2016 IEEE/ACM 13th Working Conference on Mining Software Repositories

Judging a Commit by Its Cover

Correlating commit message entropy with build status on Travis-CI

Eddie Antonio Santos
Department of Computing Science
University of Alberta
Edmonton, Canada
easantos@ualberta.ca

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
hindle1@ualberta.ca

ABSTRACT

Developers summarize their changes to code in commit messages. When a message seems "unusual", however, this puts doubt into the quality of the code contained in the commit. We trained *n*-gram language models and used cross-entropy as an indicator of commit message "unusualness" of over 120,000 commits from open source projects. Build statuses collected from Travis-CI were used as a proxy for code quality. We then compared the distributions of failed and successful commits with regards to the "unusualness" of their commit message. Our analysis yielded significant results when correlating cross-entropy with build status.

CCS Concepts

• **Software and its engineering** → **Software configuration management and version control systems; Software version control;** *Open source model;*

Keywords

commit message, github, travis-ci, cross entropy, n-gram language model, build status, open source

1. INTRODUCTION

Commit messages are summaries written by developers describing the changes they have made during the development process. Our experience working within the open source community has given the impression that developers tend to use a fairly limited vocabulary and restricted structure when writing commit messages. Alali *et al.* [1] provide empirical evidence for this observation reporting that over 36% of all commit messages contained the word "fix" and over 18% contained the word "add". Thus, developers may regard short, terse, and to-the-point messages such as "Add test for visibility modifiers" to be *usual* when browsing the commit log of a code repository.

MSR'16, May 14-15 2016, Austin, TX, USA

 ISBN 978-1-4503-4186-8/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2901739.2903493

When a developer reads an *unusual* commit message that defies their expectations, such as "Cargo-cult maven", they may ask themselves a number of questions. "Why did they write 'cargo cult maven'?" "What changes prompted this cryptic commit message?" *"Should I trust the code behind this commit message?"* Such *unusual* messages may induce suspicion; a developer reading this message may question the commit's quality.

Should developers trust their instinct? This paper seeks to answer the question: Are unusual commits hiding bad code? We break this down into the following research questions:

RQ1: How can we measure *unusualness* of a commit?

RQ2: Is the unusualness of a commit message related to the quality of the code committed?

2. METHODOLOGY

In natural language processing, *n*-gram language models are used to answer questions about frequency, surprise, and unusualness. We use the *cross-entropy* of a commit message with respect to a language model in order to quantify its unusualness. To determine the quality of a commit, we use its *build status* as provided by Travis-CI,[1] a continuous integration service popular among open source projects. The build status of any commit can then be evaluated with respect to the unusualness of its message as measured by *cross-entropy*.

To ensure that the commits indeed come from software development projects, we employed a number of strategies described in Section 2.1. We describe how commit quality is determined using data from Travis-CI in Section 2.2. In Section 2.3, we describe how each commit message was *tokenized* so that they could be used as input for the *n*-gram language models. Finally, in Section 2.4, we describe how the tokenized commit messages were used to train language models, and how these language models were used to calculate each commit's unusualness via cross-entropy.

2.1 How were commits chosen?

To obtain commit messages applicable for this study, we used Boa [2] to query the September 2015 GitHub dataset. The Boa query and its results are available online.[2] This query used the following criterion to establish if a given project was to be considered in the study.

1. Boa must have parsed abstract syntax trees of the projects. As of this writing, Boa only parses Java code,

[1] https://travis-ci.org/

[2] http://boa.cs.iastate.edu/boa/?q=boa/job/public/30188

meaning that only projects that contained parsable Java code were obtained. Thus, repositories that are unlikely to be used for software development are pruned.
2. The project must have more than 200 abstract syntax tree nodes. This filters out stub projects.
3. The project must have more than 6 commits, to avoid personal and other stub projects, following the recommendations given in "The Promises and Perils of Mining GitHub" [6].
4. Finally, the project must have a file named `.travis.yml`, indicating that project uses Travis-CI to track per-commit build status.

From each project that passed the above criteria, commits were chosen only if the commit had an associated Travis-CI status (described in Section 2.2), and if it was not a *merge*. Merge commits were excluded due to their messages being automatically generated by Git by default. Auto-generated merge commits are identifiable by their message starting with "Merge branch", "Merge pull request", or "Merge remote." The message text was used to detect merge commits since commit parent information is not available in Boa. Were this data available, merges would be detectable if the commit has more than one parent.

After filtering, 120,822 commits from 2,679 projects fit the criteria described above and were used in our analysis.

2.2 Establishing commit quality: Travis-CI

To establish the quality of a commit, we mined Travis-CI. Travis-CI.org is an online *continuous integration* service that is free for use by open source projects. When a commit is pushed to any branch on GitHub—be it the main branch, a derivative branch, or a pull request—Travis-CI will clone, build, and test that project's commit in a clean virtual machine or Linux container. The *install* phases sets up the machine by installing the project's dependencies, and populates test databases, if any. The *script* phase follows, in which the project is built (compiled) and its test suites are run. A Travis-CI build may result in one of these statuses:[3]

errored An error occurred in the *install* phase. For example, a Java project using the Maven build system[4] may error due to a mis-configured `pom.xml` file.

failed An error occurred in the *script* phase. This usually means the project either failed to build or was successfully built, but failed its test suite.

passed The project built successfully and passed its tests.

A build may also be manually *cancelled* by a developer, but such commits were omitted from our analysis.

2.3 Tokenization

We used n-gram language models which use *tokens* as input. A token is an indivisible unit of meaning that makes up a message. The process of *tokenization* transforms a series of characters into a series of tokens which can then be used with a language model. To avoid undue surprise, tokens fed to the language model must accurately represent the notion of "unusualness" defined in this paper. Thus, we performed the following steps to tokenize each commit message.[5]

[3]https://docs.travis-ci.com/user/customizing-the-build/#Breaking-the-Build

[4]https://maven.apache.org/

[5]https://github.com/eddieantonio/judging-commits/blob/msr2016/tokenize_commit.py#L425

NFC Normalized	Updating Manifest.txt. Related to f75a283 (#495)
Lower-cased	updating manifest.txt. related to f75a283 (#495).
Split	[updating] [manifest.txt] [related] [to] [f75a283] [#495]
Substitutions	[updating] [FILE-PATTERN] [related] [to] [GIT-SHA] [ISSUE-NUMBER]

Table 1: Each step of the tokenization process

Substitution	Meaning	Examples
ISSUE-NUMBER	GitHub Issue numbers[6]	#22, #34
FILE-PATTERN	Filenames and globs	db.c, **/*.jpg
METHOD-NAME	Method names	build_indexes()
VERSION-NUMBER	Version numbers[7]	2.2, 2.1.0-rc.1
GIT-SHA	SHA1 commit hashes	d670460

Table 2: Examples of semantic substitutions

1. First, the message text is normalized using Unicode Normalization Form C to ensure character sequences that "look the same" are compared equal.
2. The message text is transformed into lower-case.
3. The text is then split on whitespace and separating punctuation.
4. Tokens with similar *meaning* are substituted with generic tokens. Upon manually observing over 2500 commit messages, a number of patterns were observed in their text. For example, one commit would read "Update README.md"; another would read "Update pom.xml". Such messages are *not unusual*, yet the amount of individual variety may be immense, due to the amount of different filenames possible. To capture this and other regularities, tokens that were obviously similar were substituted with a generic token such as FILE-PATTERN. Many such *semantic substitions* were defined and employed, as demonstrated in Table 2.

Table 1 shows the tokenization applied to the message, "Updating Manifest.txt. Related to f75a283 (#495)."

2.4 Training the n-gram language model

For each project in the dataset, an n-gram language model was trained on the commit messages of all *other* projects in the dataset. This is called the "leave-one-out" method. Excluding each project from its training set ensures that commit messages from the current project are not already "known" to the language model, thus biasing their "unusualness" score. The *order* or n of the n-gram model was set to three. n-gram models with an order of 3 are also known as *trigram* models. A trigram model was chosen because predictive performance on English text does not significantly improve for larger values of n [4].

The n-gram implementation used was MITLM [5], which implements modified Kneser-Ney smoothing to interpolate

[6]https://help.github.com/articles/autolinked-references-and-urls/#issues-and-pull-requests

[7]According to Semantic Versioning: http://semver.org/

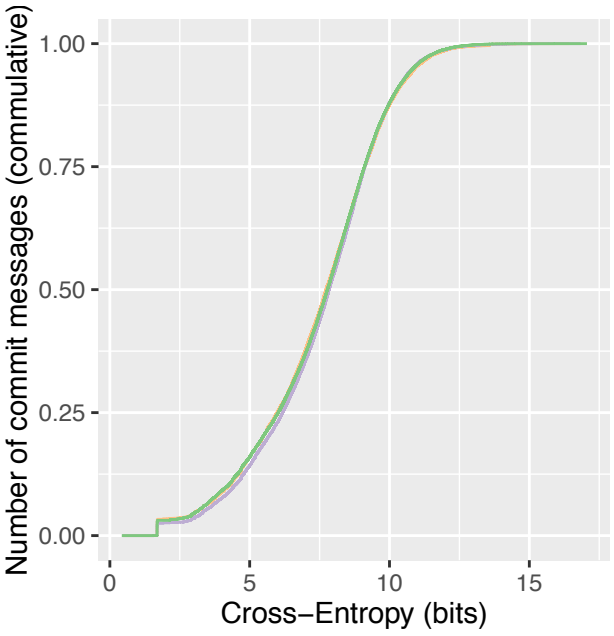

Figure 1: ECDF of the number of passed (in green), failed (in purple), and errored (in orange) commits as cross-entropy ("unusualness") increases. Note that failed, initially grows slower than passed and errored; by 10 bits, however, failed is indistinguishable from passed and errored.

probabilities in the very likely case of data sparsity. With Kneser-Ney smoothing, the model is not "infinitely surprised" when it encounters a trigram it has never seen before—a trigram that is actually composed of one or more bigrams (2-gram) or unigrams (a single token) that it *has* seen before. Instead, Kneser-Ney smoothing interpolates the probability with a penalty—that is, the model is surprised, but not too surprised.

To quantify the "unusualness" of each commit message, the text of each message is tokenized and evaluated using MITLM to calculate the *mean cross-entropy* of the commit message with respect to a language model trained on all other projects. That is, the cross-entropy of each trigram in a message is calculated with respect to the leave-one-out language model and averaged to produce the cross-entropy of the entire message. Intuitively, cross-entropy measures how much information a distribution needs to explain an observation. The higher the cross-entropy, the more difficult it is for the model to explain a given observation. **The higher the cross-entropy a commit message has, the more unusual it is**.

3. RESULTS

Figure 2, left, is a histogram displaying the cross-entropies of all 120,822 commits. The width of the bins in the histogram were chosen using the Freedman-Diaconis rule [3]. What is instantly notable from this histogram are the four major "outlier" bins; in general, the distribution of cross-entropies tends to follow a somewhat normal distribution; however, these bins (quite literally) stick out.

Examining the messages in these bins revealed that, despite the previous discipline in removing merge commits due to their automatically generated commit messages, these automatic messages still crept in. For the exception of the second largest bin, which contains the messages of the form "Update FILE-PATTERN", the messages in these outliers are caused by tools which automatically make commits. Specifically, the Maven release plug-in, and its clone, the Gradle release plug-in both, of which generate messages of the form, "[maven-release-plugin] prepare for next development iteration". The largest outlier bin was filled with 2880 messages of the form, "PROJECT-ISSUE Build version advanced to BUILD-VERSION." These commits were automatically generated by a bot that modifies build configuration; most of these commits had a build status of errored.

Lesson: Not all commit messages are written by hand; even non-merge commits may be automatically generated.

We were curious about "very unusual" messages—those belonging to the rightmost bins in Figure 2, left. Manual inspection of these bins revealed that the majority of the messages in the right-tail were not written in English. Since most of the corpus is in English, the language models would report commits in other languages as very unusual indeed; additionally, this would affect our distribution and hypothesis, since, if a message is usual in its respective language, it would skew the results of unusual English commits.

Lesson: commit messages on GitHub cannot be assumed to be written in English.

To determine if the cross-entropies of passed, failed, and errored commits had different distributions, we calculated the pairwise comparisons using the Wilcoxon rank sum test. Letting $\alpha = 0.01$, we obtained a significant p-value near zero for all comparisons of passed vs. failed, passed vs. errored, and errored vs. failed, meaning that the cross-entropy distributions differed depending on build status.

However, we were suspicious of the effects of the outlier bins; note that the tallest bin visually seems to have a disproportionate amount of errored results. Thus, we repeated the pairwise Wilcoxon rank sum test with all five outlier bins *removed*. Only passed vs. failed and passed vs. errored distributions resulted as significantly different with a p-values near zero and 0.0021, respectively. Errored vs. failed, were not significantly different ($p = 0.2518$). This means that the cross-entropy distributions of passed vs. "broken" (either errored or failed) may be different. Still, since the bins were merely removed from the significance tests *after* the models had already been trained, the cross-entropies analyzed in these tests are still affected by commit messages in the outlier bins.

Given the significant effect of outliers and the fact that the tail had commits in languages other than English, it became obvious that the leave-one-out methodology must be recalculated without such confounding factors.

We curated a list of auto-generated commits, and omitted such commits when constructing the new corpus. To filter for English-language commits, we used `langid.py` [7] to estimate the probability that a whole project's commits are in English. We found that in some cases, `langid.py` would misclassified English commit messages (often as German or Dutch). Hence, we assisted it by manually verifying over 100 projects, letting `langid.py` handle the rest. The resultant corpus contained 108,989 commits from 2,529 projects.

The results are in Figure 2, right. One outlier remains, which (as in the previous histogram) is filled with messages

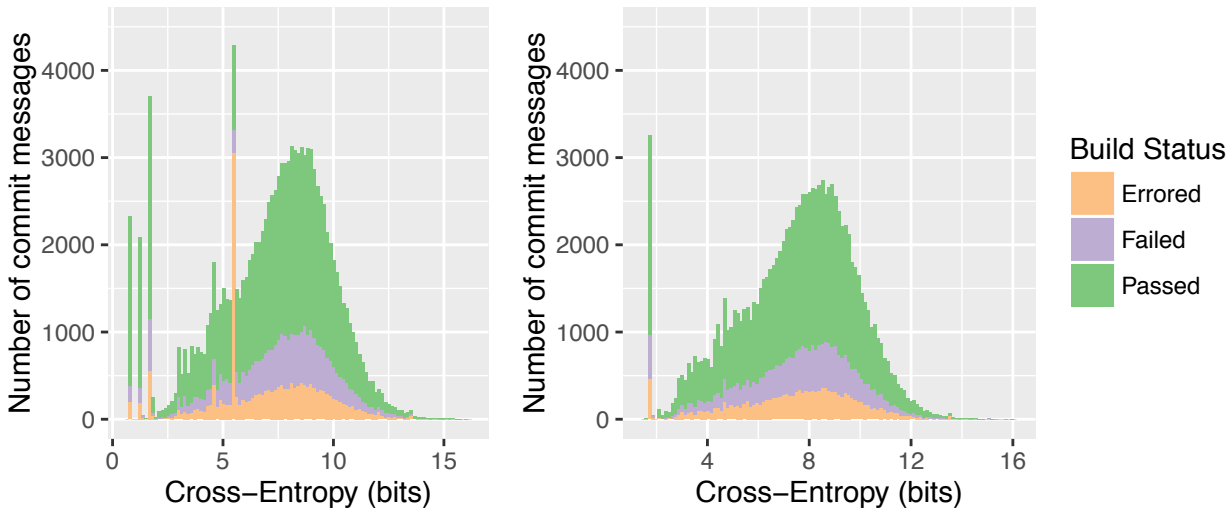

Figure 2: Histograms of commit message cross-entropies. Note the tall bins (left), which contain a large number of auto-generated commit messages that were not foreseen when training this model. We recalculated the histogram (right), removing auto-generated commits, as well as many non-English commit messages.

of the form "Update FILE-PATTERN". A portion of these may be auto-generated, however we deemed these messages as plausibly hand-written; we did not discard them. Pairwise Wilcoxon rank sum test was retried and we found that passed vs. failed are still different distributions with a p-value near zero; similarly, failed vs. errored are different with a p-value of 0.0015. Given a p-value of 0.7527, we fail to reject passed vs. errored as different distributions. Strikingly, we plotted the empirical cumulative distribution function (ECDF) of passed, errored, and failed as seen in Figure 1. We found that failed *is* different, for lower values of cross-entropy. Additionally, it steadily closes the gap between itself and passed and errored. This means that failure rate is *lower* given a more usual commit, and gradually increases. Calculating Pearson's product-moment linear correlation coefficient yields a 99% confidence interval of $(0.007, 0.468)$. Since zero is not in the interval (zero would indicate no correlation) we conclude that build failure and "unusualness" may be positively correlated—but only marginally.

4. DISCUSSION

Does this mean that developers can use commit messages to predict build failure? Can we cancel our continuous integration subscriptions? In short, no.

Though the results are statistically significant, we conclude that they are not *practically helpful* for the average developer. For example, which of the following commits failed its status check? "added init.d test to travis config" (cross-entropy = 5.08), or "I'm sloppy" (cross-entropy = 12.9)? The latter has a far more unusual commit message than the former, yet it passed its status check; the "usual" commit failed. Thus, as a heuristic for estimating the probability of build failure, commit messages are not very useful.

5. CONCLUSIONS

RQ1: **How can the unusualness of a commit message be measured?** Using n-gram language models, one is able to use cross-entropy as an analogue for unusualness. Automatically-generated messages and non-English commits become easy to spot out.

RQ2: **Is the unusualness of a commit message related to the quality of the code committed?** Despite some evidence to suggest that the "unusualness" of a commit message is positively correlated with build failure, the slope is so gradual that it is infeasible for an average developer to judge a commit by simply reading its log message.

6. REFERENCES

[1] A. Alali, H. Kagdi, and J. I. Maletic. What's a typical commit? A characterization of open source software repositories. In *16th IEEE International Conference on Program Comprehension*, ICPC 2013, pages 182–191.

[2] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering*, ICSE 2013, pages 422–431, May 2013.

[3] D. Freedman and P. Diaconis. On the histogram as a density estimator: L 2 theory. 57(4):453–476.

[4] A. Hindle, E. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *34th International Conference on Software Engineering*, ICSE 2012, pages 837–847.

[5] B.-J. P. Hsu and J. R. Glass. Iterative language model estimation: efficient data structure & algorithms. In *INTERSPEECH*, pages 841–844.

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101. ACM.

[7] M. Lui and T. Baldwin. langid. py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 system demonstrations*, pages 25–30. Association for Computational Linguistics.