Overview of the IRSE track at FIRE 2022: Information Retrieval in Software Engineering

Srijoni Majumdar[1,2,*,†], Ayan Bandyopadhyay[1,†], Samiran Chattopadhyay[1,3], Partha Pratim Das[2], Paul D Clough[4,5] and Prasenjit Majumder[1,6]

[1]TCG CREST, West-Bengal, India
[2]IIT Kharagpur, West-Bengal, India
[3]Jadavpur University, West-Bengal, India
[4]TPXimpact London, UK
[5]Sheffield University, Sheffield, UK
[6]DA-IICT Gandhinagar, Gujarat, India

Abstract

Code Comments increase the readability of the surrounding code if they highlight concepts that are not evident from the source code itself. Hence, evaluation of the quality of code comments is important to de-clutter large code bases and remove not useful comments. The Information Retrieval in Software Engineering (IRSE) track aims to develop solutions for automated evaluation of code comments. In this track, there is a binary classification task to classify comments as useful and not useful. The dataset consists of 9048 code comments and surrounding code snippet pairs extracted from open source *github* C based projects. Overall 34 experiments have been submitted by 11 teams from various universities and software companies. The submissions have been evaluated quantitatively using the F1-Score and qualitatively based on the type of features developed, the supervised learning model used and their corresponding hyper-parameters. The best performing architectures mostly have employed transformer architectures coupled with a software development related embedding space.

Keywords

bert, GPT-2, Stanford POS Tagging, neural networks, abstract syntax tree

1. Introduction

Assessing comment quality can help to de-clutter code bases and subsequently improve code maintainability. Comments can significantly help to read and comprehend code if they are consistent and informative. Comment analysis approaches have mainly focused on detecting inconsistent comments [1, 2] but not appreciably on the quality and relevance of the information contained in a comment. A poorly written or superfluous comment duplicating the information evident from source code identifiers can hinder the readability of code, even though it may be consistent [3, 4].

Forum for Information Retrieval Evaluation, December 9-13, 2022, India

*Corresponding author.
†These authors contributed equally.
majumdar.srijoni@gmail.com (S. Majumdar); bandyopadhyay.ayan@gmail.com (A. Bandyopadhyay); samiran.chattopadhyay@jadavpuruniversity.in (S. Chattopadhyay); ppd@cse.iitkgp.ac.in (P. P. Das); p.d.clough@sheffield.ac.uk (P. D. Clough); prasenjit.majumder@gmail.com (P. Majumder)

Several approaches have been proposed to classify comments based on *explicit* syntactic information, such as the presence of specific tags (e.g., @param, @deprecated, etc.), words, and symbols; or *implicit* details, such as the type of associated code construct, length of the comment, parts of speech (POS) and dependency relations of comment words or the cosine similarity of vector representation of words in code-comment snippets [5, 6, 7].

These approaches do not target comment quality evaluation based on the interpretation of the information contained in comments. The syntactic methods used for comment classification need to be augmented so as to extract the semantics of a comment in order to develop an overall quality assessment model.

Further, the perception of quality in terms of the 'usefulness' of the information contained in comments is relative and hence is perceived differently based on the context. Bosu et al. [8] attempted to assess code review comments (logged in a separate tool) in the context of their utility in helping developers write better code through a detailed survey at Microsoft. A similar quality assessment model is important to analyse the type of source code comments that can help for standard maintenance tasks but is largely missing.

Majumdar et al. [4] proposed a comment quality evaluation framework wherein comments were assessed as 'useful', 'partially useful', and 'not useful' based on whether they increase the readability of the surrounding code snippets. The authors analyse comments for concepts that aid in code comprehension and also the redundancies or inconsistencies of these concepts with the related code constructs in a machine learning framework for an overall assessment. The concepts are derived through exploratory studies with developers across 7 companies and from a larger community using crowd-sourcing.

The IRSE track of FIRE 2022, extends the work in [4] and empirically investigates comment quality with a larger set of machine learning solvers and features. The track is targeted to automate program comprehension tasks and subsequently reduce code maintenance overhead. In its first edition, the IRSE track is based on a task for quality evaluation of comments into two clusters - 'useful' and 'not useful'. A 'useful' comment (refer Table 1) contains relevant concepts that are not evident from the surrounding code design, and thus increases the comprehensibility of the code. The suitability of analysing comment quality using various vector space representations of code and comment pairs along with standard textual features and code comment correlation links are evaluated. A total of 34 experiments have been submitted by 11 teams.

2. Related Work

Several approaches exist that attempt to assess the quality of comments by detecting inconsistencies with source code or by classifying comments based on syntactic properties.

Tan et al. [1, 9] use the sequence of occurrence of words (from an enumerated set) in a comment and the surrounding code to develop rules for detecting inconsistent comments related to memory errors.

Ying et al. [10] undertake an empirical study to derive the attributes of the various categories of task comments in Java codes used for developer communication. Storey et al. [11] presented a detailed study to understand how the task comments are interpreted in larger projects during

Table 1
Useful and Not-Useful comments in context of code comprehension

#	Comment	Code	Label
1	/* uses png_calloc defined in pngriv.h*/	/* uses png_calloc defined in pngriv.h*/ PNG_FUNCTION(png_const_structrp png_ptr) { if (png_ptr == NULL || info_ptr == NULL) return; png_calloc(png_ptr); ...}	U
2	/* serial bus is locked before use */	static int bus_reset (. . .) /* serial bus is locked before use*/ { .. update_serial_bus_lock (bus * busR); }	NU
3	// integer variable	int Delete_Vendor; // integer variable	NU

U: Useful; NU: Not Useful

the different phases of the software lifecycle.

Comment quality evaluation: Steidl et al. [7] propose a comment quality detection method by comparing the similarity of words in code-comment pairs using the Levenshtein distance and length of comments to filter out trivial and non-informative comments. Rahman et al. [12] detect useful and non-useful code review comments (logged in review portals) based on attributes identified from a survey conducted with developers of Microsoft [8]. They use textual features (Table 2) and train using a set of 1,200 review comments for automated quality assessment using decision tree and naive bayes algorithms. Recent work in the Declutter Challenge of DocGen2 by Liu et al. [13] detects 'not useful' comments using textual and structural features (Table 2) in a machine learning framework. [4] proposed a framework to evaluate comments based on concepts that are relevant for code comprehension. They developed textual and code correlation features using a knowledge graph for semantic interpretation of information contained in comments (Table 2).

The available approaches mostly target to evaluate the quality of the comments by mining for irrelevant words and phrases, coupled with the repetitiveness in the surrounding constructs. The context based on which the quality is defined is essential, like Rahman et al. [12] assess comments based on attributes limited to code review comments only. Similarly, Majumdar et al. [4] analysed comments by mining concepts that are relevant to code comprehension and can aid in software maintenance tasks. The IRSE track extends the approach proposed in [4] to explore various vector space models and features for binary classification and evaluation of comments.

3. IRSE Track Overview and Data Set

The following section outlines the task descriptions and the characteristics of the dataset.

Table 2
Characteristics of the available quality assessment approaches

Parameters	**steidl et al. [7]**	**rahman et al. [12]**	**liu et al. [13]**	**majumdar et al. [4]**
Textual Features	Length of comments	Numerical estimation of reading ease Stop word ratio % of interrogative sentences Percentage of code constructs in review comments Number of commits on a file authored / reviewed by a developer % of external libraries in review comments	Comment Length Type of Comment - Document level, Block and Line	Comment length Construct Names in Comments Significant Words ratio Descriptive / Conditional % occurrence: Concept Mining for software development domains occurrence of commits, version details
Code Correlation Features	*c_coeff*: Similar words between code and comment based on levenshtein distance < 2	Lexical similarity between words of code and review comments using cosine distances	Similarity between words of code and review comments using cosine distances Percentage of exact match of words between code and comment	Syntactic and semantic matches using vector space structural match using abstract syntax tree in form of knowledge graph
Code in Scope	comment before or after method $\rightarrow$ method name; inline comment $\rightarrow$ constructs in the same line,	code constructs mentioned in review comments	code constructs in next line of comment	
Vector Space Representation	None	tf-idf	fastText (english corpora)	Word2Vec, ELMo (software concepts corpora)
Dataset	1,330 comments (C++ and Java)	1,200 review comments logged in Codeflow	1,194 annotated comments from JabRef project [14]	20206 C comments (from 5 github projects)
Quality classes	Not Useful	Useful and NotâUseful	Informative and Non-Informative	Useful, NotâUseful and Partially Useful

3.1. Task Description

Comment Classification: A binary classification task to classify source code comments as *Useful* or *Not Useful* for a given comment and associated code pair as input.
`Input`: A code comment with surrounding code snippet (written in C)
`Output`: A label (Useful or Not Useful) that characterises whether the comment helps developers comprehend the associated code

Therefore in this classification task, the output is based on whether the information contained in the comment is relevant *and* would help to comprehend the surrounding code, i.e., it is *useful*.

Useful: Comments have sufficient software development concept → Comment is Relevant, and these concepts are not mostly present in the surrounding code → Comment is not Redundant, hence the comment is *Useful*

Not Useful: Comments have sufficient software development concept → Comment is Relevant, and these concepts are mostly present in the surrounding code → Comment is Redundant, hence the comment is *Not Useful*

It may also be the case that comments do not contain sufficient software development concepts → Comment is Not Relevant, hence the comment is *Not Useful*.

It is left to the participants to decide on the threshold value for how many concepts retrieved make a comment relevant or how many matches with surrounding code make a comment redundant.

3.1.1. What is a Relevant Code Comment?

The notion of *relevant* comments refers to those that developers perceive as important in comprehending the associated or surrounding lines of code. These concepts are related to the outline of the algorithm, data-structure descriptions, mapping to user interface details, possible exceptions, version details, etc. In the below examples, the comments highlight useful details about the input data to the function, which is not evident from the associated code itself.

```
1  # works on a two dimensional data matrix (each of size 8) generated from the light rider
     bot module
2    int* flood_fill(self, position, visited) {...}
```

```
1  /* uses png_calloc defined in pngriv.h*/
2 PNG_FUNCTION(png_voidp,PNGAPI
3 png_calloc,(png_const_structrp
4   png_ptr, png_alloc_size_t size),PNG_ALLOCATED)
5 { }
```

Therefore, a relevant comment provides more information for the surrounding code and subsequently aids better comprehension that can improve software maintenance.

Relevant, but Redundant

However, in the example below, even if the comment contains relevant information, it is already available in the associated code rendering the comment redundant.

```
1 // PHP Shutdown method to destroy the global php hash map, using zend hash api's
2 PHP_MSHUTDOWN_FUNCTION(hash) { ... zend_hash_destroy(&php_hashtable); . }
```

3.2. Dataset

We select 5 projects from Github and use the *modified random sampling approach* of Cochran [15] to sample source files with equal probability and hence provide an unbiased representation of the population (C code files with comments). We gather a total of 318 files with 20,206 comments.

Ground Truth Generation: For every comment, a label (Useful or Not Useful) has been generated by a team of 14 annotators. Every comment has been annotated by 2 annotators with a kappa (κ) value of 0.734 (Cohen's metric [16]). The annotation process has been supervised through weekly meetings and brainstorming sessions and peer review. Out of the total 16,000 comments, 2,285 comments were annotated by every individual annotator. A total of 156 man-hours were required to complete the annotation process.

For the IRSE track, we use a set of 9048 comments (from Github) with comment text, surrounding code snippets, and a label that specifies whether the comment is useful or not. Sample data has been characterised in Table 1.

- The development dataset contains 8048 rows of comment text, surrounding code snippets, and labels (Useful and Not useful).
- The test dataset contains 1,000 rows of comment text, surrounding code snippets, and labels (Useful and Not useful).

Table 3
Ranking F1-Scores

Team Name	No. of Runs	Model	Macro F1-Score
MentorX, Mentor Graphics, Siemens	4	Transformer - BERT (base: CodeBERT)	0.9073
Empathy AI Team, Amazon	2	Transformer - GPT-2	0.90472
CITK, Kokrajhar	2	Transformer - BERT (base: uncased BERT)	0.90471
iRel, Indian Institute of Technology Hyderabad	6	Transformer - BERT (base: ALBERT)	0.8961
Conquerers, Indian Institute of Technology Dhanbad	1	SVM - Radial Basis Function	0.8810
Charusat University	6	Random Forest	0.834
Hunters, Jadavpur University	3	Logistic Regression	0.83
Boomerang, Indian Institute of Technology Kharagpur	2	SVM - Radial Basis Function	0.78
SpechTech, Indian Institute of Technology Kharagpur	2	Logistic Regression	0.688
BioNLP, IISER Bhopal	5	SVM - Radial Basis Function	0.53
YA, IISER Kolkata	2	Naive Bayes	0.26

4. Participation and Evaluation

In its first edition, IRSE 2022 received a total of 34 experiments from 11 teams. As this track is related to software maintenance, we received participation from companies like Amazon and Mentor Graphics along with several research labs of educational institutes.

The various teams with the details of their submissions are characterised in Table 3.

Evaluation Procedure: Candidates were asked to submit predicted labels ('useful' or 'not useful') for every data point in the test set of 1000 comments. This was used by our script to generate the precision, recall, and the F1-Score (Macro) using the annotated (golden) labels.

Features: Apart from evaluating the prediction metrics, we analysed the types of features the teams have used to devise the machine learning pipeline. The teams have performed routine pre-processing and have retained the significant words or letters only for both the code and comment pairs. Further, some of the teams have also used morphological features of a comment like a length, significant words ratio, parts of speech characteristics, or occurrence of words from an enumerated set as textual features. To correlate code and comment and detect redundancies, the teams mostly used grep-like string match to find similar words.

Vector Space Representations: Code and comments belong to different semantic granularity which is unified by a vector space representation. The participants have used various pre-trained embeddings to generate vectors for the words like those based on one hot encoding, tf-idf based, word2vec or context aware like ELMo and BERT. Each of the employed embedding models are trained or finetuned using software development corpora.

5. Results and Analysis

The F1-Scores have been analysed based on the machine learning models used, the features and pre-trained embeddings for projection to vector space.

The dataset provided was balanced and had 4015 useful comments and 4033 not useful comments. The textual features used by the teams were mostly related to mining specific words and determining significant words. Similarly, almost all teams used string matching to locate overlapping words between code and comment.

Significant differences were analysed in terms of the pre-trained embeddings used and the machine learning models which contributed to the improvement in the F1-Score.

Machine Learning Architectures: The best F1 score was obtained using GPT architecture, although it is resource critical and can be afforded by software companies. The other machine learning models commonly used are recurrent neural networks, support vector machines, random forest and logistic regression with textual and correlation features. The F1-Score obtained using recurrent neural networks, support vector machines are comparable to the ones obtained from BERT. This is because the dataset is balanced and also due to the use of various textual features apart from numerical vectors (as features).

Pre-Trained Embeddings: Both context-aware and context-independent pre-trained embeddings trained from scratch or finetuned with software development concepts have been used. Results are better with the recently released codeBERT pre-trained embeddings [17] where natural language and programming language pairs are used from software projects of different domains to train using masked language modeling. Comparable results have been obtained by using CodeELMo [4] which trains ELMo from scratch using software development corpora from books, journals, and code repositories.

6. Conclusions

The IRSE track in its first edition empirically investigates various approaches in a machine-learning framework for automated comment quality evaluation. The comments are evaluated based on whether they contain information that can aid in understanding the surrounding code. A total of 11 teams participated and submitted 34 experiments that used various types of machine learning models, embedding spaces, and features. The best F1-Score of 90.8 was reported by experiments conducted using GPT-2 architecture with textual and numerical features from CodeBERT vector space embeddings, to classify comments as 'useful' and 'not useful'.

References

[1] L. Tan, D. Yuan, G. Krishna, Y. Zhou, icomment: Bugs or bad comments?, Association for Computing Machinery's Special Interest Group on Operating Systems Review (SIGOPS), ACM, 2007, pp. 145–158.

[2] I. K. Ratol, M. P. Robillard, Detecting fragile comments, International Conference on Automated Software Engineering (ASE), IEEE, 2017, pp. 112–122.

[3] J. L. Freitas, D. da Cruz, P. R. Henriques, A comment analysis approach for program comprehension, Annual Software Engineering Workshop (SEW), IEEE, 2012, pp. 11–20.

[4] S. Majumdar, A. Bansal, P. P. Das, P. D. Clough, K. Datta, S. K. Ghosh, Automated evaluation of comments to aid software maintenance, Journal of Software: Evolution and Process 34 (2022) e2463.

[5] L. Pascarella, A. Bacchelli, Classifying code comments in java open-source software systems, International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 227–237.

[6] D. Haouari, H. Sahraoui, P. Langlais, How good is your comment? a study of comments in java programs, International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2011, pp. 137–146.

[7] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[8] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, Working Conference on Mining Software Repositories, IEEE, 2015, pp. 146–156.

[9] L. Tan, D. Yuan, Y. Zhou, Hotcomments: how to make program comments more useful?, in: Conference on Programming language design and implementation (SIGPLAN), ACM, 2007, pp. 20–27.

[10] A. T. Ying, J. L. Wright, S. Abrams, Source code that talks: an exploration of eclipse task comments and their implication to repository mining, ACM SIGSOFT software engineering notes, ACM 30 (2005) 1–5.

[11] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, J. Singer, Todo or to bug, International Conference on Software Engineering (ICSE) , IEEE, 2008, pp. 251–260.

[12] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 215–226.

[13] M. Liu, Y. Yang, X. Peng, C. Wang, C. Zhao, X. Wang, S. Xing, Learning based and context aware non-informative comment detection, International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 866–867.
[14] M. Alver, N. Batada, Jabref: Cross-platform citation and reference management software, Open Source, 2003. https://github.com/JabRef/jabref, Last Accessed: December 12, 2020.
[15] J. Kotrlik, C. Higgins, Organizational research: Determining appropriate sample size in survey research, Information technology, learning, and performance journal 19 (2001) 43.
[16] N. Gisev, et al., Interrater agreement and interrater reliability: key concepts, approaches, and applications, Research in Social and Administrative Pharmacy 9 (2013) 330–338.
[17] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).