Evaluating Solution Quality and Problem Difficulty Utilizing Code Metrics

Gurpreet Singh (# *250674134*) GSINGH95@UWO.CA
The University Of Western Ontario

Abstract

Discovering the effects of code style on code functionality and problem structure. This study aims to showcase how well formatted, reusable and maintainable code is fundamentally better in all circumstances by looking at over 1 million results from a competitive programming website and analyzing the correlation between question and solution. The goal is to promote code quality checking in online 'just in time' teaching resources to improve the performance of interviewers, researchers, and students.

1. Description of Applied Problem

1.1. Problem identification

A large amount of educational material related to programming exists on the internet but the majority of which is not well structured or presented. An applied problem that can be observed from educational material found online is that code quality is often left mutually exclusive from code functionality. Astrachan (2004)

This leads to some students believing it is acceptable to write code that produces the correct result even if the process behind it is not correct. Online code challenge websites like CodeChef.com do not take into account the style and quality metrics of a code submission when judging competitions. CodeChef (2017) Cutting corners in the learning process advances into a complete disregard for best practices in open source software and in the workplace which results in a larger amount of errors. Readability of code is an essential metric in software engineering and can be improved even with simple additions of whitespace between lines. Buse and Weimer (2010)

Computer code written by researchers and other individuals who are just trying to accomplish a result in any way possible is often of the worse quality. This is because they learn using the 'just in time' mentality and the resources online that promote this mentality disregard code quality. Astrachan (2004) Lack of code quality directly reduces it's re-usability because other programmers have a harder time understanding what the code is doing. If these teaching resources could use questions and checks that promote better code quality, many technical innovations could be made. Researchers would be more educated on how to create reusable code and this would influence developers to take their ideas and apply them to real life use cases. Gandhi and Bhatia (2010)

Code quality post processing software is often used in production development environments to ensure good style choices. These checks are much less useful at this senior level than they would be at an educational level. If programming style can be judged on a submission, companies conducting technical interviews will be able to better judge applicants and make a more informed decision.

This study will focus on proving that code quality can have an influence on code functionality, as well as which kinds of questions influence good or bad code styles.

1.2. Approach Strategy

A solution to these problems is linking the scoring process in programming problems to a metric derived from running code quality checks on the submission.

Not only will this analysis benefit educational institutes but also companies and competitions that judge people on their code submissions.

Project report for CS4437/CS9637: Intro to Data Science.
University of Western Ontario, Winter 2017.

2. Data Description

2.1. CodeChef Dataset

CodeChef.com is a competitive programming web application that has posted all of their questions and solutions onto the data science website, Kaggle. The data consists of a questions comma separated file, a solutions comma separated file and 3 files that show the code associated with each solution id. The set contains about 1000 problem statements and over 1 million code solutions submitted. This should be a more than sufficient to make a training and test data set.

2.2. Code Submission Language Density

The code submissions are written in many different programming languages and each language has it's own code analysis tool. Therefore, to make the process simpler and come up with higher quality results, the data will need to be filtered by the top languages used. Figure˜1 shows that C++, Java, C and Python are the most popular submissions in this dataset. The majority of the dataset is describing submissions in the C family of languages. They are also the easiest to group together and process allowing for fair comparison of results, therefore, this study will focus only on the C family. This reduces the dataset to just under 600000 rows of valid data.

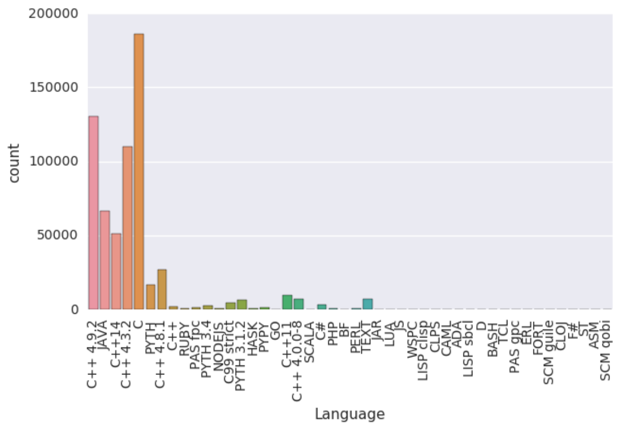

Figure 1. Frequency of each programming language that occurs in the dataset of solutions

Wang (2016)

2.3. Feature Selection

2.3.1. Question Data

The most useful features in the question data files are question ID, title, difficulty level, question statement, and time limit. This data will need to be cleaned by removing some common text that exists in every question. For example: "All submissions for this problem are available". Then, this data can be inner joined to the solutions dataset with the key 'Question ID' to determine insights about how the questions effect code style.

2.3.2. Solution Data

The most useful features in the solution data files are solution ID, status, time taken, memory taken, and language. First this data will be filtered to contain only the C family of languages. The C family is any language that starts with a C followed by a space. Then solution ID will be used to outer merge data to the relevant code data provided in separate files so we have more information about each individual solution.

2.3.3. Code Data

The code data files only contain two columns, the first one containing the solution ID and the second containing the code string. The solution ID will be used to merge the data with the solution data file and that data frame will act as the base for the rest of the analysis on code.

In order to make the connection between code functionality and code style, a large part of the experiment is obtaining an accurate and unbiased metric for code quality. The decision to narrow the dataset to only the C family shows it's importance here as we can use a single tool to evaluate quality

The tool we will be using is `cpplint`. It is open source and follows Google's C style guide. We wrote a ruby script to run this tool on each row of the merged solutions + code dataset and add another feature to each instance. The feature is an integer representing the number of code style errors that exist in the code.

3. Analysis

3.1. Experimental Methods

The main relationship we are interested in is the effect of Quality on the success rate. To begin to solve that problem we first look at the success rate grouped by UserID. Figure 2 shows that large chunks of users are placed at very low success and very high success. This is what we would expect to see if code quality was influencing results because users who write better code would be correct more often. Other factors may be causing this behaviour so we continue analyzing the data.

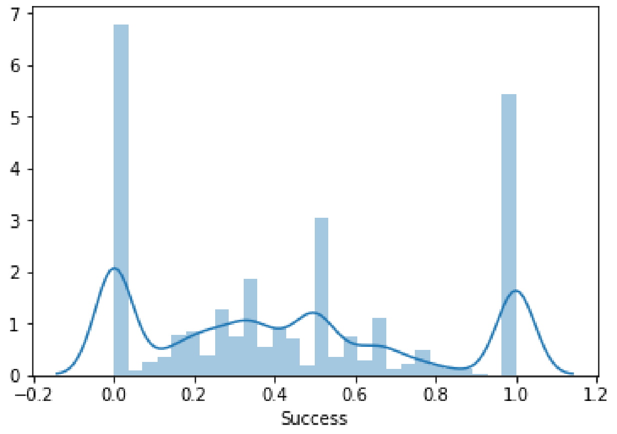

Figure 2. Histogram showing the distribution of users and their success rates

Next, we will see if you can predict the amount of style errors based on weather a solution will be "accepted" or end in one of the many error codes. This will be done by creating a model using linear regression so we can do predict the Quality value based on the status feature. Before creating the model the data was split into 2 parts for training and testing. The training data frame was 75% of the data and the testing data frame was the remaining 25%. Figure 3 shows a visual representation of the binary classification. It is clear that the unaccepted answers had more code style errors in their submissions.

After developing the model, the test set was used to determine the models error. Root means squared error was used for error calculation so there is extra weight given to large errors. The error value we received for this model was 76. The dependent variable was 'Number of Errors' therefore this error is low because RMSE has the same units as the dependent variable. The RMSE of the training data frame is 78 which leads us to believe the model is not over-fitting. This shows that given a correct or incorrect solution our model can guess how bad the code style will be.

Next, we will use logistic regression to predict the chance that a solution will be correct based on the Quality feature. After using `glm` with Quality and Success features, the summary of the model showed that Quality is statistically significant with a low p-value. The same 75% training data 25% testing data split was used in this model as well. Using the testing set to predict and calculate error using MSE with a 0.5 decision boundary, the error returned was 0, therefore, the model was unsuccessful.

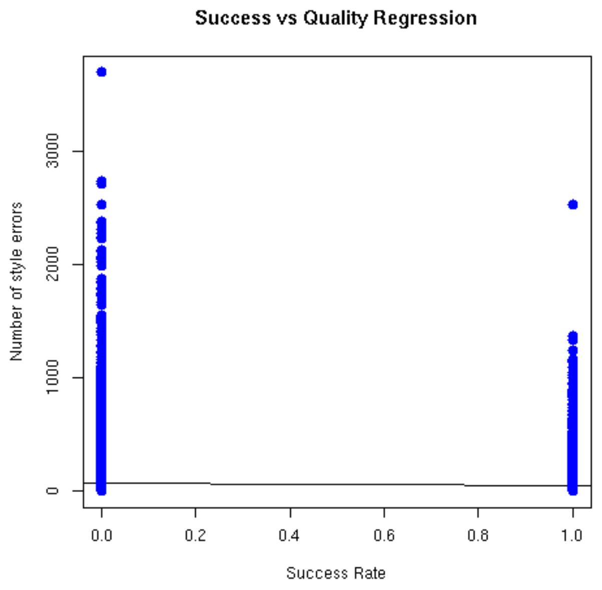

Figure 3. Predicting Quality based on Success

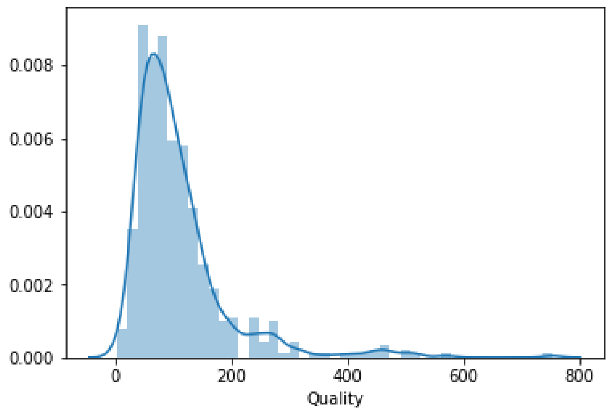

Figure 4. Histogram showing frequency of questions over quality

Since we now know about how solutions are effected by code quality, next we will see how different questions influence the outcome's code quality. Figure 4 shows a right skewed plot indicating that most of our questions resulted in submissions that had between 0 and 200 errors. This was generated by grouping by QCode and getting the mean of the Quality feature. Using that data, we can make 4 bins of size 50. Less than 50, 50-100, 100-150, and greater than 150 errors. Next the actual count of data is min-max normalized using preprocessing methods from `sklearn`. An issue that was encountered here was that we ran into NaN values in our data frame. We resolved this using the np.median of non-NaN instances.

After getting the data in a workable format, the first analysis we ran is K-Means clustering with cluster number of 3, and fit to the values of a data frame containing the normalized error count and the difficulty level of the question. Figure 5 shows points visually clustered into 5 main sections with 3 clusters. From this plot we can determine that the largest cluster is in the low to medium difficulty and has under 100 error. The second largest cluster is above medium difficulty and has the most points in the "bad" category of code quality (above 150). The third cluster is in the hard difficulty and has the highest absolute number of errors. After these 3 clusters we can see there is a drop off in code errors with even harder questions.

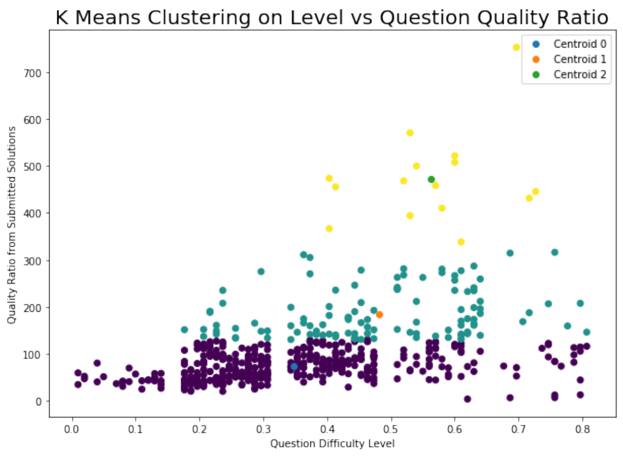

Figure 5. K means clustering comparing difficulty of a question vs its mean number of errors

4. Discussion & Conclusion

Figures 2 & 4 show how the users of code evaluation websites have very dispersed skill levels and questions are receiving answers with lots of code style errors. Figure 2 should ideally be a more normal distribution if all users had an equal learning curve. We feel this is achievable with more education about code quality because over time code quality is one of the most important building blocks for a programmer.

Figure 3 shows how the correct answers had much less code quality errors regardless of their level of difficulty. Our model was able to predict how many error would result by just knowing submission status which shows how closely these 2 features are related.

Figure 5 further proves the direct relationship between code quality and code correctness by showing a linear increase in quality mistakes as question difficulty gets harder. This model also shows how the number of quality mistakes decreases in the very hard difficulty category. The drop off is most likely caused by more experienced programmers in that segment. This could imply that programmers are able to tackle harder problems if they can keep their quality mistakes to a minimum

In conclusion, the analysis attempted in this study provide some evidence that modern code quizzing websites and other "just in time" teaching resources can enhance their material by adding code quality checks into their evaluation.

References

Owen L Astrachan. Non-competitive programming contest problems as the basis for just-in-time teaching. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages T3H–20. IEEE, 2004.

Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010.

CodeChef. Codechef competitive programming, October 2017. URL http://www.codechef.com. [Accessed Oct 26, 2017].

Parul Gandhi and Pradeep Kumar Bhatia. Reusability metrics for object-oriented system: An alternative approach. *International Journal of Software Engineering (IJSE)*, 1(4):63–72, 2010.

Justin Wang. Nlp and ml experiments, December 2016. URL https://www.kaggle.com/justwjr/nlp-and-ml-experiments/notebook. [Accessed Oct 26, 2017].