Analyzing Factors Impacting Open-Source Project Aliveness

Rudi Chen
David R. Cheriton School of Computer Science
University of Waterloo
200 University Avenue
Waterloo, ON, Canada
rudi.chen@uwaterloo.ca

Ivens Portugal
David R. Cheriton School of Computer Science
University of Waterloo
200 University Avenue
Waterloo, ON, Canada
iportugal@uwaterloo.ca

ABSTRACT

Developers would like to know if an open-source project will be alive in the future before contributing or using its source code, so that they know that contributions will impact other developer's work, and that security updates or bug fixes will be implemented. To address this issue, we study the impact of several different metrics in the prediction of open-source project aliveness. We analyzed 8 million GitHub projects from the Boa's dataset and studied almost 70,000 projects. Our results show that date features, such as the number of days since the last commit, are good predictors of project aliveness, with more than 80% accuracy. Additionally, the impact of the percentage of founder commits is significant, and the presence of documentation files, when not named "README", correlate with future project aliveness.

CCS Concepts

•General and reference → Empirical studies; •Software and its engineering → *Open source model;*

Keywords

Mining repositories; Open-source project; GitHub; Machine Learning; Decision Tree; Big Data

1. INTRODUCTION

Open-source projects, whose source code is publically available, allow developers to use existing code, make changes, and contribute back. Modern services such as GitHub, a free Git server for open-source projects, provide useful features (e.g. pull requests, issue tracking) to synchronize multiple development efforts. Some open-source projects are actively maintained by the community. For example, new bugs in the LLVM compiler tool suite[1], their fixes and feature improvements are periodically discussed in mailing lists among developers. These discussions are not only important for the contributors, but also for the users of the source code, because they can be assured that a particular open-source project being used has updated security fixes, has the necessary features, and works with the latest versions of its dependencies, compilers and operating systems so that any developer can work or contribute without issues.

However, not all projects remain active. Developers may lose interest in an open-source project, the departure of the key maintainers may lead to a lack of direction or structure in the project or users may shift to another solution. This leaves the project outdated, perhaps no longer functional and obsolete. This implies that developers and users may have a difficult decision to make when investigating open-source projects to work with. Prospective contributors want to work on open-source projects that will last, so their contributions have a long term impact. Users of the source code, i.e. developers who use external, open source libraries in their own projects want to know if that particular open-source project is reliable, maintainable, and that it will not be abandoned in the future, so they can safely run and work with the project and have their concerns and questions addressed. With these two problems in mind, we decided to investigate the factors that most contribute to open-source project aliveness and discover whether it is possible to predict the chances of a particular project being alive in the future.

In our approach, we use a data set of almost eight million projects from Boa. We first start by brainstorming factors that are expected to contribute to project aliveness. For example, we hypothesize that the number of recent commits by the founder of the project is an important factor. From these, we compute a set of features and labels over the revision history of repositories and then, use a machine learning approach and create a project aliveness predictor. In particular, we choose to use those features to train a decision tree and gain insights about the most useful features. We also examine aliveness with individual features to better understand how certain factors correlate or fail to correlate.

2. RELATED WORK

As previously explained, open-source users and developers seek projects that are expected to stay alive (or active) in the future, so that their work can improve or be improved by the project. The concept of "project aliveness" can be understood in many different ways and approached from many different angles. Our notion of aliveness, which will be for-

[1]http://llvm.org

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

malized later in this paper, emphasizes the point of view of users of the source code and contributors, who care that the code is maintained, even if it involves a relatively small number of commits per year. In this section, we make a distinction between concepts similar to project aliveness and refer to some study available in the literature.

Consider a user who wants to use a new open-source library for their project. This user may use popularity as a heuristic to assume that a particular library has the most recent updates, is more stable, and will be alive in the future. After all it seems unlikely that a given project will abruptly lose most of its developers. However, popular open-source projects are not necessarily maintained (or alive) in the future [6], and therefore not the main object of study of this research. Open-source project popularity has also been studied in [11] and [17].

An alternative solution to the user at the beginning of this section would be to choose a successful open-source project. It is assumed this project has overcome several problems such as fixing bugs, budget (if applicable), performance issues, and lack of interest from the community. All of these factors may indicate that this project will not be abandoned in the near future, and more contributions are expected. However, how can users or developers know beforehand that a project is successful?

The concept of a project being successful is still fuzzy, mainly because of the definition of success. Some studies [4, 5] investigate the best way to describe project success. The first of the studies goes deeper and proposes a taxonomy, based on other studies published in the literature, that describes success in 6 dimensions: product quality (task completion), user interest (popularity), project performance (being effective and efficient), project effectiveness (capable of producing), project efficiency (being economical), and project activity (quantity of output generated).

As noted, project success is an umbrella term that encompasses several other secondary terms, which can be measured using different metrics about projects. Previous work has examined many of these different secondary terms [15, 16, 18, 20]. In this study, we focus on one aspect of project success by measuring the chances that a project will be alive in the future, will receive updates, new features, and will not be ignored.

In this study, open-source project aliveness is defined as the characteristic of a project having some activity in the future. These activities reflect an interest from the project creator, or other developers and are aimed at maintaining the project for the future. Note that the quantity of activity is of secondary concern since many projects reach a level of completeness, and requires less active feature development. Under our project aliveness definition, any contribution made by the community aimed at improving the project means that the project is alive.

Other studies in the literature have investigated open-source project aliveness from different views. One study [19] adopts a social network view on project aliveness and investigates whether having an explicit structure among project contributors increases the chances of more contributions in the future. The study uses the network centrality measure to capture the structure among contributors.

A more in-depth pair of studies [10, 13] investigates project aliveness using statistical tools to predict whether projects will have activity in the future. According to their results, open-source projects with a long history of activity are more difficult to be abandoned. This may indicate that projects that reach a certain level of maturity, trust or establishment need to worry less about their aliveness for the future than recently created open-source projects. Moreover, adding new programmers to the development is positively correlated to project aliveness in the future.

A third work [12] published in the literature describes project aliveness in terms of its viability. The study formally defines project viability as "the ability of a project to grow and maintain its structure in the presence of perturbations". The metric is composed of three dimensions: vigor (related to growth), resilience (related to recovering from disturbances), and organization (related to project structure).

All studies previously mentioned provide different ways of analyzing project aliveness, but having few fixed factors under investigation. Therefore, little is known about alternative features that may be determinant in project aliveness. Our study tries to close this gap by studying different factors and their impact in project aliveness prediction.

3. STUDY DESIGN

This section explains and defines the terminology used in this paper and describes each step taken, from concepts to execution, on this study. After discussing the terminology, we present the research questions, the data and tools that we use, and a description of the approach.

3.1 Terminology

In informal communication, software developers and engineers use the terms "project" and "repository" interchangeably to refer to the source code and the collection of artifacts used in the development of a computer program (e.g. commit comments, documentation, etc). In common usage, a "repository" tends to emphasize the storage space used to store the project's source code and its artifacts. For example, one may say "the compiler project is stored in the GitHub repository."

Each project has a founder. They are usually the person who either idealized or created the infrastructure for the project. However, other terms may also be used to refer to this role, such as project author or creator. We decided to use the term project founder as the one who started the project. In our approach, the project founder is the one responsible for the first contribution of the project.

Several are the terms used to refer to the act of contributing to an open-source project. Some of the terms software developers and engineers often use are "commits", "contributions", "revisions", or simply "activity". We decided to use the term "commit", since it seems to be the most widely used, and it is part of GitHub's terminology.

Lastly, each commit has a person associated with it. Here, we simply refer to him or her as the commit author, or the committer. We do not expect any confusion with the terms, since they are widely used and the concepts are fairly straightforward.

3.2 Research Questions

To formally define the directions of this study, we describe our goals in three research questions, shown below:

RQ1: What contribution activity impacts open-source project aliveness prediction?

Open source projects that receive regular contributions are likely to stay active, and therefore alive in the future [8]. However, this is not the only activity factor that affects project aliveness. Thus, it is worthy to investigate what historical contribution activities correlate with and can be good predictors of project aliveness. This may include the number of commits in a month, the number of different contributors, the distribution of contribution among those contributors, and the artifacts (e.g. commit comments) associated with these commits.

RQ2: What is the impact of the founder on open-source project aliveness prediction?

Our hypothesis is that the survival of open-source projects depends heavily on its founder, that is, the initial project creator. A founder holds significant part of the domain knowledge of the project, owns a considerable portion (if not the majority) of the code, and is responsible for its direction and vision. Once the founder leaves the project, we expect that its survival chance drastically reduce, as no one in the community may be have enough knowledge to conduct the project in the future.

RQ3: How does documentation quantity and activity affect open-source project aliveness?

New developers who want to start contributing to a particular open-source project, may not have enough knowledge about some part of the project, or even what tasks are available for them to get started, tips, and expectations. They generally start by looking at the documentation files and are only occasionally motivated to contact existing developers [14]. Therefore, documentation is an important issue for open-source projects, which help developers have a clear view of the project and allow them to keep the project alive in the future [1]. Investigating the influence of documentation activity on open-source projects may give insights about how much effort should be spent on this activity.

3.3 Dataset

The dataset used in this study comes from the Boa project [2], hosted by the Iowa State University. The project is publicly available online and is used in the 2016 MSR challenge[2]. It contains data from almost eight million GitHub projects dating back to early 1990s until September 2015. These projects have a history of more than 23 million commits and were written in more than 20 programming languages.

The Boa project is composed of a distributed query infrastructure and a domain-specific programming language with the same name. The Boa language allows researchers to query open-source projects' data, commit history, and associated files with a few lines of code. Scripts in Boa are compiled to MapReduce code to be run in Hadoop clusters. Their scalability and performance makes it suitable for processing this type of Big Data.

However, the expressivity of the Boa language is limited. In particular, it does not provide a way to write any arbitrary "reduce" operation. Furthermore, the online infrastructure makes quick iteration and debugging very difficult. For these reasons, we decided to use the Boa infrastructure mostly for simple investigatory queries. For more complex queries, such as using machine learning, we extracted the relevant information about projects and their commit history to process them locally.

[2]http://msrconf.org

Table 1: Project information extracted from Boa dataset

Project Information	Description
ID	An unique identifier for the project
Name	The name of the project (including "/" and the username)
Number of commits	Number of commits since the creation of the project
Presence of documentation files	Whether this project contains documentation files
Project creation time	Project creation time
Number of programming languages	Number of programming languages in the source code of the project

Table 2: Commit information extracted from teh Boa dataset

Commit Information	Description
ID	An unique identifier for the commit
Project ID	The ID of the project this commit belongs to
Commit time	The time this commit happened
Author username	The username of the committer

Additionally, only Java projects come with a commit history, which is the primary indicator of contribution activity. For this reason, we restrict our investigation to Java projects. With this restriction, the size of our local dataset is around one gigabyte, a manageable size. At this order of magnitude, it is also faster to run queries locally than on a distributed system which has a lot more overhead. Tables 1 and 2 show what project and commit information was extracted from Boa online dataset. This information was stored in a local SQL database.

Every project in Boa has a unique identifier. The project name is also important to understand the nature of projects. All project names have the format "userid/projectname", which allows us to manually inspect the projects on GitHub, and to have additional insight about the purpose of the project (e.g. weekend hackathon project). Projects with the same name may also indicate that the project was forked (e.g. a project named 'spark' is likely a fork of the Apache Spark project).

The number of commits a project has is an important measurement of activity. It is used in the analysis of project commit history and in the definition of project aliveness. Equally importantly is the presence of documentation in the project. In Section 4.4, we investigate the presence of documentation files that start with certain names, such as "README" and "CONTRIBUTING", among other project artifacts, which we selected based on their frequency of occurrence and an estimate of their importance.

Each commit has an identifier, which is the same as the commit ID created by Git. It is a chained SHA-1 hash of all changes made in the commit, the parent commit hash, and commit header information. The project ID is also extracted, so it is possible to identify the project each commit is associated with. Additionally, we extracted the commit time. It is relevant to analyze the evolution and density of the any project's commit history. The commit time is formatted identically to the project creation date previously explained. A last piece of information about commits is their authors. We extract the username of the commit author which allows us to investigate how commit activity varies by participant, including the project founder (author of the first commit).

3.4 Approach

In our approach, we construct a set of features and we generate instances and labels from project and commit information, to be used with the J48 decision tree algorithm, present in the Weka framework [7]. In machine learning a feature is an attribute of a data point, an instance is the value of all the features for a data point, and the label is the value we try to predict from our data. This approach is expected to provide insights into the impact of various projects attributes on the prediction of open-source project aliveness. Each instance is generated via backtesting over the commit history in one-month intervals. That is, we first set a cutoff date. This represents a user looking at the project on that date and asking the question "will the project be alive?" Commits before that date becomes our known history, from which we compute features. Commits after that date indicate whether the project indeed turned out to have activity and is used to compute the label. Then, the cutoff date is iteratively set to a month back until the creation of the project, recalculating the features and labels every time.

3.4.1 Labels

In this study, a project is said to be alive if there is at least one commit after the cutoff date. One may argue that projects that have a single commit in a year are not really alive. However, in the context of the motivation for this study, we care that the project is at least being maintained, even if it has reached a state of completion such that active improvement is no longer needed. For example, it may be the case that the project needs only a yearly update to work with the latest Linux release and fix an occasional security bug. We would not call such a project "dead".

3.4.2 Features

We decided to analyze 15 features to discover their importance on open-source project aliveness. These features are calculated using information extracted from Boa as explained in the previous section. Table 3 lists each feature and gives a brief description of them.

Features were divided in three types, depending on what is processed. People features relate to the project founder and other committers. Commit features analyze properties of the number of commits in the project. Lastly, some features pertained to the project and were listed under the type Project.

3.4.3 Data Preprocessing

To ensure that we measure the data in a way that represents what we want to measure, some preprocessing steps were taken.

- We remove all data after September 2013. The reason is that the September 2015 GitHub dataset Boa provides was obtained by mining GitHub on two occasions, 2013 and 2015. However, the 2015 dataset is much smaller, despite the growth of GitHub in recent years, and not all projects mined in 2013 were updated to their latest 2015 snapshot, which we confirmed with the Boa authors. This is a problem: while some projects have commit activity after 2013, the absence of commit activity after 2013 does not imply that the project died. To have an accurate definition of aliveness, we therefore need to ignore data after September 2013.

Table 3: Features included in this study

Type	Commit Information	Description
People	Number of committers	The number of different committers in the project
	Number of committers with multiple committs	The number of different committers who have committed more than one to the project
	Number of committers sticking at least a day	The number of different committers that committed in different days
	Number of committers sticking at least a week	The number of different committers that committed in different weeks
Commits	Number of commits	The number of commits in the project
	Percentage of commits - founder	The percentage of founder commits in the project
	Percentage of commits - top 2	The percentage of commits done by the 2 most contributing commiter
	Percentage of commits - top 3	The percentage of commits done by the 3 most contributing commiter
	Percentage of commits - top 5	The percentage of commits done by the 5 most contributing commiter
	Days after last commit	The number of days between the last commit and the cut off date
	Days after last founder commit	The number of days between the last founder commit and the cut off date
	Largest gap between two consecutive commits	The greater number of days between two consecutive commits
	Density of commits	The density of commits in the project
Project	Project age	The number of days since the creation of the project
	Number of programming languages	The number of programming languages the project work with

- In GitHub, some projects are forked, which means that a developer copies the project and all of its commit history to his or her account and evolves the forked project independently of the original one. This is a problem is our study, since projects with many forks (e.g. popular projects) may be disproportionately represented in the dataset. Therefore we decided to look for forked projects and keep only one copy of them. The Boa dataset does not indicate which projects are forks of others. Instead, we identify forks using the ID of the first commit, which is the Git commit hash. Because forked projects also copy the whole history of commits, the first commit will be the same in the original and the forked project. However, while we can find duplicates this way, this does not identify the original ("upstream") repository. Instead, we keep the project with the most commits, and discard the other duplicates. Generally, people fork projects and add changes with the intent of making a pull-request with those changes, i.e. merge the changes with the upstream repository. Therefore, the original repository is most likely to have the latest changes and developers rarely update their forks with the latest commits from the upstream. Furthermore, in the rare event that a fork has more commits than the original, this may indicate that the original was abandoned but someone else took over the maintenance of the project by maintaining their own fork. In that case, the project is still being maintained, conceptually.

- We remove all projects whose commits are all made by a single developer. The concern is that many, if not most projects on GitHub are not meant as reusable code for other developers to use or improve on. Developers use GitHub for many other reasons including as a free web host (via GitHub pages), to learn version control, to store personal projects, or as a portfolio for employers to see. This is especially common for students. These projects typically do not show continuous development activity, which means that it would be possible to obtain a very high classification accuracy simply by checking if more than one developer contributed to a project. However, that would provide no insight, since developers do not expect to use or contribute to such projects and would not ask themselves whether those projects will be alive. As such, our study tries to avoid these projects by analyzing only projects that have more than one committer. We expect that after this filter, the remaining projects are truly collaborative and more suitable to be relevant to our research questions.

- Then, for every project, we calculate the number of days between the first and the last commit observed and remove those whose activity span less than a week. This is for similar reasons as removing projects with a single committer. GitHub is often used as a collaboration tool on short-term projects, such as weekend Hackathons or class homework. Those are also not meant as reusable components for a wider audience to use or contribute to. We expect that projects with a long term vision will include polishing and bug fixing work, especially as new users slowly come in and identify new issues. This back and forth process will take a lot of time. As such, a week is a very conservative lower bound.

3.4.4 Analysis

After all data is generated, it is ready to be processed by the decision tree algorithm, which will identify the most impacting features on project aliveness. However, the number of positive or negative samples, i.e. the number of projects with the label alive or dead, is not equally represented. It means that when providing the algorithm with the current data, results may be biased to one of the label values due to the number of instances with that label. To avoid this, we select the same number of projects from each label to be analyzed, by randomly discarding projects with the most represented label. By doing so, the baseline accuracy will be 50%.

Once data is ready, we feed it into Weka's decision tree algorithm and analyze the results. We also iterate with different values for the parameter minimum number of instances per leaf. It is useful to set a large number here since we would like a tree of reasonable size, amendable to human interpretation. It also helps prevent overfitting. Generally, setting the value such that the decision tree has depth 2-4 is a good balance.

It is also possible to analyze the distribution of feature values for each feature individually. In particular, we can compare the distribution of values for projects marked as 'alive' and projects marked as 'dead', using tools such as boxplots. This can help gain more insight into individual features. In the paper, we use this approach for features that help us understand the impact of project founders.

Documentation is also examined separately from the rest of the features. For technical and practical reasons, it was difficult to obtain information about the presence of documentation files at cutoff date during backtesting from Boa. We can only query files in the latest snapshot of the project. Therefore, it would not make sense to include documentation features in the instances generated during backtesting since the instances would then contain both past and future data. Instead, to analyze the impact of documentation, we change our notion of aliveness and correlate the presence of documentation files such as README with presence of a commit within a week or a month of September 2013. Only the filename is used, since the only type of file content that can be accessed through Boa is Java code AST nodes.

4. RESULTS

This section starts by explaining the data cleaning necessary for our analysis and then presents the results of our studies, answering the research questions stated in Section 3.2.

4.1 Data Cleaning

Following our approach in Section 3.4, we first extracted all the relevant metadata from Java projects in Boa's September 2015/GitHub dataset. This gives us a total of 355,641 projects. Among those, 97,317 did not have any commit history attached at all, and 122,639 did not have any commit history after the September 2013 cutoff date. Those projects were removed since there would be no commits to compute features from.

Our fork detection algorithm marked 5,046 projects (2%) as duplicates. Among those, most projects have only a few

Table 4: Log-scaled histogram of the lifespan of projects in the dataset

Number of days	Number of projects
[0,0]	86838
[1,2]	16087
[3,6]	16918
[7,14]	18267
[15,30]	19342
[31, 62]	19187
[63, 126]	17539
[127, 254]	13984
[255, 510]	11745
[511, 1022]	7422
[1023, 2046]	4153
[2047, 4096]	1265
[4095, 8190]	194
[8191, 16382]	61

copies. For example, 2,014 projects are duplicated once, sometimes by the owners themselves as a way of having two "versions" of the project instead of using branches. From manual inspection, projects with many copies tend to be well-known, impactful projects. For example, the Android Cyanogen project android_system_core, a popular distribution of the Android operating system has 81 copies. We also find, as a side effect of this inspection, that Boa's data is not a full snapshot of GitHub on September 2013. The original repository CyanogenMod/android_system_core is not present in the dataset, and 81 forks seems on the low side given that the GitHub page currently lists more than 800 forks, even if it was over two years ago. Furthermore, no project has more than 100 copies which is unlikely given that the most popular projects tend to have thousands of forks.

Next, as we hypothesized, most people use GitHub not as an open-source collaboration tool, but as a free file storage service. For each project, we calculated its lifespan, the number of days elapsed between the first and last commits. As shown in Table 4, almost 90,000 repositories have all their commits in less than one day. This usually means that either 1) the project was a short-term experiment or 2) a project not previously under version control was retroactively put on GitHub in a single commit (if it was already a local Git repository, the original commits would kept once uploaded to GitHub).

Among projects with commits before September 2013, there were 135,126 single-owner repositories, which were removed from our study. Among the remaining projects, 29,577 projects had commit activity spanning less than a week, which were also removed. Applying these filters left us with 68,299 projects (20%). Those are open-source projects that may plausibly be intended to be used by other developers.

4.2 Decision Tree

Using these projects, our next step is to compute instances, using backtesting as described previously. This produced 119,318 instances, of which 49,730 had the label "alive" and 69,588 had the label "dead". To balance the dataset, we dropped 19,856 instances with the "dead" labels. We then used Weka to construct a decision tree using the J48 algorithm with 10-fold cross validation. We choose to use 2000 for the minimum number of instances per nodes.

```
date_before_last_commit <= 37
|   date_before_last_commit <= 6: alive
|   date_before_last_commit > 6
|   |   project_age <= 154
|   |   |   date_before_last_commit <= 17: alive
|   |   |   date_before_last_commit > 17: dead
|   |   project_age > 154: alive
date_before_last_commit > 37
|   date_before_last_commit <= 134
|   |   project_age <= 223: dead
|   |   project_age > 223
|   |   |   date_before_last_commit <= 85: alive
|   |   |   date_before_last_commit > 85
|   |   |   |   project_age <= 546: dead
|   |   |   |   project_age > 546: alive
|   date_before_last_commit > 134: dead
```

Figure 1: Decision tree from J48 algorithm, 80.72% accuracy

```
density_of_commits <= 0.211111
|   largest_gap_between_consecutive_commits <= 43: dead
|   largest_gap_between_consecutive_commits > 43
|   |   percentage_top5_contributors_commits <= 0.995775: alive
|   |   percentage_top5_contributors_commits > 0.995775
|   |   |   density_of_commits <= 0.068548: dead
|   |   |   density_of_commits > 0.068548: alive
density_of_commits > 0.211111
|   largest_gap_between_consecutive_commits <= 18
|   |   density_of_commits <= 0.939239: dead
|   |   density_of_commits > 0.939239: alive
|   largest_gap_between_consecutive_commits > 18: alive
```

Figure 2: Decision tree without date features, 71.06% accuracy

This particular choice is an adequate balance between the complexity of the tree and the accuracy of the classifier. This produced the decision tree shown in Figure 1 with 80.72% classification accuracy.

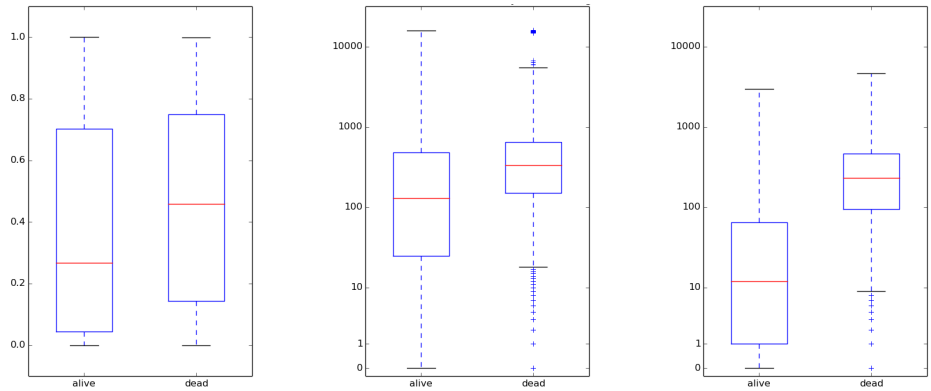
The feature "date before last commit" accounts for almost all of the correct classification. If the number of minimum instances per node is set to be such that the tree has only one node, the classification accuracy becomes 78%. Since we seek to gain insight into features with predictive power, we also repeat this procedure by removing that feature. Similarly, "date before last commit by founder" is highly correlated with "date before last commit" and can achieve a 70% accuracy alone. Without these date features, the decision tree obtained in Figure 2 has 71% accuracy.

The next more useful feature is the density of the commits. This is both seen directly through "density of commits" which is a ratio of number of commits over a time period, and indirectly through "largest gap between consecutive commits" which measures periods of inactivity. Thus, projects that are actively maintained are also more likely to be alive in the future than projects that only show occasional commits. These results tell us that while there are many features that are useful in predicting, date features are usually more effective.

4.3 Impact of the Project Founder

To understand the impact of the founder on the project, we examine two features in more depth, "percentage of founder commit" and "date before last commit by founder". Since the latter is strongly correlated with "date before last commit" (the founder is likely to have made the last commit), we also examine the distribution of that feature, as seen in Figure 3.

The boxplot for the date before the last commit is con-



(a) Percentage of class commits by founder when the project is alive or dead.

(b) Number of days since the last commit by the founder.

(c) Number of days since the last commit by any developer.

Figure 3: Box plots of the distribution of the values of features related to the project founder.

structed in log scale. This is in part because most of the data points are within a year of two of the last data point. Since GitHub is relatively new and has seen exponential growth, there are very few old projects, and the outlier oldest projects tend to be ones that used Git before the creation GitHub and was retroactively uploaded.

From the box plots, we see that the two founder commit features (a) and (b) are able to discriminate to some extent between projects that are alive and projects that are dead. When the percentage of commits by the founder is low (a), the project is more likely to be alive. The explanation we suggest is that when the percentage is low, it means that the founder has successfully created a community of committers around the project and delegated responsibilities, meaning that there are more people to maintain the project.

The date before last commit by founder is also correlated with aliveness (b), but not as much as the date before the last commit (c), regardless of who made it. However, this still suggests that the founder is important.

4.4 Documentation Activity

We are interested in looking at documentation, since we expect their presence to be a potential indicator of the willingness of the project founder to accept commits and of interest of maintaining the project in the long term. For example, we hypothesize that "contributing.md", a file usually placed in the root folder of projects containing contribution guidelines for new committers, is a strong indicator of willingness to take committers. Our first step was to run a Boa query to get a list of the most common files with the extensions "*.md", "*.txt", and no extension, which we expect to be the most common documentation file formats, and to provide some insights into which files to look into specifically.

We found that README files are by far the most common name used for documentation files and all other names have lower representation. Table 5 shows a sample subset of the most common documentation file names.

We choose to examine a particular set of files, whose file name are one of {readme, license, todo, install, contributing, changelog}, with either no extension, ".md" or ".txt", and examine if their presence correlates with the project being alive. Due to restrictions of the Boa infrastructure, we can only determine whether the repository contains the file at the time of the last snapshot, September 2013. Thus, we cannot include documentation in our set of features obtained from backtesting, since backtesting requires that data used to compute features comes strictly before the cutoff date. Adding documentation in the features would combine future data with past data, which is why documentation is examined separately.

Filename	Number of projects
readme.md	105270
readme	39852
license	14963
readme.txt	10506
license.txt	7676
copying	3077
notice	2965
todo	1459
changelog	1260
install	804
docs/readme.txt	257
contributing.md	170
maintainers	133
examples/readme	114

Table 5: A sample of the list of most common file related to documentation for demonstration, and the number of projects containing them.

Since we have no future commit history to determine aliveness, we instead check for the presence of a commit in the week and month prior to September 2013, which is reasonable given that our machine learning classifier found that date features were the best predictor of future aliveness. We calculate the baseline percentage of alive projects and compare it with the percentage of project that contain those files and that have a commit within the last 7 or 30 days.

We can make a few observations from Table 6. First, projects with a README file, either with or without an extension, are not significantly more likely to have recent commit activity (5.1% vs 5.0% for 7 days, 12.6% v.s. 11.8% for

Table 6: Likelihood of a project having recent contribution activity given the presence of documentation files.

	Last commit within 7 days			Last commit within 30 days		
	Alive	**Total**	**%**	**Alive**	**Total**	**%**
readme	2147	41782	5.1%	5252	41782	12.6%
license	1035	16210	6.3%	2392	16210	14.8%
todo	582	8943	6.5%	1256	8943	14.0%
install	359	4855	7.4%	680	4855	14.0%
contributing	218	3028	7.2%	448	3028	14.8%
changelog	243	3407	7.1%	498	3407	14.6%
One of the above	2835	53904	5.3%	6752	53904	12.5%
Two of the above	1203	16641	7.2%	2641	16641	15.9%
All projects	3394	68299	5.0%	3394	68299	5.0%

30 days). This is understandable given that most projects have a README file and GitHub has the option "Initialize this repository with a README" when creating new repositories. However, the presence of all the other files correlate with a higher likelihood of the project having recent commit activity. This can be explained by the fact that files such as "CONTRIBUTING.md" are indicators of the willingness of the project founder to accept external commits. The results show that projects that include "contributing.md" are 25% to 45% more likely to be alive (7.2% v.s. 5.0% for 7 days, 14.8% v.s. 11.8% for 30 days). Projects that contain any of the above files have almost the same likelihood of having recent commit activity as projects with only a README file, which is again explained by README files being by far the most common documentation files. However, projects that have at least two of the above are the most likely of all project types to have recent commit activity.

5. THREATS TO VALIDITY

Following the taxonomy in [3] and [9], we define the main threats to validity of our approach in the following sections. It is worthy noting that no internal validity was identified.

5.1 Conclusion validity

During our analysis, we decided to use a decision tree algorithm to investigate the features that had the greater impact in the prediction of project aliveness. However, other analysis techniques were available, and could indicate a different result. The decision tree approach had an accuracy of above 80% when processing all features, which indicates that the results have a good margin of confidence. Another reason for choosing a decision tree algorithm is that it clearly reveals the most important features in project aliveness prediction making it easier to have insights and draw conclusions. Other approaches, with other analytical tools or machine learning algorithms, can also be performed in the future to strengthen our results.

When using the decision tree algorithm, we set the number of instances per node to be 2000. However, other values for this parameters can affect the results. To mitigate this threat, we tested the parameter with different values, and observed that the accuracy did not change significantly.

Because the Boa infrastructure has a limited set of project information, we were not able to analyze other metadata about the project that could be useful in measuring project aliveness. This includes project ratings, the relationship between developers (especially with the project founder), number of pull requests, and number of GitHub issues raised. Those may play an important role in project aliveness prediction, but could not be included in the study.

5.2 Construct validity

The concept of project aliveness can be defined in several ways, depending on the perspective of the research. A study may state that a project is alive when it has many developers interested in it, or when the project is successful, in which case a definition of project success is still required. We had a discussion about the precise meaning of the concept of project aliveness that we use in this study and how it differs from other similar concepts. We believe that the distinction was clear and objective enough to reduce the chances of misinterpretation during the study.

Our formal definition of project aliveness states that a project is alive when there is at least one commit after the cutoff date for observation. Although objective, this definition may incorrectly label some projects and affect the results (e.g. a project with scarce commit activity may be alive, but labeled as not alive). To address this concern, we used backtesting when generating features, so that a project is analyzed during many different time periods until its creation.

5.3 External validity

As stated in Section 3.3, the Boa infrastructure has historical commit information for Java projects only. This forced us to limit our investigation of projects containing that language. Although the number of projects studied is high enough to provide significant results (68,299 projects), projects written in different languages may show different commit activity patterns.

The most recent dataset available in Boa contained projects from the GitHub repository. Although GitHub may be the most popular repository today and may include most of the active open-source projects, there are projects are stored in different repositories (e.g. SourceForge, Google Code). This prevents us from completely generalizing our results to any open-source project, and opens new research opportunities in the future.

In our study, it is important that we analyze open-source projects coming from different domains, to assert that our results are not specific to a particular one. We believe that our dataset is diverse enough to generalize the result and mitigate this concern.

6. CONCLUSION

When choosing open-source projects to work with, either while contributing or using its source code, developers would like to know whether a particular project will be alive in the future, so they can know that their commits will last and impact other developers, or that the project's code is reliable enough to be reused. Therefore, it is worthy to study the factors that impacts open-source project aliveness, and have a list of the features developers should inspect before choosing a project.

In our study, we analyzed GitHub open-source projects available in the Boa's dataset. After filtering, we studied data from 68,299 open-source Java projects. The study was divided into three research questions that investigate the commit behavior in the project, the impact of its founder, and the presence of documentation files. By using Weka's J48 decision tree algorithm, we found that features measuring the number of days since the last commit has a better predictive power than every other feature in the prediction of open-source project aliveness, with 80.72% accuracy. The second most significant type of feature is the density of the commits and the related feature, the largest number of days between two consecutive commits.

When analyzing the impact of the project founder on project aliveness, we found that when the percentage of the founder commits is low, the project is more likely to remain alive in the future. This may be the result of having a community of developers around the project with clear roles, and delegated responsibilities, contributing to a better maintenance of the project and its aliveness in the future.

Documentation files may help new developers familiarize themselves with a project and increase the chances of committing. When analyzing the presence of documentation files in open-source projects, we found that the majority of these files are named "readme", followed by any extension. Our results show that the presence of README files in projects does not indicate that a project is likely to remain alive in the future. We believe that the results were affected by the GitHub's README file generation feature when creating new projects. However, when inspecting documentation file with other names, we found that their presence correlates with future project aliveness. In fact, projects having "contributing.md" files are 50% more likely to be alive in the future.

We expect that these results help developers in their choice of open-source project in GitHub, when looking for projects to work with. Additionally, future studies, with more recent data, can be done to increase the confidence of our results.

7. REFERENCES

[1] N. Carvalho, A. Simões, and J. Almeida. Dmoss: Open source software documentation assessment. *Computer Science and Information Systems*, 11(4):1191–1208, 2014.

[2] R. Dyer, H. Nguyen, H. Rajan, and T. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. pages 422–431, 2013.

[3] R. Feldt and A. Magazinius. Validity threats in empirical software engineering research - an initial survey. pages 374–379, 2010.

[4] A. Ghapanchi. Investigating the interrelationships among success measures of open source software projects. *Journal of Organizational Computing and Electronic Commerce*, 25(1):28–46, 2015.

[5] A. Ghapanchi, A. Aurum, and G. Low. A taxonomy for measuring the success of open source software projects. *First Monday*, 16(8), 2011.

[6] A. Guimarães, H. Korn, N. Shin, and A. Eisner. The life cycle of open source software development communities. *Journal of Electronic Commerce Research*, 14(2):167–182, 2013.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[8] C. Kolassa, D. Riehle, and M. Salim. The empirical commit frequency distribution of open source projects. 2013.

[9] D. E. Perry, A. A. Porter, and L. G. Votta. Empirical studies of software engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 345–355, New York, NY, USA, 2000. ACM.

[10] N. Radtke, M. Janssen, and J. Collofello. What makes free/libre open source software (floss) projects successful? an agent-based model of floss projects. *International Journal of Open Source Software and Processes*, 1(2):1–13, 2009.

[11] A. Rahardjo Emanuel. Statistical analysis of popular open source software projects and their communities. 2015.

[12] U. Raja and M. Tretter. Defining and evaluating a measure of open source project survivability. *IEEE Transactions on Software Engineering*, 38(1):163–174, 2012.

[13] I. Samoladas, L. Angelis, and I. Stamelos. Survival analysis on the duration of open source projects. *Information and Software Technology*, 52(9):902–922, 2010.

[14] I. Steinmacher, M. Graciotto Silva, M. Gerosa, and D. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85, 2015.

[15] C. Subramaniam, R. Sen, and M. Nelson. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46(2):576–585, 2009.

[16] J. Tsay, L. Dabbish, and J. Herbsleb. Social media and success in open source projects. pages 223–226, 2012.

[17] S. Weber and J. Luo. What makes an open source code popular on git hub? volume 2015-January, pages 851–855, 2015.

[18] J. Wu, K. Goh, and Q. Tang. Investigating success of open source software projects: A social network perspective. 2007.

[19] J. Wu and Q. Tang. Analysis of survival of open source projects: A social network perspective. 2007.

[20] X. Yang, D. Hu, and D. Robert. How microblogging networks affect project success of open source software development. pages 3178–3186, 2013.