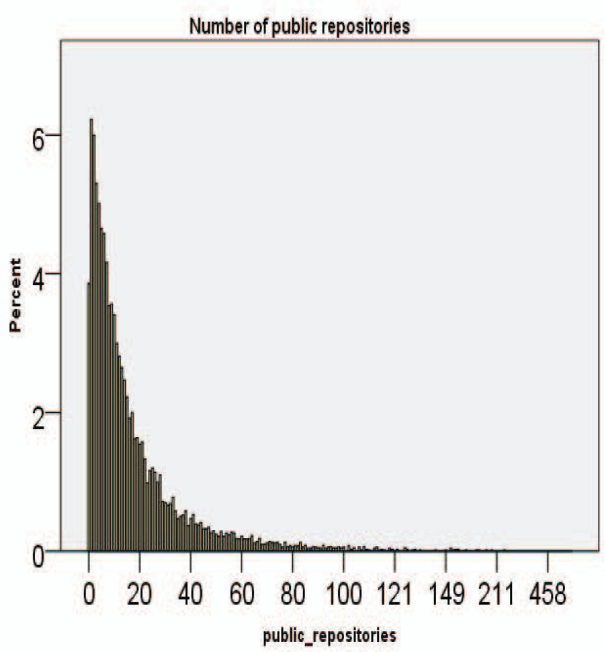

figure5.Number of public repositories

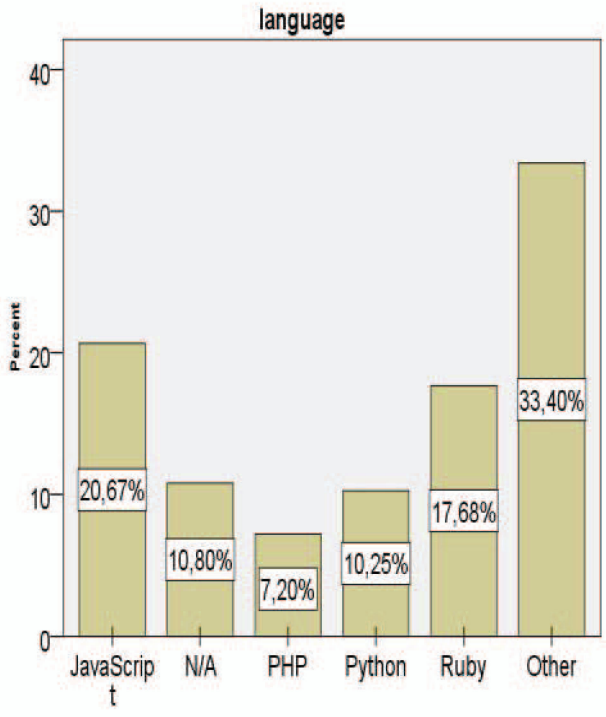

figure4.Repository languages

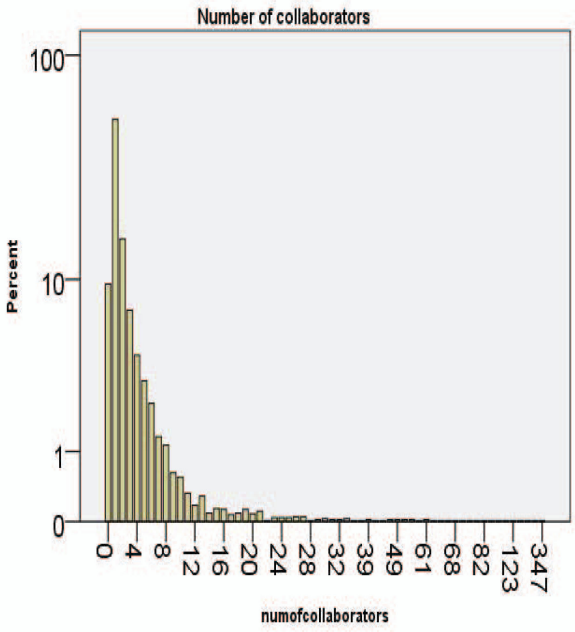

figure 6. Number of collaborators

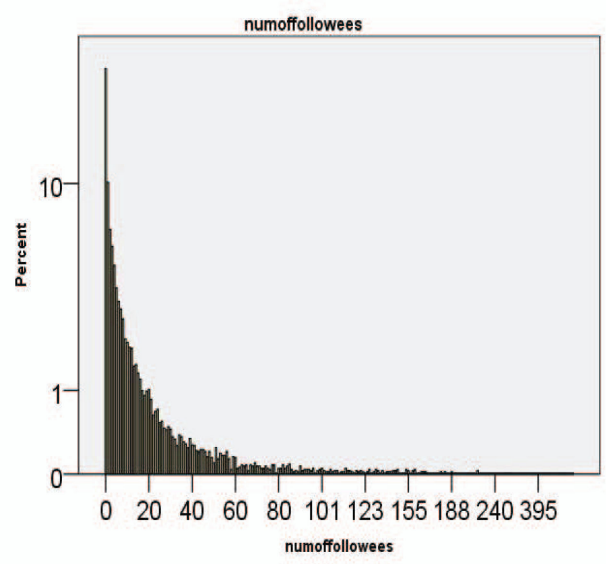

figure 7. Number of followees

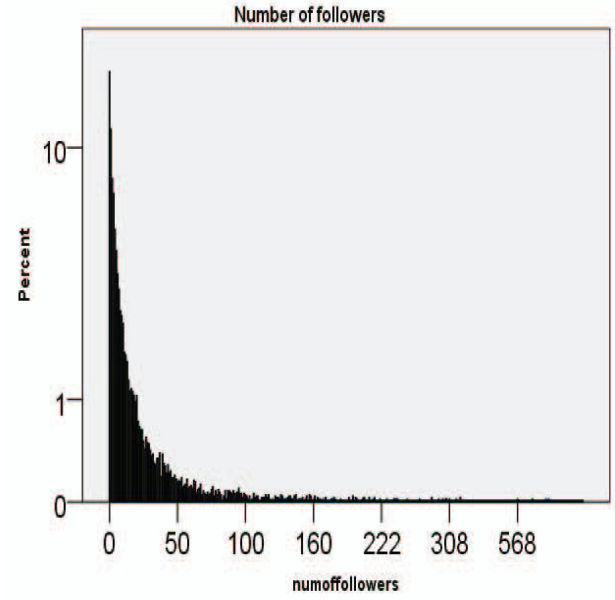

figure 8.Number of followers

Success Rules

In this section we simply list the most successful association rules detected.

1. **openIssues=near_26 userLastAction=near_4 numOfFollowees=near_8==> downloads=1000andMore confidence 0.99**
2. **repoAge=old_36 userLastAction=near_4 numOfCollaborators=near_2 numOfFollowees=near_8 ==> downloads=1000andMore confidence 0.81**
3. **repoAge=midle_aged_24 userLastAction=near_4 UserPublicRepositories=near_12 numOfFollowers=near_13 numOfFollowees=near_8 ==> downloads=1000andMore confidence 0.78**
4. **repoAge=midle_aged_24 userLastAction=near_4 UserPublicRepositories=near_12 numOfCollaborators=near_2 numOfFollowers=near_13 numOfFollowees=near_8==> downloads=1000andMore confidence 0.78**
5. **hasWiki=true repoAge=old_36 numOfFollowees=near_8 ==> downloads=1000andMore confidence 0.77**
6. **repoAge=old_36 isHirable=false 6142 ==> downloads=1000andMore confidence 0.77**

Discussion

Following our approach we eventually discovered 6 success rules. The first rule informs us that if the project has a number of open issues close to 26 and the user has made some action in GitHub environment in the past 4 months and user has a number of followees near 8 then the project has downloads more than 1000 with probability 99%. In other words this rule says that if a project is active, its owner is active and follows a small number of other users it has a large probability to be successful. At this point we may observe that successful projects seem to have a number of open issues which is neither zero nor very large. Intuitively, zero open issues probably reveal an inactive project, while a large number of open issues means either a project that went inactive or that is of low quality.

The second rule informs us that if project's age is near 36 months, which may be considered as not a new project, and project user made an action in the last 4 months, has number of collaborators near 2 and follows near 8 other users then this project has downloads more than 1000 with probability 81%. In other words, in addition to the features that were taken into account by rule 1, rule 2 examines the maturity of the project and the size of development team. As we see, mature projects with relatively small development teams seem to be more successful. This finding deserves further research to understand why projects with four or five collaborators appear less successful.

The third rule dictates that if project age is near 24 months, project owner made some action in the last four months, project owner has a number of public repositories near 12, project owner has a number of followers near 13 and a number of followees near 8 then project has more than 1000 downloads with probability 78%. This rule associates the success of a project with project's maturity, user activity, user number of repositories and user social activity. In this rule appears the new item user public repositories near 12, not observed in the previous rules. This item show that if a user has a small amount of public repositories it is more probable that these repositories will be successful. This sounds reasonable, because users with few public repositories may be more committed to them compared to users with a lot of public repositories. Another notable element of this rule is that the users of successful projects appear not to be social enough.

The forth rule is an extension to the third one, thus it has in the left side the same itemset with rule 3 plus the size of the development team. With this rule we see that, as before, small teams have higher probability to run successful projects.

The fifth rule says that if project has wiki, is mature and its users follows a small number of other users, then it has more than 1000 downloads with probability 77%. A new item in this rule is Wiki, a very important feature of an open source project, because it helps other users to download it and use it, making the project more popular. The rule suggests that, if a project is

mature and its owner is not willing to be hired then it has large probability to be successful. This rule makes sense too, because if a developer has already a project that has more than 1000 downloads then probably he wouldn't be interested to work in another project.

Conclusions and Future work

The present study tried to discover paterns which associate open source project and owner features with project success. We retrieved data for approx. 100K projects and 10K users from the highly popular GitHub repository and analyzed them after preprocessing them to achieve value discretiazation. We have been able to produce six association rules for successful projects, using number of downloads as a proxy for project success. As a first conclusion, by combining all six rules, GitHub projects that are successful appear to exhibit the following characteristics.

- maturity
- high activity
- support to other users
- active owners
- owners with a small amount of other projects
- small development teams
- owner with small amount of followers
- owner with small amount of followees
- owner without desire to be hired

There are some validity threats in our study at this time. Success of a project may be measured differently, for example by considering the number of languages it has been translated to. Also we set as threshold of downloads that divide projects into successful and non-successful the number 1000 by observing the data distribution. At this point it would be a good idea for future work to search for more elaborate approaches for selecting this threshold. Another point of the study that can be improved is the way in which we discretized the values of the dataset features, by seeking better ways to discretize continuous values which follow zipf distribution compared to k-means algorithm. Other mining algorithms for extracting rules might give better results. Finally there may be some issues with GitHub API data themselves, for example levels and intensity of project activity are not distinguishable, since a single simple commit by a user decides the value of last action date.

We plan to download the complete GitHub dataset and repeat our analysis, addressing the above mentioned research issues. Our current data are available in http://sweng.csd.auth.gr/githubData.zip and can be used in future studies by researchers who wish to analyze this specific subset of GitHub.

REFERENCES

[1] Georgios Gousios, MSR '13. *The GHTorent dataset and tool suite(2013)*

[2] Sanjay Chawla, Bavani Arunasalam, and Joseph Davis.2003.*Mining Open Source Software(OSS) Data using Association Rules Network, PAKDD'03 Proceedings of the 7th Pacific-Asia conference on Advances in knowledge discovery and data mining, Pages 461-466* 2003

[3] Ding,Yongqin Gao, Yingping Huang, Greg Madey, *Data Mining Project History in Open Source Software Communities, NAACSOS Conference 2004, Pittsburgh, PA, 2004*

[4] Uzma Raja and Marietta J. Tretter J. 2006. *Investigating Open Source Project Success: A Data Mining Approach to Model Formulation, Validation and Testing,* SAS User Group International Conference, paper 070-31, SUGI 2006

[5] Andi Wahju Rahardjo Emanuel, Retantyo Wardoyo, Jazi Eko Istiyanto, Khabib Mustofa.2010.*Success Rules of OSS Projects using Datamining 3-Itemset Association Rule,* IJCSI International Journal of Computer Science Issues, 7 (6), 2010

[6] Stamatia Bibi, Ioannis Stamelos, Lefteris Angelis .2008. *Combining probabilistic models for explanatory productivity estimation,* Information and Software Technology, 50 (7-8), Pages 656-669, 2008

[7] Stamatia Bibi, Ioannis Stamelos. *Analogy Based Cost Estimation Configuration with Rules,* Proceedings of the Eighth Joint Conference on Knowledge-Based Software Engineering, Pages 317-326, 2008

[8] *James Dougherty, Ron Kohavi, Mehran Sahami, Computer Science* Department Stanford Univercity, *Supervised and Unsupervised Discretization of continuous features,* in Armand Prieditis & Stuart Russell, eds., Machine Learning: Proceedings of the Twelfth International Conference, 1995

[9] Rakesh Agrawal and Ramakrishnan Srikant.1994. *Fast algorithms for mining association rules in large databases*. Proceedings of the 20th VLDB Conference, 1994