Repo Velocity: Diagnosing Git Repository Health

Tyler Brown
NUID: 001684955

1 Introduction

Understanding the health of open source projects is important to industry because it helps them assess risk associated with their technology stack. Forecasting health is important to investors because this information can help them make profitable investments in open source. Academia is interested to know if there are links between theory, such as programming language design [1], and the health of open source software. A previous metric used to assess open source software health is the Truck Factor [2]. Truck Factor is the smallest subset size of developers who contributed 50% of the code in an open source project. The underlying intuition is that open source projects with a lower Truck Factor are more susceptible to project disruption in the event of adverse circumstances.

By borrowing from Physics, this paper [1] contributes to the advancement of understanding the health of open source projects, studying a subset of GitHub repositories, with the following:

- Introducing a health measure that can be assessed at time $t_{i\pm k}$ where k is a multiple of i.
- New health measure can be used in forecasting as well as description
- Health measure is rooted in Physics so we can derive related measures using preexisting theory.

2 Related Work

Truck Factor reflects robustness of project [2] by computing the minimum number of developers required to comprise 50% of file ownership. The underlying intuition is that a project with a low number of dominant developers will be more susceptible to failing due to external shocks. Projects where file ownership is shared by a large number of developers are interpreted as more healthy with this approach.

Not all projects reflects software which requires support. Researchers have classified many project types such as 'software development projects', 'solutions for homework', 'projects with educational purposes', 'data sets', and 'personal web sites' [3]. The observation that many projects are not software development is also a noted peril when analyzing GitHub [4]. We may also find repositories containing code duplicates [5]. These factors may comprise threats to validity which are important to address in this analysis.

3 Methods

The Truck Factor assesses Git repository health at time t_i. We would like to better understand health at different points in time $t_{i\pm k}$ where $k \geq 1$. We can borrow from Physics to find such a measure. Velocity, $v(t)$, is a measure of distance over time. What is a proxy for distance in the context of Git

[1]Code available on GitHub: https://github.com/tbonza/EDS19

repository health? Total lines of code changed in a project during time t_i can tell us how quickly the code base is changing. We apply this definition of velocity, $v(t)$, for our new measure Repo Velocity.

$$\text{Repo Velocity } \approx v(t) = \frac{d}{t} = \frac{\text{lines added } - \text{ lines deleted}}{t} \qquad \forall i \pm k \in t | i \in \mathbb{R}, k \in \mathbb{R}_{>0} \quad (1)$$

Equation 1 defines our target measure, Repo Velocity, and posits that the Physics concept of 'distance' can provide insight into how a GitHub repository is changing over time. We can compute velocity, $v(t_i)$ at time t_i where i is a real number. We can also compute average velocity, $v_{avg}(t_{i \pm k})$, at time t_i to time $t_{i \pm k}$. The Repo Velocity measure also has the advantage of allowing aggregation and disaggregation. Truck Factor is a measure which can only be analyzed at an aggregate level. There can be a lot of interesting variation below a Repository level aggregation. Repo Velocity can also be used to capture individual contributor velocity. The ability of Repo Velocity to be applied over time as well as at aggregated and disaggregated levels make the measure a unique contribution to the understanding of Git repository health. The remaining methodology seeks to establish Repo Velocity as an informative descriptive and predictive target or dependent variable.

3.1 Establishing Baseline Repo Velocity

In our effort to establish the Repo Velocity measure, we first examine behavior on Git repositories known to be either (a) active and currently in development, (b) previously active and not currently in development. The repositories I chose for the baseline subset are as follows:

1. torvalds/linux (a) - Linux kernel, a fundamental component in many cloud computing server operating systems such as Amazon Linux.
2. docker/docker-ce (a) - Community edition of Docker including the open repositories Docker Engine and Docker client, a popular enterprise application container platform currently heavily utilized in projects such as Kubernetes.
3. apache/spark (a) - Unified analytics engine for Big Data. Allows for an alternative to Map/Reduce and is currently widely favored by industry. One of the most active projects in the Apache Software Foundation [6].
4. apache/lucene-solr (a) - Mirror of Apache Lucene + Solr. Apache Lucene is a high-performance, full featured text search engine library written in Java. Apache Solr is an enterprise search platform written using Apache Lucene. Notably, Lucene is a major dependency for Elasticsearch [7]
5. apache/attic-lucy (b) - Mirror of Apache Lucy, search engine library provides full-text search for dynamic programming languages. Previously active project in the Apache "attic".
6. apache/attic-wink (b) - Apache Wink, simple yet solid framework for building RESTful Web services. Previously active project in the Apache "attic".

All GitHub repositories chosen for the baseline sample would be classified as 'software development projects', or *DEV*[2] using the ClassifyHub designations [3]. This paper establishes baseline Repo Velocity expectations using descriptive and predictive (forecasting) methodologies.

3.1.1 Descriptive Baseline

We establish a descriptive baseline for the Repo Velocity measure by examining changes the measure over time (2013-2018) across our baseline sample. Figure 1 shows diverging behavior within the baseline sample for repositories which are (a) active and currently in development, or (b) previously active and not currently in development. The common trend with (b) type repositories is that Repo Velocity is converging to zero. Apache Lucy (b), shows a decrease in velocity then a convergence to zero between 2013 and 2015. Apache Wink (b) has already converged to zero within the 2013-2018 time window; a longer term window shows similar behavior to Apache Lucy.

[2]The baseline distribution containing only DEV repositories is a fundamental part of the results presented in this paper.

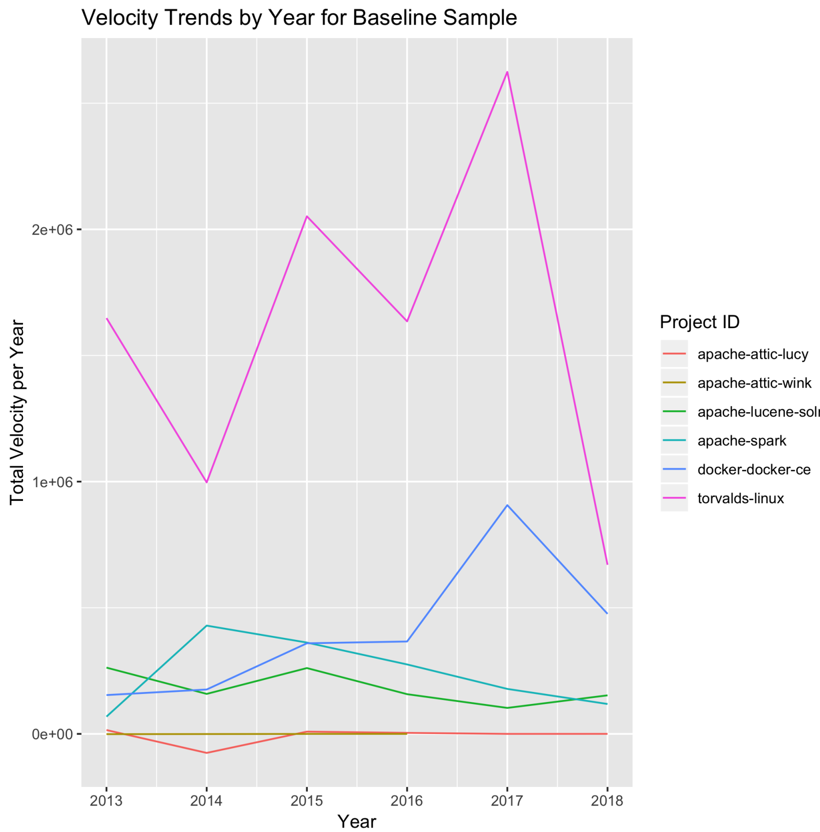

Figure 1: Repo Velocity Trends for Baseline Sample

Type (a) projects show considerable variance in Repo Velocity, such as 'torvalds/linux'. However, on average, we can see that Repo Velocity is not converging to zero. There is a clear and interesting trend with Apache Spark that is showing a downward trend in Repo Velocity between 2014 and the end of sample at December 2018. This downward trend may reflect a peril of mining GitHub [4], in that not all repositories are projects. For example, Yahoo is currently developing TensorFlowOnSpark [8] which allows Tensorflow programs to operate using Apache Spark clusters. The Yahoo project is directly competing with an offering by Google Cloud which does not appear to involve Spark [9]. We can see that GitHub repositories not representing the scope of a project is a potential threat to validity we must consider here.

Trends viewed in this descriptive baseline helped to inform the following hypothesis:

$H1_a$: As the absolute value of velocity diverging from zero (acceleration) increases, Git repo health will also increase.

Hypothesis $H1_a$ should hold when reviewing the predictive baseline, as well as at scale.

3.1.2 Predictive Baseline

The predictive baseline uses Repo Velocity as a target or dependent variable and seeks to explain which features contribute to this measure. This paper posits the following features play a strong role in explaining Repo Velocity:

- Number of authors in a given time period.
- Number of mentors; number of authors in a year that have made monthly commits for at least 6 months.
- Number of organizations; organization affiliations taken from authors email addresses (i.e. RedHat).

Figure 2 shows that these features are almost perfectly correlated in the baseline sample containing only 'software development type', DEV, projects. The almost perfect correlation between features means that we must adjust the forecasting model for multicollinearity so that predictions are not higher at the expense of not being able to control for possible intervening features during future analysis.

	# Authors	# Organizations	# Mentors
# Authors	1.0	1.0	0.99
# Organizations	1.0	1.0	0.99
# Mentors	0.99	0.99	1.0

Figure 2: Correlation Matrix for Features in Baseline Sample

The high collinearity of these features, seen in Figure 2, provides us with an expectation of a linear relationship best captured by a Linear Regression model. Equation 2 defines the linear model.

$$\hat{y} = \alpha + \beta_{1,\#Authors}x_1 + \beta_{2,\#Organizations}x_2 + \beta_{3,\#Mentors}x_3 + \epsilon \tag{2}$$

The linear model in Equation 2 is based on correlations of trends visually observed in Figure 1. We should not consider Equation 2 as a model which can establish causality.

3.2 Validating Repo Velocity at Scale

In order to validate the Repo Velocity measure at scale, we want to use a subset of GitHub repositories that have a higher probability of being classified as 'software development projects', DEV. This paper assumes that all projects posted on GitHub by these organizations are intended to be classified as actively maintained DEV projects [3]. The repositories were collected from: Apache Foundation, edX Online Learning, Facebook, Google, Microsoft, and Tensorflow (deep learning framework from Google). We will know if Repo Velocity is supported at scale if the descriptive trends and predictive modeling results associated with the baseline set are able to be replicated using Git repository data collected from these organizations.

3.3 Data Engineering

A majority of the work for this analysis went into Data Engineering. To streamline this work, the author wrote a Python package called Okra [4]. Data Engineering involved two phases of (1) data collection, and (2) data processing.

Repository names were retrieved using SQL queries from the GHTorrent relational database [5] hosted on Google BigQuery. GitHub repository names were compiled into a small text from the organizations: Apache, edX, Facebook, Google, Microsoft, and Tensorflow. Relevant information was then extracted using the 'git log' utility and processed using Okra.

3.3.1 Collecting Data

Initial exploratory efforts were undertaken to see if downloading all repositories from GitHub would be possible. This required an architecture which could clone and process the GitHub repositories in parallel.

2,869 repos collected and processed; mention the ones that failed

3.3.2 Processing Data

using Apache spark, using "hack/reduce"

[3] Later work should absolutely question this assumption.
[4] Distribution: https://pypi.org/project/okra/, Development: : https://github.com/tbonza/EDS19
[5] http://ghtorrent.org/gcloud.html

4 Experiment

4.1 Results from Baseline Repo Velocity

4.1.1 Results from Predictive Baseline

The training/test/validation sets are broken out by year 2015/2016/2017 rather than pooling time and selecting observations randomly. We expect strong trends over time so random selection wasn't used when specifying the training/test/validation sets. Results from the linear model specified by Equation 2 are reported in Figure 3.

Dataset	Features	r^2 Value
Test	All	0.8291
Validation	All	0.9253
Test	# Authors	0.8104
Validation	# Authors	0.8726

Figure 3: Test/Validation Baseline Set Results from Equation 2

4.1.2 Results from Scaled up Prediction

4.2 Results from Repo Velocity at Scale

4.2.1 Modeling Outliers

bad

5 Discussion

had mixed results, should have better classified input GIGO

5.1 Threats to Validity

yup

References

[1] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[2] Guilherme Avelino, Marco Tulio Valente, and Andre Hora. What is the truck factor of popular github applications? a first assessment. Technical report, PeerJ PrePrints, 2015.

[3] Marcus Soll and Malte Vosgerau. Classifyhub: An algorithm to classify github repositories. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 373–379. Springer, 2017.

[4] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[5] Cristina V Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. Déjàvu: a map of code duplicates on github. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):84, 2017.

[6] Apache. The apache software foundation announces apache™ spark™ as a top-level project : The apache software foundation blog. `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50`, Feb 2014. (Accessed on 04/26/2019).

[7] Apache. elastic/elasticsearch lucene server dependency. https://github.com/elastic/elasticsearch/blob/0dca576a35fa6fb199ae8dbb7d01603c21991f7e/server/build.gradle, Apr 2019. (Accessed on 04/26/2019).

[8] Yahoo. yahoo/tensorflowonspark: Tensorflowonspark brings tensorflow programs to apache spark clusters. https://github.com/yahoo/TensorFlowOnSpark, Apr 2019. (Accessed on 04/26/2019).

[9] Google. Running distributed tensorflow on compute engine | solutions | google cloud. https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine, Apr 2019. (Accessed on 04/26/2019).