Automatic Project Rator

Goal, approach, anticipated results and related work

Hebi Li

November 11, 2016

1 goals of the project

Research into the possibility of predict project quality by only textual features. There are numerous projects hosted on Github. However, open source projects have different qualities. Projects developed by in-experienced developers tend to have lower quality compared to projects with high reputation. Knowing the quality of a project is beneficial to developers, students interested in learning the project, as well as the potential users. Judging the quality of a project is non-trivial, and may require manual effort and domain knowledge.

This work aims to provide a general prediction model for the quality of projects, based on their textual features.

2 Proposed Approach

We have collected projects with different quality via Github API. We collect 1000 top rated project for each of the following languages: c, java, javascript, python, ruby, php, c++, c#, objective-c, shell. The rating ranging from hundreds of stars to over one hundred thousand of stars. The different programming languages gives us a broader view of the potential properties of differnet communities. Also, we collect the data by the order of rating, thus do not suffer from bias caused by crawling by users relationship. The current projects number in total is 10,000, if we find this is not enough, we might use the crawling approach but with bread-first approach, to be as unbiased as possible.

Projects can be rated by different metrics: a) download times b) lifetime length c) commit number d) a rating system: e.g. github star e) number of contributors f) number of watchers. Specifically, we propose to mine the meta data from github as our corpus, and use the number of stars as the quality ground truth. We may also use other metrics, such as download times, along or combined together. Considering the huge number of projects on github, we may choose to set to threshold, i.e. only projects with at least certain number of stars will be included into the corpus.

We propose to use mainly the file-level and textural features to predict the quality. The proposed features include: 1) number of tests 2) number of documentation 3) number of source files 4) total source line of code 5) total source line of code in source directory (excluding libraries) 6) average length of source files 7) every depth of the file structure 8) average branching factor of the directories.

Some attributes of a project may affect the effectiveness of these metrics. For example, the language used in the project may affect the rating model. C languages tend to have many standalone test files, while Java projects have unit test method mixed into source code. There are also features that are hard to collect the may influence the precision of the ground truth we selected. For example, a lot of javascript projects on github have many stars, partially because they have a

dedicated webpage as a demo of their project. It is interesting to see the difference induced by these attributes.

2.1 Classifiers

The second phase of this work is to fit a statistic model for the data we collect.

2.1.1 Logistic Regression

We apply several machine learning classifiers on the features extracted from training data set. Logistic regression[**?**] can be used to learn the model. The probability of the output to the problem is true is modeled by

$$p(X) = Pr(Y = 1|X) \tag{1}$$

We have p features, thus

$$log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p \tag{2}$$

Where $X = (X_1, ..., X_p)$ are p predictors for p features. This equation can be rewrite so that our *logistic function* is

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_p X_p}} \tag{3}$$

To fit this model, we use the *maximum likelihood*[**?**, p. 130] method to estimate $\beta_0, ..., \beta_p$.

2.1.2 Support Vector Machine

SVMs have been shown to perform well in a variety of settings, and are often considered one of the best "out of the box" classifiers [**?**, p. 337].

In a p-dimensional space, a *hyperplane* is a flat affine subspace of dimension $p - 1$.

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0 \tag{4}$$

defines a p-dimensional hyperplane, which classifies the space into two. To fit the model, a natural choice is the *maximal margin hyperplane*. But it does not work well because it is not stable. The following figure taken from [**?**, p. 345] is a good explaination of why it is not working.

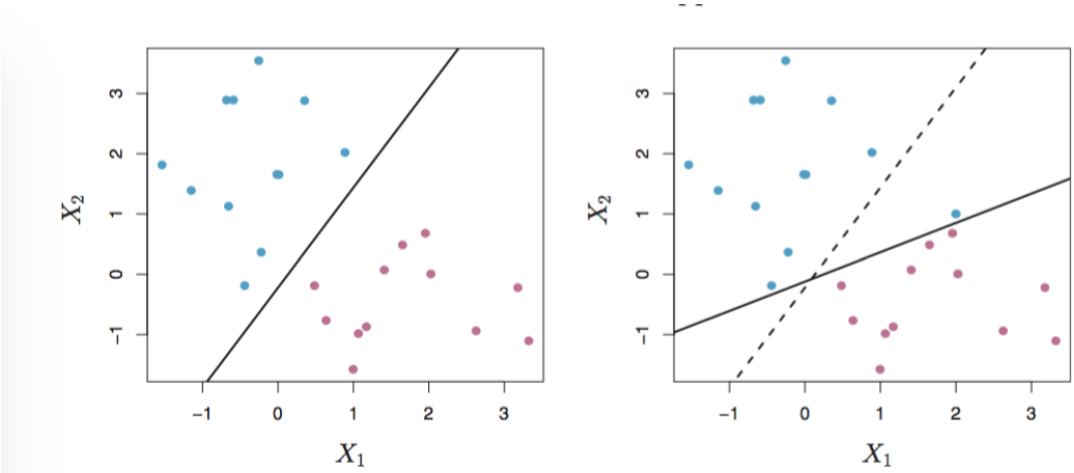

FIGURE 9.5. Left: *Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane.* Right: *An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.*

Support vector classfier, sometimes called a *soft margin classifier* want the data not only on the correct side of the hyperplane but also on the correct side of the margin. Thus it is 1) greater robustness 2) better classification.

The *Support Vector Machine* is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using *kernels*. Formally the classifier function is

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \tag{5}$$

where $K(x_i, x_{i'})$ is the *kernel*, which quantifies the similarity of the two observatiosn x_i and $x_{i'}$. β and α are parameters, S is the collection of indices of these support points.

We list several kernels we are intereted to experiment:

- Linear Kernel.

$$K(x, x_i) = x * x_i \tag{6}$$

This kernel is very efficient when dealing with large sparse data vectors as is usually the case in text cate- gorization.

- Polynomial

$$K(x, x_i) = (\gamma * x * x_i + coeff)^{deg} \tag{7}$$

This kernel is generally used in classification of images.

- Radial Basis

$$K(x, x_i) = exp(-\gamma|x - x_i|^2) \tag{8}$$

This is a general purpose kernel and is typically used when no further prior knowledge is available about the data.

- Gaussian Radial Basis

$$K(x, x_i) = exp(-\sigma|x - x_i|^2) \tag{9}$$

This is a general purpose kernel and is typically used when no further prior knowledge is available about the data.

- Bessel

$$K(x, x_i) = \frac{Bessel^n_{(v+1)}(\sigma|x - x_i|)}{(|x - x_i|)^{-n(v+1)}} \tag{10}$$

This is a general purpose kernel and is typically used when no further prior knowledge is available about the data and is mainly popular in Gaussian process com- munity.

- Laplace Radial Basis

$$K(x, x_i) = exp(-\sigma|x - x_i|) \tag{11}$$

This is a general purpose kernel and is typically used when no further prior knowledge is available.

3 anticipated results

A model learned from 95% of the data to fit the rating of the projects. Using the 5% part to validate the precision and recall of the model. Or, if no model can achieve desired precision and recall, we want to be able to tell what's the potential issues and how can we learn from it.

4 Related Work

A large body of work has been done for mining interesting data from software repositories. In face, the conference Mining Software Repositories(MSR) is dedicated for this. Our work is most related to this body of work.

The most related work is OpenHub [5] by Farah and Tejada. The work presents a scalable and extensible architecture for the static and runtime analysis of open source repositories. They use more exepnsive techniques, and try to analyze the performance on a given of quality attributes. Their current analysis is limited to Python, while our technique aims to provide a way that is independent with programming languages, but instead rely on textual features only. Our model, once learned, can be used with little overhead. The learning process will be a one time cost.

A lot of work has been done on characterization the hosting [14] platform(like github [3, 7]), the code sharing and colaborating architecture [2, 19], as well as the project attributes [16, 11].

Contributors are the core of development, and they are the key to determine the quality of a project. Robles et al [13] conducted a survey about free software contributors to over 2000 people,

and conclude their findings for challenges of curating, sharig, and combing. Code review for the community [17, 8, 18] is well studied, showing the community-based characterization. Although open source projects are "open", the acceptance of contributions is strictly controlled [12]. This kind of characteristic might show interesting patterns in projects of different qualities. For example, is the higher quality projects tend to be hard to contribute for newbees? Or the high quality of projects are achieved due to an open mind to newcomers? Finally, an interesting feature that can be potential useful to us is the Microblog [15], a new trend to communicate and to disseminate information amoung open source communities.

One important textual feature is the documentation [10], and the quality of documentation of a project can somehow decide the quality of the project. Testing [6] is another crucial part of modern developement that produce predictable and stable code. Both doucmentation and test cases ensures the maintainability of the projects, which is especially critical for open source projects.

Our code technique can be used to help prioritize the repository for mining. For example, the recomandation system proposed by Diamantopoulos and Thomopoulos [4] can benefit from our approach by first prioritize the repositories by their predicted ratings, and provide a less overhead recommandation. Evolution study [9, 1] can also benefit from the predicted rating in the sense that the top rating projects often have more contributors, longer history, as well as high quality in-progress commit.

References

[1] Boris Baldassari and Philippe Preux. Understanding software evolution: The maisqual ant data set. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 424–427, New York, NY, USA, 2014. ACM.

[2] Olga Baysal, Reid Holmes, and Michael W. Godfrey. Mining usage data and development artifacts. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 98–107, Piscataway, NJ, USA, 2012. IEEE Press.

[3] Valerio Cosentino, Javier Luis, and Jordi Cabot. Findings from github: Methods, datasets and limitations. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 137–141, New York, NY, USA, 2016. ACM.

[4] Themistoklis Diamantopoulos, Klearchos Thomopoulos, and Andreas Symeonidis. Qualboa: Reusability-aware recommendations of source code components. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 488–491, New York, NY, USA, 2016. ACM.

[5] Gabriel Farah, Juan Sebastian Tejada, and Dario Correal. Openhub: A scalable architecture for the analysis of software quality attributes. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 420–423, New York, NY, USA, 2014. ACM.

[6] María Gómez, Romain Rouvoy, Bram Adams, and Lionel Seinturier. Mining test repositories for automatic detection of ui performance regressions in android apps. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 13–24, New York, NY, USA, 2016. ACM.

[7] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 384–387, New York, NY, USA, 2014. ACM.

[8] Daniel Izquierdo-Cortazar, Lars Kurth, Jesus M. Gonzalez-Barahona, Santiago Dueñas, and Nelson Sekitoleko. Characterization of the xen project code review process: An experience report. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 386–390, New York, NY, USA, 2016. ACM.

[9] Patrick Kreutzer, Georg Dotzler, Matthias Ring, Bjoern M. Eskofier, and Michael Philippsen. Automatic clustering of code changes. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 61–72, New York, NY, USA, 2016. ACM.

[10] Parisa Moslehi, Bram Adams, and Juergen Rilling. On mining crowd-based speech documentation. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 259–268, New York, NY, USA, 2016. ACM.

[11] Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. A large-scale study on repetitiveness, containment, and composability of routines in open-source projects. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 362–373, New York, NY, USA, 2016. ACM.

[12] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 332–335, New York, NY, USA, 2014. ACM.

[13] Gregorio Robles, Laura Arjona Reina, Alexander Serebrenik, Bogdan Vasilescu, and Jesús M. González-Barahona. Floss 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 396–399, New York, NY, USA, 2014. ACM.

[14] Megan Squire. Data sets: The circle of life in ruby hosting, 2003-2015. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 452–459, New York, NY, USA, 2016. ACM.

[15] Yuan Tian, Palakorn Achananuparp, Ibrahim Nelman Lubis, David Lo, and Ee-Peng Lim. What does software engineering community microblog about? In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 247–250, Piscataway, NJ, USA, 2012. IEEE Press.

[16] Harold Valdivia Garcia and Emad Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 72–81, New York, NY, USA, 2014. ACM.

[17] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. Mining the modern code review repositories: A dataset of people, process and product. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 460–463, New York, NY, USA, 2016. ACM.

[18] Alexey Zagalsky, Carlos Gómez Teshima, Daniel M. German, Margaret-Anne Storey, and Germán Poo-Caamaño. How the r community creates and curates knowledge: A comparative study of stack overflow and mailing lists. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 441–451, New York, NY, USA, 2016. ACM.

[19] Jiaxin Zhu, Minghui Zhou, and Hong Mei. Multi-extract and multi-level dataset of mozilla issue tracking history. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 472–475, New York, NY, USA, 2016. ACM.