A dataset for identifying actionable feedback in collaborative software development

Benjamin S. Meyers[†], **Nuthan Munaiah**[†], **Emily Prud'hommeaux**[‡§], **Andrew Meneely**[†], **Cecilia Ovesdotter Alm**[‡], **Josephine Wolff**[‡], **and Pradeep K. Murukannaiah**[†]

[†]Golisano College of Computing and Information Sciences, Rochester Institute of Technology

[‡]College of Liberal Arts, Rochester Institute of Technology

[§]Morrissey College of Arts and Sciences, Boston College

bsm9339,nm6061,emilypx,axmvse,coagla,jcwgpt,pkmvse}@rit.edu

Abstract

Software developers and testers have long struggled with how to elicit proactive responses from their coworkers when reviewing code for security vulnerabilities and errors. For a code review to be successful, it must not only identify potential problems but also elicit an active response from the colleague responsible for modifying the code. To understand the factors that contribute to this outcome, we analyze a novel dataset of more than one million code reviews for the Google Chromium project, from which we extract linguistic features of feedback that elicited responsive actions from coworkers. Using a manually-labeled subset of reviewer comments, we trained a highly accurate classifier to identify "acted-upon" comments (AUC = 0.85). Our results demonstrate the utility of our dataset, the feasibility of using NLP for this new task, and the potential of NLP to improve our understanding of how communications between colleagues can be authored to elicit positive, proactive responses.

1 Introduction

As in many other work environments, such as hospitals and law firms, employees in software development must communicate through written feedback and comments to develop functional and secure code. Developers elicit feedback from their collaborators on the code that they write through the code review process, which is an integral part of the mature software development lifecycle. Most large software development organizations, including Microsoft (Lipner, 2004) and Google (Chromium, 2017), mandate the review of all changes to the code base. Code reviews identify potential bugs or errors in software, but not all of the comments made by reviewers are acted upon by developers.

Some code reviews are taken seriously by developers and prompt significant fixes, while many others are overlooked or dismissed. In some cases, such as when code reviewers misunderstand the purpose of a proposed change or identify an unimportant issue, it may be appropriate to ignore their comments. At other times, however, the presentation and language of the reviewer's feedback may cause the problems it identifies to be overlooked. Understanding which linguistic characteristics of code reviews influence whether reviews are taken seriously can aid developers in providing effective feedback that is acted upon by their peers. In turn, this can contribute to our general understanding of how to provide meaningful written feedback in a collaborative workplace setting.

With this in mind, we present a dataset of over one million code review comments from the Chromium project (Chromium, 2017), designed with the goal of discovering the linguistic features associated with actionable developer feedback. We describe the dataset, along with an array of linguistic features capturing characteristics of complexity, content, and style, extracted from that dataset. Using a labeled subset of this large dataset, we develop a highly accurate classifier for identifying examples of actionable feedback that performs better than the keyword and sentiment features previously explored for similar tasks.

The contributions of this work are: (1) the introduction of a new NLP task: identifying actionable feedback in collaborative work conversations; (2) a large structured dataset of automatically linguistically annotated software developer conversations for feature exploration[1]; (3) a smaller manually-labeled subset of that dataset for hypothesis test-

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 126–131
Melbourne, Australia, July 15 - 20, 2018.

ing[1]; and (4) a demonstration of the feasibility of using NLP for this task in the form of a high-accuracy classifier of actionable feedback.

2 Background

A typical code review is initiated by a developer (*change-author*) who wishes to have a collection (*patchset*) of local changes (*patches*) to the source code merged into the software product. The patchset is reviewed by other developers (*reviewers*) who provide feedback to ensure that the change does not negatively impact the overall quality of the product. In response to this feedback, the change-author can submit one or more additional patchsets for further review. The process repeats until the *owner* of the source code approves the change. The Chromium project, which underlies Google's Chrome browser and Chrome OS, follows this typical code review process, requiring all changes to the source code to be reviewed before being accepted into the repository. Rietveld (The Chromium Project, 2017), an open-source tool, facilitates the code review process in Chromium.

The process of providing direct assessment of an individual's actions or performance, known as feedback intervention, has been widely studied in a number of domains (Judd, 1905; Kluger and DeNisi, 1996, 1998; Xiong et al., 2010; Xiong and Litman, 2010), but previous work applying NLP to the specific task of evaluating code review feedback is somewhat limited. Rahman et al. (2017) examined a small set of text features (e.g., reading ease, stop word ratio) in a small set of code review comments but found associations between those features and comment usefulness to be mostly insignificant. Pletea et al. (2014) examined sentiment as an indicator of comment usefulness, while Bosu et al. (2015) considered both sentiment and the presence of pre-defined keywords in feedback. While these studies offer insights into the language used by developers, they are limited to sentiment and basic lexical attributes. In contrast, we explore more subtle linguistic features that more accurately characterize actionable feedback.

3 Data

Our dataset consists of written natural language conversations among developers working to find

[1] https://meyersbs.github.io/chromium-conversations/

```
409     return found;
```

susan 2015/06/25 07:31:02
```
return nullptr;
```

william 2015/06/25 07:56:32
```
On 2015/06/25 07:31:02, susan wrote:
> return nullptr;
No, if line 398 breaks, |found| can be non-null.
```

susan 2015/06/25 08:04:40
```
On 2015/06/25 07:56:32, william wrote:
> On 2015/06/25 07:31:02, susan wrote:
> > return nullptr;
> No, if line 398 breaks, |found| can be non-null.
Ops. I didn't realize it.
line 398 should `return found`, instead of `break`.
```

william 2015/06/25 08:23:38
```
On 2015/06/25 08:04:40, susan wrote:
> On 2015/06/25 07:56:32, william wrote:
> > On 2015/06/25 07:31:02, susan wrote:
> > > return nullptr;
> > No, if line 398 breaks, |found| can be non-null.
> Ops. I didn't realize it.
> line 398 should `return found`, instead of `break`.
Done.
If the code doesn't break there, line 409 should
return nullptr, as the original comment suggests.
```

Figure 1: Example code review comment thread.

flaws in proposed changes to software. An example is shown in Figure 1. We used Rietveld's RESTful API to retrieve, in JSON formatted documents, publicly-accessible code reviews in the Chromium project spanning eight years (2008-16). We processed the JSON documents and extracted reviews with their associated patchsets, patches, and comments, saving them to a PostgreSQL database. Of the 2,855,018 comments, 1,591,431 were posted by reviewers. We refer to this set of comments, for which we provide values for the 9 linguistic features (described in Section 4), as the **full dataset**.

With the goal of characterizing the linguistic attributes of actionable feedback, we created a **labeled dataset**, which reflects the overall distribution of actionable comments in the full dataset by including 2,994 comments automatically identified as *acted-upon* and 800 comments manually identified as *not (known-to-be) acted-upon*. We automatically identified *acted-upon* comments using the Rietveld functionality that allows change-authors to respond to feedback by clicking a link labeled "Done", which automatically posts a special comment containing only the word '*Done.*'. We consider comments by reviewers that elicit this '*Done.*' response to be *acted-upon*. Of the 1.5 million comments posted by reviewers, 690,881 (43%) were identified as *acted-upon* using the '*Done.*' metric. We independently verified a subset of 700 of these comments (Cohen's $\kappa = 0.89$) and found that in 97% of instances when a devel-

oper posted a comment with '*Done.*', there was an associated code change implemented.

To identify comments that were not acted upon, we manually inspected code review comments that did not terminate in a '*Done.*' comment. We randomly sampled a set of 2,047 such comments and inspected the line of code associated with a comment across all patchsets and the source code commit associated with the code review. Comments for which the authors could not find evidence that the developer acted upon the feedback were labeled as *not (known-to-be) acted-upon*. Within the sample, 800 (39.08%) were manually identified as *not (known-to-be) acted-upon*.

4 Feature Extraction

We extract nine linguistic features from the reviewer comments (examples in Table 1) that capture structure, information content, style, and tone. Before extracting features, we automatically replace all sequences of source code tokens with a single custom token. Since comments can span multiple sentences, we aggregate sentence-level features at the comment level as described below for each feature.

Syntactic complexity: Previous work (Rahman et al., 2017) attempted to measure structural complexity using readability metrics, such as Flesch reading ease (Flesch, 1948), which approximate complexity using word and sentence length. We instead evaluate the structure of comments by calculating YNGVE (Yngve, 1960) and FRAZIER (Frazier, 1987) scores, two complimentary approaches derived from constituent parses (as in Roark et al. (2011); Pakhomov et al. (2011)) that approximate the cognitive load of sentence processing (Baddeley, 2003; Sweller and Chandler, 1991). We take the maximum over all sentences in a comment for each of these scores.

Information content: We calculate both content density (C-DENSITY), which measures the content of text using the ratio of open-class to closed-class words, and propositional density (P-DENSITY), which is the ratio of propositions to the number of words in a text (Roark et al., 2011). We use an approach similar to that used by Brown et al. (2008) to detect propositions, and we aggregate both scores over the sentences in a comment.

Style and tone: We explore several features characterizing style and tone to learn whether the way reviewers choose to communicate their feedback has an influence on how their colleagues respond to that feedback.

SENTIMENT: We extract the sentiment of a code review comments using Stanford CoreNLP (Manning et al., 2014). In contrast to previous work (Bosu et al., 2015; Agarwal et al., 2011), we use only three values, merging the two *positive* classes and the two *negative* classes, and introduce a fourth class, *non-neutral*, which ignores the sentiment polarity. The sentiment at a comment level is the ratio of negative/neutral/positive/non-neutral tokens to all tokens.

FORMALITY: We use the dataset provided by Lahiri (2015) to train a logistic regression model for estimating the formality of a sentence, with precision and recall of 83%. We reduce the 7-point rating scale to a binary (formal vs. informal) scale. The features used to train the model included parts-of-speech, character n-grams, chunking tags, and other features used in predicting uncertainty Vincze (2014). For this feature and POLITENESS, we find the maximum and minimum values over all sentences in a comment.

POLITENESS: To measure politeness, we use a corpus of Wikipedia editor and Stack Exchange user conversations annotated for politeness (Danescu-Niculescu-Mizil et al., 2013). We reimplemented their logistic regression model with newer programming languages and frameworks, yielding 94% precision and 95% recall.

UNCERTAINTY: Uncertainty in natural language has been studied by Vincze (2014) and Farkas et al. (2010), who worked with Szarvas et al. (2012) to compile the Szeged Uncertainty Corpus. While Vincze (2014) trained a binary (certain vs. uncertain) model on the corpus, we trained a multi-label logistic regression model using the same features to predict the type of uncertainty exhibited by each word in a comment.

5 Results

Using the labeled dataset described in Section 3, we evaluated the association between each feature and the class labels. For continuous valued features, we used the non-parametric Mann-Whitney-Wilcoxon to test for association and Cliff's δ to assess the strength of that association. For boolean-valued features, we used the χ^2 test to test for independence between the feature and class label. Our results show that *acted-upon* code review comments were shorter, more polite, more formal,

Feature	Example Sentence from Chromium dataset
FRAZIER	Low: *This 'if' can be done more elegantly with Min(x,y)*
	High: *Please see this warning about adding things to NavigationEntry.*
YNGVE	Low: *The description is a little confusing.*
	High: *The only time we call one but not the other is in the destructor, when we don't need to call needsNewGenID, but setting two fields needlessly might be a low price to pay to ensure we never accidentally call one without the other.*
P-DENSITY	Low: *In addition to what I suggested earlier about testing for the non-existence of a third file, we could also verify that the contents of the sync database files are not nonsense.*
	High: *I tried patching this in locally and it doesn't compile.*
C-DENSITY	Low: *Slight reordering: please put system modules first, then a blank line, then local ones (PRESUBMIT).*
	High: *Please check that given user_id is child user, not currently active user is child.*
FORMALITY	Low: *But yeah, I'm just being an API astronaut*; I think that what I wrote up there is neat, but after sleeping, don't worry about it; it's too much work to go and rewrite stuff.*
	High: *Moving this elsewhere would also keep this module focused on handling the content settings / heuristics for banners, which is what it was originally intended for.*
POLITENESS	Low: *You don't actually manage the deopt table's VirtualMemory, so you shouldn't act like you do.*
	High: *Thanks for writing this test, getting there, but I think you could do this in a more principled way.*
SENTIMENT	Negative: *That's not good use of inheritance.*
	Neutral: *Are we planning on making use of this other places?*
	Positive: *It looks slightly magical.*
UNCERTAINTY	Epistemic: *This seems a bit fragile.*
	Doxastic: *I assume we added this notification purely for testing purposes?*
	Investigative: *Did you check whether it was needed?*
	Conditional: *Another possible option, if it does not cause user confusion, would be to automatically select those projects in the Files view when the dialog closes.*

Table 1: Example code review comments for a subset of the linguistic features.

Feature Set	Precision	Recall	F_1	AUC
# Tokens	0.793	0.996	0.883	0.610
# Sentences	0.792	0.999	0.884	0.584
All	0.829	0.926	0.872	0.849
Significant	0.805	0.953	0.871	0.805
Relevant	0.802	0.963	0.874	0.819

Table 2: Results of $10 \times 10-$fold cross-validation.

less uncertain, and had a lower density of propositions than those that were *not acted-upon*.

We then trained a classifier to identify code review comments that are likely to be acted upon. In training the classifier, we considered three sets of linguistic features: (1) *all* features, (2) *significant* features from association analysis, and (3) *relevant* features from recursive feature elimination. Through recursive feature elimination, we found MAX POLITENESS, P-DENSITY, MIN FORMALITY, and MAX FORMALITY to be the four most relevant features for discriminating between *acted-upon* and *not acted-upon* comments.

We trained logistic regression classifiers with these three sets of linguistic features, evaluating performance using 10x10-fold cross validation. We compare these with two baseline classifiers using only token count and sentence count. Table 2 shows the average precision, recall, F_1-measure, and AUC. The classifiers trained on the linguistic features, while performing near the baselines on the first three measures, substantially outperform the baselines on AUC, with all three yielding values over 0.8. Given these results and the imbalanced nature of the dataset, it seems that the classifiers trained on the linguistic features are able to identify both classes of comments with high accuracy, while the baseline classifiers perform only marginally better than a majority class baseline.

6 Discussion & Future work

Overall, we find that the way in which coworkers communicate feedback to each other strongly influences whether their peers will act on their advice. Remarkably, politeness and formality, two high-level discourse features, are among the most effective in distinguishing acted upon feedback. It seems that the manner in which feedback is deliv-

ered has more impact on the actions of developers than might be expected given the practical and impersonal nature of written code reviews. These results point to the critical importance of how feedback is phrased and delivered in workplace settings, beyond just the content of the feedback itself.

In our future work, we plan to explore whether these and other features can be incorporated into a code review tool like Rietveld to automatically flag feedback that is less likely to be acted upon and to encourage more effective communication strategies. We also plan to use our methods to analyze the linguistic patterns of individual reviewers to identify those with particularly effective or weak communication styles.

Our work demonstrates the potential of applying NLP to the task of identifying actionable feedback in collaborative work scenarios and the utility of our two datasets for this task. More broadly, these results speak to the importance of training code reviewers–and indeed all employees working in highly collaborative environments–not just in how to do their jobs effectively but also how to communicate their findings and feedback to their coworkers in a way that will elicit proactive responses.

References

A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau. 2011. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38.

A. Baddeley. 2003. Working memory and language: An overview. *Journal of Communication Disorders*, 36(3):189–208.

A Bosu, M Greiler, and C Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156.

C. Brown, T. Snodgrass, S.J. Kemper, R. Herman, and M.A. Covington. 2008. Automatic measurement of propositional idea density from part-of-speech tagging. *Behavior Research Methods*, 40(2):540–545.

Chromium. 2017. Chromium OS Developer's Guide. `https://www.chromium.org/chromium-os/developer-guide`. [Online; accessed 25-Aug-2017].

C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts. 2013. A computational approach to politeness with application to social factors. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 250–259.

R. Farkas, V. Vincze, G. Móra, J. Csirik, and G. Szarvas. 2010. The CoNLL-2010 shared task: Learning to detect hedges and their scope in natural language text. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning—Shared Task*, pages 1–12.

R. Flesch. 1948. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221.

L. Frazier. 1987. Syntactic Processing: Evidence from Dutch. *Natural Language & Linguistic Theory*, 5(4):519–559.

C.H. Judd. 1905. Practice without knowledge of results. *The Psychological Review: Monograph Supplements*.

A.N. Kluger and A. DeNisi. 1996. The effects of feedback interventions on performance: a historical review, a meta-analysis and a preliminary feedback intervention theory. *Psychological Bulletin*, 119:254–284.

A.N. Kluger and A. DeNisi. 1998. Feedback interventions: Toward the understanding of a double-edged sword. *Current Directions in Psychological Science*, 7(3):67–72.

S. Lahiri. 2015. SQUINKY! A Corpus of Sentence-level Formality, Informativeness, and Implicature. *CoRR*, abs/1506.02306.

S. Lipner. 2004. The Trustworthy Computing Security Development Lifecycle. In *20th Annual Computer Security Applications Conference*, pages 2–13.

C.D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S.J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

S. Pakhomov, S. Chacon, M. Wicklund, and J. Gundel. 2011. Computerized assessment of syntactic complexity in Alzheimer's disease: A case study of Iris Murdoch's writing. *Behavior Research Methods*, 43(1):136–144.

D. Pletea, B. Vasilescu, and A. Serebrenik. 2014. Security and Emotion: Sentiment Analysis of Security Discussions on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 348–351.

M.M. Rahman, Chanchal K. Roy, and R.G. Kula. 2017. Predicting Usefulness of Code Review Comments Using Textual Features and Developer Experience. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 215–226.

B. Roark, M. Mitchell, J. Hosom, K. Hollingshead, and J. Kaye. 2011. Spoken Language Derived Measures for Detecting Mild Cognitive Impairment. *Transactions on Audio, Speech, and Language Processing*, 19(7):2081–2090.

J. Sweller and P. Chandler. 1991. Evidence for cognitive load theory. *Cognition and Instruction*, 8(4):351–362.

G. Szarvas, V. Vincze, R. Farkas, G. Móra, and I. Gurevych. 2012. Cross-genre and cross-domain detection of semantic uncertainty. *Computational Linguistics*, 38(2):335–367.

The Chromium Project. 2017. Code Review — Chromium. `https://codereview.chromium.org/`. [Online; accessed 25-Aug-2017].

V. Vincze. 2014. *Uncertainty Detection in Natural Language Texts*. Ph.D. thesis, University of Szeged.

W. Xiong and D. Litman. 2010. Identifying problem localization in peer-review feedback. In *Intelligent Tutoring Systems*, pages 429–431. Springer.

W. Xiong, D.J. Litman, and C.D. Schunn. 2010. Assessing reviewers performance based on mining problem localization in peer-review data. In *Third International Conference on Educational Data Mining*, pages 211–220.

V.H. Yngve. 1960. A model and an Hypothesis for Language Structure. In *Proceedings of the American Philosophical Society*, volume 104, pages 444–466.