Predicting Usefulness of Code Review Comments using Textual Features and Developer Experience

Mohammad Masudur Rahman   Chanchal K. Roy   §Raula G. Kula

University of Saskatchewan, Canada, §Osaka University, Japan

{masud.rahman, chanchal.roy}@usask.ca, §raula-k@ist.osaka-u.ac.jp

Abstract—Although peer code review is widely adopted in both commercial and open source development, existing studies suggest that such code reviews often contain a significant amount of non-useful review comments. Unfortunately, to date, no tools or techniques exist that can provide automatic support in improving those non-useful comments. In this paper, we first report a comparative study between useful and non-useful review comments where we contrast between them using their textual characteristics, and reviewers' experience. Then, based on the findings from the study, we develop RevHelper, a prediction model that can help the developers improve their code review comments through automatic prediction of their usefulness during review submission. Comparative study using 1,116 review comments suggested that useful comments share more vocabulary with the changed code, contain salient items like relevant code elements, and their reviewers are generally more experienced. Experiments using 1,482 review comments report that our model can predict comment usefulness with 66% prediction accuracy which is promising. Comparison with three variants of a baseline model using a case study validates our empirical findings and demonstrates the potential of our model.

Index Terms—Code review quality, review comment usefulness, change triggering capability, code element, reviewing experience.

I. Introduction

Peer code review has been considered as one of the best engineering practices in software development for over last 35 years [7, 30]. It works as a quality safeguard for the patches (i.e., software code changes) that are to be integrated into the master codebase of a software system. It detects simple source code defects (e.g., logical errors) and coding standard violations (e.g., lack of descriptive identifier names) in the early phases of the development, which reduces the overall cost [23]. Despite the proven benefits, the formal code inspection (also called Fagan's inspection) was never widely adopted due to its high costs and inapplicability to distributed software development (i.e., demands synchronous meeting among developers) [7]. Interestingly, *modern code review*, a lightweight version of the peer code review that is assisted with various tools, has gained remarkable popularity in recent years both in the industry (e.g., Microsoft, Google) and in the open source development (e.g., Android, LibreOffice) [7]. The main building blocks of modern code review are the associated review comments provided by the reviewers during code review [10]. Such comments are generally consulted by the patch submitters before correcting their code changes and then submitting the next patch. Hence, the quality of the review comments is crucial, and the comments are expected to be accurate, thorough [22] and useful [10]. Inaccurate, ambiguous or low quality review comments could mislead the submitters, kill valuable development time unnecessarily, and thus, could simply defeat the purpose of the code review. A recent study [10] reported that 34.50% of the code review comments from five major projects of Microsoft are *non-useful*. Automatic filtration of those review comments could have saved the development time of the patch submitters [10]. Alternatively, meaningful insights on the quality (e.g., prediction of usefulness) of those comments accompanied by helpful suggestions could have assisted the reviewers in improving their comments in the first place. Unfortunately, to date, no tools or techniques exist that predict usefulness of the review comments during the submission of a code review.

In fact, Bosu et al. [10] first explore professional developers' perceptions on the usefulness of code review comments using a qualitative study, and identify several characteristics of the useful and non-useful review comments at Microsoft. According to them, developers find those review comments *useful* that trigger code changes within the vicinity (i.e., 1–10 lines) of the comment locations. Kononenko et al. [22] conduct a similar study involving professional developers from open source domain–*Mozilla*, and analyze the perceptions of the developers on the quality of code reviews. They suggest that the quality of code reviews is primarily associated with the thoroughness (i.e., completeness) of review feedback, i.e., comments. While both studies offer insights on the usefulness of review comments by employing qualitative methods, there has been a marked lack in the application of such insights in practical settings using automated approaches. Although Bosu et al. develop a classifier model for review comments, their feature set is constrained, and many of their features are not available during review comment submission. Thus, the model is not capable of predicting usefulness of the new review comments to be submitted. On the other hand, we adopt their definition of *usefulness of comments* [10] and provide automated supports for identifying and thus improving the non-useful review comments during review submission. Besides, given the earlier findings [10, 22], textual features of the review comments are not properly studied yet which might be linked to their usefulness.

In this paper, we (1) report a comparative study on the usefulness of code review comments using their textual features and reviewers' expertise, and then (2) propose a prediction model –*RevHelper*– based on the findings from our study. We first motivate our idea using a comparative study between
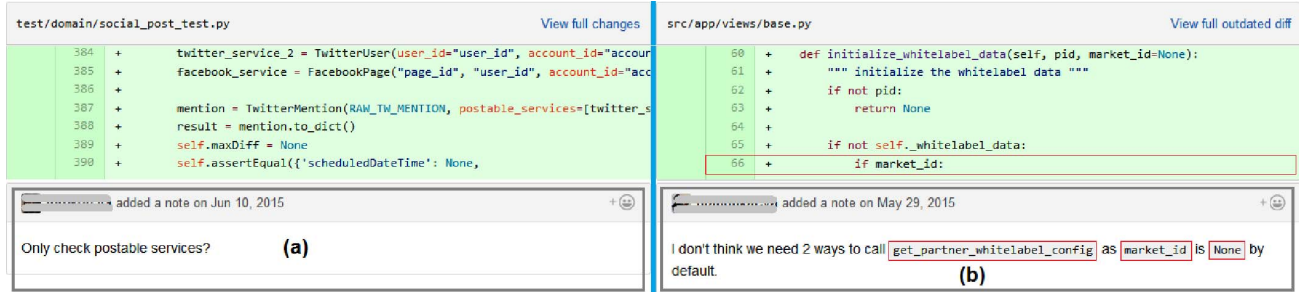
Fig. 1. Review comments from *ABC* code reviews–(a) Non-useful review comment, and (b) Useful review comment

Fig. 2. Code change triggered by the useful review comment from Fig. 1-(b)

useful and non-useful review comments, and answer two research questions. Review comments are marked as *useful* or *non-useful* based on the suggested heuristics of Bosu et al. (e.g., change-triggering capability) that are derived from an interview with professional developers from Microsoft. Then, we develop a machine learning model for predicting usefulness of a review comment based on the perceived signals from the comparative study. Such a model not only provides an approximate view on the quality of review comments, but also can assist the code reviewers in improving their review comments, especially the non-useful ones, through meaningful insights and complementary information.

For example, the review comment in Fig. 1-(a)(bottom row)–*"Only check postable services?"*–did not trigger any code change, and the patch submitter also found the comment non-useful [10]. We note that the comment does not warrant any clear action (i.e., change of code) rather asks for a clarification. On the other hand, the review comment in Fig. 1-(b)(bottom row)–*"I don't think we need 2 ways to call* `get_partner_whitelabel_config` *as* `market_id` *is* None *by default"*– triggered a code change within its vicinity (i.e., next line, Fig. 2), and the patch submitter also found the comment useful. Please note that the comment contains relevant code artifacts (e.g., `market_id`), and also suggests a code change on invoking the method– `get_partner_-whitelabel_config()`. Such a warranted action along with references to the artifacts or concepts of interest from code could have led the review comment to be useful. In our research, we study several of such textual features of the review comments along with the reviewers' expertise, contrast between useful and non-useful comments, and then capture their essence into a usefulness prediction model–RevHelper. To the best of our knowledge, ours is the first study that not only contrasts between useful and non-useful comments extensively (i.e., using two dimensions and eight indepen-

dent features) but also automatically predicts usefulness of the review comments during code review submission. The major focus of the earlier studies [10, 22] was to analyze the review comments qualitatively by involving professional developers. The datasets used for our comparative study and the experiments are hosted online [2] for replication purposes.

Our investigation using 1,116 code review comments from four commercial subject systems of *ABC Company* (blinded for double-blind reviewing) reported that (1) 44.47% of those comments were found non-useful (i.e., did not trigger any code changes [10]), (2) useful review comments share more vocabulary with the changed code, and contain more relevant code elements (e.g., program entity names) and relatively less stop words, and (3) reviewers' past experience positively influence the usefulness of their comments. Experiments using 1,116 code review comments from *ABC Company* (i.e., ground truth is based on change-triggering heuristic of Bosu et al.) suggest that our model can predict comment usefulness with 66% overall accuracy, 66% precision and 66% recall which are promising given that ours is the first complete model of its kind. Finally, a case study using 366 new review comments and comparison with three variants of a baseline model [10] not only validates our empirical findings but also demonstrates the potential of our prediction model. Thus, we make the following contributions in this paper.

- A comparative study that contrasts between useful and non-useful review comments using five textual characteristics and three reviewers' experience features.
- A novel and promising model for predicting the usefulness of a new review comment during review submission.
- Comprehensive evaluation of the proposed model, and validation using a separate case study and three variants of a baseline model from the literature.

II. COMPARATIVE STUDY BETWEEN USEFUL AND NON-USEFUL REVIEW COMMENTS

Given that existing findings are mostly derived from qualitative studies involving software developers [10, 22], we are interested to further understand the topic of usefulness of review comments from a comparative perspective. In particular, we want to find out what items are missing from the non-useful comments or what distinguishing features of the useful comments are leading them to be useful. Hence, we conduct a comparative study using 1,116 code review comments taken

TABLE I
OVERVIEW OF THE VARIABLES AND DIMENSIONS USED

Independent Variable	Dimension	Description
Reading Ease (RE) [15, 26]	textual	numerical estimation of ease with which a comment can be read
Stop Word Ratio (SWR) [40]	textual	frequently occurring words in natural language texts with little or no semantics
Question Ratio (QR) [10]	textual	percentage of interrogative sentences in each review comment
Code Element Ratio (CER) [16]	textual	percentage of source code related tokens in each review comment
Conceptual Similarity (CS) [6]	textual	lexical similarity between changed source lines and each review comment
Code Authorship (CA) [10, 37]	experience	number of commits authored on a file by a developer
Code Reviewership (CR) [10, 37]	experience	number of commits on a file reviewed by a developer
External Library Experience (ELE) [29]	experience	percentage of external libraries a developer familiar with from a reviewed file

from four commercial subject systems of *ABC Company*. In our study, we answer the two research questions as follows.

- **RQ$_1$**: Are useful review comments different from non-useful comments in terms of their textual properties?
- **RQ$_2$**: How does experience of the developers help them write useful review comments during a code review?

A. Dataset for Study

Dataset Collection: We collect 1,200 recent code review comments for our study from four commercial subject systems of *ABC Company*. In GitHub code reviews, reviewers generally submit two types of review comments–*in-line comments* and *pull request comments*. While an in-line comment refers to a particular code change directly, the pull request comment targets the whole collection of patches submitted in the request. In this research, we focus on *in-line* comments of a code review as was also done by Bosu et al. [10]. We collect 300 most recently posted *in-line* review comments from each of the four subject systems by accessing GitHub API using a client library–*github-api* [1]. Once collected, we request the company developers for initial feedback on their comments. The company reported that few of the pull requests were not originally intended for merging commits rather than for scaffolding purposes. We thus discarded such pull requests along with their review comments from our dataset, and finally ended up with a total of 1,116 review comments.

Annotation of Review Comments: We manually annotate each of the collected comments as either *useful* or *non-useful* by applying *change triggering* heuristic of Bosu et al. [10]. That is, if any comment triggers one or more code changes within its vicinity (i.e., 1–10 lines) in the subsequent commits (i.e., patches) of a pull request, the comment is marked as *useful*. On the other hand, if the comment either triggers code changes in unrelated places or does not trigger any change at all, the comment is marked as *non-useful*. Not only the heuristic was recommended by the professional developers from *Microsoft* but also it was found highly effective for comment classification [10], which justify our choice of the heuristic for the manual annotation. The whole annotation task took about 20 man-hours. We found 55.53% of our collected comments as *useful* and 44.47% of them as *non-useful*. Table II shows the details of our dataset used for the study.

B. Determining Independent Variables and Dimensions

According to Kononenko et al. [22], a significant fraction (i.e., 38%) of the developers consider that *clear* and *thorough feedback* is the key attribute of a well-done code review. The clarity of any natural language artifact can be

TABLE II
DATASET FOR COMPARATIVE STUDY

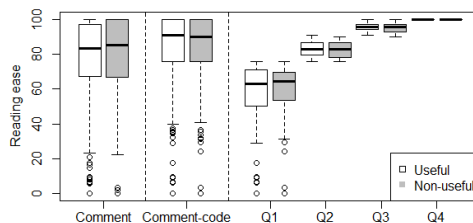| System	Pull Requests	Useful Comments	Non-useful Comments	Total
CS	80	153 (59.77%)	103 (40.23%)	256
SM	97	164 (58.36%)	117 (41.64%)	281
MS	88	155 (53.82%)	133 (46.18%)	288
SR	101	146 (50.17%)	145 (49.83%)	291
Total	366	618 (55.53%)	498 (44.47%)	1,116

Fig. 3. Reading ease of code review comments (**Q1**=0.25 quantile, **Q2**=0.50 quantile, **Q3**=0.75 quantile, and **Q4**=1.00 quantile)

determined based on a popular metric called *readability* as was also applied by several existing studies [15, 26, 28]. The thoroughness of a comment might be linked to the conciseness and salience of the comment texts [38]. Bosu et al. [10] report positive impact of developers' prior experience upon their code reviews. Kononenko et al. also suggest that having enough domain knowledge by the reviewers is a prerequisite for effective code reviews. Such experience can be estimated by using their code authorship, review activities [10] or even familiarity with the external libraries used in the changed source files under code review [29]. Given that many of these findings on code reviews are based on qualitative studies, we want to revisit them using a comparative study between useful and non-useful comments. We thus consider a total of eight independent variables from two dimensions and one response variable–*usefulness of comment* for our study. Table I provides an overview of our selected independent variables where first five are related to *textual properties* of the review comment and the remaining three are proxies for *developer's experience*. For each variable, we present our method for comparison between useful and non useful comments. We apply several statistical tests to determine the differences between the both groups.

C. Textual Features

Reading Ease: Flesch-Kincaid reading ease is one of the most popular metrics for determining readability of any regular texts [15, 26]. It is based on a ranking scale from 0 to 100 where the higher value represents lower complexity of the texts and vice versa. Fig. 3 summarizes our findings on reading ease. We see that the median ease is between 80 and 90 which suggests that both comment types are easy to read. Unfortunately, unlike the qualitative findings suggest [22], we
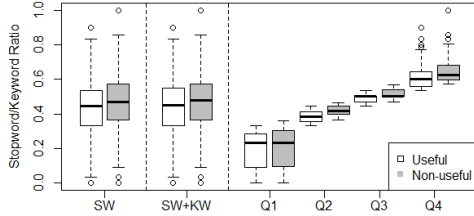
Fig. 4. Stop word and keyword ratio (**SW**=Stop words, and **KW**=Keywords, **Q1**=0.25 quantile, **Q2**=0.50 quantile, **Q3**=0.75 quantile, and **Q4**=1.00 quantile)

Fig. 5. Question ratio of review comments (**Q1**=0.25 quantile, **Q2**=0.50 quantile, **Q3**=0.75 quantile, and **Q4**=1.00 quantile)

did not find any significant difference in the reading ease (i.e., *Mann-Whitney Wilcoxon test, p-value=0.32>0.05, Cohen's D=0.04<0.20*) between useful and non-useful comments. We then manually analyze the comments, and found that comment texts contain both regular texts and structured code elements (e.g., identifier names, code fragments). We discard the code elements from the comments using appropriate regular expressions, and calculate the reading ease again. As Fig. 3 shows, we found that overall reading ease significantly increased for both useful (i.e., *p-value=0.00, Cohen's D=0.28*) and non-useful comments (i.e., *p-value=0.00, Cohen's D=0.25*) with medium effect sizes. We note that useful comments are a bit easier to read than non-useful comments with only regular texts considered. However, their reading ease difference is not statistically significant (i.e., *p-value=0.80>0.05, D=0.00*).

We also divide the reading ease measures for each of the comment types into four quartiles– Q1, Q2, Q3 and Q4, and then compare them using box plots, as done by relevant studies [9]. However, the readability difference between useful and non-useful comments was not found statistically significant (i.e., *p-values>0.05*) for any of the four quartiles.

Stop Word Ratio: Stop words are frequently occurring words in the regular texts with very little or no semantics [40]. Given the limited size of a review comment, we are interested to find out if there exists any distributional difference of stop words between useful and non-useful comments. We determine stop word ratio over all words from each comment, and compare the ratios from both comment types. We use a standard list [4] of stop words for our analysis. Fig. 4 summarizes the comparative analysis between both types. We see that the median stop word ratio is relatively higher for non-useful review comments than that of useful comments. In fact, the ratio difference is statistically significant (i.e., *p-value=0.01<0.05*) although the effect size is small (*Cohen's D=0.14*). This partially supports our conjecture that non-useful comments contain relatively less meaningful words which might have reduced their usefulness.

Programming keywords can be considered as an equivalent to stop words in the source code [17]. Since review comments sometimes contain source code elements along with regular texts, we recompute the stop word and keyword ratios (i.e., of Python language) and compare the ratios for both comment types. As Fig. 4 shows, the ratio is still higher for non-useful comments than that of useful comments, and their difference is statistically significant (*p-value=0.01<0.05*). However, the effect size of the difference is still small (*Cohen's D=0.14*).

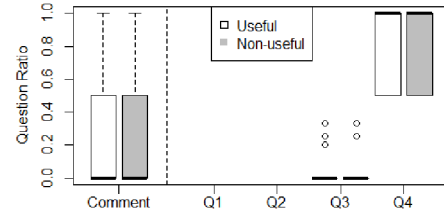We also divide the stop word ratios for each comment type into four quartiles, and compare between useful and non-useful comments. While the difference for Q1 is not significant (*p-value=0.55>0.05*), we found statistically significant difference for Q2 (i.e., *p-value=0.00, Cohen's D>1.00*), Q3 (i.e., *p-value=0.00, Cohen's D=0.89*) and Q4 (i.e., *p-value=0.00, Cohen's D =0.54*) with medium to large effect sizes, which are interesting, and support our conjecture.

Question Ratio: Reviewers often ask clarification questions during code review. Bosu et al. suggest that such questions might be helpful for knowledge dissemination among the developers but do not improve the quality of a submitted patch. Thus, review comments as a form of clarification questions are not considered useful by the professional developers [10]. We revisit that proposition in our study, and investigate if that is supported by our collected data or not. In particular, we want to see if there exists a difference in the ratio of interrogative sentences over all sentences from a review comment between useful and non-useful comments. We thus determine the number of questions from each comment using appropriate regular expression, and compare the question ratio between both comment types. Fig. 5 summarizes our comparative findings. We see that about 50% of the review comments from both categories do not contain any questions (i.e., median question ratio = 0). More importantly, the distributions of question ratio of both comment types look similar, and we did not find any statistically significant difference (i.e., *p-value=0.08>0.05, Cohen's D=0.12*) between them. An investigation using quartile analysis also revealed the same finding. Finally, a Kruskal-Wallis test confirmed that the distribution of question ratio for both comment types are identical (i.e., *p-value=0.08>0.05*), and the usefulness of a review comment is not influenced by such distribution.

Code Element Ratio: Since our investigation suggests that code segments or code-like elements in the review comments could affect their readability (i.e., reading ease) significantly, we are interested to find out whether they also affect the usefulness of a comment or not. Our interest also stems from the fact that the usefulness of review comments is determined based on their code change triggering capabilities in practice [10]. Ercan et al. [16] also report that a good code explanation in Stack Overflow posts generally overlaps (i.e., shares artifacts) with the code snippet. We thus believe that presence of relevant code elements in the review comment might improve its chance of being useful to the patch submitters during change implementation. A comparative analysis between useful and non-useful comments on the code elements

System	All	Useful Comments		Non-useful Comments		Both	
		CC	CWC	CC	CWC	CC	CWC
CS	256	94 (61.44%)	59	46 (44.66%)	57	140 (54.69%)	116
SM	281	87 (53.05%)	77	42 (35.90%)	75	129 (45.91%)	152
MS	288	91 (58.71%)	64	70 (52.63%)	63	161 (55.90%)	127
SR	291	66 (45.21%)	80	77 (53.10%)	68	143 (49.14%)	148
Total	1,116	338 (54.69%)		235 (47.19%)		Avg=51.34%	

CC=Review comments with code elements, **CWC**=Review comments without code elements
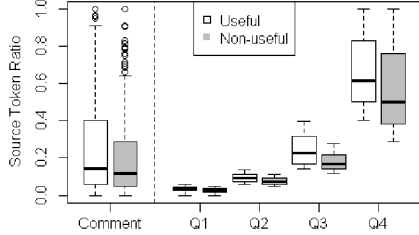

Fig. 6. Source token ratio of review comments (**Q1**=0.25 quantile, **Q2**=0.50 quantile, **Q3**=0.75 quantile, and **Q4**=1.00 quantile)

can offer us further meaningful insights. We thus collect all code segments and code-like elements from each of the review comments using a set of custom regular expressions, and discard false positives through a careful manual analysis. We then perform the comparative analysis where we investigate how frequently and how extensively the code elements are used in both types of review comments.

From Table III, we see that about 51% of all review comments contain one or more source code elements which is a significant amount. We also note that code elements are more frequently used in the useful review comments than in the non-useful comments for each of the four systems except one (i.e., SR). For example, about 45%–61% of the useful comments from our selected subject systems refer to at least one code element whereas such ratios reach up to 36%–53% for the non-useful comments. However, our statistical analysis suggested that these ratios are not significantly different (i.e., *Paired t-test, p-value=0.27>0.05*) between both comment types.

We also consider the extent to which code elements are used within each review comment, and investigate if the two comment types differ on that aspect. We thus split the source code artifacts from each comment, discard all the punctuation marks, and then determine the source token ratio over all tokens from the comment. Fig. 6 shows box plots of source token ratio between both comment types. We see that median ratio of useful comments is relatively higher than that of non-useful comments. More interestingly, we found the ratio difference statistically significant (i.e., *p-value=0.00<0.05, Cohen's D=0.17*) which is interesting. However, we further decompose our analysis, and our investigation with quartiles reports that source token ratio is significantly higher for useful comments for all four quartiles with large effect size– Q1 (i.e., *p-value=0.00<0.05, Cohen's D=0.61*), Q2 (i.e., *p-value=0.00<0.05, Cohen's D=0.70*), Q3 (i.e., *p-value=0.00<0.05, Cohen's D=0.92*) and Q4 (i.e., *p-value=0.00<0.05, Cohen's D=0.42*). Finally, Kruskal Wallis test confirmed that usefulness of a review comment is significantly affected (i.e., *p-value=0.003*) by its source token ratio, which supports our initial conjecture.
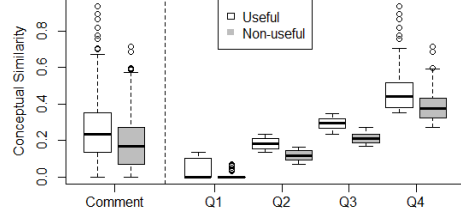

Fig. 7. Conceptual similarity between review comments and the changed code

Conceptual Similarity: According to Bosu et al., usefulness of review comments is mostly determined based on their code change triggering capability in practice. Relevance of concepts (e.g., concerns) discussed in the comment with the submitted code changes could be a proxy to such triggering capability. Concept or feature location communities also hold a similar assumption in slightly different contexts [17]. We thus are interested to find out how relevant the review comments (i.e., containing code elements and associated texts) are to the changed code submitted for review, and whether such relevance significantly affects their usefulness or not. In information retrieval, relevance or conceptual similarity between two text documents is generally determined based on their lexical similarity. Cosine similarity is a widely used metric to determine lexical similarity between two entities where each entity is considered as a vector of words [6]. We thus consider each in-line review comment and each changed source line as two individual entities, convert them into two vectors of terms by applying standard natural language preprocessing (i.e., stop word removal, splitting), and then determine their cosine similarity. Since the in-line review comment targets a particular source line, we treat each of the changed lines (i.e., added, deleted, modified) as an individual entity during similarity calculation. For each review comment, we then analyze all the changed source lines of the same file from earlier commits of the same pull request, and collect the maximum cosine measure. The measure is based on the shared vocabulary between the review comment and the changed source line, and thus, we consider the measure as a proxy to the relevance of the comment to the changed code.

From Fig. 7, we see that useful review comments are more relevant, i.e., lexically or conceptually similar, to the changed code submitted for review. That is, they refer to the concepts embedded in the changed code more extensively than the non-useful comments, which is intuitive and also supports our initial conjecture. According to our analysis, useful comments have a median relevance measure of 0.24 which is 0.17 for non-useful comments. Statistical tests report that the similarity measures of useful comments are significantly higher (i.e., *p-value=0.00*) with medium effect size (i.e., *Cohen's D=0.42*) than that of the non-useful comments. Further investigation using quartile analyses (Fig. 7) reported that relevance measures are also significantly higher for useful comments than the counterpart for each of the four quartiles–Q1 (i.e., *p-value=0.00, Cohen's D=0.91*), Q2 (i.e., *p-value=0.00, Cohen's D>1.00*), Q3 (i.e., *p-value=0.00, Cohen's D>1.00*) and Q4 (i.e., *p-value=0.00, Cohen's D=0.83*).
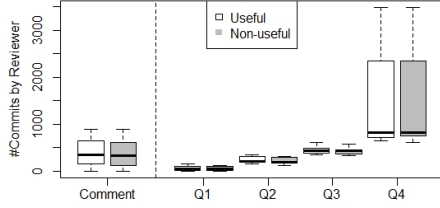
Fig. 8. Authorship of developers providing useful and non-useful review comments

Summary of the Findings–Answering RQ$_1$: Useful code review comments are significantly different than non-useful comments in terms of several textual properties of the comments such as *code element ratio, stop word ratio* and *conceptual similarity*, and not significantly different in terms of *reading ease* and *question ratio*.

D. Developer Experience

Code Authorship: Code authorship is a traditional metric for estimating developer's experience which is reported to positively influence the usefulness of code review comments [10, 14]. According to Bosu et al., reviewers even with minimum authoring experience on a target source file (i.e., changed once) can provide higher number of useful comments than the reviewers without any authoring experience. However, these findings were derived from the systems of *Microsoft* of different application domains. We thus revisit their findings in our domain (i.e., brand analytics), and investigate the correlation between code authorship and comment usefulness.

We calculate the number of total commits submitted by the developers on a target file under review as well as on the whole subject system. Please note that only commits submitted before the date of the comment are considered during file level experience approximation. We found that reviewers of 46.12% of the useful comments changed the target file at least once whereas such statistic for non-useful comments is 49.00%. The finding is a bit counter-intuitive, and it also does not support the earlier finding [10]. However, our Kruskal-Wallis test on these statistics reported that comment usefulness is affected by the presence or absence of authorship experience of the reviewers (i.e., *p-value=0.00<0.05*). We then consider the file level commits of the reviewers of both comment types, and investigate their impact upon comment usefulness. We performed a Kruskal-Wallis test on file level commit counts for the reviewers, and found that their distributions are significantly different (i.e., *p-value=0.03<0.05*). This suggests that the usefulness of review comments is indeed influenced by the authorship of the reviewers, which supports the earlier finding [10]. We also repeat the investigation for commits on the whole system (i.e., overall authoring experience), and found similar finding, i.e., significant difference (i.e., *p-value=0.00<0.05*) in the distributions of commit count. Unfortunately, Mann-Whitney Wilcoxon test reported the opposite (i.e., *p-value=0.27>0.05, Cohen's D=0.12*). Thus, we further investigate, and divide the commit counts on whole system into four quartiles. Fig. 8 summarizes our findings from the investigation. Outliers are omitted from the plot to preserve visual clarity. We performed Mann-Whitney Wilcoxon tests,
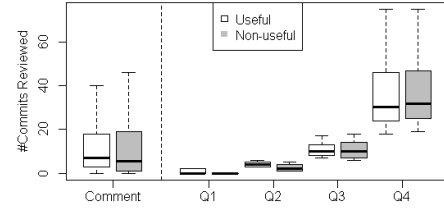

Fig. 9. Review experience of developers making useful and non-useful review comments

and found that authorship (of the reviewers) difference is significantly different between useful and non-useful comments for two quartiles–Q2 (i.e., *p-value=0.00<0.05, Cohen's D=0.42*) and Q3 (i.e., *p-value=0.74>0.05, Cohen's D=0.56*), and not significantly different for the rest two quartiles– Q1 (i.e., *p-value=0.74>0.05, Cohen's D=0.08*) and Q4 (i.e., *p-value=0.80>0.05, Cohen's D=0.21*). That is, too little or too much authoring experience on the subject system did not make a difference on the usefulness of review comments, i.e., each of these experience levels produced similar amount of useful and non-useful comments. On the contrary, medium level experience made a difference significantly. However, higher level authoring experience definitely helped the code reviewers to provide more useful review comments.

Code Reviewership: Bosu et al. suggest a strong relationship between code reviewing experience of the developers and the usefulness of their provided review comments. That is, experienced reviewers are likely to provide more useful comments than first time reviewers. Thongtanunam et al. [37] even suggest review activities to be considered as a metric for code ownership. We revisit the earlier findings in the context of *ABC Company*, and investigate how prior experience of the reviewers affect the usefulness of their comments.

We determine review experience of the developers by calculating number of earlier commits on a target file currently under review, total commits, and pull requests reviewed by them from a subject system. We found that code reviewers reviewed the same file at least once for 87.22% of the useful comments whereas such statistic for the non-useful comments is 84.54%. We also found that 86.25% of the reviewers reviewed more than once for useful comments compared to 79.09% for the non-useful comments. Although our findings do not strongly support the earlier findings, they align with them comfortably [10]. When actual commits on the target file for each reviewer are considered, we also find that they are significantly different (i.e., *p-value=0.02<0.05*) for useful and non-useful review comments. However, the effect size of their difference is small (i.e., *Cohen's D=0.05*). We thus perform Kruskal Wallis test on them, and found that comment usefulness is significantly affected (i.e., *p-value=0.00<0.05*) by the reviewed commit counts of the reviewers. We further investigate into this, and perform quartile analysis [9]. Fig. 9 shows our comparative analysis between useful and non-useful comments. Outliers are omitted from the plot to preserve visual clarity. We perform Mann-Whitney Wilcoxon tests on each of the quartiles, and found significant difference for Q1 (i.e., *p-value=0.00<0.05, Cohen's D=0.99*) and Q2 (i.e., *p-value=0.00<0.05, Cohen's D>1.00*), and not significant
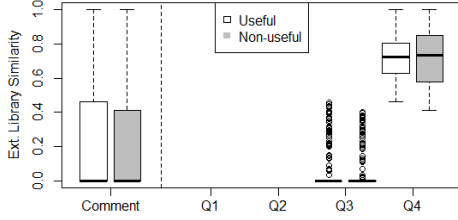
Fig. 10. Reviewer's experience on external libraries included in the file under review

Fig. 11. Schematic diagram of the proposed prediction model–RevHelper

difference for Q3 (i.e., *p-value=0.72>0.05, Cohen's D=0.04*) and Q4 (i.e., *p-value=0.18>0.05, Cohen's D=0.27*). That is, reviewers with high level reviewing experience did not provide significantly higher number of useful comments than their non-useful comments. On the contrary, reviewers with low to medium level experience did exactly that. This might be partially explained from the earlier finding [10] that suggests that developers generally improve their reviewing experience during the *"ramp up period"* (i.e., first one-year of hire), and then the experience reaches into plateau. However, according to our findings, increased reviewing experience on the target file definitely helps the reviewers provide more useful comments. For example, reviewers with top level experience (i.e., Q4) provided 160 useful review comments compared to 126 non-useful comments by them or 136 useful comments by the least experienced reviewers (i.e., Q1).

We also consider the total number of commits reviewed by the reviewers from a target system, and found that they are not significantly different (i.e., *p-value=0.63>0.05, Cohen's D=0.01*) for two comment types. However, our Kruskal Wallis test reported that such review statistic has a significant influence (i.e., *p-value=0.00<0.05*) on the usefulness of review comments. We also observed similar findings when total number of pull requests reviewed by the reviewers is considered. We found that the statistic does not differ significantly (i.e., *p-value=0.33>0.05*) for useful and non-useful comments, but Kruskal Wallis test reported its significant influence (i.e., *p-value=0.00<0.05*) on comment usefulness.

External Library Experience: Rahman et al. [29] suggest that experience with the external libraries included in a source file is an effective metric to determine the eligibility of a developer as the reviewer of that file. That is, the developers who have working experience with the dependencies of a target file are likely to provide better reviews than the ones with no such experience [29]. We are interested to find out if such experience of the developers is connected to the usefulness of their review comments or not. We thus determine similarity between the imported libraries in the target file (i.e., attached to a review comment) and that from other files recently authored by the reviewer, and consider such similarity measure as a proxy to her experience with the external libraries. Then we contrast corresponding measures between useful and non-useful comments.

From Fig. 10, we see that reviewers' minimum library experience is a bit higher for useful comments than that for non-useful review comments. However, our Mann-Whitney Wilcoxon test reported that their difference is not statisti-
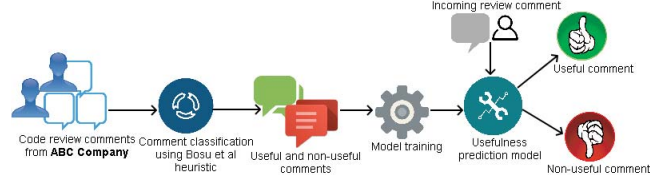
cally significant (i.e., *p-value=0.29>0.05, Cohen's D=0.05*). Interestingly, a Kruskal Wallis test reported that experience with external libraries can significantly influence the usefulness of the review comments (i.e., *p-value=0.00<0.05*). We thus compare between two comment types using quartile analysis. The median measure is 0 for both comments which leads the comparison to only Q3 and Q4. We found that Q3 for useful comment is significantly higher (i.e., *p-value=0.03<0.05, Cohen's D=0.18*) than that for non-useful comments, which partially supports our initial conjecture about reviewers' experience on external libraries. For Q4, we did not find any significant difference (i.e., *p-value=0.62>0.05, Cohen's D=0.12*), and thus, does not support our conjecture.

Summary of the Findings–Answering RQ$_2$: Prior experience with a file being reviewed in terms of authorship and reviewing activity of the developers can significantly influence the usefulness of their review comments. However, such experience of the developers and the usefulness of review comments provided by them always do not have a linear relationship, which further supports the earlier findings [10].

III. REVHELPER: AUTOMATIC PREDICTION MODEL FOR REVIEW COMMENT USEFULNESS

Given a significant part (i.e., 34.5%) of the code review comments are non-useful [10], an automatic technique is warranted for identifying and improving them before the review submission. Our comparative study reported that useful review comments significantly differ from non-useful review comments in terms of several textual properties, and their reviewers have more experience. We thus develop a model for automatic usefulness prediction (i.e., *useful* or *non-useful*) of review comments by applying machine learning on these textual and experience based features. In this section, we discuss different steps of our model development. Fig. 11 shows the schematic diagram of our prediction model.

A. Feature Calculation

We select a total of 15 independent variables (i.e., several features yield multiple variables) from two dimensions– comment texts and developer experience– and one response variable (i.e., useful or non-useful) for our model. Although useful and non-useful comments do not vary significantly for some of the features according to the comparative study, we still wanted to investigate their predicting capabilities in a practical setting. We thus apply all the features to our model. While the textual features come from comment texts, the corresponding reviewers' experience is determined based on the analysis of version control history and code review history of the subject systems. Each of the comment instances is

manually annotated as either *useful* or *non-useful* (details in Section II-A). Less than 4% of the samples in our dataset contain missing values for one or more features, and such values are either discarded or replaced with the mean of corresponding features.

B. Development of the Usefulness Prediction Model

Since the relationships between the usefulness of a review comment and corresponding features might be complex according to our comparative study, we choose three algorithms with different learning strategies–*Random Forest* [12], *Logistic Regression* [13] and *Naive Bayes* [28]– to train our classification model. While Random Forest assumes the non-linear relationship between independent variables and the response variable, Logistic Regression does the opposite and returns binary outcomes. On the other hand, Naive Bayes assumes the independence among the predictor variables, and determines the response based on the *maximum log-likelihood* based training of the model parameters. We apply 10-fold cross-validation for training and testing of the model when Naive Bayes and Logistic Regression are considered. That is, nine folds of our data are used for training and one fold is applied to testing at any moment, and this process iterates until all data are used for training and testing. Then the results are averaged from the 10 models, and the final performance is reported. On the other hand, Random Forest algorithm is robust enough to avoid over-fitting, and it does not warrant a cross-validation [3, 12]. However, it randomly selects the predictor variables for tree generation, and thus, warrants multiple test runs. We run our Random Forest based model 10 times, train with 65% of the randomly sampled data (i.e., different seed points) and a set of 2,000 decision trees, and then report the average test performance. We use machine learning workbench–WEKA [5]–and R–for training, testing and validation of our models.

IV. EXPERIMENT

We conduct experiments using 1,482 code review comments from *ABC Company* with 10-fold cross-validation tests and a separate case study. Thus, we evaluate our model with real code review data from the commercial subject systems against a set of popular performance metrics. In order to further validate our performance, we compare with three baseline comment classification models that were employed by Bosu et al. [10]. We thus answer three research questions as follows:

- **RQ$_3$**: How does the proposed model perform in predicting the usefulness of code review comments?
- **RQ$_4$**: How effective are the dimensions–comment texts and corresponding reviewers' experience–as a proxy to the usefulness of a review comment?
- **RQ$_5$**: Can RevHelper outperform the baseline models in predicting usefulness of the code review comments?

A. Experimental Dataset

Dataset Collection: We conduct experiments using a total of 1,482 code review comments extracted from the review history of four commercial subject systems–CS, SM, MS and
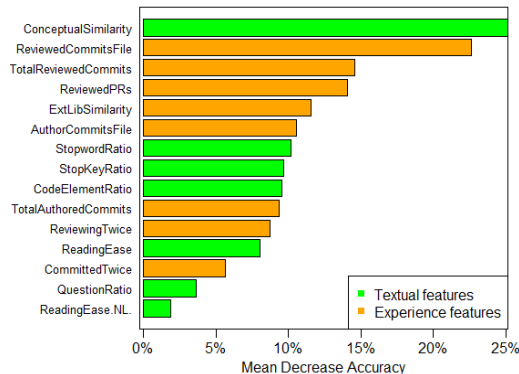

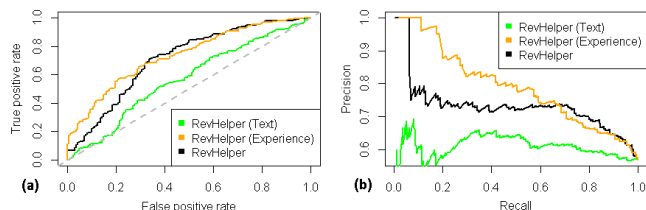Fig. 12. Importance of 15 independent variables in RevHelper$_{RF}$

Fig. 13. (a) ROC and (b) Precision-Recall curves of RevHelper$_{RF}$

SR–of ABC Company. While 1,116 of them were previously collected for comparative study (Section II-A), we extend the dataset for evaluation and validation following the same steps.

Ground Truth Selection: Each of our selected review comments is manually annotated based on the *change triggering* heuristic of Bosu et al. [10]. The heuristic is objective and derived from the interviews with professional developers from the industry, and thus, our choice of the heuristic for ground truth selection is justified. That is, if the review comment triggers a code change to its proximity (i.e., 1–10 lines) in the subsequent commits, the comment is considered *useful* and vice versa. Please check Section II-A for more detailed steps.

Replication: All experimental data and supporting materials are hosted online [2] for replication or third-party reuse.

B. Evaluation of RevHelper

Performance of RevHelper–Answering RQ$_3$: We evaluate our prediction model with a dataset of 1,116 review comments using four appropriate performance metrics–*precision*, *recall*, F_1-*score* and *classification accuracy*. We perform 10-fold cross-validation on Naive Bayes (NB) and Logistic regression (LR) based models and run Random Forest (RF) based model 10 times. Table IV summarizes our performance details.

From Table IV, we see that our model predicts usefulness of a review comment with up to 66% overall accuracy which is promising according to related literature on classification [10, 13, 28]. Given that ours is the first automated support of its kind, such performance can make a difference during code review. Our model exploits the non-linear relationships between comment usefulness and the independent variables using Random Forest learning algorithm, and provides better performance than the two other variants. For example, the model variants–RevHelper$_{NB}$ and RevHelper$_{LR}$–provide a maximum of 59% overall accuracy even after applying

TABLE IV
PERFORMANCE OF REVHELPER

Learning Algorithm	Dimension	Feature Set	Useful Comments		Non-useful Comments		All Comments			
			Precision	Recall	Precision	Recall	Precision	Recall	F$_1$-score	Accuracy
Naive Bayes (NB)	textual +	{all features}	60.90%	57.10%	50.60%	54.40%	56.30%	55.90%	56.10%	55.91%
	experience	{all features}$_{PCA}$	61.30%	66.00%	53.30%	48.20%	57.70%	58.10%	57.90%	**58.06**%
Logistic Regression (LR)	textual +	{all features}	60.70%	71.40%	54.60%	42.80%	58.00%	58.60%	58.30%	**58.60**%
	experience	{all features}$_{PCA}$	60.20%	72.80%	54.50%	40.40%	57.70%	58.30%	58.00%	58.33%
Random Forest (RF)	textual	{RE, SWR, QR, CER, CS}	60.52%	65.49%	50.49%	45.18%	56.04%	56.43%	56.23%	56.58%
	experience	{CA, CR, ELE}	66.73%	**76.29**%	63.03%	51.29%	65.08%	65.13%	65.10%	**65.37**%
	textual +	{all features}	65.68 %	73.30%	59.73%	50.82%	63.02%	63.27%	63.14%	**63.45**%
	experience	**{CS, CA, CR, ELE, SWR}**	**67.93**%	75.04%	**63.06**%	**54.54**%	**65.76**%	**65.89**%	**65.82**%	**66.06**%

certain feature optimization technique such as Principal Component Analysis (PCA). On the other hand, the final model–RevHelper$_{RF}$–provides 66% overall accuracy with 66% overall precision and 66% overall recall. While decision tree based model might be less explanatory, feature importance graph (e.g., Fig. 12) demonstrates the relative importance of the predictors of our model. Instant usefulness prediction during review submission along with insightful rationale behind the prediction (e.g., details of predictor variables) is likely to help the reviewers improve their review comments. Especially, the non-useful comments lacking salient features (e.g., relevant code elements) could be benefited from our instant supports.

Thus, to answer **RQ$_3$**, our model predicts usefulness of a review comment with 66% overall accuracy, 66% precision, recall and F$_1$-score which are promising.

Role of Dimensions–Answering RQ$_4$: We consider two distinctive dimensions – comment texts and reviewer's experience – as a proxy to the usefulness of code review comments. As investigation details in Table IV and performance curves in Figures 12, 13 suggest, experience of the reviewers is a better predictor of the usefulness of their comments than the textual characteristics of the comments. Unfortunately, prediction based on only reviewer's experience is of limited insights, less actionable and might not help much in improving the review comments in practice. Hence, a model that analyzes the actual content of a review comment and predicts instantly and reliably is warranted. Thus, although the textual features have relatively less prediction capability and addition of this dimension to the model yields marginal improvement in performance, our choice of introducing and combining both dimensions is possibly justified. Besides, from Fig. 12, we see that exclusion of one or more textual features (e.g., *conceptual similarity*) can significantly decrease the overall accuracy of the prediction model. This signals the importance of the textual dimension and makes it indispensable to the model.

Thus, to answer **RQ$_4$**, reviewer's experience is more effective than textual content of a review comment in predicting usefulness of the comment. However, textual features could be more meaningful to the developers, and its addition to the model also marginally improves the performance.

C. Comparison with Baseline Models using Case Study

Although the above evaluation with several performance metrics and industrial data extensively analyzed the performance of our prediction model, we also wanted a comparison with the state-of-the-art. However, the existing model of Bosu et al. [10] cannot predict the usefulness of a new review
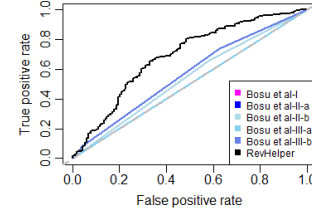

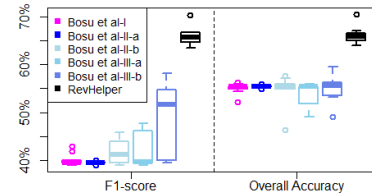Fig. 14. Comparison of ROC with the baseline models

Fig. 15. Comparison of F$_1$-score and prediction accuracy with baseline models

comment due to the nature of its learned features. Many of their features are not available during the submission of a code review. Hence, comparing with Bosu et al. [10] as is might be neither feasible nor fair. Thus, we develop three variants of Bosu et al. using a subset of their features–*keyword count* and *comment's sentiment*–only which are available during review comment submission like ours. Please note that the omitted features from Bosu et al. are not available during submission of a new review comment and thus, could not be used for its usefulness prediction. Their methodologies, tools and learning algorithms are also carefully adopted. In particular, we used their keyword list [10], sentiment analysis tool (i.e., MSR-Splat [27]) and machine learning algorithm (i.e., Classification and Regression Tree–CART [11]) to implement those variants. Then we prepare a separate validation set containing 366 code review comments totally unseen by any model, and compare our model with those variants using a case study.

Table V summarizes our comparative analyses using both test dataset (i.e., cross-validation) and validation dataset. We see that our model – RevHelper – performs significantly better than the baseline variants in terms of several performance metrics with both datasets. The baseline models perform well for useful comments with about 70% precision and up to 100% recall. Unfortunately, they perform very poorly with non-useful comments. Bosu et al.-I (i.e., applies keyword counts) and Bosu et al.-II-a (i.e., applies comment sentiment group) returned 100% false-positives for them. Since our goal is to support developers in improving their review comments, especially the non-useful ones, performance with non-useful comments is of crucial importance. The best performing baseline model– Bosu et al.-III-b (i.e., applies both keyword counts

TABLE V
COMPARISON WITH BASELINE MODELS

Classifier	Test Dataset						Validation Dataset						Significance
	Useful Comments		Non-useful Comments		All Comments		Useful Comments		Non-useful Comments		All Comments		p-value & Cohen's D
	Precision	Recall	Precision	Recall	F_1-score	Accuracy	Precision	Recall	Precision	Recall	F_1-score	Accuracy	
Bosu et al.-I	55.30%	98.50%	35.70%	1.00%	40.09%	55.02%	71.60%	**100.00%**	0.00%	0.00%	0.00%	**71.58%**	–
Bosu et al.-II-a	55.40%	**100.00%**	0.00%	0.00%	0.00%	55.38%	71.60%	100.00%	0.00%	0.00%	0.00%	71.58%	–
Bosu et al.-II-b	55.40%	95.10%	44.40%	4.80%	42.64%	54.84%	71.90%	98.50%	42.90%	**2.90%**	61.05%	71.31%	–
Bosu et al.-III-a	55.00%	91.30%	40.70%	7.40%	43.60%	53.85%	71.60%	100.00%	0.00%	0.00%	0.00%	71.58%	–
Bosu et al.-III-b	56.80%	79.00%	49.40%	**25.50%**	**51.61%**	55.11%	70.20%	77.50%	**23.40%**	**17.30%**	58.39%	**60.38%**	–
RevHelper$_{txt}$	60.52%	65.49%	50.49%	45.18%	56.12%	56.58%	73.74%	71.40%	32.29%	34.90%	61.47%	61.15%	0.11>0.05 & 0.22
RevHelper$_{exp}$	**66.73%**	76.29%	**63.03%**	**51.29%**	**64.66%**	**65.37%**	**76.02%**	69.10%	**35.86%**	**44.20%**	63.07%	62.11%	*0.01<0.05 & 0.56
RevHelper	65.68%	73.30%	59.73%	50.82%	62.87%	**63.45%**	74.09%	78.98%	35.27%	29.30%	63.83%	**65.03%**	*0.00<0.05 & 0.46

a = model uses comment sentiment group, b = model uses comment sentiment score, and *=significantly higher performance than the best baseline model–Bosu et al.-III-b

and sentiment scores)–predicts the non-useful comments with 23%–49% precision and 17%–26% recall. On the other hand, our model provides 35%–63% precision and 29%–51% recall with an overall prediction accuracy of 65%. While our model has a significant error rate (35%), it is still preferable to the existing alternatives given their poor performance (i.e., 57%–100% false positives) with non-useful comments. Besides, ours is the first model that predicts usefulness during a review submission. The ROC curves in Fig. 14 also show that our model is more effective (i.e., bigger AUC) than each of the baseline models. The box plots in Fig. 15 also clearly demonstrate that our model performs significantly higher (i.e., *p-value=0.00, D>1.00*) than any of the baseline models.

Thus, to answer **RQ$_5$**, our model –RevHelper– outperforms the baseline models [10] significantly, and such findings clearly demonstrate a higher potential of our model for automatic developer support during code reviews.

V. THREATS TO VALIDITY

First, threats to internal validity relate to experimental errors and biases [40]. Our dataset selection involves manual analysis where the *change triggering* heuristic of Bosu et al. [10] was carefully applied. While human errors cannot be avoided during analysis, we train each participant with pilot examples and conduct the analysis under constant supervision. Besides, we perform random cross-check to avoid false-positives in the dataset. Finally, we collected a reasonable sample set of 1,482 review comments as an outcome of 30–35 man-hours work.

Second, the baseline model–Bosu et al.-II-a– returned 100% false-positives for non-useful comments. Our investigation revealed that the suggested sentiment analyzer (i.e., MSR-Splat [27]) found none of our comments *negative* which is unlikely to be true. To mitigate this threat, we repeated experiments using another parallel tool–Stanford Sentiment Analyzer [33], but the overall validation findings did not change.

Third, our study data are from one company and the systems are Python-based. However, Rigby et al. [32] suggest that organizations and frameworks share common reviewing principles, and thus, our findings might also be generalizable.

VI. RELATED WORK

Existing studies on code review explore various dynamics of the available code review practices [18, 20, 30, 31], mine code review analytics [7, 9, 14, 21, 22, 23, 24, 36, 37] or review comment analytics [8, 10, 22, 25], and recommend appropriate reviewers for a code review [8, 19, 29, 34, 35, 39, 41].

While most of these studies perform qualitative or quantitative analysis to derive meaningful insights, there has been a marked lack in the application of such insights to developing tools or techniques that can actually support the code review activities. To the best of our knowledge, there exist not many tools or techniques that can automatically support a code review except the recently emerged code reviewer recommendation systems [29, 35, 41]. In this work, we provide automated support in improving code review comments, and two existing studies–Bosu et al. [10] and Kononenko et al. [22]–are closely related to ours. Bosu et al. first study the perceptions of the professional developers from *Microsoft* on the usefulness of code review comments, and identify several characteristics of useful and non-useful review comments from the qualitative study. Kononenko et al. conduct a similar study involving professional developers from open source domain–*Mozilla*, and analyze the perceptions of the developers on the quality of code reviews. While both studies offer insights on the usefulness of review comments by employing qualitative methods, applicability of such insights (i.e., using automated methods) is yet to be explored. We harvest such learned insights on comment usefulness from them, and provide automated support to code reviews through a model for comment usefulness prediction. Such prediction could also be justified with meaningful rationale (e.g., comment feature values). Our automatic support is found promising and likely to help improve code review comments, especially the non-useful ones. Such exact support was not provided by any of the earlier studies.

VII. CONCLUSION AND FUTURE WORK

To summarize, we propose a novel model for automatic and instant usefulness prediction of code review comments. It exploits two dimensions–comment texts and developer experience–and uses machine learning for the prediction. Our comparative study between useful and non-useful comments first motivates the model. Experiments using 1,482 review comments from four commercial subject systems report 66% overall prediction accuracy which is promising. Comparison with three variants of a baseline model using a case study and a validation set not only validates our empirical findings but also demonstrates the higher potential of our model. In future, we plan to extend our automatic supports for code reviews to personalized recommendation systems.

REFERENCES

[1] Github API library. URL http://github-api.kohsuke.org.

[2] RevHelper portal. URL http://homepage.usask.ca/~masud.rahman/revhelper.

[3] Does modeling with random forests requre cross-validation? URL http://datascience.stackexchange.com/questions/6510.

[4] Stop words. URL https://code.google.com/archive/p/stop-words.

[5] WEKA. URL http://www.cs.waikato.ac.nz/ml/weka/.

[6] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. *TSE*, 28(10):970–983, 2002.

[7] A. Bacchelli and C. Bird. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proc. ICSE*, pages 712–721, 2013.

[8] V. Balachandran. Reducing Human Effort and Improving Quality in Peer Code Reviews using Automatic Static Analysis and Reviewer Recommendation. In *Proc. ICSE*, pages 931–940, 2013.

[9] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The influence of non-technical factors on code review. In *Proc. WCRE*, pages 122–131, 2013.

[10] A. Bosu, M. Greiler, and C. Bird. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proc. MSR*, pages 146–156, 2015.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, C., 1984.

[12] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.

[13] R. P. L. Buse and W. R. Weimer. Learning a metric for code readability. *TSE*, 36(4):546–558, 2010.

[14] J. Czerwonka, M. Greiler, and J. Tilford. Code reviews do not find bugs. how the current code review best practice slows us down. In *Proc. ICSE*, pages 27–28, 2015.

[15] M. Duijn, A. Kucera, and A. Bacchelli. Quality questions need quality code: Classifying code fragments on stack overflow. In *Proc. MSR*, pages 410–413, 2015.

[16] S. Ercan, Q. Stokkink, and A. Bacchelli. Automatic assessments of code explanations: Predicting answering times on stack overflow. In *Proc. MSR*, pages 442–445, 2015.

[17] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic Query Reformulations for Text Retrieval in Software Engineering. In *Proc. ICSE*, pages 842–851, 2013.

[18] K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida. Who does what during a code review? datasets of oss peer review repositories. In *Proc. MSR*, pages 49–52, 2013.

[19] C. Hannebauer, M. Patalas, S. Stnkelt, and V. Gruhn. Automatically recommending code reviewers based on their expertise: An empirical comparison. In *Proc. ASE*, pages 99–110, 2016.

[20] N. Kitagawa, H. Hata, A. Ihara, K. Kogiso, and K. Matsumoto. Code review participation: Game theoretical modeling of reviewers in gerrit datasets. In *Proc. CHASE*, pages 64–67, 2016.

[21] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey. Investigating code review quality: Do people and participation matter? In *Proc. ICSME*, pages 111–120, 2015.

[22] O. Kononenko, O. Baysal, and M. W. Godfrey. Code Review Quality: How Developers See It. In *Proc. ICSE*, pages 1028–1038, 2016.

[23] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proc. MSR*, pages 192–201, 2014.

[24] R. Morales, S. McIntosh, and F. Khomh. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In *Proc. SANER*, pages 171–180, 2015.

[25] S. Panichella, V. Arnaoudova, M. D. Penta, and G. Antoniol. Would Static Analysis Tools Help Developers with Code Reviews ? In *Proc. SANER*, pages 161–170, 2015.

[26] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving Low Quality Stack Overflow Post Detection. In *Proc. ICSME*, pages 541–544, 2014.

[27] Chris Quirk, Pallavi Choudhury, Jianfeng Gao, Hisami Suzuki, Kristina Toutanova, Michael Gamon, Wen-tau Yih, Lucy Vanderwende, and Colin Cherry. Msr splat, a language analysis toolkit. In *Proc. NAACL HLT*, pages 21–24, 2012.

[28] M. M. Rahman and C. K. Roy. An insight into the unresolved questions at stack overflow. In *Proc. MSR*, pages 426–429, 2015.

[29] M. M. Rahman, C. K. Roy, and J. Collins. CORRECT: Code Reviewer Recommendation Based on Cross-Project and Technology Experience. In *Proc. ICSE*, pages 222–231, 2016.

[30] P. C. Rigby and C. Bird. Convergent contemporary software peer review practices. In *Proc. ESEC/FSE*, pages 202–212, 2013.

[31] P. C. Rigby and M. A. Storey. Understanding broadcast based peer review on open source software projects. In *Proc. ICSE*, pages 541–550, 2011.

[32] P.C. Rigby, B. Cleary, F. Painchaud, M. Storey, and D.M. German. Contemporary Peer Review in Action: Lessons from Open Source Development. *TSE*, 29(6):56–61, 2012.

[33] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *Proc. EMNLP*, 2013.

[34] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida. Improving Code Review Effectiveness Through Reviewer Recommendations. In *Proc. CHASE*, pages 119–122, 2014.

[35] P. Thongtanunam, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto. Who Should Review my Code ? In *Proc. SANER*, pages 141–150, 2015.

[36] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. Investigating code review practices in defective files: An empirical study of the qt system. In *Proc. MSR*, pages 168–179, 2015.

[37] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proc. ICSE*, pages 1039–1050, 2016.

[38] E. Wong, J. Yang, and L. Tan. AutoComment: Mining Question and Answer sites for Automatic Comment Generation. In *Proc. ASE*, pages 562–567, 2013.

[39] Y. Yu, H. Wang, G. Yin, and C. Ling. Reviewer Recommender of Pull-Requests in GitHub. In *Proc. ICSME*, pages 609–612, 2014.

[40] T. Yuan, D. Lo, and J. Lawall. Automated Construction of a Software-Specific Word Similarity Database. In *Proc. CSMR-WCRE*, pages 44–53, 2014.

[41] M. B. Zanjani, H. Kagdi, and C. Bird. Automatically recommending peer reviewers in modern code review. *TSE*, 42(6):530–543, 2016.