Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

JONATHAN FRIESS
MASTER THESIS

PREDICTORS OF SUCCESSFUL OPEN SOURCE PROJECTS

Submitted on 7 January 2019

Supervisor: M.Sc. Michael Dorner
Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 7 January 2019

License

Erlangen, 7 January 2019

Abstract

The number of open source software projects has continued to grow for many years now, with a portion of them being very successful and broadly adopted. For companies and individuals involved in an open source project, it is advantageous to estimate its success and longevity. There are some high-level models of success to be found in literature. However, other than growth, factors of success derived from commit data have been barely studied by academics. This thesis searches for predictors that can contribute to a predictive model. We use an exploratory approach on a large set of over 11,000 active projects to find predictors. During the course of this work, we implemented a duplicate correlation filter in order to account for forked projects, and we designed a general ranking metric. We compared the distribution of features between the high ranking projects and all others. Through that, we found that the amount of initial contributors and the number of active months correlates positively with success. An in-depth evaluation of the explored predictors is not part of this work.

Contents

1 Introduction

1.1 Original Thesis Goals

This thesis aims to find predictors of success for open source projects through an exploratory data analysis approach. As part of this approach, an overview of available literature is provided. Further goals include the development of a ranking metric for open source software projects, and the evaluation of the predictors.

1.2 Changes to the Thesis Goals

In the course of this research, the need for further data conditioning has arisen; hence this thesis also aims to design and implement a duplicate filter. Second, due to time constraints, the evaluation was shortened.

2 Research

2.1 Introduction

Open source development of software is considered efficient and has been increasingly adopted by companies over the last decade. Indeed, around half of open source software (OSS) development is actual paid work (Riehle, Riemer, Kolassa & Schmidt, 2014). With the increase of OSS in commercial fields, the interest in the success prediction of OSS projects has, accordingly, also risen. This is a complex task since various qualitative and quantitative factors can play a role in success – ranging from community dynamics over licensing to development activities. Many studies on these different factors can be found in the work of Ghapanchi, Aurum and Low (2011).

This work focuses on quantitative commit data and provides features for supporting the development of a predictive model. By analyzing a large data set of open source projects, we aim to isolate factors that can be used to predict the chances of success.

2.2 Related Work

2.2.1 Concerning Growth

Several, well-received papers have been published regarding the growth of OSS. All of the following research has employed a type of *line of code* (LoC) metric for the growth models. Although we use only pure commit numbers in this work, these studies are still relevant since the number of commits correlates closely with the LoC. Kolassa, Riehle and Salim (2013) have shed light on the distribution of the LoC per commit, finding a median of 16 LoC.

Godfrey and Tu (2001) have examined the growth and evolution of the Linux Kernel, wherein they have found a super-linear process of growth. They have

postulated the hypothesis that ”successful open source software seems to have a development dynamic — distinct from that of most industrial software — that allows some systems to grow at a super-linear rate for prolonged periods.”
Robles, Amor, Gonzalez-Barahona and Herraiz (2005) have also examined the growth of the Linux Kernel, among 18 other large *Free/Libre and Open Source Software* (FLOSS) projects. They have also discovered a super-linear increase for the Kernel, while 15 out of the 18 other projects showed linear growth.
Covering the small scale, Roy and Cordy (2006) have not found any super-linear growth in the two small-sized projects they evaluated.
Succi, Paulson and Eberlein (2001) have compared the growth rates of three commercial, and three open source software projects, including Linux. The growth rate of the majority of projects they analyzed can be described by means of linear approximation, with the only exception being Linux.

While all of these studies have focussed on a few individual projects, Koch (2007) analyzed the growth of 4,047 open source projects, by using data from SourceForge. He was interested in deriving a growth model to fit the individual projects of his dataset. Opposed to the others, his study, interestingly, has indicated that a quadratic growth model fits better than a linear growth model.

In contrast, we do not focus on growth directly, but rather on success, which is induced by long-term growth. Further, instead of LoC, we use the number of commits.

2.2.2 Concerning Success

Many previous studies in the field of FLOSS have provided a definition of *success* for OSS. The definitions vary greatly since there is no established definition as yet. Ghapanchi et al. (2011) have undertaken an extensive literature review, including 45 studies and their respective definitions of successful open source software. In their approach to create a measurement taxonomy for success, they have isolated six ”success areas”, all of which have been studied by the papers they used. Namely, these areas are product quality, project performance, user interest, project efficiency, project effectiveness, and project activity. This has created a high-level view on success, whereby factors such as governance model, license model, and the ratio of paid vs. volunteer contributors can be embedded on a lower level.
From this perspective, this research can be considered to reside in the area of project activity. We solely focus on commit data, deriving an activity measure from it, and creating a ranking metric to reflect the relative success. Other factors that might have a large influence on the overall success, are beyond the scope of this work.

Research regarding the *prediction* of success can be found within the scope of start-ups. Amar Krishna and Choudhary (2016) have build several models for predicting the success of start-ups. Their work has been based on a dataset of 11,000 failed and successful companies. After defining several key factors – such as the amount of seed funding, the time to fund, the time in the market and the burn rate – they have used supervised learning classifiers to model success-prediction.
This is interesting for this research, because in some aspects, an emerging OSS project is comparable to a start-up. Contributors can be seen as employees, and commits representing the work done, what might be seen as the revenue stream. We relate to their research for initial feature selection. Also, the failure rate of start-ups is around 90% (Amar Krishna & Choudhary, 2016), in comparison to the 63% failure rate that Krishnamurthy (2002) has found among SourceForge projects. When one considers the large increase in numbers of OSS since then, it seems quite likely that the failure rate of OSS has increased as well – matching that of start-ups even more closely.

Krishnamurthy (2002) has looked at the top 100 mature open source projects, where he found that most projects were developed and maintained by an exceedingly small number of people. He used data from SourceForge, which classifies projects into stages, including a mature stage. Among 480 mature projects, a ranking was created through use of the activity percentile provided by SourceForge. According to SourceForge, the activity percentile is composed out of a traffic, a development, and a communication component. Since these components are not further defined, it is difficult to comprehend the underlying metric. In contrast, this work defines a transparent ranking metric, that can be applied to data from various sources. In addition, a much larger data set of over 11,000 active OSS projects has been used.

2.3 Research Question

The broader question of the research is: *how can the success of emerging open source projects be predicted?*
With the purpose of contributing to that problem, we focus on the particular question: *what are useful success predictors in commit data?*
The answer delivers one component for a predictive model of OSS success.
During the research, further questions arose. For example, *how can one identify forked OSS projects through commit data?* And *what is a suitable metric for ranking OSS projects based on commit data?* These questions are answered in this thesis.

2.4 Research Approach

2.4.1 Methodology

In this work, we used the *exploratory data analysis* (EDA) approach that has been established by Tukey (1977). For the most part, we employed visual and descriptive techniques, while using correlations to identify duplicates. Overall, the method entailed an iterative process of exploring the data and implementing or, otherwise, adapting the pre-filters. Hence, we only describe the final iteration in this written work.
The data set was provided by the professorship. At the beginning of the research, we merged the given data and metadata, which contains the development history of over 238,000 OSS projects. After examining the distribution of the projects in a scatter plot, we found that mapping these to the best fit line can result in a useful relative ranking. As many duplicates where apparent at this point, we identified them through correlation. Furthermore, a graph visualization revealed a clustering of some duplicates, which we then used to manually identify the original projects.

2.4.2 Brief Background

Time Series Analysis

For the duplication filter, we used time series analysis techniques. When one deals with time series data, one important property to consider is stationarity. A time series is stationary if its properties do not depend on the time of observation. In other words, if the *common probability distribution* of a process does not change over time, then it is considered stationary (Kirchgässner, Wolters & Hassler, 2008). In the simplest case, this is true for white noise, a succession of uncorrelated, random values that have a zero mean. But for real-world data, this is almost never the case.
The project data in our set shows obvious trends, as the activity on projects grows and shrinks. Therefore we needed to transform the time series into stationary data. One plain method to achieve this is computing the difference between two subsequent points – known as *differencing*. The differenced time series can be written as $Y'_t = Y_t - Y_{t-m}$, where m is the lag. The lag determines the distance of the left shift. We used the *random walk model* for our data; this means differencing with a lag of 1. For closely correlated time series, such as the project data, the resulting series will approach white noise. This means we will loose information on the trend, but it enables us to apply further statistical methods

that require stationary data.
To test the de-trended time series for stationarity, we applied the widely used *augmented Dickey-Fuller* (ADF) test, since it is valid for large samples. It tests for the presence of a unit root; accordingly, in the absence of it, the process is stationary.
To measure the correlation between two time series, we used the *Pearson correlation coefficient* (PCC). It ranges from -1 to 1, with a value of 0 denoting no correlation at all, while a value of 1 indicates a perfect match. In this work, we looked for a high positive PCC to find duplicate projects. A high PCC implies that the variables move in a most similar way, which is the case for duplicated / forked projects.

Supervised Machine Learning

To test the predictors, we made use of supervised machine learning. "A supervised scenario is characterized by the concept of a teacher or supervisor, whose main task is to provide the agent with a precise measure of its error (directly comparable with output values). With actual algorithms, this function is provided by a training set made up of couples (input and expected output)."(Bonaccorso, 2017) With this information, an algorithm is trained, constantly correcting its parameters to reduce the output error. After all, the goal is that the algorithm works with samples never seen before.
In this work, we used the *Gaussian Naive Bayes* (GNB) algorithm for classifying projects as successful / non-successful. The GNB is a simple probabilistic classifier, with 'naive' assumptions: It assumes strong independence and Gaussian distribution of the variables. Although these oversimplified assumptions are almost never true, the GNB has proven to work reasonably well for real-world data in the past.

Performance Metrics

An intuitive and broadly used way to assess the performance of classifiers is the *confusion matrix* (Bonaccorso, 2017). A confusion matrix is shown in Figure 1. From that, several performance metrics are derived. It depends on the use case towards which metric the algorithm should be optimized.

Precision is a measure for the portion of items classified as positive, that are actually positive. It is defined as:

$$Precision = \frac{TP}{TP + FP}$$

	Actual positive	Actual negative
Predicted positive	True positive (TP)	False positive (FP)
Predicted negative	False negative (FN)	True negative (TN)

Figure 1: Confusion Matrix

Recall is the portion of positive items being correctly classified as positive by the algorithm.

$$Recall = \frac{TP}{TP + FN}$$

Recall gives us information about the false negatives, indicating how many items the classifier missed, whereas *precision* tells us how many items it caught out of all positives. When the focus is on minimizing false positives, *precision* should be as high as possible. Otherwise, when aiming to minimize the false negatives, *recall* should be as high as possible.

Finally, the *F1* score combines the two previous metrics by calculating the harmonic mean, which is useful for finding a balance between *precision* and *recall*:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.4.3 Definitions

Throughout this paper, we use the following definitions, based on those of Riehle et al. (2014):

- A *(code) repository* is used as a synonym for a version management system.
- To *commit* is the process of putting a piece of code into a repository.
- A *commit* is the piece of code submitted.
- A *contributor* is a person who has submitted at least one commit and has, hence, contributed to the project. One should note that often this is regarded as a *committer*, while the contributor denotes the author of the code, who does not have the access rights to commit the code by himself or

herself. In this paper, we do not distinguish between *committer* and *contributor* since one cannot ascertain who undertook the actual work through an examination of the data.

Furthermore, we use the definition of *active projects*, as defined by Daffara (2007) and subsequently rephrased by Kolassa:
"A project is active at a given point in time if the number of commits in the preceding 12 months is at least 60% of the number of commits in the 12 months before that."(Kolassa et al., 2013).

This means that a project can be considered active if the following condition is met:

$$0.6 \leq \frac{\sum_{i=1}^{n} commits_i}{\sum_{i=n+1}^{2n} commits_i} \text{ , with window } n = 12$$

For projects with less than 24 months worth of data, we have adapted the definition so as to use the following formula for the comparing windows:

$$n = \frac{length(commits)}{2}$$

This ensures that younger project can also be classified as active / non-active.

2.4.4 Used Tools

As the main development environment, we used *Jupyter Notebooks* in conjunction with *Python 3.6*. This ensures an easy comprehension of all processing steps taken, and decreases the effort required for anyone wishing to reproduce the results. We utilized the *pandas* library for handling, pre-conditioning, and filtering the data, while using *numpy* and *scipy* for statistical analysis. The visualization was done by *matplotlib*, *seaborn*, and *networkx*. To embed the generated figures into this document, we utilized the *pgf*-backend for *matplotlib* together with *pdflatex*.

All tools used are open source and freely available for all major platforms.

2.5 Used Data

2.5.1 Data Sources

The main data source used for this research is the platform Open Hub (formerly called Ohloh) by the company Black Duck Software, Inc. (2014). This platform catalogues open source projects from many different version control services. All

data published on Open Hub was originally crawled and downloaded through the use of their provided API. The raw data set contains more than 238,000 projects, within a total of 2.9 million data points. The data is quantized in time slices of months.
We cross-validated the data set against data directly from GitHub. For this, we picked a sample of known, large projects hosted on GitHub. We then parsed the commit data from the *Git log* and compared their development histories to the data from Open Hub (see Figure 2). Most of the data matched extremely closely, with some minor quantitative differences. This is because Open Hub keeps track of multiple repositories per project and merges the data. This is also why there is a significant discrepancy apparent for *Mozilla Firefox*. Firefox mainly uses a *mercurial* repository for development, with the repository on GitHub only being a mirror, which does not include all branches. However, Open Hub includes a merge of all commits from all possible branches.
Finally, the data from GitHub is more up to date. This is due to the delay introduced by the crawling interval of Open Hub, and additionally, the last time we updated the local data set. Since we are focussing on the development history, this is not critical for this work.

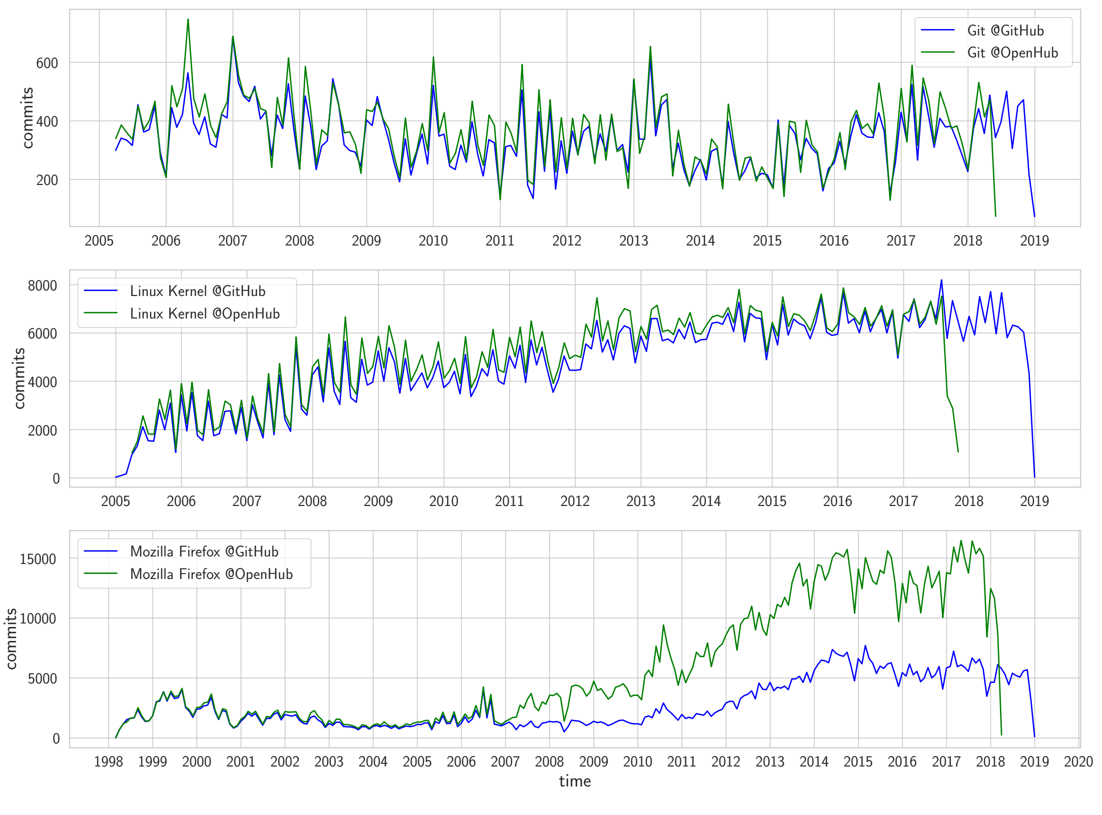


Figure 2: Comparing sample projects against GitHub data

2.5.2 Descriptive Statistics

Every data point represents the monthly activity of a project. It contains the number of commits, the number of contributors, and a time stamp for the month. A unique ID identifies the corresponding project that the data relates to.

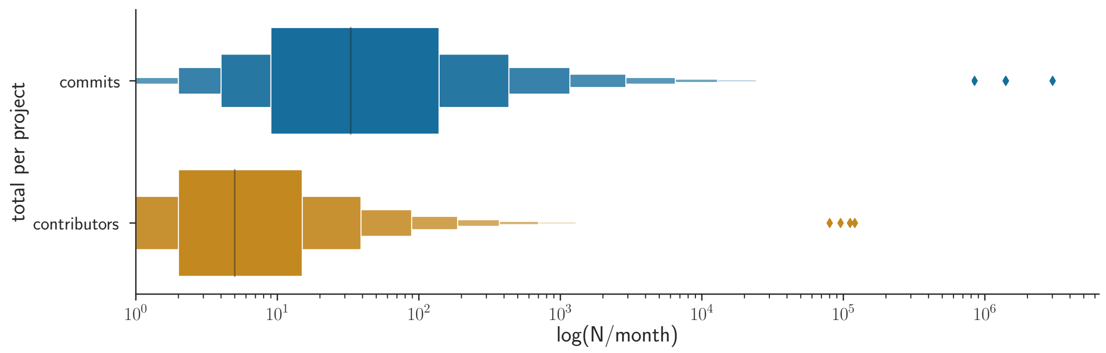
Descriptive statistics of the projects are presented in Table 1. The percentiles indicate that there are many tiny projects in the set. The medians of total commits and contributors per projects are 33 and 5, respectively, while the 95th percentile shows that there are projects on the high end, which have numbers magnitudes larger. This is to be expected from a nearly complete data set since such a set should include many personal code dumps and projects by hobbyists, as well as the large, vivid OSS projects. This is also reflected in the length – respectively the number of data points per project – where the median percentile is at 4, while the top 5% of the projects are above 55.

Table 1: Descriptive statistics of all projects

	commits	contributors	length
count	237,972.00	237,972.00	237,972.00
mean	670.24	44.08	12.41
std	10,631.59	653.59	23.43
min	1.00	1.00	1.00
25%	9.00	2.00	2.00
50%	33.00	5.00	4.00
75%	139.00	15.00	12.00
95%	1,594.00	114.00	55.00
max	3,013,656.00	120,531.00	258.00

Figure 3 shows a graphical summary of the distributions as a letter-value plot. Since the data points are wide spread, with only a minority scoring high numbers, the letter-value plot conveys further information about the tail behaviour in comparison to a simple boxplot (Hofmann, Wickham & Kafadar, 2017).

The outliers can be found in Table 2 and Table 3, wherein *KVM* appears as the project with the most contributors in total. Given the nature of KVM, one suspects it to be mostly a duplicate of the *Linux Kernel* with only a few additional commits and contributors. We look into this later on in chapter 2.6.1.



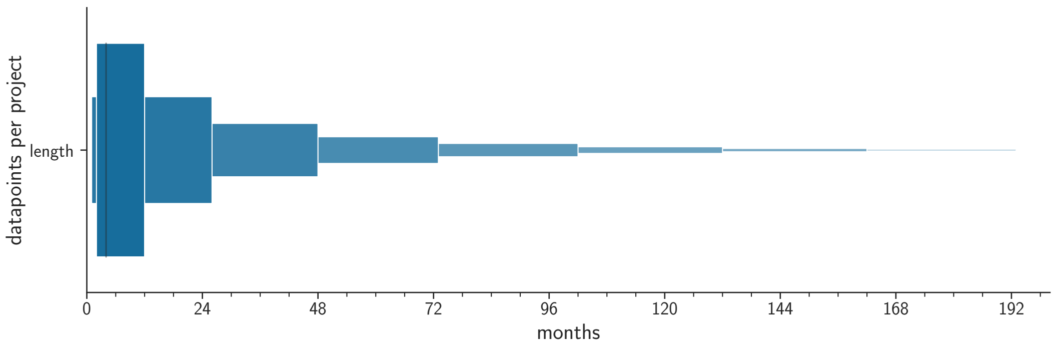


Figure 3: Letter value plot of descriptive statistics (using tukey stopping rule)

Table 2: Upper outliers in total commits

	commits	contributors	length
KDE	3,013,656	59,240	247
NetBeans IDE	1,406,202	14,253	226
Mozilla Firefox	1,405,371	50,834	242
KVM	845,738	120,531	158

Table 3: Upper outliers in total contributors

	commits	contributors	length
KVM	845,738	120,531	158
Linux Kernel	772,157	111,562	152
XFS Filesystem	667,096	95,585	232
Linux NTFS file system support	562,988	80,065	152

2.6 Results

2.6.1 Duplicate Filtering

The dataset turned out to include many duplicates. While there were no exact duplicates in the set, many stemmed from forked projects, with only minor additional work having been undertaken. This can be seen, in particular, for projects related to the Linux Kernel. There are many ports of the Kernel for different platforms. We want to identify these in order to prevent distortion in the further analysis.

To detect these forks, a correlation filter was implemented. For every two projects that overlap more than 33% in their time series, the Pearson correlation coefficient (PCC) of this overlapping window is calculated. One assumption of the Pearson correlation is, that the respective processes are stationary. Since most of the time series in the data set shows an obvious trend – and are, therefore, non-stationary – a difference-transform with a lag-1 difference was applied. Evaluating the stationarity of the resulting time series is done by testing the null hypothesis with the augmented Dickey-Fuller test (ADF). The results show that after the transformation, more than 90% of all the time series are stationary ($p \leq 0.05$). For the purpose of finding duplicates, that value is reasonable. Having some trend left in the data is not critical since two duplicates are extremely likely to share the same trend, and, therefore, correlate significantly despite being considered non-stationary.
The computation of the PCCs can be quite resource intensive, considering the huge amount of possible permutations of the projects. Because of that, only a subset of 5,000 projects was used here. This consisted of the top 5,000 projects, as ranked in chapter 2.6.2. We considered projects with a $PCC > 0.97$ to be duplicates. This threshold delivered useful results. Linking all correlated pairs together creates a network of the suspected duplicates. Figure 4 visualizes a section of the graph, containing the eight largest clusters. We then manually identified the original projects of the clusters that were part of the top 100 in the ranking. For all other clusters we defined the oldest project as the original.
By far the largest cluster consisted of Linux kernel based projects. We chose this cluster to evaluate the results of the duplicate filter. By manually examining the projects, reading their repository descriptions, as well as comparing their commit time series directly (see Figure 5), we found no false positives among them. One should note that only PCCs greater than 0.97 were considered for clustering. It might be possible to lower the threshold, although at the cost of increasing false-positives.

Almost all of the duplicates were forked projects, with additional work undertaken based on the original. This can, for example, be observed in Figure 5 for the

Linux Kernel and KVM. Accordingly, instead of removing all forks from the set, we decided to take the difference of the forks and the originals. Hence, we could treat them as independent projects and could keep their work in the set.

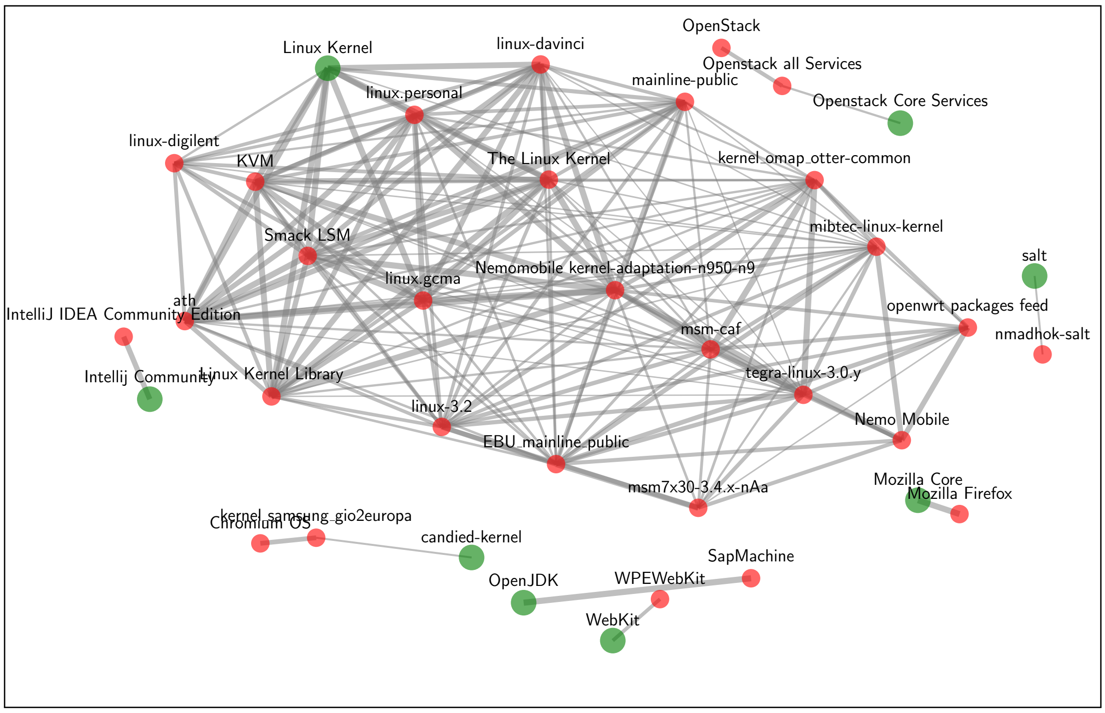


Figure 4: Section of the correlation network - original projects in green

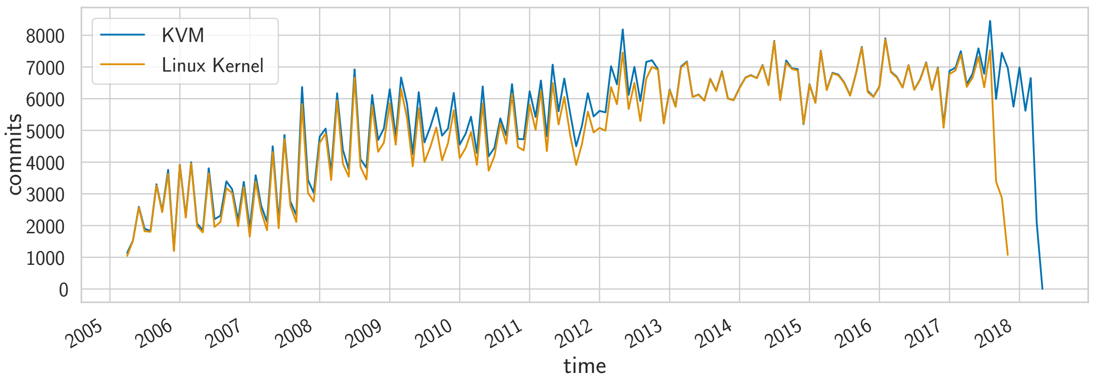


Figure 5: Linux kernel and KVM time series

2.6.2 Ranking Metric

A project with a high activity, which means that many people are constantly working on it, is considered successful. To quantify the success of a project, a ranking metric was developed. In order to keep this metric simple and robust, only three dimensions were considered: commits, contributors, and time. Time is involved implicitly through filter conditions, while commits and contributors were merged into a ranking score.

We applied several pre-filters so as to only include projects that met the following conditions:

- The median of contributors per month was higher than 1.3.
- There were data points in the previous 2 years.
- The projects were "active", by the definition in 2.4.3

The first condition removed most of the tiny projects that were not of interest for this research. The value of 1.3 was initially derived from the lower 1/4 percentile, and it was tweaked later in the process. This median value seemed to work well for this simple pre-filter.
The second condition sets the time window to which the ranking metric was applied. Two years was reasonable for obtaining a stable but dynamic ranking. Smaller windows, however, could be used to attain a momentary view.
Last, we filtered out any projects that were not considered active, as from the date of their last data point. After all, this leaves us with over 11,000 currently active projects.

When one examines the scatter plot of median commits over median contributors in Figure 6, it is apparent that many well-known projects scored highly on both dimensions. The best fit line was used as a trajectory for an ideal development. By orthogonally projecting the points onto the best fit in normed space, and by taking the Euclidean distance, we obtained a relative ranking. The *best fit* denotes the weight of commits and contributors for this particular data set. The color indicates the lifetime of a project until the present day, ranging from green for ≥ 10 years to red for ≤ 1 year. A list of the ranking results can be found in the appendix.
As one might expect, many projects with high numbers in commits and contributors are also very mature – most having a lifetime of over 10 years. There were some outliers: Microsoft's *.NET* can be seen to have an lifetime of under five years, which seems odd since it was published in 2002. But .NET was not open sourced until 2014 and, therefore, was not listed in any public repository. This matches the earliest data point of it in the set.
Another outlier was *cctx*, which despite its young age, scored noticeably high,

ranking quite close to *QT 5*. A quick glance at the Git repository reveals that it is a trading library for cryptocurrency exchange markets, and that it is broadly adopted in the that field. Given the background of the cryptocurrency boom at the time, this rapid development seems plausible.

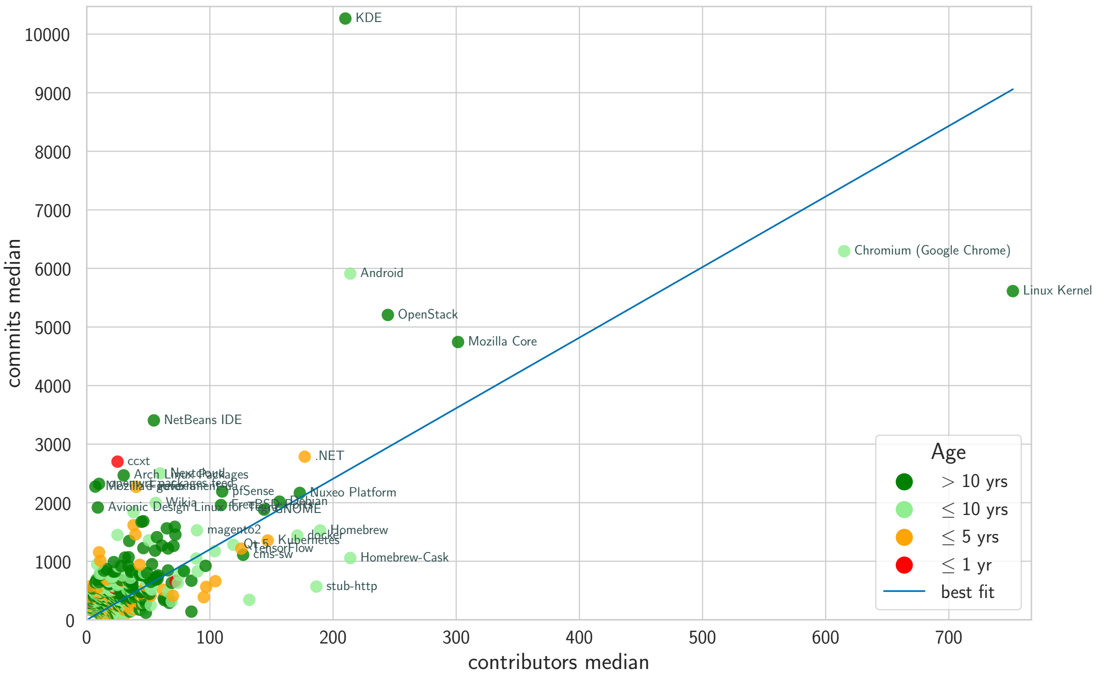


Figure 6: Scatter plot of active projects

2.6.3 Predictors

To compare potential predictors, we split the data into two groups: the successful group, consisting of the top 500 projects in the ranking, and all other projects that were active for at least one month in their lifetime. The latter contained around 50,000 projects. The top 500 reflected the largest, currently active projects; at the low end, there were projects with around ten contributors per month.

Selecting Features

Inspired by the features used by Amar Krishna and Choudhary (2016) for predicting the success of start-ups, we investigated the *quantity of active months* per project, the *number of initial contributors* and the *rate of commits per contributor*.

To obtain an active-status for every point in a given project's lifetime, we applied the definition of active as a rolling function over the time series. After aggregation, we could tell for how long a given project was considered active. Comparing the descriptive statistics of this feature in the two groups shows differences in distribution. For the top 500 group, this resulted in 75% of projects with more than 48 active months, while 75% of all the other projects were below 28 active months (see Table 4). This difference indicated that it could be a useful feature for classification.

We calculated the second feature, the *number of initial contributors*, as the sum of contributors over the first six months of a given project's lifetime. An examination of the distribution between the two groups shows that 75% of the top 500 had more than 14 initial contributors, whereas 75% of all the others had less than 7 (see Table 5). We tried different time windows for the summation of contributors, ranging from 1 to 24 months. Increasing the window enhanced the difference in distribution – but only significantly up to a period of six months.

To feature the work done per contributor to a given project, we calculated the median *rate of commits per contributor* (see Table 6). In comparison to the other features described before, the distribution of the rates between the two groups was far more similar. Despite the fact that contributors committing three times as much in the top 500 group in median, its the weakest of the feature so far.

Table 4: Descriptive statistics of active months

	top500	all
count	500.00	54,368.00
mean	96.09	22.47
std	63.69	29.11
min	0.00	1.00
10%	20.90	2.00
25%	48.00	5.00
50%	83.00	12.00
75%	138.00	28.00
90%	195.10	56.00
max	246.00	246.00

Table 5: Descriptive statistics of initial contributors

	top500	all
count	500.00	237,972.00
mean	77.70	5.87
std	174.36	12.48
min	6.00	1.00
10%	9.00	1.00
25%	15.00	2.00
50%	36.00	5.00
75%	77.25	7.00
90%	145.50	11.00
max	1,960.00	1,960.00

Table 6: Descriptive statistics of commits per contributor rate

	top500	all
count	500.00	54,368.00
mean	21.65	8.88
std	19.73	25.02
min	1.81	1.00
10%	6.48	2.00
25%	9.69	3.00
50%	15.11	5.00
75%	26.06	10.00
90%	44.57	19.00
max	217.00	5,100.00

Testing Features

To test the performance of the features, we chose the *Gaussian Naive Bayes* (GNB) classifier in a supervised learning setup. The training set we used is constructed of successful / non-successful projects, with the same frequency as they occur in the data set. Using the same amount for both classes would introduce a bias towards the successful class. We used the standard ratio of 20:80 to split the data in a training and a test set, resulting in 10,973 projects in the training set.

As the items for the training set are randomly selected, we ran 1000 iterations of training and testing for each feature. This minimizes the impact of "lucky" constellations in training items and delivers reproducible results.
Table 7 lists the median performance results of all iterations per feature set.

Among the single features, we found the number of active months to perform best in classifying the successful projects correctly, with a *recall* of 30%. However, using any of the features individually resulted in a rather poor performance, ranging between 30% and 8% for recall, which indicates many false negatives. When a combination of features was used, the GNB performed better, with matching rates from 36% up to 57%. The highest result was obtained by using all the features to train the GNB.
The recall tells us what portion of actual successful projects have been classified as successful. However, for our case, the reduction of false positives is more interesting. We want as little projects as possible to be falsely classified as successful. The feature that produced the smallest amount of false positives is *initial contributors*, with a *precision* of 30%. This is still better than a random classifier guessing half of all projects to be successful, which scores 0.8% in precision.

The *rate of commits per contributor* scores lowest throughout all metrics. This supports the implications of the descriptive statistic in Table 6, that it is not significant enough as a predictor.

Note that the *overall* performance – that is the percentage of all projects classified correctly – is not sound for our data. The data set is vastly imbalanced, through the successful projects being a tiny minority.

Table 7: GNB feature performances

	used features	overall	precision	recall	F1
0	'active', 'init_contributors'	0.98	0.21	0.57	0.30
1	'init_contributors'	0.99	0.30	0.32	0.30
2	'active', 'init_contributors', 'rate'	0.97	0.19	0.56	0.28
3	'init_contributors', 'rate'	0.98	0.21	0.34	0.26
4	'active'	0.98	0.14	0.34	0.20
5	'active', 'rate'	0.97	0.13	0.39	0.19
6	'rate'	0.98	0.06	0.07	0.07

2.7 Discussion

With the duplicate filter we implemented, it is easy to identify forked projects in a reliable way. This helps to condition time series data of OSS and also opens up the possibility of finding similar or connected projects. Advanced tweaking of the thresholds could improve the results further, but with the current values we found no false positives.
A future approach for increasing the resource efficiency of the filter might be to

roughly pre-select pairs of projects, for example, by matching projects with a similar median in commits. This would avoid computing the PCC of pairs that are highly unlikely to be duplicates, due to the fact that duplicate projects are close to each other in absolute commit numbers. This approach might also open up the possibility of running the filter on the complete data set.

The ranking metric is suitable for establishing a relative ranking between projects at a given point in time. This metric provides us with an easy tool to generally compare projects based on their activity. When one expands the time window to the lifetime of all projects, an absolute ranking value is delivered; this makes it possible to compare projects between datasets. We used it to obtain the current most projects, within a window of 2 years, and we defined the top 500 projects as successful samples. For future work, it might also be interesting to look at changes in rankings over time and, for example, identify those projects that outpace others.

The GNB performed quite poor on classifying, however, as intended it gave an orientation on which of the features are most useful for prediction. The number of *initial contributors* and *active months* found to be promising. But clearly, further evaluation is needed.
One could suspect that projects funded by companies or those that include many paid developers have a higher chance of being successful in the long run. This might be a reason for the number of initial contributors being higher among the top 500 projects since funded projects are likely to have more contributors right from the start, whereas projects started by individuals or small groups of volunteers should take longer to attract contributors. Hence, identifying projects in the top 500 that are funded or supported by companies would be an informative task for future work.

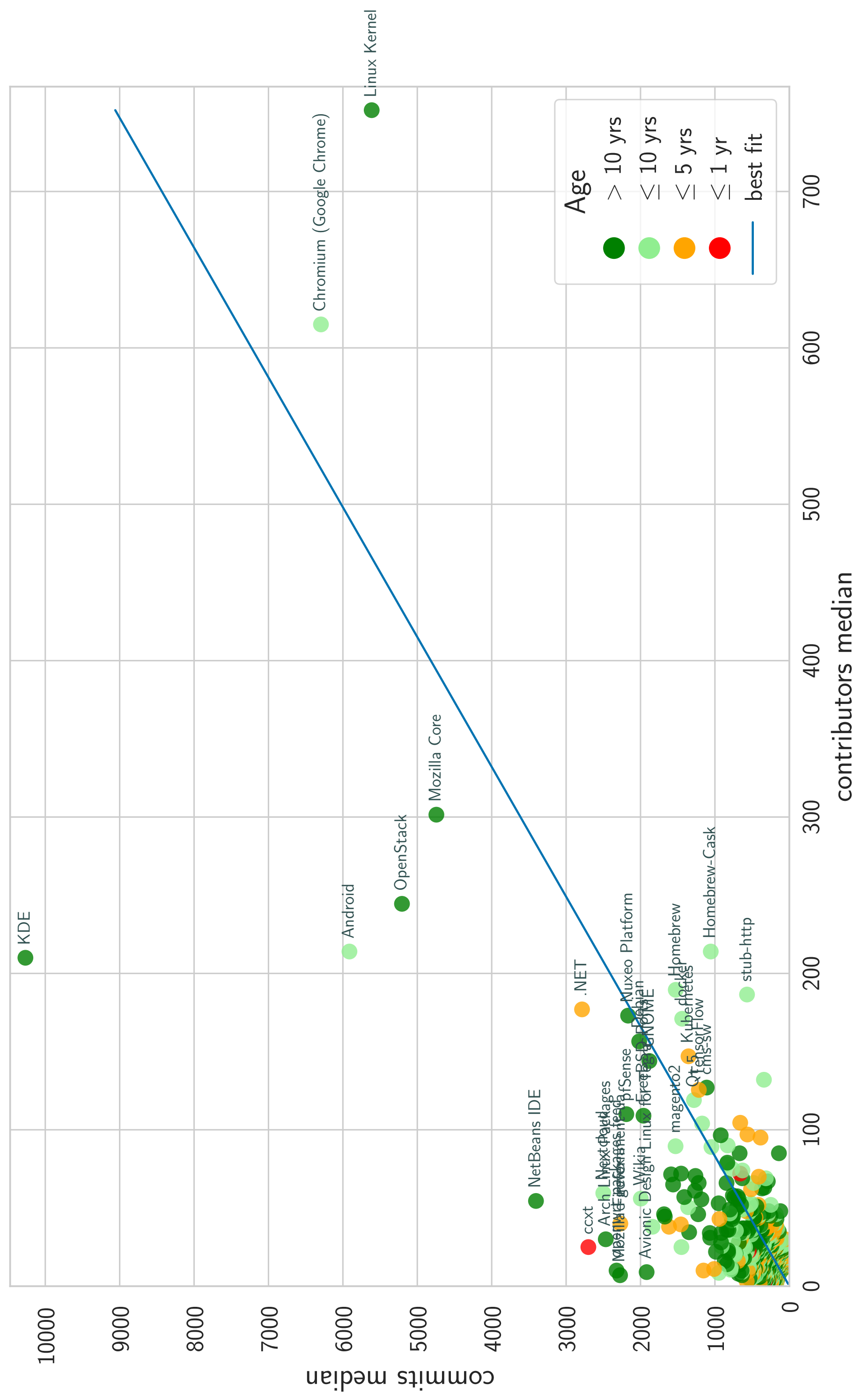
One general limitation to consider for the given data set is that commits and contributors are not independent variables but are, rather, closely correlated. Accordingly, the informational gain of using both is limited. Having additional data to connect commits to particular contributors should provide the possibility for interesting features, such as the presence and size of a highly active core development team.

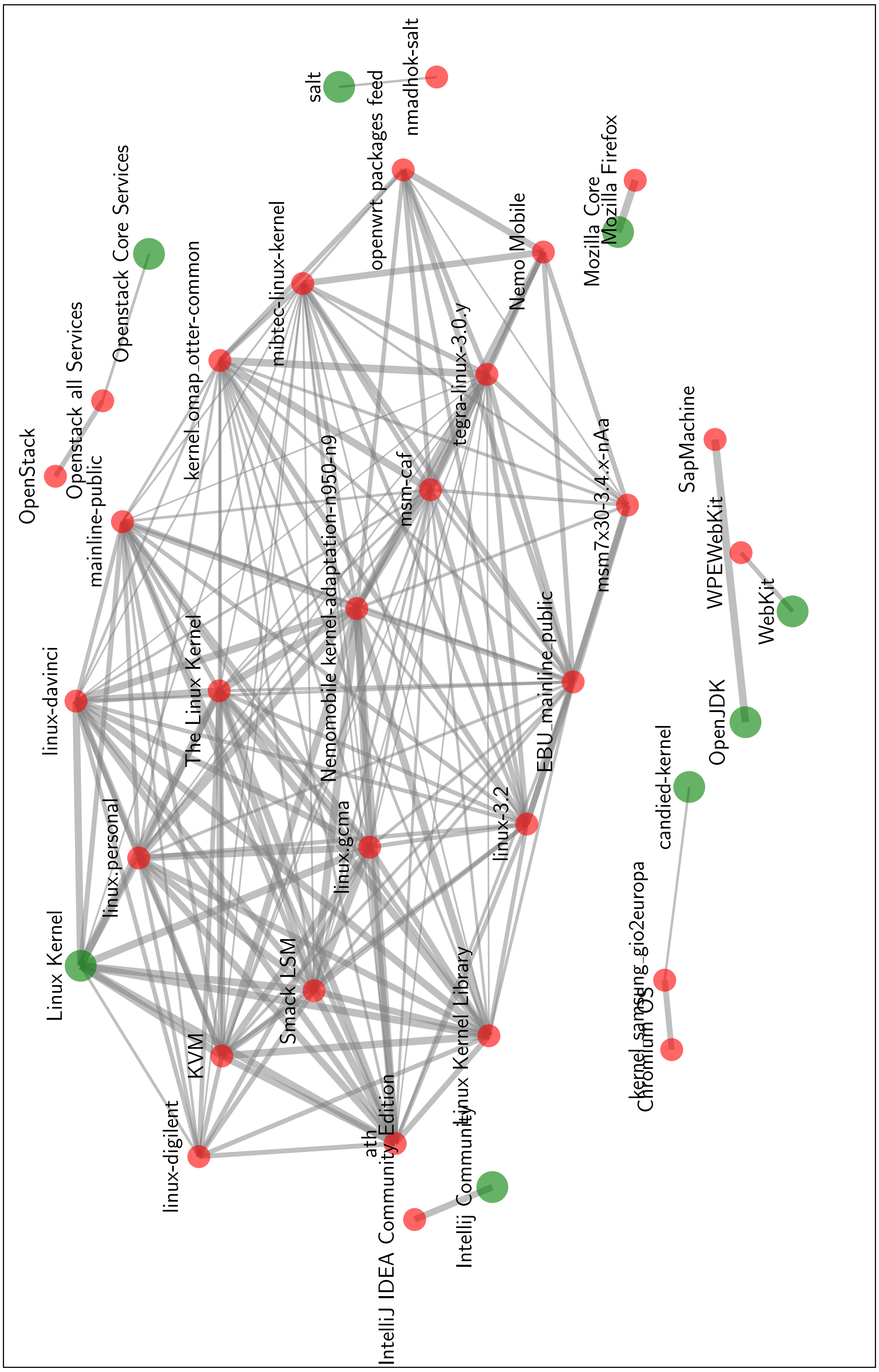
Another limitation we discovered concerns Daffara's definition of when a project is active. It could lead to implausible behaviour under certain conditions. For example, a pattern often seen on large projects entails a drop in activity after either an initial hype or the project has reached a stable state. The number of commits shrinks as it gets mature and transitions to the maintenance phase. These drops can be more than the 60% threshold that Daffara has used in his definition. Therefore, even some large projects such as Debian, which still has over one hundred contributors per month, turned out as inactive by the definition at times.

2.8 Conclusion

In this work, we found a way of identifying and linking forked projects to their originals based on commit data. Furthermore, we designed a ranking metric with which we defined the most successful projects. Based on that, we explored predictive features of success and found the *initial contributors* and *active months* to be the most promising out of the predictors we tested.
The prediction results of the GNB turned out rather poor. More optimization is needed. However, this also might not be enough for a reliable predictive model purely based on commit data. Moreover, there are too many other factors that have a huge impact on a project's success, as described by Ghapanchi et al. (2011). Nevertheless, using features derived from commit data as part of a predictive model seems worthwhile. With the tools and findings provided, this work has provided a solid foundation for further exploration of predictors in commit data – and eventually the development of an predictive model of success for open source software projects.

Appendix A Enlarged Plots





OpenStack
Openstack all Services
Openstack Core Services
Opencv-public
mainline-public
kernel_omap_otter-common
mibtec-linux-kernel
salt
nmadhok-salt
openwrt packages feed
Nemo Mobile
Mozilla Core
Mozilla Firefox
tegra-linux-3.0.y
msm-caf
msm7x30-3.4.x-nAa
Nemomobile kernel-adaptation-n950-n9
linux-davinci
The Linux Kernel
EBU mainline public
SapMachine
WPEWebKit
WebKit
OpenJDK
candied-kernel
linux.personal
linux.gcma
linux-3.2
Linux Kernel
Smack LSM
Linux Kernel Library
kernel_samsung_gio2europa
Chromium OS
KVM
linux-digilent
ath
Community Edition
Intellij Community
IntelliJ IDEA Community

Appendix B

Listing 2.1: Pseudocode - Duplicate Filter

```
duplicate_pairs = []

for every project pair in permutations:
        project1 = de-trend(project1)
        project2 = de-trend(project2)
        overlap = intersect(project1, project2)
        if overlap < 6 months or overlap < 33%:
                ignore

        coefficent = correlate(project1[overlap], project2[overlap])
        if coefficent > 0.98:
                duplicate_pairs.append(project1, project2)

duplicate_cluster = link all duplicate_pairs
```

Table 8: Top 100 projects in ranking

	name	commits median	contrib. median	ranking score
0	Linux Kernel	5,613.00	752.00	7,576.41
1	Chromium (Google Chrome)	6,295.00	615.00	6,944.87
2	KDE	10,267.00	210.00	5,937.21
3	Mozilla Core	4,743.50	301.50	4,132.46
4	Android	5,912.00	214.00	4,051.11
5	OpenStack	5,206.00	244.50	3,948.24
6	.NET	2,786.00	177.00	2,426.27
7	Nuxeo Platform	2,167.00	173.00	2,127.14
8	Homebrew	1,526.50	189.50	1,957.89
9	Debian	2,018.00	156.50	1,949.53
10	Homebrew-Cask	1,056.00	214.00	1,917.71
11	NetBeans IDE	3,405.00	54.50	1,865.62
12	GNOME	1,881.00	144.00	1,804.39
13	docker	1,440.00	171.00	1,794.15
14	pfSense	2,191.00	110.00	1,709.49
15	FreeBSD Ports	1,960.00	109.00	1,601.21
16	Kubernetes	1,355.00	147.00	1,593.69
17	stub-http	568.50	186.50	1,516.62
18	Nextcloud	2,500.50	59.50	1,502.22
19	TensorFlow	1,216.00	125.50	1,386.49
20	Qt 5	1,282.00	119.00	1,371.31
21	ccxt	2,701.00	25.00	1,355.82
22	cms-sw	1,110.50	127.00	1,350.33
23	Arch Linux Packages	2,467.00	30.00	1,287.00
24	magento2	1,529.00	89.50	1,279.32
25	e-government-ua	2,271.00	40.00	1,268.86
26	Wikia	1,996.00	56.00	1,256.79
27	Boot To Gecko	1,172.00	104.00	1,221.03
28	PotPlayer	1,589.00	71.50	1,183.33
29	Liferay Portal	1,454.50	72.00	1,127.64
30	LibreOffice	1,562.00	65.00	1,127.29
31	openwrt packages feed	2,322.00	10.00	1,087.36
32	OroPlatform	1,839.00	38.00	1,065.48
33	salt	1,047.00	89.00	1,064.16
34	ResourceLib: C# File Resource Management	922.00	96.50	1,060.22

Continued on next page

Table 8: Top 100 projects in ranking

	name	commits median	contrib. median	ranking score
35	Intellij Community	1,684.00	46.00	1,051.75
36	tigerbrew	341.50	132.00	1,046.47
37	Mozilla Firefox	2,275.00	7.00	1,046.32
38	IntelliJ IDEA Community Edition	1,673.50	44.50	1,036.95
39	Eclipse IDE for Java	1,261.50	70.50	1,032.66
40	OpenBSD	1,412.00	57.00	1,007.02
41	terraform-providers	660.00	104.50	999.49
42	WebKit	1,218.50	66.00	983.18
43	OpenDaylight	828.50	90.00	974.96
44	Fuchsia OS	1,268.00	61.00	970.94
45	grpc	1,617.00	38.00	967.94
46	DART language	1,359.50	50.50	939.78
47	OroCommerce	1,458.00	39.50	908.29
48	pytorch	563.00	97.00	905.90
49	Avionic Design Linux for Tegra	1,918.00	9.00	903.08
50	FreeBSD	833.00	79.00	902.18
51	Rust (programming language)	844.00	78.00	900.21
52	WebKitForWayland	1,181.50	55.50	895.56
53	Ansible	668.00	85.00	870.47
54	ownCloud	1,225.00	46.00	850.10
55	Open edX	767.00	75.00	845.99
56	MariaDB	1,346.50	34.50	825.32
57	Odoo	844.00	66.00	818.65
58	react-native	387.00	95.00	814.98
59	hifi	1,450.00	25.00	806.22
60	open-liberty	661.00	72.00	779.03
61	SlackBuilds.org	630.00	74.00	779.01
62	codership-mysql	950.00	53.00	776.86
63	GNU Compiler Collection	633.00	69.00	746.34
64	NetBSD	759.00	58.00	726.93
65	MySQL	738.00	57.00	710.91
66	Paddle	939.00	43.00	704.06
67	llvm-or1k	1,069.00	34.00	700.00
68	LLVM/Clang C family frontend	699.00	57.50	697.17
69	pkgsrc: The NetBSD Packages Collection	799.00	49.00	683.33

Continued on next page

Table 8: Top 100 projects in ranking

	name	commits median	contrib. median	ranking score
70	cheri-clang	682.00	56.00	679.51
71	Zend Framework	1,063.50	31.00	677.20
72	HipHop Virtual Machine for PHP	799.00	46.50	666.34
73	OpenStack Nova	480.00	66.00	658.73
74	wp-calypso	411.00	70.00	655.61
75	mulle-clang	705.00	50.00	648.83
76	.NET Core Runtime	515.00	62.00	646.92
77	OpenPandora-Console	140.00	85.00	638.50
78	Boost C++ Libraries	648.50	52.00	637.61
79	Cloud Foundry	758.50	44.50	634.96
80	PHP	914.50	34.00	632.13
81	The LLVM Compiler Infrastructure	768.00	43.00	628.94
82	WildFly	776.50	42.00	625.87
83	Talend	651.00	49.50	621.71
84	MediaWiki	840.00	36.00	612.99
85	Symfony2	630.00	49.00	609.09
86	SBo-git	316.00	69.00	607.07
87	Exherbo	817.00	36.00	602.88
88	CiviCRM	789.50	37.50	601.00
89	MediaWiki extensions hosted by WMF	917.50	28.00	592.66
90	Project Atomic	532.00	52.00	586.42
91	Laravel	516.00	53.00	586.19
92	OpenJDK 9	368.50	62.50	585.96
93	Kokua Viewer	986.00	22.00	581.98
94	vbatts's SlackBuilds	289.00	67.00	581.62
95	Microsoft Azure PowerShell Cmdlets	581.00	48.00	580.76
96	Ruby on Rails	336.00	63.00	575.08
97	YetiForceCRM	1,153.50	10.00	574.00
98	PrestaShop	911.00	25.50	572.82
99	Bitrig	533.00	49.00	566.47

References

Amar Krishna, A. A. & Choudhary, A. (2016). Predicting the outcome of startups: Less failure, more success., IEEE.

Black Duck Software, Inc. (2014). Open Hub. Retrieved September 23, 2018, from https://www.openhub.net/

Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing.

Daffara, C. (2007). Estimating the number of active and stable floss projects. Retrieved 2018, from http://robertogaloppini.net/2007/08/23/estimating-the-%20number-of-active-and-stable-floss-projects

Ghapanchi, A. H., Aurum, A. & Low, G. (2011). A taxonomy for measuring the success of open source software projects. *First Monday*, *16*(8). doi:10.5210/fm.v16i8.3558

Godfrey, M. & Tu, Q. (2001). Growth, evolution, and structural change in open source software. (pp. 103–106). IWPSE '01. doi:10.1145/602461.602482

Hofmann, H., Wickham, H. & Kafadar, K. (2017). Letter-value plots: Boxplots for large data. *Journal of Computational and Graphical Statistics*, *26*(3), 469–477. doi:10.1080/10618600.2017.1305277. eprint: https://doi.org/10.1080/10618600.2017.1305277

Kirchgässner, G., Wolters, J. & Hassler, U. (2008). *Introduction to modern time series analysis*. Springer.

Koch, S. (2007). Software evolution in open source projects – a large-scale investigation. (Vol. 19, *6*, pp. 361–382). doi:10.1002/smr.v19:6

Kolassa, C., Riehle, D. & Salim, M. A. (2013). A model of the commit size distribution of open source. (pp. 52–66). Springer.

Krishnamurthy, S. (2002). Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*, *7*(6), 1–12.

Riehle, D., Riemer, P., Kolassa, C. & Schmidt, M. (2014). Paid vs. volunteer work in open source. (pp. 3286–3295). IEEE Press.

Robles, G., Amor, J., Gonzalez-Barahona, J. & Herraiz, I. (2005). Evolution and growth in large libre software projects. *8th International Workshop on Principles of Software Evolution*. IWPSE '05, *0*, 165–174. doi:10.1109/IWPSE.2005.17

Roy, C. K. & Cordy, J. R. (2006). Evaluating the evolution of small scale open source software systems. (Vol. 23, pp. 123–136). National Polytechnic Institute, Mexico.

Succi, G., Paulson, J. & Eberlein, A. (2001). Preliminary results from an empirical study on the growth of open source and commercial software products. (pp. 14–15).

Tukey, J. W. (1977). *Exploratory data analysis.* Addison-Wesley.