TRACKING THE SUCCESS OF OPEN SOURCE PROJECTS

Submitted by:

Sarah O'Connell BSc, MSc

Supervisor:

Aisling O’Driscoll

MSc. Cloud Computing

December, 2013

ABSTRACT

As the adoption of Open Source Software by commercial entities continues to grow, it is becoming increasingly important to be able to measure the current health of an open source project. Utilizing machine learning techniques, this thesis aims to define a classification model to predict Open Source Software success. By leveraging related work, existing metrics of success will be aggregated and applied along with newly defined metrics which fit the chosen domain. GitHub, the leading hosting platform for open source software, will be used as the data source domain to identify these key success indicators. In this way, a classifier of open source software success will be developed as a project's current state of development is measured. The first goal of this thesis is to identify which projects can be considered successful, and what features contribute to a higher success ranking. The second goal is to develop a supervised prediction model that will ascertain the future state of a project based on the first year's development. This model can be used to aid the early adoption of emerging technologies which often proves to be a differentiating factor among competing products.

ACKNOWLEDGEMENTS

I would like thank to my supervisor, Aisling O'Driscoll, whose encouragement, guidance and support contributed greatly to this thesis. I am also grateful to the lecturers who have taught me over the past eighteen months.

Lastly, I offer my thanks and appreciation to all of those who supported me in any respect during the completion of the project.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS & DEFINITIONS

BSD	Berkeley Software Distribution
FLOSS	Free/Libre and Open-Source Software
FSF	Free Software Foundation
GitHub	GitHub is a web-based hosting service for software development projects that use the Git revision control system.
GNU	GNU is a Unix-like operating system that is free software
IaaS	Infrastructure as a Service
IDE	Integrated development environment
MIT	Massachusetts Institute of Technology
OSBC	Open Source Business Conference
OSS	Open Source Software
R	Language and environment for statistical computing
RapidMiner	Application for Machine Learning
Repository	A project hosted on GitHub
RStudio	User interface for programming in R
SaaS	Software as a Service
SPSS	IBM Statistics Analysis Software
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

Over the past number of years, Open Source Software (OSS) has become a dominant force in the IT industry. A 2012 survey conducted by The Open Source Leadership Panel at OSBC [1] highlights the benefits of OSS as freedom from vendor lock-in, lower costs and higher quality; with 56% of respondents claiming that more than half of software products adopted over the next five years will be open source.

Driving this growth are emerging technology segments such as Software-as-a-Service (SaaS), private cloud, public cloud, and mobile. As Cloud Service Providers begin to develop services on top of available open source cloud platforms it becomes increasingly important [2] to track the various competing technologies and platforms.

There are many examples of successful open source projects available. Large well-established projects include Linux, Apache, MySQL, OpenStack & Hadoop while smaller projects trending today include Bootstrap, Rails, Pidgin and HomeBrew. However most free software projects fail, and although a failed project is difficult to define, Karl Fogel, author of "Producing Open Source Software", estimates the failure rate is as high as 90-95% [3]. Although free software projects might fail for the same reasons as proprietary projects, Fogel states that, more often, the free software project fails because the developers do not properly appreciate the unique problems associated with open source development. Given that project failure is difficult to define, the ability to assess the health of an open source project is growing in significance.

Although there is a wealth of literature on the topic of measuring traditional commercial software (or closed system) success (see Chapter 2), there is a lack of literature which deals specifically with open source software. The recent rise in the number open source projects and their growing importance to commercial enterprises has meant that researchers are now beginning to look at open source projects and analyse their productivity and the factors of their success.

This thesis aims to provide a method of measuring the health of a project using a current snapshot of development information. However, to assess new and emerging technologies this thesis also aims to provide a method of prediction based on a project's first year of development.

1.1 Problem Statement

In this thesis, the discriminative features of open source projects will be analysed to build a model that will help predict whether a project will be successful or not. The work will attempt to define a number of key indicators of success derived from software developer activity. Firstly, classification based on the current snapshot of a project's development data will be performed with the aim of labelling projects as successful or failed. This work will inform the development of a ranking algorithm to measure the success of a project at a point in time. Secondly, using the labelled sample data set, historical development data is retrieved for each project with the aim of predicting the future success or otherwise of the project at quarterly intervals within the first year of development.

By leveraging prior work, in the area of software project success measurement, this thesis will collate existing metrics of success and provide new metrics as appropriate. The ultimate goal is prediction of open source software success and will employ supervised learning techniques to build a prediction model.

As such, this thesis will contribute, to the current state of the art, the following goals:

- Firstly, a reasoned method of classifying and ranking open source projects through new and existing success metrics. This will assist interested parties in measuring the current health of an open source project.

- Secondly, a supervised prediction model that will ascertain the future state of a project based on the first year of development. This will assist early adopters to make more informed choices when adopting an open source project.

1.2 Thesis Outline

The rest of this thesis is organised as follows:

Chapter 2 outlines the background to the research proposed in this thesis and presents the current literature available for quantifying success of open source software and how this can be measured. This chapter also introduces machine learning and why it was chosen to help predict the success or failure of a project. **Chapter 3** examines the literature proposed in Chapter 2, and collates existing measures of success, as well as defining a new set of metrics specific to projects hosted in GitHub, the chosen domain. **Chapter 4** presents the implementation and data mining aspect associated with this project. Utilizing the findings from Chapter 3, a labelled sample set of projects is generated. **Chapter 5** introduces the supervised learning techniques performed through the RapidMiner modeller application. The SVM algorithm is chosen to aid in the ranking of projects, and the prediction model used to determine project success, based on the first 12 months of development data. **Chapter 6** provides a conclusion to this thesis, along with an overview of possible further development opportunities for this research.

CHAPTER 2

BACKGROUND RESEARCH

This chapter is organized as follows: section 2.1 provides an introduction into the origins of open source software and code hosting platforms. Section 2.2 describes the GitHub code hosting platform which has been chosen as the domain space for this research. Section 2.3 discusses the topic of open source software success and provides an overview of the current literature available. Finally, section 2.4 describes the machine learning techniques utilized in this thesis to achieve its research goals, as well as a review of literature with similar goals to this work.

2.1 OPEN SOURCE SOFTWARE

Open Source Software (OSS) is defined as software that makes its codebase publicly available for anyone to use or modify. Well known open source products include Mozilla FireFox, Android and OpenOffice.org. Distribution & modification rights are granted through the use of licensing agreements, some of which are more permissive than others, such as Apache License, BSD License, GNU General Public License or MIT License. Whether a license can be labelled as open source is determined by the "*Open Source Definition*" a document published by the Open Source Initiative [4].

Open Source has its origins in the common practise of software sharing in the 1970s. Manufacturers of early computers encouraged the distribution of software updates & patches by its customers as profits were generated from the hardware produced not the software it ran. Richard Stallman, a programmer with MIT in the 1970s and early 1980s, described the hacking culture that developed:

> *"We did not call our software "free software", because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program."* [5].

In the 1980s, as the industry matured, the importance of software grew which resulted in manufacturers enforcing copyrights on their code. In response to the growth of proprietary software, Stallman resigned from MIT and started the GNU Project and the Free Software Foundation (FSF). Over the next decade, more high-profile free projects were developed (e.g. Linux, Apache HTTP Server, MySQL) and attracted the attention of the commercial software industry. However, by the late 1990s it was felt that a new label was required to counteract the ideological and confrontational connotations of the term "*free software*". As such, the label "*Open Source*" was adopted [3].

The Open Source Initiative is an organisation, founded in 1998 by Bruce Perens and Eric S. Raymond, dedicated to promoting open source software. The "*Open Source Definition*" is originally based on the Debian Free Software Guidelines written by Perens in 1997 [4]. Raymond (president of the Open Source Initiative, until 2005) authored the well-known book "*The Cathedral and the Bazaar*", [6], in which he introduces two development models, the Cathedral model and Bazaar model. The Cathedral model represents software that publishes its source code along with each release. The model restricts development to an exclusive group of core programmers. Raymond proposes the Bazaar model as an alternative to the closed Cathedral system. He suggests that developing a project transparently results in better quality code, so that the more accessible the codebase is, for public testing and scrutiny, the faster bugs are discovered. It is this bazaar model that forms the basis for open source projects developed today using online code repositories.

2.2 GitHub

Online hosting platforms for open source projects are now easily accessible to developers who support the open source movement and want to contribute. They offer a centralized location for developers to control and manage their projects. Among the most popular hosting platforms are SourceForge, GitHub, Google Code and BitBucket. GitHub is the chosen code hosting platform for the work conducted in this thesis, as it is the most widely used hosting platform at the time of writing.

GitHub Inc. was founded in 2008 by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett and is based in San Francisco, California [7]. Compared with SourceForge's launch back in 1999, GitHub is a relative new comer to the code hosting market. However, as of May 2011, GitHub was listed as the most popular open source code repository site [8]. According to Brian Doll, GitHub's Vice President of Marketing, GitHub has over 2 million people working on over 4 million repositories [9]. This results in an average of 125,970 events generated on GitHub every day. This information is collected and archived by the GitHub Archive project. In May 2012, the data collected in the GitHub Archive was made publicly available, providing records of many forms of GitHub activity such as commits, comments, forking and follows. This archive is available for analysis via Google BigQuery adding to the analytical opportunities available.

Providing insight into the wealth of information available from GitHub, Brian Doll & Ilya Grigorik [9], a developer advocate at Google and open source devotee, analysed millions of GitHub commits to help answer many questions regarding the anatomy of the GitHub ecosystem such as “What were the hot new projects today?”, “Did anyone commit something interesting or controversial?” and “What are the emerging projects, or languages?”. To answer these questions and more, Doll & Grigorik issued a “GitHub Data Challenge” involving the GitHub archive and

Google's BigQuery. Entries included a project analysing the emotional impact of programming languages, language association and an analysis of OSS development using DNA sequencing tools. The variety of projects entered demonstrates the analytical opportunities available for this dataset.

2.3 Defining Open Source Project Success

Although there is a wealth of literature on the topic of software or information system success, ([10], [11], [12], [13]) there is a lack of literature which deals specifically with open source software. The recent rise in the number open source projects and their growing importance to commercial enterprises has meant that researchers are now beginning to look at open source projects and analyse their productivity and the factors of their success.

One such group of researchers, Kevin Crowston *(Syracuse University)*, James Howison *(Syracuse University)* and Hala Annabi *(University of Washington)* have produced a number of papers dealing with the success of open source software [14], [15], [16]. In particular, in their paper entitled, "Information Systems Success in Free and Open Source Software Development: Theory and Measures", [14], the authors investigate information system success in terms of Free/Libre and Open Source Software (FLOSS). They review existing models of information system success and aim to identify processes to improve or enable distributed open source team performance. Leveraging work by DeLone and McLean, [10], they developed a model of FLOSS success.

The original model developed by DeLone and McLean is a well-known and accepted model for evaluating information system success and focuses on success indicators such as system quality, use, user satisfaction and organizational impacts. To improve open source software processes, firstly, Crowston et al. identify factors that influence the success of a project. To this end, they look at the appropriateness of factors of success from traditional software systems (DeLone and McLean) and supplement these factors with new measures. Using data from SourceForge they measure a project's effectiveness with respect to team building, the project's bug report responsive rate and the project's popularity. Crowston proposes that measuring OSS success is more difficult than traditional systems as the goal and intended user base of OSS is often subjective and difficult to define. It is also suggested that the literature focusses on measuring aspects of the development environment such as project activity, since for OSS the development environment is transparent and accessible while the production or "in-use environment" is not. This is the opposite phenomenon to traditional systems which have a hidden codebase and a well-defined in-use environment.

Lee et al. [17] seek an answer to two research questions: (1) what are the factors determining OSS success? (2) How do these factors influence each other? This work attempts to answer these questions by reviewing existing literature and by conducting surveys targeting users and developers of OSS over the Internet. Lee et al. developed a model based on the following determinants of success: software quality, community service quality, user satisfaction, OSS use and individual net benefits.

Ghapanchi et al. [18], aims to provide an overview of the current state-of-the-art research into OSS success and looks at different approaches to measure OSS success. By searching academic databases using specific keywords, Ghapanchi builds a measurement taxonomy of OSS success based on 45 directly related publications. Their analysis of the literature highlights two categories: metrics based on product success and metrics based on project success. By examining each category, they found that studies on project success can be further broken down into four success areas: project activity, project efficiency, project effectiveness, and project performance. Meanwhile, studies based on product success are mainly focused on product quality. This thesis will look at metrics based on project success, following on from research initially carried out by Crowston.

In his 2007 MSc thesis, Wang [19] presents research which is similar to the second goal of this thesis, based on machine learning techniques and data drawn from the SourceForge code repository. Expanding on Crowston's work, Wang lists several measures used to determine project success and formulates a success predictor based on the K-Means clustering method.

Wang attempts to validate the metrics introduced by Crowston and formulates a number of his own to predict the success of a project within 9 months of the project's creation date. Wang focusses on validating the metrics by performing unsupervised learning on a sample set of projects retrieved from SourceForge. In contrast to Wang, this thesis uses supervised learning techniques to develop a prediction model, and the data is mined from GitHub, the fastest growing open source community of the day.

Both Wang and Crowston use SourceForge as the source of their data, while more recent research has begun to look at the increasingly popular GitHub platform code repository. In their 2013 paper, "Performance and Participation in Open Source Software on GitHub" [20], McDonald et al. attempt to determine through interviews, with prominent GitHub developers, what factors have influenced the platform's rapid rise in popularity since its introduction in 2008. They conclude the key indicators of success to be contributor growth, community involvement and visible activity while code quality is at best a secondary metric. They also conclude that the GitHub platform itself is a major contributor to the success of the projects that it hosts. The low barriers to participation and the consistent and transparent nature of GitHub development, which is the same for all projects, provides a visible means for measuring community activity which encourages user participation. One interviewee states that *"...the tools that GitHub provides give the project better visibility for potential contributors, and the ability for them to file pull requests that can be reviewed by the team makes it easier to bring them onboard."*

This thesis will leverage this prior work by collating a number of relevant metrics defined by Crowston, Wang and McDonald and aims to introduce new metrics as appropriate to measure open source projects hosted on the GitHub platform.

2.4 Machine Learning

Machine learning is a branch of artificial intelligence and deals with pattern recognition and model building. In 1997, the computer scientist, Tom Mitchell authored one of the first textbooks in this field entitled "*Machine Learning*" [21]. In this seminal work, Mitchell describes machine learning as “*the study of computer algorithms that improve automatically through experience.”* Machine learning algorithms are data driven and their application is proven in a number of domains e.g. spam filtering, handwriting recognition, face recognition and recommendation systems. The core objective of a machine learning algorithm is often generalization. This is a measure of the algorithm's ability to accurately classify new input based on a generalized model that was learned based on other examples within a training set.

The two main categories of machine learning algorithms include supervised learning algorithms and unsupervised learning algorithms. Unsupervised learning algorithms (e.g. cluster analysis) look for patterns in data but have no examples of the required output. The objective is to discover a structure within the data using unlabelled examples. Supervised learning algorithms are trained using labelled examples and attempt to build a generalized model based on the labelled training data which can correctly classify the output of unseen inputs. This thesis aims to build a list of metrics which define project success, and will utilize these metrics to label a sample set of projects retrieved from GitHub. Thus, having generated a labelled set of projects, supervised machine learning techniques will be employed to create a classifier that can be used to generalize from new instances. The process of applying supervised machine learning to a real-world problem can be seen in Figure 1. This process will be followed in this thesis.

As seen in Figure 1, the first step in building a classifier is identification of the required data. The simplest method is that of “brute-force” which means every feature is measured in the hope that relevant features can be isolated. However this can lead to a high level of noise and missing feature values which will require significant pre-processing [22]. As such, it will be seen in this thesis that, having reviewed the literature, an informative set of features will be used to define the date set retrieved. Chapter 5 will address the selection of the machine learning algorithm (SVM) as well as the training and evaluation processes performed on the data set.

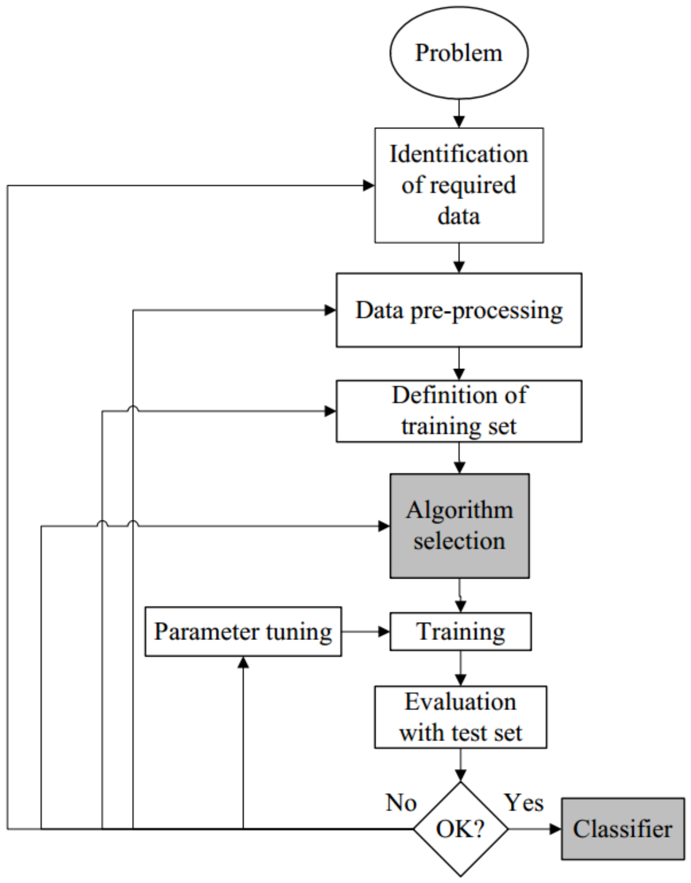


Figure 1: The process of supervised machine learning [22]

2.4.1 Support Vector Machines (SVM)

Although introduced in 1963 by Vapnik, [23], Support Vector Machines (SVM) are among the newest supervised learning techniques. In this thesis, the SVM technique will be used both in the identification of feature weights in the development of a ranking algorithm and to build a predictive model of project success. Support vector machines (SVMs) take a set of training examples as input with each example marked as belonging to one of two categories. SVM can analyse the labelled data set to find which features in the set contribute most to a particular category, providing the baseline for ranking. The SVM training algorithm can also build a model that predicts which category a new example belongs to. SVM is a powerful machine learning technique and performs well in a wide range of non-linear classification problems and regression analysis.

SVM works by constructing a model to represent each input as a point in space. These points are mapped according to the class they belong to. The aim is to separate both classes by a clear gap or decision boundary which is as wide as possible. Each new input is mapped onto the feature space and its class is predicted based on which side of the gap they are mapped to. The term hyperplane is often used to describe the gap or decision boundary produced by the SVM model and is constructed based on the area that is maximally further away from any data point. The distance between the hyperplane and the closest data point is called the margin. This results in a hyperplane defined by a small subset of data points that are on the margin called support vectors. Figure 2 visualizes the hyperplane, along with the margin and the support vectors that define it.

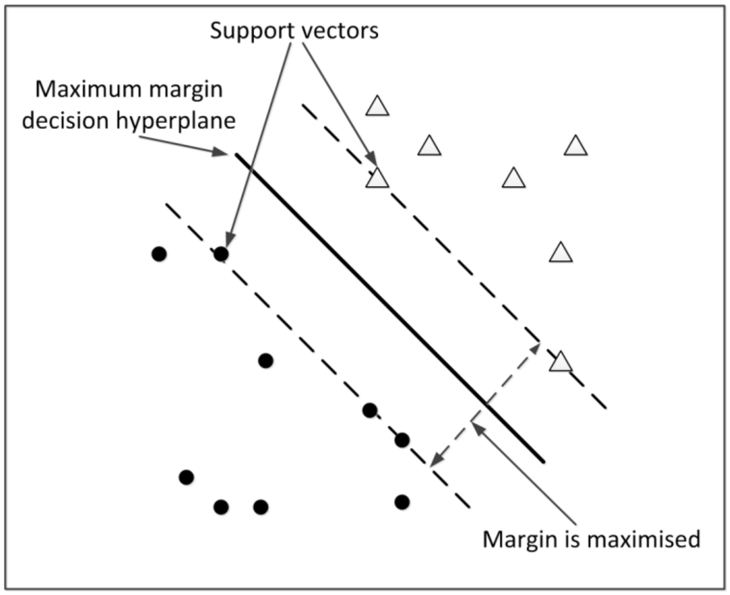


Figure 2: SVM Hyperplane Visualisation

2.4.2 MACHINE LEARNING IN THE LITERATURE

As stated previously, machine learning algorithms have been successfully applied to many problem domains. In this thesis, the following tasks are performed: attribute weighting, classification of known samples and prediction of unknown samples. Machine learning techniques, including the SVM algorithm, have proven to be well suited to these tasks. A review of various supervised machine learning algorithms is provided by Kotsiantis in his paper "Supervised Machine Learning: A Review of Classification Techniques" [22]. The SVM algorithm has been chosen as the supervised learning technique to support this work and will inform the machine learning literature reviewed.

In this thesis, open source projects will be classified as successful or failed; an example of a binary classification problem. Similarly, binary classification of email as spam and non-spam is a popular application of machine learning techniques. Druker et al. study the use of support vector machines for email spam identification and demonstrate good results for the algorithm [24]. According to Sculley et al. Support Vector Machines give state-of-the-art performance for text classification and their work seeks a resolution to the high performance cost of SVMs which has led to Bayesian methods being utilized for online spam filtering [25]. While Sculley experiments with the SVM algorithm to produce a relaxed version with improved performance, combining machine learning techniques e.g. SVM and *k*-nearest neighbour, is growing in popularity [26].

Attribute weighting is an important feature of this thesis as it helps to achieve the goal of ranking GitHub repositories. SVM calculates feature weights by using the coefficients of the hyperplane. Chang and Lin have studied feature ranking using weights from linear SVM models and their experiments have yielded good performance from the algorithm [27].

With respect to measuring the success of software projects, in his thesis, "Prediction of Success in Open Source Software Development" Wang utilizes the K-Mean clustering algorithm to group unlabelled sample projects into successful and unsuccessful clusters based on their degree of similarity. In this way he uses the unsupervised learning technique of clustering to predict the viability of a project. In this thesis, the supervised learning SVM algorithm will be used to predict open source project success and provide attribute weightings for the ranking algorithm.

2.4.3 RapidMiner

The freely available open source tool RapidMiner [28] was chosen as the platform for running the SVM algorithm. Other open-source and proprietary platforms were also assessed including IBM's SPSS [29], SAS [30] and RStudio IDE [31]. The proprietary platforms are SAS which provides business analytics software and IBM's SPSS which is a software suite for predictive analytics. Both applications are expensive to purchase and trial versions were investigated. While both tools are powerful it was quickly evident that the trial period would not be long enough to conduct this research. As such open source tools were deemed more suitable for this work and both RStudio IDE & RapidMiner were considered. RStudio IDE is a user interface for R [32], the free software environment for statistical computing and graphics. RStudio has a huge library of features available online, however, RapidMiner's intuitive “drag and drop” UI is easy to use and also has many highly-configurable algorithms and operations built-in. A rich community has built up around RapidMiner providing many tutorial resources which minimize the learning curve considerably.

2.5 Summary

In this chapter, a comprehensive review of the literature was provided and the key concepts behind this thesis were introduced. The next chapter will introduce the metrics used to determine project success and how they can be applied to GitHub projects.

CHAPTER 3

DEFINING SUCCESS FIRST

To achieve the goals as outlined in the problem statement in Chapter 1, a sample set of Open Source projects is required. As already stated, the popular online source repository, GitHub, will be used as the source of the sample set. For each repository sampled from GitHub, a large amount of descriptive data has been retrieved; this includes historical data from the project's first 12 months of development as well as a snapshot of the current state of the project. The current snapshot will help determine the current health of the project as of the date of retrieval. The historical data enables an examination of the project's first year of development to determine if success or failure of a project can be predicted during this critical time in the life cycle of an open source project, as further discussed in Chapters 4 and 5

The remainder of this chapter is organized as follows: Section 3.1 will introduce the metrics that determine project success and how they can be applied to GitHub projects. Section 3.2 lists additional metrics that are devised specific to the historical dataset

3.1 Applying Success Metrics to GitHub

From the literature review a list of metrics are collated to determine project success and are matched to the data available in GitHub projects. As discussed in Chapter 2, the initial goal of this research is to investigate the factors that determine open source project success. Crowston et al. introduce the importance of attracting developers to a project and suggest that a successful project should have at least 7 core developers committing code. The research also suggests that the number of issues logged against a project should be at least 100. Wang expands on Crowston's work and defines metrics based on project activity (e.g. the number commits and issues logged), the bug fixing speed of the project and also the concept of project and developer outdegree. Wang also suggests that frequent releases are a key component of open source software.

McDonald et al. specifically address the factors that have influenced the rapid growth in the popularity of GitHub as a platform for open source development. McDonald applied these success factors to the popularity of GitHub as a platform. In this thesis, these metrics will be applied to the success of projects themselves. Through interviews with prominent GitHub users, various metrics are presented such as the number of watchers, stars and forks a project has attained as well as how recent the last update has been. These metrics and how they relate to GitHub projects are presented Table 1. The GitHub REST API endpoints required to access this information are also listed.

Success Metric	Source	GitHub API	Detail
Developers	Crowston et al.	GET /repos/:owner/:repo/contributors	> 7 Core developers
Bugs reported	Crowston et al.	GET /repos/:owner/:repo/issues	> 100 Issues logged In GitHub Issues also include Pull Requests
Releases	Wang	GET /repos/:owner/:repo/tags	> 5 Releases

Success Metric	Source	GitHub API	Detail
Updated Date	Author Defined	GET /search/repositories?q=pushed	Project should be updated in the past month
Stars	Author Defined	GET /repos/:owner/:repo	Greater than average number of Stars
Watchers	Author Defined	GET /repos/:owner/:repo/subscribers	Greater than average number of Watchers
Forks	Author Defined	GET /repos/:owner/:repo	Greater than average number of Forks
Pull Requests	Author Defined	GET repos/:owner/:repo/pulls	Greater than average number of Pull Requests

Table 1: Success metrics (as defined in the literature and by the author)

Since the metrics defined by Crowston et al. and Wang have been discussed in detail in their respective works, [14], [19], the GitHub specific metrics, which are author defined and influenced by the research carried out by McDonald et al., [20], will be explained in more detail here to give an understanding of their significance when defining the success of a GitHub Project.

3.1.1 Updated Date

GitHub provides visible access to the last commit date on a project. According to one prominent JavaScript developer interviewed by McDonald, regular commit activity is a sign of the vitality of a project and the likely-hood of becoming involved in a project that has not been updated recently is very slim. The developer states *"If I haven't seen activity in – even like the last month – I get a little concerned."*

3.1.2 Stars

This is a metric used by GitHub to determine popular projects. A person will "*star*" a project to indicate their interest, and endorsement. This level of interaction is a passive indication of interest in that a user who stars a project is not required or expected to perform any subsequent action. This action is analogous to the well-known "*like*" or "*+1*" actions on the social networks Facebook & Google+ respectively.

3.1.3 Watchers

This metric is used to measure the interest of a user, it goes beyond starring a project since the user is signing up to receive notifications that occur when the project updates. This action is passive but displays a more active interest than starring since the user will now receive regular updates based on the activity of the project.

3.1.4 Forks

Another metric used by GitHub to determine popular projects. A "*fork*" is a copy of a GitHub repository. A person will fork a project in order to make changes to the codebase. These changes are often merged back into the original codebase as a pull request. This level of interaction is an active indication of interest as it requires the user to download the repository source and indicates that they may be willing to contribute back to the original project, sharing any improvements or changes they implement on their own copy.

McDonald states that the GitHub platform, through the use of forks, has "*lowered the barriers to participation*". Since any user registered on GitHub can fork any project, they do not require permission to clone a project's codebase and can make changes as they see fit. Subsequently, pull requests allow the user to submit patches back into original codebase.

3.1.5 Pull Requests

This metric is related to forks, once a user has forked a project and has performed some changes on the forked codebase they may wish to merge their changes back into the original project. This active metric requires more effort from the user than the other metrics listed above and also provides some insight into the quality and popularity of the original project. Donnie Berkholz, a data scientist with RedMonk, the developer focussed industry analyst firm, argues that using pull requests as a metric is better than forks [33], as it is an actual action, rather than a statement of intent. Berkholz states that "*that the true measure of an open-source project is the ability to merge code regularly — reviewing all of the open pull requests and acting on the ones that are ready to join the primary codebase.*" According to Berkholz, failing to accept contributions will only encourage competing projects to develop which could undermine and overtake the original project's community.

McDonald explains the effect of pull requests on the GitHub platform as introducing a level of democracy not previously seen on other platforms. Submitting a patch does not require permission, although merging the code requires a user with project commit privileges. Pull requests also allow comments which means that the process of adding code to the project is transparent. This has the effect of making it easier for new contributors to join the project as well as allowing users who do not wish to contribute code to voice an opinion on the future direction of the project.

In summary, the success metrics defined above can be used to determine the status of projects based on data retrieved from GitHub. This is the initial step in achieving the first goal outlined for thesis i.e. a reasoned method of classifying the current health of open source projects through new and existing success metrics, used to develop a ranking algorithm. This data is also used in the second goal of this thesis, where the success of a project is determined based on the first 12 months of a project's life cycle. The next section will detail additional metrics which can be applied to historical data to determine the success of a project at an earlier stage in its development cycle.

3.2 SUCCESS METRICS FOR THE HISTORICAL DATASET

As discussed in Chapter 2, GitHub provides an extensive API giving access to a vast array of development data and project information. Much of this data is available from a historical standpoint however at the time of writing some metrics defined in the previous section are unavailable. The total number of stars, forks and watchers is not available from a historical point of view. While it is possible to retrieve these statistics based on the current state of the project, it is not possible to ascertain the numbers of stars, forks or watchers a project had achieved based on a date in the past.

Success Metric	Source	GitHub API	Detail
User Reputation	Author Defined	GET /users/:user	Number of GitHub users that follow this user
Bug Fixing Speed	Wang	GET /repos/:owner/:repo/issues	Time, in days, from when an issue is opened to when it was closed.

Table 2: Historical success metrics

Thus, to predict the success of a project after 12 months of development a second set of metrics specific to this historical timeline is introduced, as outlined in Table 2. Prediction at 12 months can then be verified using the current snapshot analysis. The data examined includes developer activity (available from a historical standpoint) already discussed in the previous section. This includes the number of commits, issues and pull requests achieved by the project over quarterly periods. To compensate for the lack of information available regarding forks, stars and watchers, the following new metrics are introduced specific to the historical dataset.

3.2.1 User Reputation

In his thesis, [19], Wang introduces the concept of "*outdegree*" which is an idea borrowed from social network analysis and is a means to measure the interactions between actors. Wang found that a successful project usually has a group of core developers who contribute most to the project's development. He states that these developers are more likely to have higher outdegree scores. He aims to measure a project's popularity by calculating the outdegree scores of its developers based on how many other projects the developers are also associated with. His premise is based on the theory that a popular project will attract high quality developers and these developers are more likely to have an association with other projects i.e. attracting high-profile developers is considered an important factor when measuring the success of a project.

Leveraging this outdegree concept, this thesis will look at the reputation of a developer to help determine whether the project is attracting high-profile developers. Reputation is based on the number GitHub users following this developer and for each unique user who committed code, or opened an issue or pull request the project's average reputation is calculated.

3.2.2 Bug Fixing Speed

The number of bugs opened and fixed is a common way of measuring the quality of software projects. Crowston and Wang, both use bug fixing speed as a measure of a project's vitality. In this work, bug fixing speed is a measure of the average issue life span in days for an issue closed in a given quarter. This is calculated based on the opened date and closed date for all closed issues.

3.3 Summary

Sections 3.1 & 3.2 identified a number of project success metrics and applied them appropriately to GitHub projects. The next chapter will detail the strategy employed for mining the data required from GitHub and building a representative sample set of projects for analysis. Once the sample set of projects has been identified, two datasets will be produced. This first dataset is termed the “snapshot” dataset for the purposes of this thesis and contains the current vital statistics of a project. Utilizing the metrics defined in Section 3.1, this snapshot dataset will label its projects as successful or unsuccessful. In this way a supervised sample set is created defining project success at the most recent point in time. As such, the current health of a project can be classified i.e. the first goal of this thesis will be achieved. To achieve the second goal, outlined in Chapter 1, of predicting project success at quarterly intervals within the first year of development, a second dataset is retrieved for each of the labelled projects. This dataset, termed the “historical” dataset, contains a breakdown of development data for the first 12 months of a project's life cycle.

CHAPTER 4

BUILDING THE DATASET

The following sections explain the GitHub data retrieval process through the GitHub REST API and how the initial sample set of repositories for analysis were identified. Section 4.1 introduces the data retrieval process from GitHub, while section 4.2 explains how the initial sample set of repositories were chosen. Section 4.3 describes the database schema used to store the data retrieved. Finally, section 4.4 details the process of identifying and labelling successful and unsuccessful projects.

4.1 GITHUB REST API

As discussed previously in Chapter 2, GitHub is the chosen domain space for retrieving data about open source projects. GitHub provides an extensive REST API to access data about the projects it hosts. The original intention was to utilize the data available via the GitHub archive [34], a project which records and archives the GitHub timeline. However, after an analysis of the success metrics required to achieve the goals of this thesis, it was found that the archive did not store the information required. The archive is focussed on timeline events and this data might prove useful for further investigations, however it was deemed outside the scope of this thesis. As such, Google's BigQuery, which hosts the archive, was no longer a viable option for data retrieval. The GitHub REST APIs were found to be the most convenient source of the information required at this time. The GitHub REST APIs used to retrieve the required data for this project are listed in Table 3.

API	URL	Description
Search Repositories	GET /search/repositories	The Search API can find repositories based on various criteria e.g. Filters repositories based on language, times of creation, when they were last updated or based on the number of stars.
Contributors	GET /repos/:owner/:repo/contributors	Lists the contributors for a particular project.
Tags	GET /repos/:owner/:repo/tags	Lists the tags (releases) associated with a particular project.
Commits	GET /repos/:owner/:repo/commits	List the commits associated with a particular project.
Issues	GET /repos/:owner/:repo/issues	List the issues associated with a particular project (also includes pull requests).
Users	GET /users/:user	Retrieve information about this user.

Table 3: The GitHub REST API used in the data retrieval process.

4.2 Building an initial sample set of GitHub projects

To build up the sample set of projects the anatomy of various GitHub projects was investigated. To compare like with like the sample set was limited to projects with the same language so that the projects retrieved would have a similar codebase and developer community, resulting in a more meaningful comparison. JavaScript was chosen as it is the most popular language for projects on GitHub, and is among the most vibrant and active of developer communities at present.

Projects were also limited to those with at least 18 months development to allow for a sufficient timeline and prevent mislabelling newly created projects as failed. The final criteria imposed on the initial sample set limited the projects retrieved to those with greater than 100 stars. 100 stars were chosen as a cut-off point to eliminate small, unknown projects and those which are simply test-beds or hobby projects. Applying these restrictions reduced the dataset from a possible 500,000 JavaScript projects to a more manageable 1700.

In summary, the initial sample set consists of 1700 projects. The number of projects retrieved from GitHub was limited according to the following criteria:

1. JavaScript projects only
2. Created before January 2012
3. Has more than 100 Stars

4.3 Database Schema

Once the initial sample set of projects was identified from GitHub, a current snapshot of development data for each of the 1700 projects was retrieved so that the metrics previously defined can be applied. In this way, a labelled dataset of successful and unsuccessful projects can be identified. Historical data was also retrieved for each project. The database schema used to store the snapshot and historical data retrieved is discussed in the following subsections. An illustration of the database schema can be seen in Figure 3.

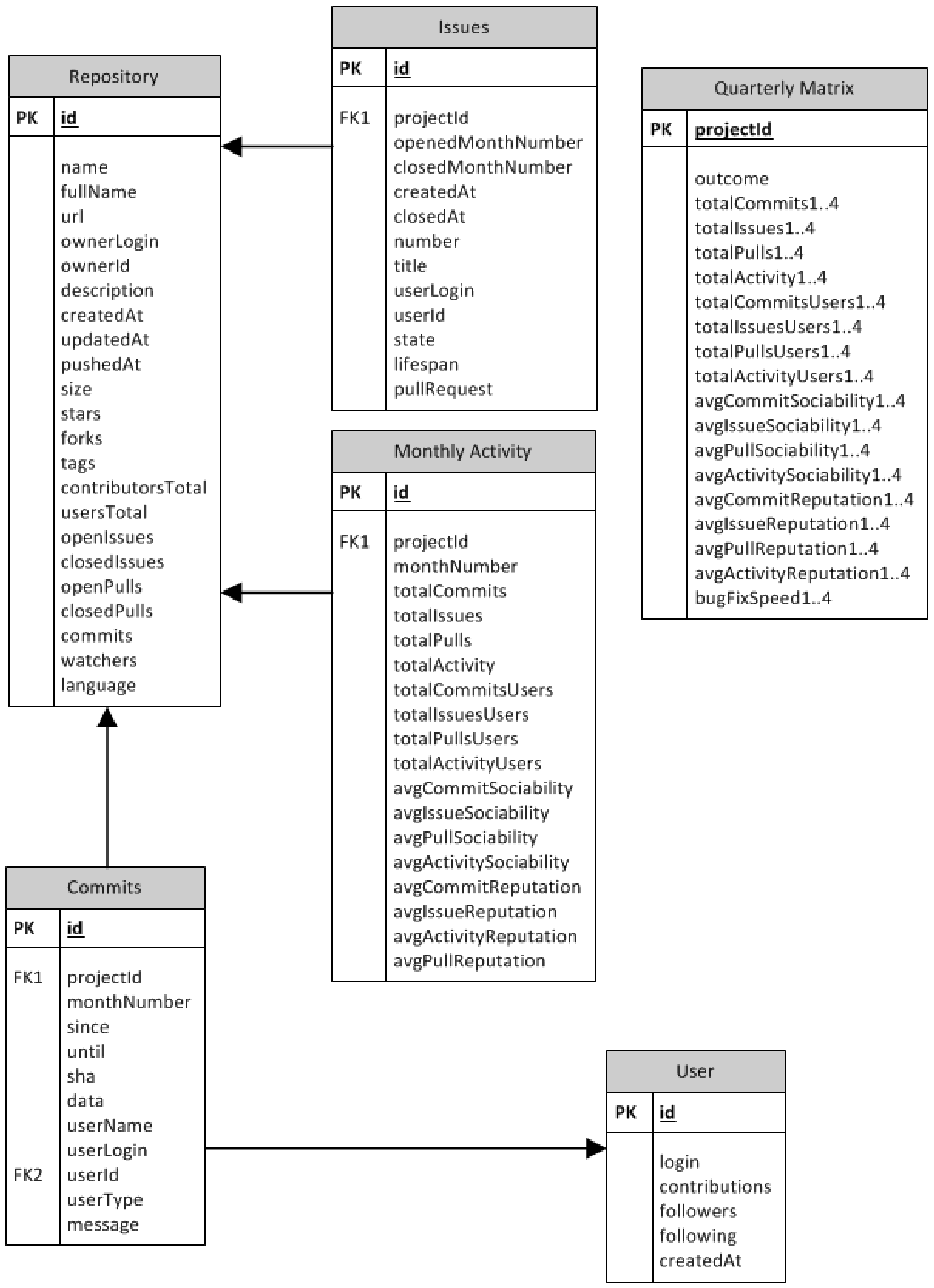


Figure 3: Database schema used to capture project information

4.3.1 Repositories Table

This table represents the current state of a project and stores the vital statistics of development activity. The primary key is the project id, as defined by GitHub, this is a unique identifier for each GitHub repository and is referenced by other tables in the schema. The table also stores the project name, description and its creation date. This table represents the current snapshot of development activity; as such it stores the total number of stars, forks, contributors, issues, commits, watchers and tags achieved by the project as of the time of retrieval. The data in this table is used to populate the snapshot dataset.

4.3.2 Commits Table

This table represents the commits that occurred for each project during the first 12 months of development. The table contains the foreign key, project id, which maps the commit back to its parent project in the Projects table. The table also contains information regarding the user who committed the code and the monthNumber in which the commit occurred.

4.3.3 Issues Table

GitHub represents bugs/defects and change requests as issues so pull requests are included in this table also. Pull requests are identified using a Boolean flag. This table represents the issues that were raised for each project during the first 12 months of development. The table contains the foreign key, project id, which maps the issue back to its parent project in the Projects table. The table also contains information regarding the user who opened the issue and the month number in which the issue was opened and the month number in which the issue was closed. The life span of the issue is also calculated and saved.

4.3.4 Users Table

For each user that committed code or raised a defect or pull request, information regarding that user was retrieved and stored in the users table. The primary key is the user id, as defined by GitHub, this is a unique identifier for each GitHub users and is referenced by the commits and issues tables. As discussed previously, the reputation of a GitHub user is an important factor to consider when measuring the success of a project. This information is stored in this table, the user's reputation is based on the number of GitHub users who are following this user.

4.3.5 Monthly Activity Table

To build the Quarterly Matrix table, information from the commits, issues and users tables is aggregated into monthly intervals and stored in the Monthly Activity table. As such, for each project identified in the sample set, this table consists of the total number of commits, issues and pull requests for each month. It also stores the total number of users who committed code, opened issues or created a pull request aggregated over the twelve months. Using this user data, reputation count for commits, issues and pull requests is calculated and stored for each month. Finally, using the issue data the average bug fixing speed for each issue is calculated and aggregated for each month.

4.3.6 Quarterly Matrix Table

To build the historical dataset used for analysis in the next chapter, the monthly project information is aggregated into quarterly segments. The information taken from the Monthly Activity table is collated into one single row of data for each project, with columns representing the required attributes for each quarter. For example, the bugFixingSpeed1 column represents the calculated bug fixing speed for the first quarter of this projects lifecycle. The columns bugFixingSpeed2, bugFixingSpeed3 and bugFixingSpeed4 represent the same data for the subsequent quarters. The outcome column is used to mark whether a project is a success or otherwise.

4.4 IDENTIFYING SUCCESSFUL PROJECTS

The first goal of this thesis is to identify which projects can be considered successful, and what attributes contribute most to a higher success ranking. For each of the 1700 projects identified in the sample set, a current snapshot of development statistics was retrieved. This snapshot of data represents the latest (at the time of retrieval) development statistics for a project and is stored in the Repositories table (see Section 4.3.1).

Project Id	2126244
Name	Bootstrap
Description	Sleek, intuitive, and powerful front-end framework for faster and easier web development.
Created At	29th July 2011
Last Update	8th September 2013
# Stars	57957
# Forks	20070
# Tags	22
# Contributors	416
# Open Issues	133
# Closed Issues	7575
# Commits	5861
# Watchers	4657

Table 4: Latest development statistics for Bootstrap (at the time of retrieval)

For illustration, Table 4 shows the snapshot development data for the Bootstrap project, a well-known GitHub project included in the sample set. Using this snapshot of development data, successful projects were identified as those with the criteria listed in Table 5.

Success Metric	Source	Detail
Updated Date	Author defined	Updated date > 2013-08
Bugs Reported	Crowston et al.	> 100 issues or pull requests logged
Pull Requests	Author defined	> 100 issues or pull requests logged
Developers	Crowston et al.	> 7 contributors
Releases	Wang	> 5 tags
Forks	Author defined	> 49 forks
Watchers	Author defined	> 18 watchers
Stars	Author defined	> 252 stars

Table 5: Criteria for a successful project

This resulted in a dataset of projects labelled as successful if the project was updated within the last month, had greater than 100 issues and/or pull requests, 7 contributors, 5 releases (tags), 49 forks, 18 watchers and 252 stars. This labelled dataset has a total of 236 successful projects, taken from the initial sample set and matching the successful criteria defined above.

4.5 IDENTIFYING UNSUCCESSFUL PROJECTS

The previous section describes the classification of successful projects, in the same way, unsuccessful projects are now defined and labelled. Since 236 successful projects have been identified, an equal number of failed projects are now added to the labelled dataset. Taken from the same initial sample set of 1700 projects, unsuccessful projects were identified using the same snapshot of development statistics and based on the criteria listed in Table 6.

Success Metric	Source	Detail
Updated Date	Author Defined	Updated date < 2013-02
Bugs reported	Crowston et al.	< 100 issues or pull requests logged
Pull Requests	Author defined	< 100 issues or pull requests logged
Developers	Crowston et al.	< 7 contributors
Releases	Wang	< 5 tags
Forks	Author defined	< 49 forks
Watchers	Author defined	< 18 watchers
Stars	Author defined	< 252 stars

Table 6: Criteria for an unsuccessful project

This resulted in a dataset of projects labelled as unsuccessful if the project was not updated within the last 6 months i.e. the project could be deemed "inactive" and has less than 100 issues and/or pull requests, 7 contributors, 5 releases (tags), 49 forks, 18 watchers and 252 stars. This labelled dataset has a total of 236 unsuccessful projects, taken from the initial sample set and matching the criteria defined above.

As such the "snapshot" dataset is defined using existing and newly defined metrics. This dataset can be utilized to identify projects which are successful and those which are not according to the latest development data available at the time of retrieval.

Figure 4 illustrates the composition of the labelled dataset, with 236 projects labelled successful and an equal number from the remainder labelled as unsuccessful. All other projects are unused in the dataset.

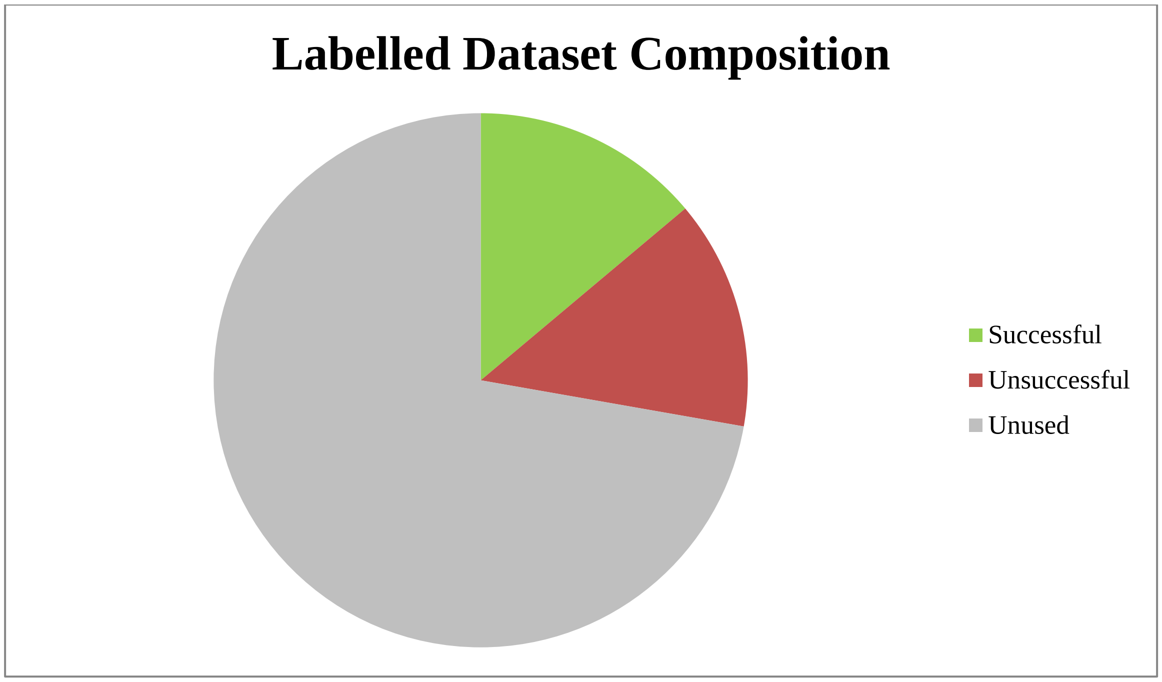


Figure 4: Breakdown of complete sample set

With a labelled sample set of 472 GitHub projects defined, the “historical” dataset was now retrieved, based on the metrics defined in Chapter 3. This data was limited to the first 12 months of the project’s life cycle and will be used to develop a predictive model in Chapter 5.

CHAPTER 5

RANKING AND PREDICTION WITH SVM

Chapter 3 looked at how various success metrics for open source projects are defined, while Chapter 4 discusses the data mining process for retrieving projects and development data from GitHub, the result of which is a labelled dataset. By leveraging the output of both chapters, machine learning techniques are now presented. This chapter will further the two primary goals of this thesis; to provide a meaningful ranking system according to the success metrics outlined in Chapter 3, and to predict the success of an open source project early in its lifecycle.

During the data mining phase, a sample set of 1700 GitHub repositories was retrieved. To devise a strategy for labelling these projects as successful or failed, the most current information available on each GitHub repository was retrieved (i.e. the "snapshot" dataset). In accordance with the current literature and the author defined metrics, this dataset was used to define the success of a project. Applying this definition to the GitHub repositories retrieved, the projects were labelled accordingly. 236 successful projects were identified and matched with an equal amount of failed projects, producing a sample set of 472 GitHub projects. This sample set is split 50/50 into projects labelled as *successful* and those labelled as *failed. S*uccessful projects match the criteria based on the aggregated set of metrics identified in Chapter 3. Meanwhile, failed projects do not meet the success metrics criteria and have been inactive for 6 months. Additionally, a second dataset was retrieved for each of the labelled projects. This dataset, termed the “historical” dataset, contains a breakdown of development data for the first 12 months of a project's life cycle, a period that is likely to be crucial in gaining community engagement.

Machine learning techniques will now be explored to validate both the “snapshot” and “historical” data sets constructed. The rest of this chapter is organised as follows: Section 5.1 introduces the Support Vector Machine (SVM) classification algorithm which will be used to identify weights for each of the attributes in the data set, and also to generate the predictive model.

Section 5.2 focusses on the first goal of this thesis, which is to build a ranking algorithm by providing different weights to each of the attributes in the dataset. Other topics of interest in this section include how the labelling strategy was validated, along with the method used to determine the attribute weights.

The development of the prediction model for project success is covered in Section 5.3. A historical dataset is created, before the process of running the algorithm is discussed, including how the datasets will be split into a training set and a smaller testing or scoring set. The section concludes with an analysis of the results of the prediction model and examines how the forecasting accuracy degrades as the historical data is limited from 12 to 6 and finally to just 3 months data**.**

5.1 CHOOSING A MACHINE LEARNING TECHNIQUE - SVM

Given the dual goals of developing a ranking algorithm, and building a classifier to predict GitHub project success, the next step involved investigating the most appropriate machine learning technique to apply to these problems.

Both the snapshot and historical datasets will be labelled so a number of supervised learning methods were investigated. As both datasets are relatively small, the attributes are numeric and the required output is a binary class label, i.e. *Success* or *Fail*, the Support Vector Machine (SVM) algorithm was chosen. SVM can also be used to deduce a ranking for each attribute, in the dataset, using the coefficients of the calculated hyperplane as feature weights.

As outlined in Chapter 2, SVM is a supervised learning technique introduced in 1963 by Vapnik. The original algorithm was a linear classifier, and was updated in 1992 by Boser, Guyon and Vapnik [35] to apply the "kernel trick" which allows non-linear input. SVM takes a set of inputs and from two possible outcomes (classes) predicts which class the input belongs to. As such, SVM is an example of a binary class classifier. A binary class problem consists of data records labelled with one of two possible labels corresponding to either class e.g. successful or failed projects

The effectiveness of the SVM algorithm relies on the configuration of the kernel chosen, and the soft margin parameter known as "C" or the Cost function. Configuration of the algorithm is discussed later in this chapter.

5.2 Developing a Ranking Algorithm

To verify the metrics identified as contributing to project success in Chapter 3 are accurate, a ranking algorithm was developed. The dataset labelled in Chapter 4 is first validated, and the snapshot data is used as input to help determine the weight of each of the dataset's attributes. This ranking algorithm adds confidence to the labelling process, and can also be used as a standalone process for the comparison of GitHub repositories.

5.2.1 Preparing the snapshot data

The "snapshot" dataset is taken and a matrix representing the most recent vital statistics (at the time of retrieval) for each labelled project is produced. Using the success metrics previously collected, the vector representation of a project is composed of the following form:

[

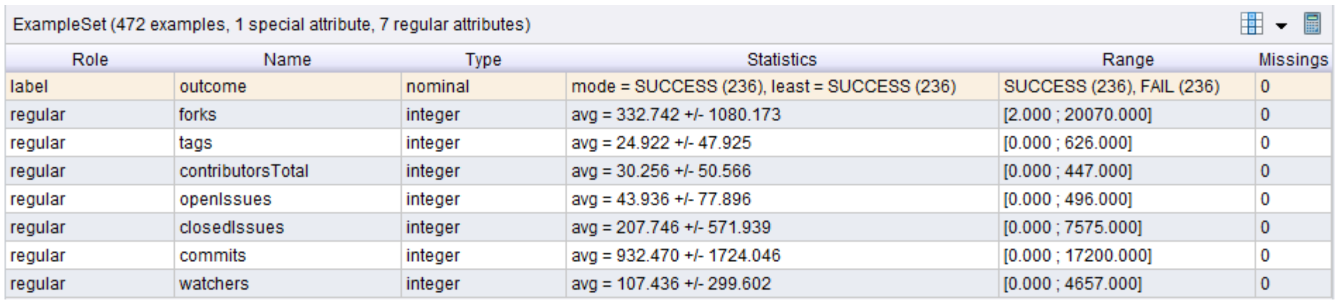
*M*1, *M*2, *M*3, *M*4, *M*5, *M*6, M7

]

where

- *M*1 is the total count of forks
- *M*2 is the total count of tags (releases)
- *M*3 is the total count of contributors
- *M*4 is the total count of open issues *(includes pull requests)*
- *M5* is the total count of closed issues *(includes pull requests)*
- *M6* is the total count of commits
- *M7* is the total count of watchers

This matrix of data is loaded into the machine learning application, RapidMiner as introduced in Chapter 2, and an analysis of the data is run. The analysis provides some statistics regarding the data including average values and the range over which the value is spread. It also confirms that no values are missing which could provide data discrepancy issues for the algorithm. A visual summary of the matrix can be seen in Figure 5.



ExampleSet (472 examples, 1 special attribute, 7 regular attributes)

Role	Name	Type	Statistics	Range	Missings
label	outcome	nominal	mode = SUCCESS (236), least = SUCCESS (236)	SUCCESS (236), FAIL (236)	0
regular	forks	integer	avg = 332.742 +/- 1080.173	[2.000 ; 20070.000]	0
regular	tags	integer	avg = 24.922 +/- 47.925	[0.000 ; 626.000]	0
regular	contributorsTotal	integer	avg = 30.256 +/- 50.566	[0.000 ; 447.000]	0
regular	openIssues	integer	avg = 43.936 +/- 77.896	[0.000 ; 496.000]	0
regular	closedIssues	integer	avg = 207.746 +/- 571.939	[0.000 ; 7575.000]	0
regular	commits	integer	avg = 932.470 +/- 1724.046	[0.000 ; 17200.000]	0
regular	watchers	integer	avg = 107.436 +/- 299.602	[0.000 ; 4657.000]	0

Figure 5: Snapshot dataset matrix visualized by RapidMiner

With the snapshot data set vector constructed and analysed for missing values through RapidMiner, the next task is to validate the labelling strategy. This is done prior to determining the weights that each attribute in the set contributes to the result, using SVM.

5.2.2 Validating the labelling strategy

Before executing the ranking algorithm, the labelling strategy must be validated to ensure a reasonable level of accuracy. The first labelling model that was processed dealt with the snapshot dataset from GitHub, which included the most recent data for each project. This model was run to see if the success metrics chosen were capable of predicting the success or failure of a project after a minimum period of 18 months, using the vital statistics of a project and the labelling strategy devised in accordance with the current literature.

To validate the model, both a training set and a testing set are required. To apply this methodology in the Rapid Miner environment the X-Validation operator is available. The X-Validation operator provides a cross validation technique for estimating the performance of a learning algorithm. In this way, the SVM algorithm can be trained and tested using the model with the cross validation technique applied. Figure 6 shows the validation process being applied in RapidMiner. The cross validation configured for this purpose is k-fold, where k = 10. This means the full dataset is utilized by algorithm by dividing it into 10 equal-sized subsets and running the algorithm 10 times. During each run, one subset is set aside for testing while the 9 other blocks are used for training. In turn, each subset is used for testing once and the result is the average performance from the 10 rounds.

Figure 6 also shows the “normalization” function available in RapidMiner and utilized by this process. Normalization is a preprocessing technique used to fit attribute values within a specific range. This technique is important when dealing with attributes of different units and scales. Statistical normalization also known as “z transformation” was chosen for this dataset.

RapidMiner also provides an internal Java implementation of the mySVM algorithm by Stefan Rueping [36]. This algorithm was used to run the SVM modelling process, as seen in Figure 7. The configuration values for the algorithm retained the default values. For example, the default value of 0 for the Cost function has been used in this instance. The Cost function variable of the SVM algorithm is related to variance so that a large value for C increases the variance and risks over fitting the problem. A small value for C risks underfitting the problem. Given the relatively small dataset being used here, the default setting of 0 was deemed most suitable as the dataset is more at risk of overfitting than underfitting. However, various types of kernel functions were used to determine the best fit e.g. linear, radial (RBF), and polynomial. If the data doesn't fit a linear model than different kernel types can be used to improve the algorithm's performance. Changing the kernel type, changes how the algorithm maps the data. Although some kernels are better suited to certain types of input e.g. RBF for images but not text [37], often the only way to choose the kernel that best fits the data is based on experimentation.

A high level view of the process defined in RapidMiner is available in Figures 6 and 7.

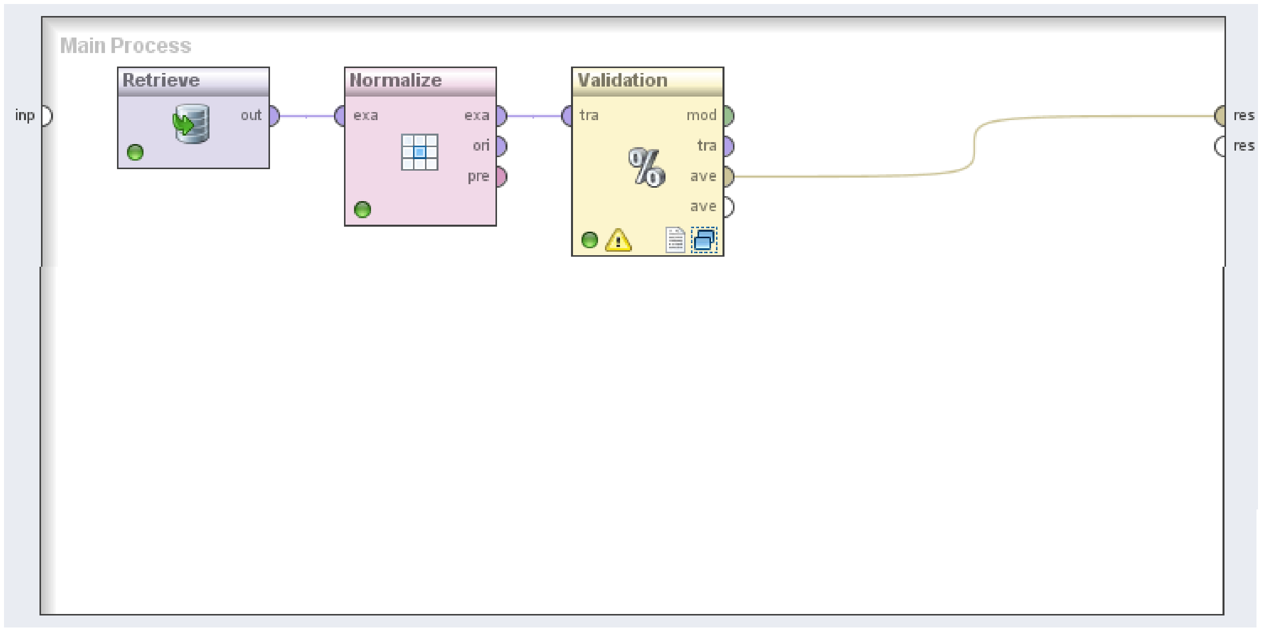


Figure 6: Illustration of SVM validation process

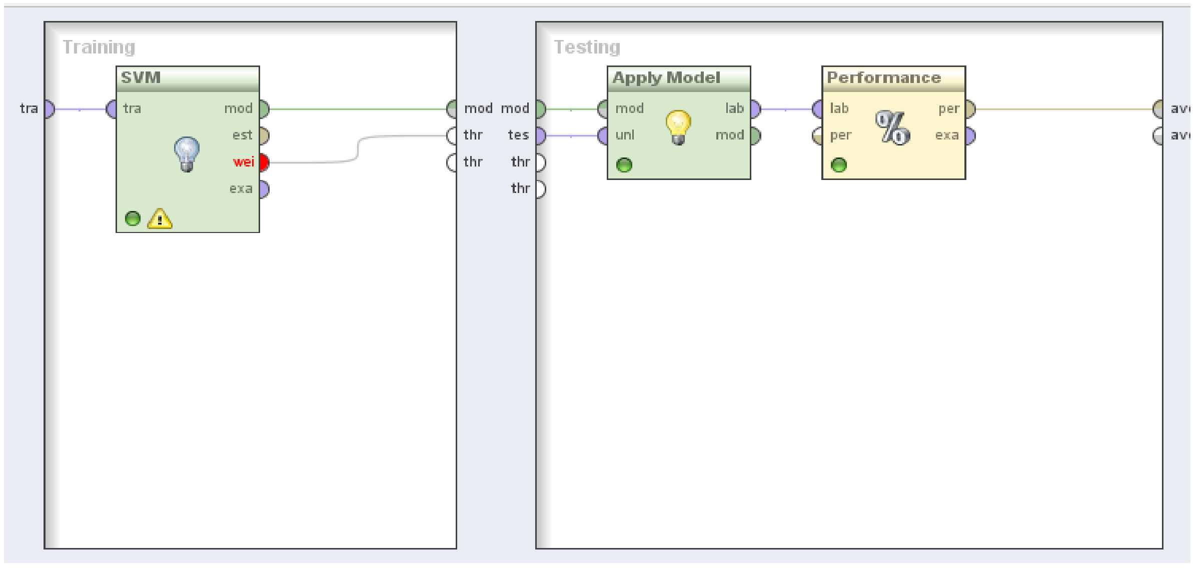


Figure 7: High level overview of SVM process

The initial test run used a linear kernel. An analysis of the kernel model produced by this run shows that 472 support vectors were used which indicates that the data is over-fitted. This might be expected on the relatively small dataset being used. Using this model, the k-fold cross validation technique was used to determine the accuracy achieved. As previously stated, choosing the correct kernel type is based on experimentation. Figure 8 shows the results of running the SVM algorithm with the different kernel types configured.

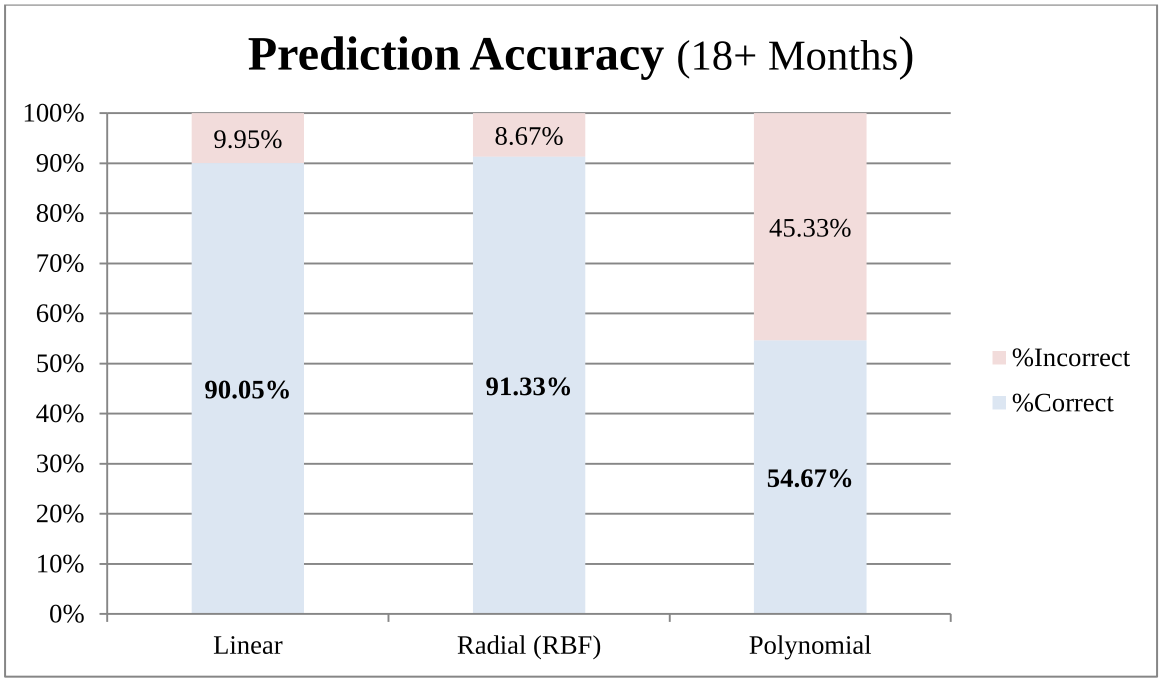


Figure 8: Accuracy comparison of the three SVM kernel types on the snapshot dataset (18+ months)

In this case, the radial kernel proved best with an accuracy of 91.33%. The radial basis function (RBF) is a popular kernel function used in SVM classification. This means that the success of a project can be determined using the model with over 91% accuracy. This high level of accuracy validates the snapshot dataset comprised of the success metrics defined in Chapter 3. Having validated the dataset, a ranking algorithm is now developed.

5.2.3 Determining Attribute Weights

With the snapshot data prepared and validated, the next step is to develop an algorithm that can rank the projects by finding the significant attributes and assigning a weight to each of these data points. In order to determine meaningful values for the weights of each attribute, RapidMiner's *Weight by SVM* operator was utilized.

Weight by SVM assigns a weight to each variable in the snapshot data set based on the coefficients of the hyperplane as calculated by the SVM. Note that the SVM kernel used for this operator in RapidMiner is linear as required by the algorithm. The resulting weight for each of the attributes is normalized, meaning that the value of each will be in the range of 0 to 1, attributes with higher values are considered the most relevant. The process was built up in RapidMiner providing the snapshot vector as input to the Weight by SVM operator, as illustrated in Figure 9.

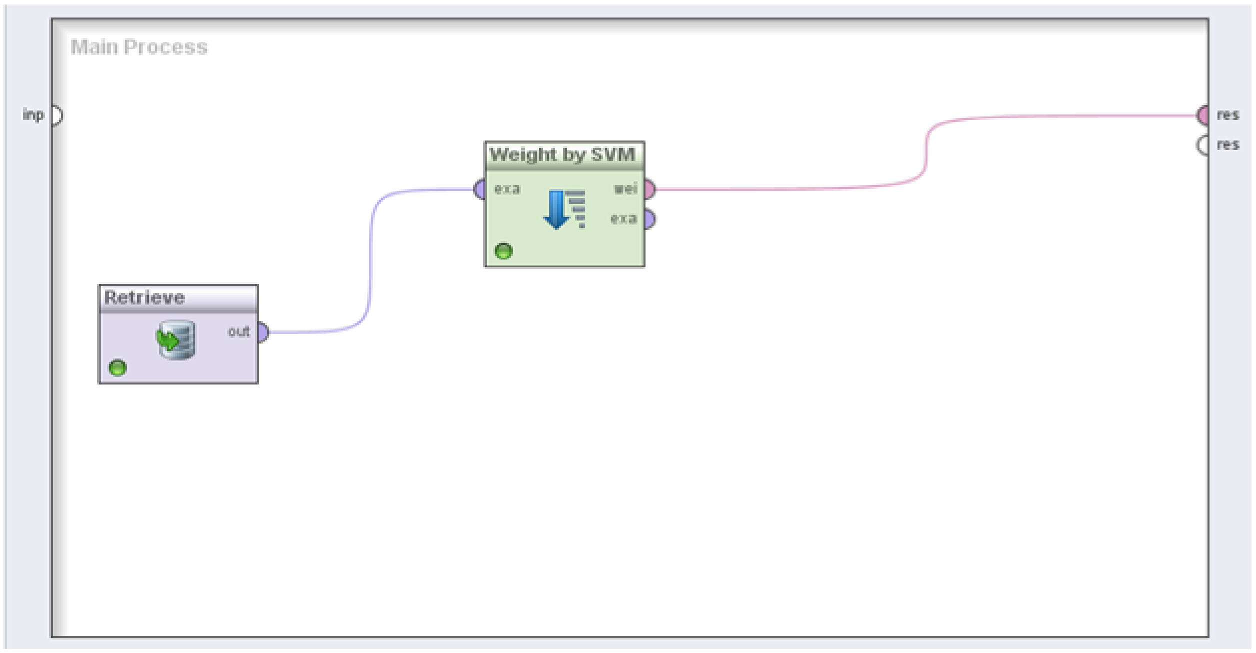


Figure 9: RapidMiner process used to determine attribute weights

This process resulted in a weight for each of the attributes that, when put together, determine a current success rating for any given GitHub project. These results are outlined in Table 7.

Attribute	**Weight**
contributorsTotal	1.0
Watchers	0.6625813716541344
closedIssues	0.24753591597490435
openIssues	0.13889798318760208
Forks	0.1281036567024759
Tags	0.04101072907549392
Commits	0.0

Table 7: Results of attribute weighting using SVM

From Table 7, it is clear that the number of contributors and watchers, the most social of the attributes, are the most relevant factors in the success of a project. A surprising outcome is that the amount of commits submitted to the codebase have made have no bearing on the success ranking for a GitHub repository.

Using the weights that were determined by the RapidMiner process, the following ranking algorithm can be devised.

```
score =      (project.getTags()* TAG_WEIGHT) +
             (project.getWatchers() * WATCHER_WEIGHT) +
             (project.getForks() * FORK_WEIGHT) +
             (project.getContributors() * CONTRIBUTOR_WEIGHT) +
             (project.getOpenIssues() * OPEN_ISSUE_WEIGHT)  +
             (project.getClosedIssues() * CLOSED_ISSUE_WEIGHT);
```

To check the validity of the attribute weights, a number of projects were selected at random from the sample set. Using GitHub’s own crude measure of popularity, the number of stars, there was an almost perfect correlation between the ranking results of the algorithm, and the ordering based on the number of stars. Table 8 provides a comparison of the ranking algorithm results with GitHubs fork and star counts.

Project	**Ranking Result**	**Stars**	**Forks**	**Algorithm Ranking**
Bootstrap	43544.25	58352	20070	1
Node.js	18286.75	24603	4746	2
Backbone	11387.82	15692	3212	3
Three.js	9759.31	12756	2844	4
Express	7775.93	10691	1747	5
Jasmine	3844.34	5502	693	6
EaselJS	2005.61	2779	659	7
audiolib.js	309.65	432	45	8
safari-json-formatter	213.62	305	36	9
Doodle-js	155.67	229	18	10
Formwizard	83.66	101	31	11
Jquery-serialize-object	73.47	101	25	12
Oauth2-server-node	70.63	101	13	13

Table 8: Comparison of ranking results with number of stars and forks

As illustrated in Table 8, the calculated ranking for each project is in line with the number of stars, a key metric used by GitHub to measure popularity. The ranking also roughly correlates with the number of forks, providing an angle on active participation. Thus, the results of this ranking calculation can be considered to be a good indicator of the current success of any GitHub repository

5.3 Developing a Predictive Model

The second stated goal of this thesis is to provide a mechanism to predict how successful a GitHub repository might be in the future. To achieve this goal, a dataset that provides information about each repository at set intervals in the past must be created. From there, a predictive model can be built which gives an insight into the potential of the project.

5.3.1 Preparing the historical dataset

The project's historical dataset was used to produce a matrix representing the first year of development for each labelled repository. Rather than consider each month in isolation, it was deemed sufficient to consider the aggregated data in 3 month segments. The development data, aggregated over each quarterly period, produces a vector representation with 40 attributes of the following form:

[

M11, M21, M31, M41, M51, M61, M71, M81, M91, M101,

M12, M22, M32, M42, M52, M62, M72, M82, M92, M102,

M13, M23, M33, M43, M53, M63, M73, M83, M93, M103,

M14, M24, M34, M44, M54, M64, M74, M84, M94, M104

]

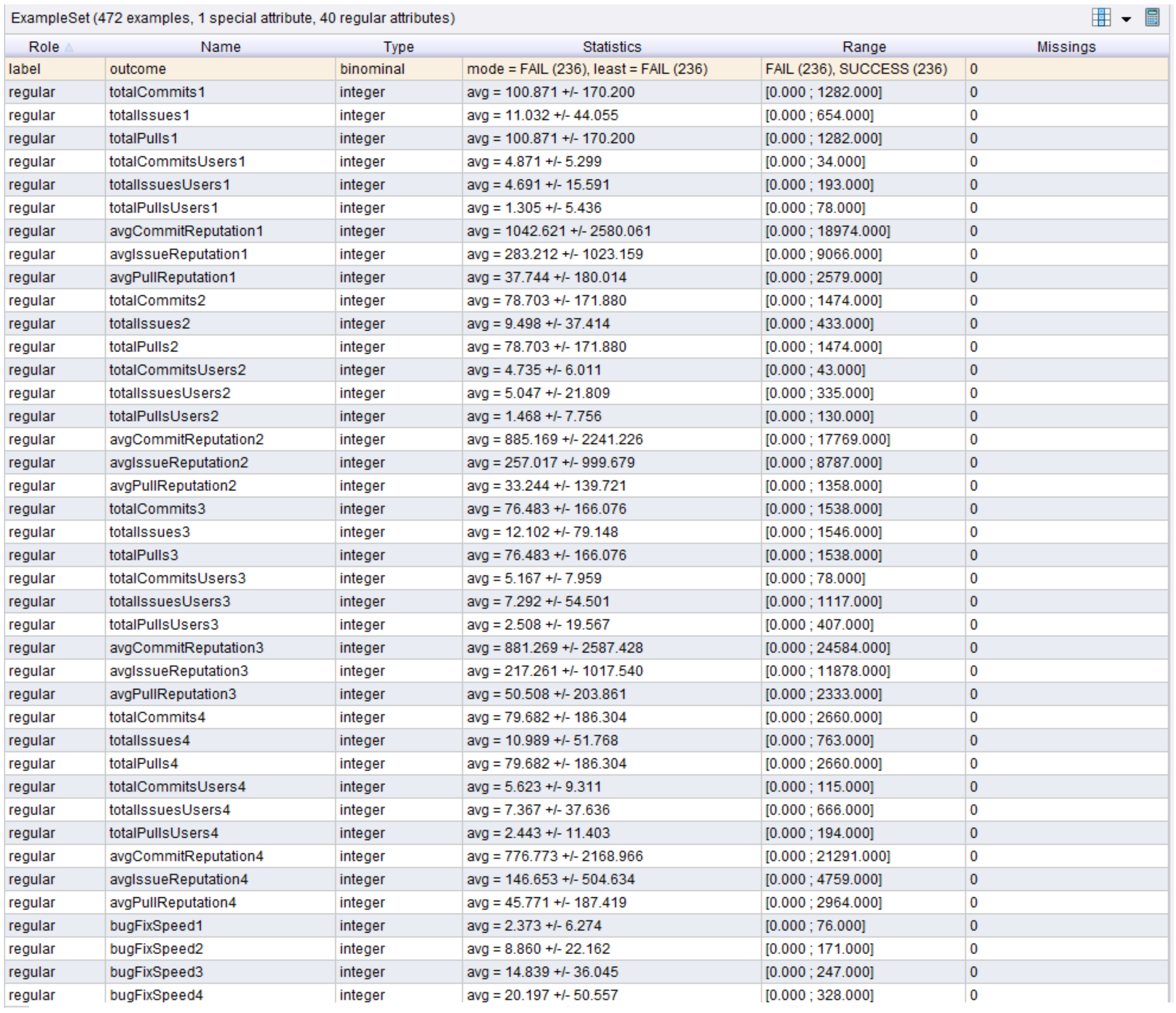
where

- *M*1 is the total count of commits over the quarter

    - e.g. M11 = total commits for first quarter

- *M2* is the total count of issues over the quarter
    - e.g. M21 = total issues for first quarter
- *M3* is the total count of pulls over the quarter
- *M4* is the total count of distinct commits users over the quarter
- *M5* is the total count of distinct issues users over the quarter
- *M6* is the total count of distinct pulls users over the quarter
- *M7* is the total count of distinct commits user reputation over the quarter
- *M8* is the total count of distinct issues users reputation over the quarter
- *M9* is the total count of distinct pulls users reputation over the quarter
- *M10* is the bug fixing speed for this quarter

As before, this matrix of data is also loaded into the machine learning application, RapidMiner, and an analysis of the data is run. The analysis provides the same statistics regarding the new dataset and again confirms that no values are missing in this dataset. A visual summary of the matrix can be seen in Figure 9.



ExampleSet (472 examples, 1 special attribute, 40 regular attributes)

Role	Name	Type	Statistics	Range	Missings
label	outcome	binominal	mode = FAIL (236), least = FAIL (236)	FAIL (236), SUCCESS (236)	0
regular	totalCommits1	integer	avg = 100.871 +/- 170.200	[0.000 ; 1282.000]	0
regular	totalIssues1	integer	avg = 11.032 +/- 44.055	[0.000 ; 654.000]	0
regular	totalPulls1	integer	avg = 100.871 +/- 170.200	[0.000 ; 1282.000]	0
regular	totalCommitsUsers1	integer	avg = 4.871 +/- 5.299	[0.000 ; 34.000]	0
regular	totalIssuesUsers1	integer	avg = 4.691 +/- 15.591	[0.000 ; 193.000]	0
regular	totalPullsUsers1	integer	avg = 1.305 +/- 5.436	[0.000 ; 78.000]	0
regular	avgCommitReputation1	integer	avg = 1042.621 +/- 2580.061	[0.000 ; 18974.000]	0
regular	avgIssueReputation1	integer	avg = 283.212 +/- 1023.159	[0.000 ; 9066.000]	0
regular	avgPullReputation1	integer	avg = 37.744 +/- 180.014	[0.000 ; 2579.000]	0
regular	totalCommits2	integer	avg = 78.703 +/- 171.880	[0.000 ; 1474.000]	0
regular	totalIssues2	integer	avg = 9.498 +/- 37.414	[0.000 ; 433.000]	0
regular	totalPulls2	integer	avg = 78.703 +/- 171.880	[0.000 ; 1474.000]	0
regular	totalCommitsUsers2	integer	avg = 4.735 +/- 6.011	[0.000 ; 43.000]	0
regular	totalIssuesUsers2	integer	avg = 5.047 +/- 21.809	[0.000 ; 335.000]	0
regular	totalPullsUsers2	integer	avg = 1.468 +/- 7.756	[0.000 ; 130.000]	0
regular	avgCommitReputation2	integer	avg = 885.169 +/- 2241.226	[0.000 ; 17769.000]	0
regular	avgIssueReputation2	integer	avg = 257.017 +/- 999.679	[0.000 ; 8787.000]	0
regular	avgPullReputation2	integer	avg = 33.244 +/- 139.721	[0.000 ; 1358.000]	0
regular	totalCommits3	integer	avg = 76.483 +/- 166.076	[0.000 ; 1538.000]	0
regular	totalIssues3	integer	avg = 12.102 +/- 79.148	[0.000 ; 1546.000]	0
regular	totalPulls3	integer	avg = 76.483 +/- 166.076	[0.000 ; 1538.000]	0
regular	totalCommitsUsers3	integer	avg = 5.167 +/- 7.959	[0.000 ; 78.000]	0
regular	totalIssuesUsers3	integer	avg = 7.292 +/- 54.501	[0.000 ; 1117.000]	0
regular	totalPullsUsers3	integer	avg = 2.508 +/- 19.567	[0.000 ; 407.000]	0
regular	avgCommitReputation3	integer	avg = 881.269 +/- 2587.428	[0.000 ; 24584.000]	0
regular	avgIssueReputation3	integer	avg = 217.261 +/- 1017.540	[0.000 ; 11878.000]	0
regular	avgPullReputation3	integer	avg = 50.508 +/- 203.861	[0.000 ; 2333.000]	0
regular	totalCommits4	integer	avg = 79.682 +/- 186.304	[0.000 ; 2660.000]	0
regular	totalIssues4	integer	avg = 10.989 +/- 51.768	[0.000 ; 763.000]	0
regular	totalPulls4	integer	avg = 79.682 +/- 186.304	[0.000 ; 2660.000]	0
regular	totalCommitsUsers4	integer	avg = 5.623 +/- 9.311	[0.000 ; 115.000]	0
regular	totalIssuesUsers4	integer	avg = 7.367 +/- 37.636	[0.000 ; 666.000]	0
regular	totalPullsUsers4	integer	avg = 2.443 +/- 11.403	[0.000 ; 194.000]	0
regular	avgCommitReputation4	integer	avg = 776.773 +/- 2168.966	[0.000 ; 21291.000]	0
regular	avgIssueReputation4	integer	avg = 146.653 +/- 504.634	[0.000 ; 4759.000]	0
regular	avgPullReputation4	integer	avg = 45.771 +/- 187.419	[0.000 ; 2964.000]	0
regular	bugFixSpeed1	integer	avg = 2.373 +/- 6.274	[0.000 ; 76.000]	0
regular	bugFixSpeed2	integer	avg = 8.860 +/- 22.162	[0.000 ; 171.000]	0
regular	bugFixSpeed3	integer	avg = 14.839 +/- 36.045	[0.000 ; 247.000]	0
regular	bugFixSpeed4	integer	avg = 20.197 +/- 50.557	[0.000 ; 328.000]	0

Figure 10: Snapshot dataset matrix visualized by RapidMiner

5.3.2 Analysing the historical dataset

The second prediction model dealt with the historical data from GitHub. This included the development data available for the first 12 months of a project. Using the same attribute weighting technique as seen in section 5.2.3, the significance of the different attributes of this dataset can be determined. In this case, the bug fixing speed over the latter quarters are among the most significant values. The weights assigned to each of the variables can be seen in Table 9.

Attribute	Weight
avgPullReputation3	1
totalCommitsUsers4	0.995183518368618
bugFixSpeed2	0.7502925041164207
bugFixSpeed4	0.7383699877441965
bugFixSpeed3	0.7368910883871183
avgCommitReputation2	0.7197776336367295
totalPullsUsers1	0.6749342368692747
totalPullsUsers4	0.6623261941505396
avgIssueReputation4	0.6455353047157337
totalCommitsUsers3	0.5899001827545685
totalIssuesUsers1	0.5545063395196742
avgPullReputation1	0.524467238554666
totalCommits1	0.4324414506993193
totalPulls1	0.4324414506993193
avgPullReputation2	0.43224180296922227
totalCommitsUsers1	0.4262206828761362
totalPullsUsers2	0.36816326198888777
totalCommits4	0.3333780615746107
totalPulls4	0.3333780615746107
totalIssuesUsers2	0.29018314493693803
totalCommitsUsers2	0.2559680982679775
totalPullsUsers3	0.25132482298876424
avgIssueReputation1	0.21184322345275805
totalIssues4	0.20027579781104113
totalCommits2	0.17934794045165975
totalPulls2	0.17934794045165975
totalCommits3	0.15062338292809663
totalPulls3	0.15062338292809663
avgCommitReputation4	0.14865753008298568
totalIssuesUsers4	0.14694189034358418
avgIssueReputation2	0.09202725472077015
totalIssues1	0.075762779494804
avgPullReputation4	0.06544488461852092
totalIssues2	0.06338452018309784
totalIssuesUsers3	0.05721658678895744
avgCommitReputation1	0.05042584658849341
bugFixSpeed1	0.044544816548964916
avgCommitReputation3	0.03212784133006459
avgIssueReputation3	0.016213043777070196
totalIssues3	0.0

Table 9: Prediction model attribute weights

The SVM prediction model is now run on the 12 month historical dataset and the k-fold cross validation technique was again used to determine the accuracy achieved. As noted previously, applying different kernel types improves the performance of the algorithm. Figure 11 illustrates the differences in levels of accuracy across each kernel type.

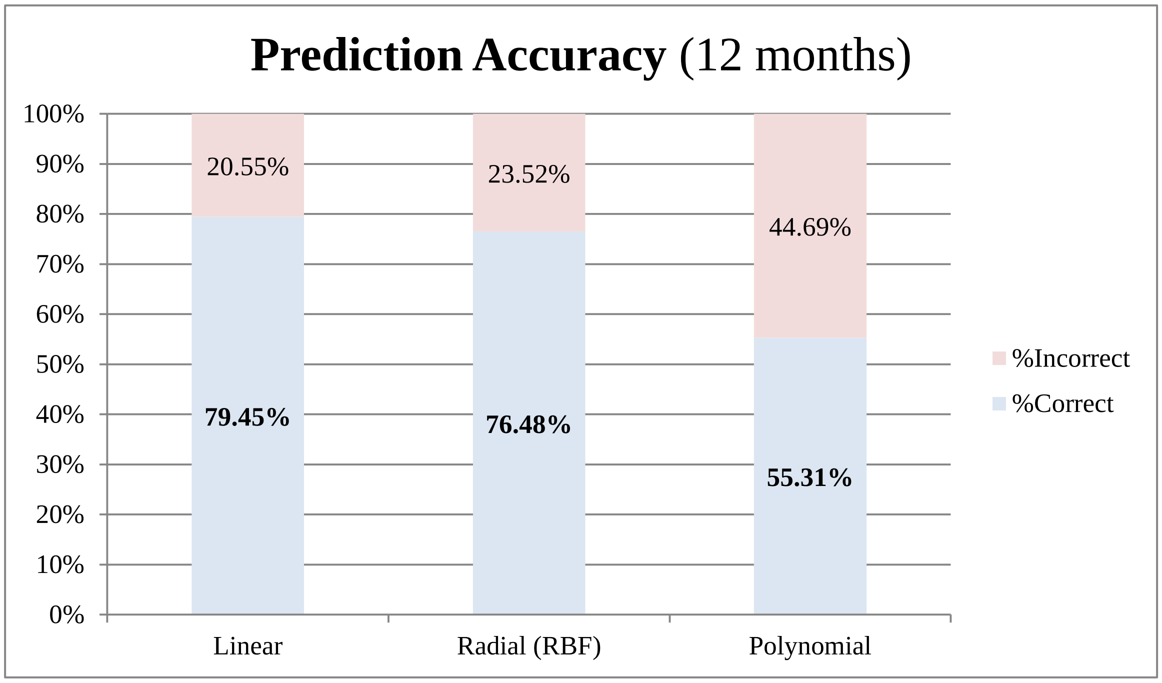


Figure 11: Accuracy comparison of the three kernel types on the historical dataset

In this case, the linear kernel proved to be best with an accuracy of 79.45%. The closest competing kernel type is radial. However it is clear that with this data set, the linear kernel is more accurate. Using the linear kernel the success of a project can be determined at 12 months with almost 80% accuracy. To measure the variance of accuracy as the development life span of the project is shortened; a further model with 9, 6 & 3 months development data was processed. Again, various kernel types were used to determine the best fit for each scenario. Table 10 provides an overview of each run, with the results for the different kernels types given.

Number of Months	Linear Kernel	Radial Kernel	Polynomial Kernel
18+	90.05%	**91.33%**	54.67%
12	**79.45%**	76.48%	55.31%
9	75.82%	**76.28%**	55.72%
6	73.71%	**75.23%**	55.72%
3	67.36%	**70.34%**	55.72%

Table 10: Summary of SVM prediction accuracy for all kernel types

Although the linear kernel provides more accuracy at 12 months, overall the radial kernel produces more consistent results over the datasets representing the first, second and third quarter, of the project’s first year of development, as well as the snapshot dataset i.e. 18+ months of activity. The highest accuracy values achieved are highlighted in Table 10 and can be seen in isolation in Figure 12. The highest SVM accuracy after 12 months is 79.45%, dropping by only 3.17% when the data is rolled back 3 months to the 3rd quarter of the first year's development span. A further 1% is lost by looking at the first 6 months development data and another 4.89% by just looking at the first quarter of the year. Although this constitutes a drop of almost 10% (from 12 to 3 months) this degradation is to be expected given the limited amount of activity that can be expected during the first quarter of development. At just 3 months, the prediction accuracy is still respectable at over 70%.

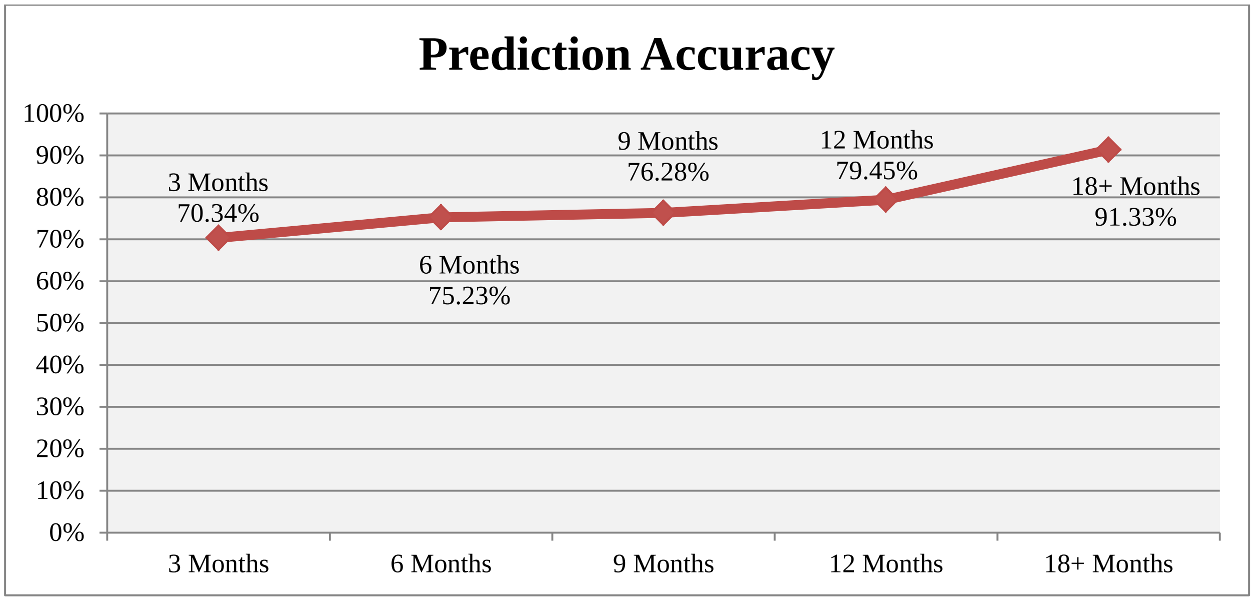


Figure 12: An overview of the variance in prediction from 3 months to 18+ months

Figure 12, illustrates how project success can be predicated with over 90% accuracy using the current snapshot dataset comprising over 18 months of development activity. The accuracy result remains high for the historical dataset at almost 80% when 12 months of activity is recorded. While it is expected that 12 months of data would produce more accurate prediction results it is interesting that even after 9 months, active projects will have gained enough data to provide a similar level of accuracy at almost 77%. Even with only 3 months of project activity, a reasonable guess can be made on the chances of success for that project. As such, the trend of increasing accuracy along the stages of a projects lifecycle suggest that this model would continue to be provide good indications into the future. The next chapter will discuss these findings in relation to the thesis goals as outlined in Chapter 1.

CHAPTER 6

CONCLUSION

At the outset of this thesis, two primary goals were identified that would assist in the understanding of what success means in open source projects. These goals, ranking and prediction, are discussed in Section 6.1, along with a summary of the findings from this research. Section 6.2 looks at potential future work, that would build on what has been achieved in this thesis and further enhance the goals of this thesis. Section 6.3 concludes this thesis with a reflection on the impact that this research provides for open source projects and its adoption.

6.1 DISCUSSION

This thesis was focused on the measurement of success for open source projects. With the proliferation of open source projects and the acceptance of these projects in commercial software, it is important for both project owners and users to have confidence in any project they are invested in. This thesis tackles the problem in two ways. Firstly, by allowing competing projects to be ranked against each other to provide a means of tracking the health of a project over time and determine which can be considered more successful. Secondly, by defining a prediction model which will determine the future success of a project at the earliest possible stage of the software life cycle.

As part of the research carried out to achieve these goals a number of new success metrics were identified which are directly based on the information that was found to be available through the GitHub API. Specifically, it was found that the GitHub data such as the number of subscribers (watchers), stars and forks were essential metrics for success; each of which should have a higher than average number. Inactivity was also identified as a crucial contributing factor to the failure of any project; for a repository to be considered successful it must have had at least one update in the last calendar month. These measures, along with the metrics concerning each contributor's reputation scores, and the bug fixing speed within a repository, give a much more detailed picture of how success can be both measured and predicted in GitHub repositories.

This builds on the initial set of metrics identified by Crowston [14] and Wang [19] in their previous works. This prior work, discussed in Chapter 2, identified key metrics that set successful projects apart from others, such as bug fixing speed and other developer related activities that can be monitored through the version control system. While research undertaken by Wang focused on SourceForge as the code sharing platform, this thesis brings this research up to date by looking at GitHub and leveraging the rich set of information available for each repository through its extensive API. In addition to the success metrics previously researched, the user reputation metric was introduced which measures the influence of the developer.

Building on the set of success metrics gathered in Chapter 3, the first goal of this thesis was to develop a ranking algorithm that could be used to compare the relative success of any number of projects, which are hosted as a GitHub repository. The algorithm developed using machine learning techniques, assigned weightings to a number of attributes made available for each repository following an intensive data retrieval process. Here it was found that the most important attributes for any repository were the number of contributors and the number of watchers. This reinforces the intuitive thinking that the social aspect of any open source project is crucial to its success. By measuring a selection of GitHub repositories, the ranking generated by the algorithm provided results that are in line with GitHub's own popularity measures, giving a high level of confidence in the calculation. As outlined in Chapter 1, tracking the success of competing technologies is becoming increasingly important as more open source projects are adopted by enterprises. The ranking algorithm developed in this thesis will allow the continual monitoring of the current health of a project and help provide a measured comparison of open source technologies.

The second goal of this thesis, outlined in Chapter 1, was to utilize machine learning techniques to build on prior research into the prediction of open source project success. As such, the original snapshot data retrieved from GitHub was extended with additional historical data taken at quarterly intervals to determine the best stage in the lifecycle to achieve an accurate prediction. Previous work by Wang utilized the K-Means clustering technique to predict project success. However, having generated a labelled dataset, using success metrics defined in the literature and supplemented with the authors own metrics specific to the GitHub platform, SVM (Support Vector Machine) was chosen as the supervised machine learning technique for this work. Running SVM on the historical datasets demonstrated that an accuracy of almost 80% is possible for any repository with twelve months of activity. However, it was also found that reasonable prediction scores are possible using this model at nine months (76%), six months (75%) or even just 3 months (70%). By using this model, companies or individuals can judge the likelihood of success of an open source project, providing a valuable insight into decisions surrounding technology adoption.

Furthermore, project owners can use this model to help identify when their project might be in danger of failing, giving them an early warning system with the chance to rectify the situation, by increasing the number of contributors or advertising the projects existence to the wider technical community.

6.2 Future Work

When this thesis was originally undertaken, the GitHub Archive project [34] was considered as a data source for all repository information. However, a number of issues were identified with this, making it unsuitable for this thesis. The archive provides information starting from February 2011, rather than the very beginning of GitHub’s history. Due to the range of the queries executed the costs of using BigQuery [38] were prohibitive for this research; there is a 1GB monthly limit on free use BigQuery, while the initial set of requests were in the range of 100MB each. Furthermore, the archive did not make all the required data available; the GitHub Archive is oriented around timeline events, rather than the level of detail required for this research. Using Google BigQuery to build up the information required for the metrics would have substantially reduced the time required in gathering the data; both in the initial writing of the queries, and the execution time. If this archive was to be enriched further, to include all the data that is provided through the raw GitHub API, and to provide a lengthier span of data, it would be a viable alternate means of extracting the information. The time taken to gather the information means that the snapshot data taken is already six months old at the time of writing. With more efficient means of gathering data either through BigQuery or through other advanced analytic and data-gathering as a service technologies, fresher sets of data could be retrieved in a fraction of the time, allowing more up to date analysis of what contributes to success.

The scope of repositories retrieved from GitHub was narrowed down to only JavaScript projects. Although this is a meaningful dataset, with JavaScript being the most popular programming language used on GitHub, there is merit in creating datasets that cover other programming languages such as Java, C# and PHP. Examination of these datasets might find further variance in the importance of metrics that have been used to define success, and might discover other relevant metrics.

At the time of writing, the GitHub API [39] was moving to version 3, with new endpoints being frequently added. It is possible that more information will be made available through the API in future that will help build an even richer data set. Also, while researching this thesis GitHub moved from a popularity chart, based on stars or forks, to a trending model [40]. As stated by GitHub *“We look at a variety of data points including stars, forks, commits, follows, and pageviews, weighting them appropriately. It's not just about total numbers, but also how recently the events happened”.* Although this approach currently fails to give a timespan outside of the current month, it is likely that historical trending data could be accessed through the API at some point in the near future.

To improve the performance of the SVM classification algorithm, the findings of the attribute weighting process could be applied to the model. Currently, all attributes were included in the SVM process however improvements might be seen by removing those attributes with lower weightings. As with all research projects, and particularly those surrounding machine learning, there is a wide variety of techniques and models that can be used. However, because of time constraints, it was not possible to evaluate and compare the results of each technique. In the case of project ranking, it would be useful to compare the performance of other feature weighting techniques such as Principal Component Analysis. Also, other models suitable for learning binary classifiers, such as decision trees, Bayesian networks and neural networks, could be explored.

6.3 Research Impact

The two goals outlined in this thesis help to further the current understanding of what success means for an open source project. It is now possible for industry analysts, project owners or commercial users to evaluate any number of open source projects to see which is regarded as the most successful. For enterprises, with a vast number of projects to choose from, each of which claim to solve a particular problem, a ranking result such as this could be the deciding factor in the technology selection process. Performing the ranking process at regular intervals will also allow stakeholders to monitor the health of the project with respect to its competitors. The open source project owner can compare the ranking result with competing projects and examine what needs to change in the project to make it more competitive, or maintain the edge over the competition.

Additionally, the prediction model developed allows project owners to look at their current repository statistics and see if they are on a trajectory for success or failure. Although this is a binary result, the findings in this thesis surrounding what factors contribute to success, allow corrective action to be taken. This could range from speeding up the time at which issues are addressed, to attracting more prominent contributors onto the development team. Once again, enterprises may use this model to become early adopters of open source projects, where knowing that the project is likely to succeed will make it more appealing to use.

REFERENCES

[1] M. J. Skok, “Slideshare,” 21 May 2012. [Online]. Available: http://www.slideshare.net/mjskok/2012-future-of-open-source-6th-annual-survey-results. [Accessed 12 September 2013].

[2] Q. Jiang, “CY12-Q3 Community Analysis — OpenStack vs OpenNebula vs Eucalyptus vs CloudStack,” 2 October 2012. [Online]. Available: http://www.qyjohn.net/?p=2427. [Accessed 22 August 2013].

[3] K. Fogel, Producing Open Source Software, O'Reilly, 2013.

[4] B. Perens, Open Sources: Voices from the Open Source Revolution, O'Reilly, 1999.

[5] R. Stallman, “The GNU Project,” [Online]. Available: http://www.gnu.org/gnu/thegnuproject.html. [Accessed 15 December 2013].

[6] E. S. Raymond, The Cathedral and the Bazaar, O'Reilly Media, 1999.

[7] “GitHub,” [Online]. Available: https://github.com/about. [Accessed 15 December 2013].

[8] K. Finley, “Github Has Surpassed Sourceforge and Google Code in Popularity,” 2 June 2011. [Online]. Available: http://readwrite.com/2011/06/02/github-has-passed-sourceforge#awesm=~oq6C9EwoibxaJh. [Accessed 15 December 2013].

[9] B. Doll and I. Grigorik, “Analyzing Millions of GitHub Commits,” 11 March 2013. [Online]. Available: www.igvita.com/slides/2012/bigquery-github-strata.pdf. [Accessed 2 August 2013].

[10] W. H. DeLone and E. R. McLean, “Information systems success: the quest for the dependent variable,” *Information systems research*, vol. 3, p. 60, 1993.

[11] W. H. DeLone and E. R. McLean, “Information systems success revisited,” in *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, 2002.

[12] P. B. Seddon, S. Staples, R. Patnayakuni and M. Bowtell, “Dimensions of information systems success,” *Communications of the AIS,* vol. 2, no. 3es, p. 5, 1999.

[13] J. S. Reel, “Critical Success Factors in Software Projects.,” *IEEE Software,* vol. 16, no. 3, pp. 18-23, 1999.

[14] K. Crowston, J. Howison and H. Annabi, “Information systems success in Free and Open Source Software development: Theory and measures,” *Software Process---Improvement and Practice,* vol. 11, no. 2, pp. 123--148, 2008.

[15] K. Crowston, H. Annabi and J. and Howison, “Defining Open Source Software Project Success,” in *ICIS 2003 Proceedings,* 2003.

[16] K. Crowston and H. Annabi, “Empirical Studies of Open Source Software Development.,” in *HICSS,* 2007.

[17] T. L. Sang-Yong, K. Hee-Woong and S. Gupta, “Measuring open source software success,” *Omega,* vol. 37, no. 2, pp. 426-438, April 2009.

[18] A. H. Ghapanchi, A. Aurum and G. Low, “A taxonomy for measuring the success of open source software projects,” *First Monday,* vol. 16, no. 8, pp. 1--15, 2011.

[19] Y. Wang, “Prediction of Success in Open Source Software Development,” 2005. [Online]. Available: http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-28.pdf. [Accessed 15 December 2013].

[20] N. McDonald and S. Goggins, “Performance and participation in open source software on GitHub,” in *CHI '13 Extended Abstracts on Human Factors in Computing Systems,* Paris, France, 2013.

[21] T. Mitchell, Machine Learning, McGrawHill, 1997.

[22] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, Amsterdam, The Netherlands, 2007.

[23] Wikipedia, “Support vector machine,” [Online]. Available: http://en.wikipedia.org/wiki/Support_vector_machine. [Accessed 15 December 2013].

[24] H. Drucker, D. Wu and V. Vapnik, “Support Vector Machines for Spam Categorization,” *IEEE Transactions on Neural Networks,* vol. 10, no. 5, 1999.

[25] D. Sculley and G. M. Wachman, “Relaxed online SVMs for spam filtering,” in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.

[26] E. Blanzieri and A. Bryl, “Instance-Based Spam Filtering Using SVM Nearest Neighbor Classifier.,” in *FLAIRS Conference,* 2007.

[27] Y.-W. a. L. C.-J. Chang, “Feature Ranking Using Linear SVM,” in *JMLR: Workshop and Conference Proceedings 3,* 2008.

[28] “rapidminer,” [Online]. Available: http://rapidminer.com/. [Accessed 15 December 2013].

[29] IBM, “SPSS software - Predictive analytics software and solutions,” [Online]. Available: http://www-01.ibm.com/software/analytics/spss/. [Accessed 15 December 2013].

[30] “SAS® Enterprise Guide®,” [Online]. Available: http://www.sas.com/technologies/bi/query_reporting/guide/. [Accessed 15 December 2013].

[31] “RStudio IDE,” [Online]. Available: http://www.rstudio.com/ide/. [Accessed 15 December 2013].

[32] “The R Project for Statistical Computing,” [Online]. Available: http://www.r-project.org/. [Accessed 15 December 2013].

[33] D. Berkholz, “Measure success by merges, not forks,” 13 July 2012. [Online]. Available: http://redmonk.com/dberkholz/2012/07/13/measure-success-by-merges-not-forks/. [Accessed 12 September 2013].

[34] “GitHub Archive,” [Online]. Available: http://www.githubarchive.org/. [Accessed 15 December 2013].

[35] “Support Vector Machine,” 5 November 2010. [Online]. Available: http://rapid-i.com/wiki/index.php?title=Support_Vector_Machine. [Accessed 12 September 2013].

[36] S. Rüping, “mySVM - a support vector machine,” 2000. [Online]. Available: http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html. [Accessed 12 September 2013].

[37] C. Martin, “Machine Learning - notes, thoughts, and practice of applied machine learning,” [Online]. Available: http://charlesmartin14.wordpress.com/2012/02/06/kernels_part_1/. [Accessed 15 December 2013].

[38] “Google BigQuery,” [Online]. Available: https://developers.google.com/bigquery/. [Accessed 15 December 2013].

[39] “GitHub Developer API,” [Online]. Available: http://developer.github.com/v3/. [Accessed 15 December 2013].

[40] J. Rohan, “Explore what is Trending on GitHub,” 13 August 2013. [Online]. Available: https://github.com/blog/1585-explore-what-is-trending-on-github. [Accessed 15 December 2013].

APPENDICES

REPOSITORIES LABELLED SUCCESSFUL

ace	cloud9	ember.js
addon-sdk	CodeMirror	emscripten
Aloha-Editor	colorbox	enchant.js
amber	compound	Ender
angular.js	connect	engine.io
autosize	consolidate.js	enyo
awssum	Crafty	es5-shim
azure-sdk-for-node	csslint	etherpad-lite
backbone	cube	everyauth
backbone-forms	cucumber-js	express
Backbone-relational	curl	ezpublish-legacy
backbone.layoutmanager	d3	fancyBox
backbone.marionette	DataTables	faye
backbone.validation	deployd	fine-uploader
beef	derby	firebug
bem-tools	dgrid	foundation
bootstrap	django-fiber	fullcalendar
brackets	django-grappelli	Gaia
browserid	EaselJS	galleria

casperjs	easyXDM	geddy
cheerio	ejs	Gm
gmaps	jquery-tokeninput	nano
goagent	jquery-ui	nanoScrollerJS
gollum	jquery-ui-bootstrap	node
greasemonkey	jquery-validation	node-amqp
h5ai	jquery-waypoints	node-browserify
haibu	jquery.terminal	node-ffi
handlebars.js	jquery_lazyload	node-http-proxy
Hapi	js-beautify	node-imap
Haraka	jsbin	node-inspector
headjs	jScrollPane	node-irc
highcharts.com	jsdom	node-memcached
html2canvas	jshint	node-mongodb-native
html5shiv	karma	node-mongoskin
i18n-js	Kibana	node-mysql
impress.js	kidsruby	node-postgres
Ink	kissy	node-restify
Jade	knockout	node-spdy
Jake	knox	node-sqlite3
jasmine	Leaflet	node-static
jasmine-jquery	learn.jquery.com	node-supervisor

Jison	less.js	node-validator
jitsu	locomotive	node-xmpp
joyride	log4js-node	Nodemailer
jqGrid	markdown-js	nodeunit
jqTree	marked	npm
jquery	mediaelement	openbadges
jQuery-Autocomplete	melonJS	openlayers
jQuery-File-Upload	mercury	Overscroll
jquery-handsontable	mobile-boilerplate	paper.js
jquery-lifestream	mobiscroll	pegjs
jquery-minicolors	mocha	persistencejs
jquery-mobile	Modernizr	phonegap-facebook-plugin
jquery-mockjax	moment	pkgcloud
jquery-placeholder	mongoose	plupload
jquery-selectBox	Monocle	popcorn-js
jquery-timeago	montage	pump.io
jquery-timepicker	mootools-more	q
jQuery-Timepicker-Addon	mxn	r.js
racer	socket.io-client	titanium_mobile
raphael	sockjs-client	ua-parser
redmine_backlogs	Spectrum	UglifyJS
remotestorage.js	Spine	underscore

request	Spm	URI.js
requirejs	sproutcore	video.js
reqwest	Statsd	vows
resourceful	Stylus	webfontloader
Restler	substance	webshim
reveal.js	Sugar	when
Rickshaw	superagent	WikipediaMobile
Ringojs	swagger-ui	winston
Roy	Swig	Wookmark-jQuery
Seajs	tablesorter	ws
sequelize	Testem	wymeditor
ShareJS	The-M-Project	yui3
should.js	Thorax	yuidoc
Sinon.JS	three.js	zepto
Sizzle	Tilemill	zeromq.node
SlickGrid	Tinymce	

REPOSITORIES LABELLED UNSUCCESSFUL

2dc_jqgrid	chess.js	Downloadify
activejs	Chroma-Hash	dygraphs
adva_cms	CNPROG	easy-fckeditor
APE_JSF	CodeMirror-old	echowaves
appengine	connect-js	elastic
askken	couchdb	emacs_chrome
awesome	cow-blog	emile
blockui	cramp	env-js
bogart	creditcard_js	fab
bookreader	css_browser_selector	facebox_render
boom_amazing	cufon	facyBox
bowline-twitter	date_input	fancy-zoom
boxy	detexify	fastladder
builder	digitarald-fancyupload	FCBKcomplete
calendar_date_select	django-mingus	firequery
cascade	django-page-cms	fireunit
celerymon	djangode	Flip-jQuery
chain.js	doodle-js	flotomatic
fluid-grid-system	jaml	jquery-expander
fluid960gs	jasmine-jstd-adapter	jquery-glow

form	javascript-debug	jquery-google-analytics
futon4mongo	javascript-last.fm-api	jquery-haml
g.raphael	javascript-tools.tmbundle	jquery-idle-timeout
geohash-js	jcarousel	jquery-meiomask
gibberish-aes	jelly	jquery-metadata
github-badges	jester	jquery-pageslide
gitweb-theme	jmapping	JQuery-PeriodicalUpdater
glee	jobberrails	jQuery-Plugins
GLGE	joint	jquery-postmessage
gordon	jparallax	jquery-smartresize
Gritter	jqfundamentals	jquery-treetable
growl4rails	jquery-address	jquery-upload-progress
haml-js	jquery-approach	jquery.entwine
hookbox	jquery-auto-geocoder	jquery.hotkeys
hotdot	jquery-bbq	jquery.livetwitter
hurl	jquery-claypool	jquery.path
ico	jquery-cluetip	jquery.ui.spinner
In-Field-Labels-jQuery-Plugin	jquery-cookie	jquery_example
instiki	jquery-css-transform	jquery_jeditable
jake	jquery-dotimeout	jquery_viewport
js-base64	MvcContrib	prettyPrint.js
js-yaml	narwhal	primer

jschat	nerve	profxmpp
jscouch	newsmonger	protolicious
jsnes	node-couch	pygowave-legacy
jsOAuth	node-persistence	quicksearch
jsPDF	node-router	rails-ckeditor
jszip	node.dbslayer.js	redis-node-client
juggernaut_plugin	node.websocket.js	redmine-stuff-to-do-plugin
keysnail	nodewiki	Reel
kiwi	oocss	remedie
lesscss-engine	opentip	RESTClient
liquid-editor	paging_keys_js	restler
livequery	parser-lib	rest_in_place
local-storage-js	pastebin	rhodes-system-api-samples
low-pro	pavlov	rifgraf
low-pro-for-jquery	phonegap	rightjs-core
masonry	phpjs	rt
meeting-ticker	picard	RTree
merb-plugins	polypage	s3-swf-upload-plugin
modalbox	postgres-js	scenejs
mooeditable	pre3d	screw-unit
scripty2	Stream	toast
scrubyt	string_score	topup

scrum-pm	studio3	transformie
selectorgadget	swfupload-jquery-plugin	tufte-graph
selenium-on-rails	swfupload_fu	uki
shadowbox	swinger	Vanadium
shapado	taberareloo	vim-recipes
shapefile-js	Tasks	visage
silver_bird	Tatsumaki	wagn
Simplegraph	TextboxList	wireit
Slidedown	the-cloud-player	wysihat
so-nice	the-sexy-curls-jquery-plugin	wysihat-engine
Sofa	Thruk	xjst
Spazcore	timeframe	xlsuite
Sproutcore	timePicker	xui
sproutcore-samples	titanium	yui3-gallery
sproutcore-ui	titanium_developer	