Research on Bug-fix Distance and Development Robustness

Group 04, CS212, LZU

1. Data Acquisition

We are interested in the relationship between bug-fix distance and development robustness. To find the distance, we need data of commits with fix tag and the commit history of the corresponding files and functions.

In terms of development robustness, our model will be based on the flowing hypothesis **(not the hypothesis to verify in the model)**: system development is more robust if its bug density is lower and new files and features are added more evenly over time.

We will not see much in Linux-stable as this is not where new features are actually developed. They are developed on the subsystems dedicated git tree and then during the merge window enter. Since we want to use the development speed of features, we will try and extract that from many subsystems at file level in Linux-next.

We have implemented four data extractor classes to extract the above indicators (The related files are in the repository https://github.com/neetneves/groupwork04).

(1) FixDistanceExtractor: get the bug-fix distance of the subsystem specified in the specified version range. Use regular expression to extract fix tags, commits, file names, and function names. Among them, the regex to find fix tags is "^\W+Fixes: [a-f0-9]{8,40} \(.*\)$", while others are not ambiguous.

(2) BugDensityExtractor: get the bug density of one file in the subsystem specified in the specified version range. The bug density is defined as the ratio of the number of commits with fix tag to the number of commits in total.

(3) FileAddExtractor: get the time difference of new files' creation in the subsystem specified. We get the history of files in the subsystem and then get the time of its first

commit. The get_timediff method returns the list of time-difference of creation of two next new files.

(4) FeatureAddExtractor: get the time difference of new features' addition in the subsystem specified in the specified version range. To get the commits adding features, we used key words "add feature" (maybe not so reasonable). And then we get the list of time-difference of addition of two next new features.

2. Data Preprocessing

For bug density, we cannot distinguish the add lines or delete lines are bug, so we use commit tags to replace the numbers of changing lines. In a file, if its history has no bug or fix commit tag, we consider that the file has no bug. According to it, we calculate the bug density of each file.

After we get data, we can know that many files have no relational tag. For those data we ignore and give up them because those are no meaning for us model. Some values are more than 100%, we also delete them because these values are error in theory. The reason why those values are more than one is maybe the history of submission is overwritten. Therefore, we delete them. To this point, we finish the data cleansing work for bug density.

3. EDA

Before modeling, we did EDA first. The result shows that data of the development speed of files and features is not useful at all. We randomly chose five subsystems to statistic on the development speed of their files and features, and we find that many of them does not have or just have one new feature or file added in the specified version (v4.4). The speed is expressed in the standard variance of time difference between two files/features' addition.

Files development speed: {'fs/afs/': nan, 'drivers/video/': nan, 'kernel/printk/': nan, 'net/nfc/': nan, 'tools/usb/': nan}

Features development speed: {'fs/afs/': nan, 'drivers/video/': nan, 'kernel/printk/': nan, 'net/nfc/': nan, 'tools/usb/': nan}

For features, this may be due to the wrong choice of regular expression. But if we use “feature” instead of “add feature”, we will get a lot of results, and the commit information shows that most of them are not adding new features. As for files, we look for the commit history of each file and the earliest commit. We found that many files were created at the same time very early. In the following time, few new files were created.

This may be because our knowledge is limited, but we have not found an effective solution. We believe that this is the case for system development itself, with very few new features and new files added, and its uniformity is not statistically significant. Therefore, in this report, we only try to model on fix distance and bug density.

4. Assumption and Hypothesis

4.1 Assumption

If there are few changes happened between introduction of a bug and the fix of it, then it would mean that the developers/maintainers are good because they can write readable code and/or understand the code they are working on, thus there would be a good development.

4.2 Hypothesis

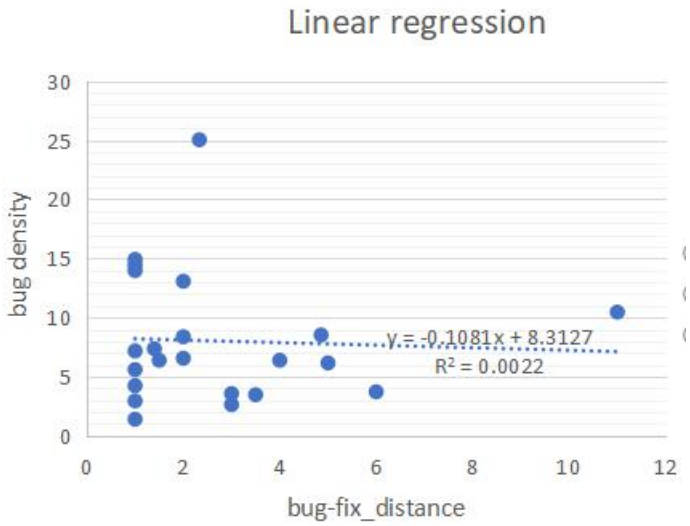
A system whose bugs are fixed in closer commit has a more robust development. The indicator of development robustness is bug density.

5. Model Development

Note: In fact, we know that this model is very bad. The indicators we selected and the data we gathered are problematic, but we have no solution. The model we built here is intended to reflect some of the modeling methods we have learned through this

semester...

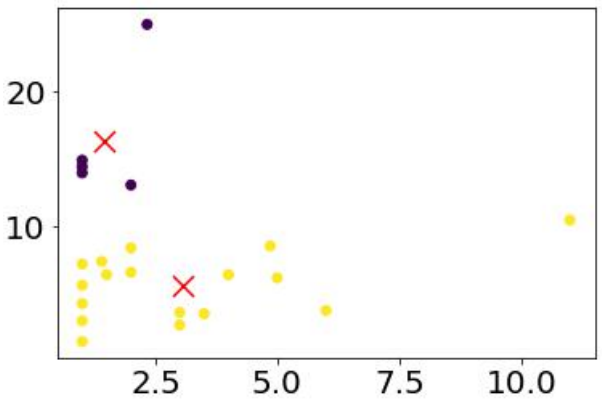
Now we have two variables to evaluate the quality of a version, bug density and bug-fix_distance. At the beginning, we thought that the bug density can indicate the quality of a certain subsystem to a certain extent. So we did a simple linear regression analysis of bug density and bug-fix_distance ,but we found that the effect was not ideal. There was no strong linear correlation between bug density and bug-fix_distance.



We conducted a goodness-of-fit test and found that R^2 is only 0.0022, which is undoubtedly a bad method.

Kmeans Clustering

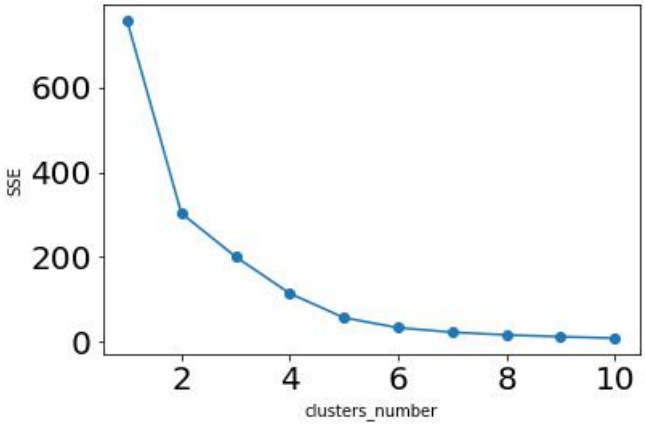
We adjust our thinking and first do a cluster analysis (binary classification) on the sample points. As shown.



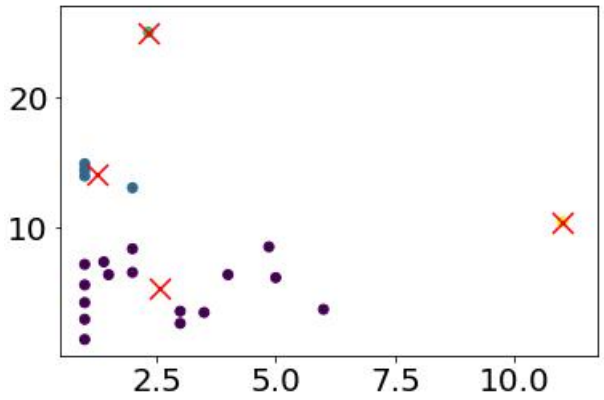
- Performance index of clustering algorithm - within-cluster sum of squared errors (SSE)

In dividing the cluster, we use SSE as the objective function to divide the cluster.

After the KMeans algorithm training is completed, we can use the inertia attribute to obtain the within-cluster sum of squared errors, without the need to calculate again.



We visualize the SSE according to the number of clusters. Through the graph, we can intuitively observe the effect of clusters_number on the SSE. We can see that when clusters_number=2, the polyline has a more obvious inflection point. After clusters_number=4, SSE did not drop significantly. So in the end we choose clusters_number of 4.



Abscissa: bug-fix_distance
Vertical coordinate: bug density

The red crosses represent the center point of each category, and different clusters are painted in different colors.

According to the information in the figure, we can artificially divide it into 4 categories. The category with lower bug density and bug-fix_distance is considered as a good quality subsystem sample because this type of sample guarantees the subsystem bug-fix_distance At the same time there is a lower bug density.

Subsystem samples with lower bug-fix_distance and higher bug density are considered to be “subsystem samples of just so-so quality”. The remaining two types of subsystems with high bug-fix_distance and high bug density are considered to be poor quality subsystems.

Logistic regression model

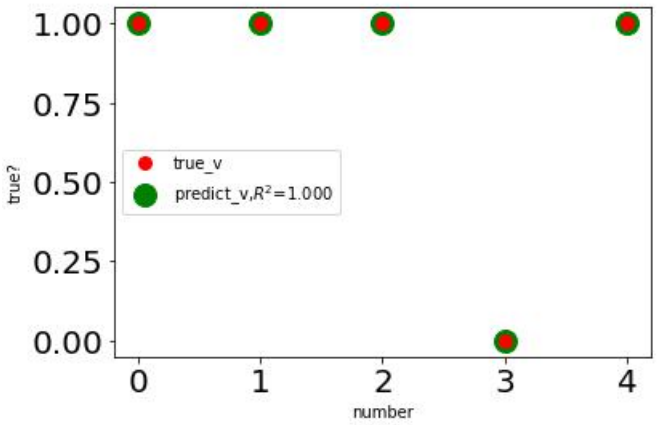
The Kmeans clustering algorithm has a very large time overhead when the amount of data is very large. The logistic regression training speed is faster, and when classifying, the amount of calculation is only related to the number of features. And it is simple and easy to understand. The interpretability of the model is very good. From the weight of features, you can see the impact of different features on the final result. Therefore, we establish a logistic regression model to classify and predict the subsystem quality.

Now we combine the labels of Kmeans clustering with the data set as a new set of data to build a logistic regression model. This way we get a new set of data.

We first divide the training set and the test set from our poor data set, and use the logistic regression tool in sklearn to build and predict the model.

Model prediction effect

The classification prediction effect on the test set, R^2 =1, is completely fitted. **Of course, it is because there is already a strong correlation between explanatory variables and response variables, and the amount of data is too small.** For this project, this result is okay on a few poor data sets.



In addition, **considering the practical significance of the data and model, we did not conduct further verification.**