Evaluating the Relevance of Code Comments Using Classical and Transformer-Based Models

Sandhya Giribabu*[1], Shinigdapriya Sathish†[1], and Vidarshanaa Saravanavel‡[1]

[1]Department of Computer Science and Engineering, SSN College of Engineering, Kalavakkam

Information Retrieval in Software Engineering (IRSE) @ FIRE 2025, December 12–15, 2025, India

Abstract. The Code Comment Classification task at the Information Retrieval in Software Engineering (IRSE)@FIRE 2025 focuses on identifying whether comments associated with C code segments are useful or not useful. This task is framed as a binary text classification problem, aiming to assess the relevance of comments with respect to their corresponding code snippets. Our team participated in this shared task and submitted multiple runs based on both classical machine learning models and transformer-based architectures. The implemented methods include Logistic Regression, Stacked Ensemble (Multinomial Naive Bayes, Decision Tree, and Random Forest), XGBoost, and several pre-trained transformer models such as DistilBERT, CodeBERT, DeBERTa, and RoBERTa. All transformer models were fine-tuned using the Hugging Face Transformers library, leveraging the combined representation of code and comments. The performance of all models was evaluated on the provided training and test corpora, and the top-performing models were selected for final submission.

Keywords: code comment classification, information retrieval, text classification

1 Introduction

A crucial part in the software development and management industry is program comprehension. It is a usual practice to add new functionalities to the existing programs to meet the dynamic requirements. Therefore understanding the relevant parts and functions of the code is extremely important to avoid redundancy and inefficiency [1]. Developers often act themselves as end users to identify any malfunctions [2] and to mine the process. Even though qualified personnel or peer code reviewer can handle this cumbersome process, the resources needed are quite high. Over the years, researchers formalized structured process for code inspection, the need for inspection and debugging and its benefits [1]. Industries like Microsoft, Google and some open source platforms use another version of the peer code review supported with tool-based approaches known as modern code review or contemporary code review [3]. The most useful block in a program review are comments. They are direct and descriptive rather than the source code. This helps reviewers to analyse, add feedback and suggestions which, in a sense, influence the effectiveness of such practices. Nevertheless, some comments can be irrelevant or redundant which make this task even more complicated. So assessing the quality of comments is also necessary as they act as a guide to the reviewer [4].

The Forum for Information Retrieval (FIRE) 2025 started a shared-task this year, named as Information Retrieval in Software Engineering (IRSE) [5], to evaluate the relevance of comments with respect to its surrounding codes. The objective of this shared task is to build a reusable benchmark for evaluating

*Corresponding author: `sandhya2370021@ssn.edu.in`
†`shinigdapriya2370023@ssn.edu.in`
‡`vidarshanaa2370047@ssn.edu.in`

the models to classify the relevance of the comments of given source codes [5]. This is a binary text classification problem to categorize the source code comments as useful or not useful, where the given codes were written in C programming language.

The proposed framework generates features from the given training corpus by using the classical bag of words (BOW) model [6] and transformer architecture based deep learning models [7, 8, 9]. The term frequency (TF) and inverse document frequency (IDF) i.e., TF-IDF [6] and Entropy based method [10] were used as the term weighting schemes. Eventually, the performance of the classifiers like support vector machine, logistic regression and random forest that uses BOW features on the training corpus have been reported. Furthermore, three different attention layer based deep learning models, viz. BERT [7], ALBERT [9], and RoBERTa [8] were implemented to generate semantic features from the given training corpus and then those features were used for comments classification. The top five frameworks that have best performance on the training corpus were chosen based on the F1 score and accuracy. Consequently, these models were implemented on the test data and submitted.

2 Related Works

This section briefly describes the related works in developing an architecture that can be used for comment classification. As comments are written in natural language while codes are programming language, detecting inconsistencies of codes with comments are often difficult. Tan et al. [11] developed a framework known as iComment which combines Natural Language Processing (NLP), Machine Learning, Statistics and Program Analysis techniques to overcome this issue. They experimented on four large code bases, namely, Linux, Mozilla, Wine and Apache. The framework has an accuracy of 90.8–100% which also detects bad comments.

Similarly, the base research work of this challenge, named as Commentprobe [12], was implemented on C codebases. First, a developer survey to study the commenting behaviour among the programmers was conducted and then the ground truth for the comment classification task was generated by manual annotation. They developed pretrained embeddings known as SWVec using the data from the posts in Stack Overflow and literature works. These features are trained using neural network architecture like LSTM and ANN to classify the comments as useful or not, or partially useful and could achieve an F1 score of 86.34%.

Apart from the comment classification, many other attempts have been done to understand and transform the code of different programming languages. Some of the works on different types of embeddings are explored in the paper A Literature Study of Embeddings on Source Code [13].

3 Experimental Design

The given training and test corpora contained comments, their corresponding C code snippets, and class labels indicating whether each comment was useful or not useful. Both corpora were released in CSV format. We extracted the comments and code contexts to form two distinct input variants: one containing only the comments, the other combining both comments and their surrounding code context.

Initially, classical machine learning models such as Logistic Regression (LR) [**?**], Stacked Ensemble Models, Extreme Gradient Boosting (XGBoost) [14], and Random Forest (RF) [15] classifiers were implemented using a TF-IDF based term-weighting scheme [6] following the bag-of-words (BOW) model. For the stacked ensemble, Multinomial Naive Bayes (NB) [16], Decision Tree (DT) [17] , and Random Forest (RF) [18] were used as base learners, with Logistic Regression serving as the meta-classifier. The hyperparameters for these classifiers were tuned using randomized search with k-fold cross-validation on the training corpus to achieve optimal performance.

Subsequently, transformer-based pre-trained models, namely DistilBERT [1] [19], CodeBERT [2] [20], DeBERTa [3], and RoBERTa [4] [8], were fine-tuned on the given training corpus. For these models, the combined representation of comment and code context was used as input. Each model was fine-tuned using 10-fold cross-validation with tokenized sequence lengths fixed to 128 tokens. Training was carried out for 3 epochs with a learning rate of 2e-5, batch size of 16, and weight decay of 0.01. The best configuration for each model was selected based on the weighted F1-score obtained on the validation folds.

The performance of all models was evaluated on the held-out test corpus, and the top-performing models (based on F1 and overall accuracy) were considered for further analysis. The complete implementation was carried out in Python, utilizing the Scikit-learn [5] and Hugging Face Transformers libraries [6].

4 Results and Analysis

The evaluation of traditional machine learning and transformer-based models on the Seed and Seed + Generated datasets shows clear differences in performance. As shown in Table 1, among classical models, the Stacked Model achieved the highest F1 Score on both the Seed (0.78) and Seed + Generated datasets (0.84), while Logistic Regression, Random Forest, and XGBoost achieved F1 Scores ranging from 0.76–0.77 on the Seed dataset and improved to 0.82–0.85 with augmented data. Transformer-based models consistently outperformed classical models, with DistilBERT, RoBERTa, and DeBERTa achieving F1 Scores of 0.83, 0.92, and 0.84, respectively, on the Seed dataset (Tables 2, 3, and 4). Augmenting the dataset further improved transformer performance, with RoBERTa reaching an F1 Score of 0.94, DistilBERT 0.87, and DeBERTa 0.88 on the Seed + Generated dataset. Epoch-wise analysis shows that both training and validation losses decreased steadily, while F1 Scores increased with more epochs, highlighting stable learning and effective generalization. Overall, the results in Tables 5 and 6 demonstrate that transformer-based models, particularly RoBERTa, achieve superior precision, recall, and F1 Scores compared to traditional machine learning approaches, and that data augmentation consistently enhances performance across all models.

Table 1: Performance Comparison of Traditional Machine Learning Models on Seed and Augmented Datasets

Model	Dataset	Precision	Recall	F1 Score	Accuracy	Support
Logistic Regression						
	Seed Dataset	0.79	0.76	0.77	0.79	2291
	Seed + Generated Dataset	0.85	0.84	0.84	0.85	3069
Random Forest						
	Seed Dataset	0.76	0.77	0.76	0.77	2291
	Seed + Generated Dataset	0.82	0.83	0.82	0.82	3069
XGBoost						
	Seed Dataset	0.77	0.78	0.77	0.78	2291
	Seed + Generated Dataset	0.82	0.83	0.82	0.83	3069
Stacked Model						
	Seed Dataset	0.79	0.77	0.78	0.80	2291
	Seed + Generated Dataset	0.85	0.84	0.84	0.85	3069

[1] `https://huggingface.co/distilbert-base-uncased`
[2] `https://huggingface.co/microsoft/codebert-base`
[3] `https://huggingface.co/microsoft/deberta-v3-small`
[4] `https://huggingface.co/roberta-base`
[5] `http://scikit-learn.org/stable/supervised_learning.html`
[6] `https://huggingface.co/docs/transformers`

Table 2: DistilBERT Training and Evaluation Results

Epoch	Training Loss	Validation Loss	F1 Score
Seed Dataset			
1	0.4789	0.4139	0.8196
2	0.3776	0.3868	0.8222
3	0.3308	0.4151	0.8251
Seed + Generated Dataset			
1	0.3794	0.3307	0.8563
2	0.2653	0.3105	0.8711
3	0.2222	0.3046	0.8702

Dataset	Class	Precision	Recall	F1 Score	Accuracy	Support
2*Seed Dataset	Class 0	0.81	0.71	0.76	2*0.83	863
	Class 1	0.84	0.90	0.87		1428
2*Seed + Generated Dataset	Class 0	0.88	0.80	0.84	2*0.87	1257
	Class 1	0.87	0.92	0.89		1811

Table 3: RoBERTa Training and Evaluation Results

Epoch	Training Loss	Validation Loss	F1 Score
Seed Dataset			
1	0.3638	0.3372	0.8864
2	0.2369	0.2126	0.9145
3	0.2035	0.2349	0.9176
Seed + Generated Dataset			
1	0.3000	0.1868	0.9306
2	0.1732	0.1574	0.9363
3	0.1479	0.1481	0.9387

Dataset	Class	Precision	Recall	F1 Score	Accuracy	Support
2*Seed Dataset	Class 0	0.93	0.85	0.89	2*0.92	863
	Class 1	0.91	0.96	0.94		1428
2*Seed + Generated Dataset	Class 0	0.96	0.89	0.92	2*0.94	1257
	Class 1	0.93	0.98	0.95		1811

Table 4: DeBERTa Training and Evaluation Results

Epoch	**Training Loss**	**Validation Loss**	**F1 Score**
Seed Dataset			
1	0.4907	0.4312	0.8140
2	0.3869	0.3816	0.8342
3	0.3379	0.4096	0.8351
Seed + Generated Dataset			
1	0.4045	0.3245	0.8642
2	0.2724	0.2941	0.8707
3	0.3222	0.3170	0.8740

Dataset	**Class**	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Support**
2*Seed Dataset	Class 0	0.82	0.73	0.77	2*0.84	863
	Class 1	0.85	0.90	0.87		1428
2*Seed + Generated Dataset	Class 0	0.89	0.80	0.84	2*0.88	1257
	Class 1	0.87	0.93	0.90		1811

Table 5: Performance on Seed Dataset

Run	Algorithm	Loss Function	Activation Function	Precision	Recall	F1 Score
1	Logistic Regression	Log-Loss	Sigmoid	0.79	0.76	0.77
2	XGBoost	Log-Loss	Not applicable	0.78	0.78	0.78
3	Stacked Model (Base: NB + DT + RF; Meta: Logistic Regression)	final estimator: Log-Loss	Sigmoid	0.79	0.80	0.79
4	Random Forest (max_depth=30, n_estimators=200)	Gini	Not applicable	0.78	0.77	0.77
5	DistilBERT	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.83	0.83	0.83
6	RoBERTa	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.92	0.92	0.92
7	DeBERTa	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.84	0.84	0.84

Table 6: Performance on Seed + Generated Dataset

Run	Algorithm	Loss Function	Activation Function	Precision	Recall	F1 Score
1	Logistic Regression	Log-Loss	Sigmoid	0.85	0.84	0.84
2	XGBoost	Log-Loss	Not applicable	0.83	0.83	0.83
3	Stacked Model (Base: NB + DT + RF; Meta: Logistic Regression)	final estimator: Log-Loss	Sigmoid	0.85	0.85	0.85
4	Random Forest (max_depth=30, n_estimators=200)	Gini	Not applicable	0.83	0.82	0.83
5	DistilBERT	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.87	0.87	0.87
6	RoBERTa	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.94	0.94	0.94
7	DeBERTa	Cross-Entropy Loss	GELU (hidden layers), Softmax (output layer)	0.88	0.88	0.88

5 Conclusion

The experiments demonstrate that transformer-based models significantly outperform traditional machine learning models in classifying the dataset, with RoBERTa consistently achieving the highest F1 Scores. Data augmentation with generated samples improves model performance across both classical and transformer-based frameworks, enhancing precision, recall, and F1 Score. Among classical models, ensemble methods such as the Stacked Model provide the best performance, but still remain below transformer-based approaches. Overall, the results confirm that leveraging pre-trained transformers along with data augmentation is highly effective for robust and generalized classification.

References

[1] M. E. Fagan, Design and code inspections to reduce errors in program development, IBM Systems Journal 15 (1976) 182–211. doi:10.1147/sj.153.0182.

[2] T. Roehm, R. Tiarks, R. Koschke, W. Maalej, How do professional developers comprehend software?, in: Proceedings of the 34th International Conference on Software Engineering, ICSE '12, IEEE Press, 2012, pp. 255–265.

[3] C. Bird, A. Bacchelli, Expectations, outcomes, and challenges of modern code review, in: Proceedings of the International Conference on Software Engineering, IEEE, 2013. `https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/`.

[4] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Comment-Mine—A Semantic Search Approach to Program Comprehension from Code Comments, Springer Singapore, 2020, pp. 29–42. doi:10.1007/978-981-15-2930-6_3.

[5] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. D. Clough, S. Chattopadhyay, P. Majumder, Overview of the IRSE track at FIRE 2022: Information Retrieval in Software Engineering, in: Forum for Information Retrieval Evaluation, ACM, 2022.

[6] C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008. doi:10.1017/CBO9780511809071.

[7] 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). `https://huggingface.co/bert-base-uncased`.

[8] Y. Liu et al., RoBERTa: A robustly optimized BERT pretraining approach, arXiv preprint arXiv:1907.11692 (2019). `https://huggingface.co/roberta-base`.

[9] Z. Lan et al., ALBERT: A lite BERT for self-supervised learning of language representations, ArXiv abs/1909.11942 (2020). `https://huggingface.co/albert-base-v1`.

[10] T. Sabbah, A. Selamat, M. H. Selamat, F. S. Al-Anzi, E. H. Viedma, O. Krejcar, H. Fujita, Modified frequency-based term weighting schemes for text classification, Applied Soft Computing 58 (2017) 193–206.

[11] L. Tan, D. Yuan, G. Krishna, Y. Zhou, /*icomment: Bugs or bad comments?*/, SIGOPS Oper. Syst. Rev. 41 (2007) 145–158. doi:10.1145/1323293.1294276.

[12] S. Majumdar et al., Automated evaluation of comments to aid software maintenance, Journal of Software: Evolution and Process 34 (2022) e2463.

[13] Z. Chen, M. Monperrus, A literature study of embeddings on source code, 2019. doi:10.48550/ARXIV.1904.03061.

[14] Bohacek, M. and Bravansky, M., 2024, June. When XGBoost outperforms GPT-4 on text classification: A case study. In Proceedings of the 4th Workshop on Trustworthy Natural Language Processing (TrustNLP 2024) (pp. 51-60).

[15] B. Xu, X. Guo, Y. Ye, J. Cheng, An improved random forest classifier for text categorization, JCP 7 (2012) 2913–2920.

[16] Xu, S., Li, Y. and Wang, Z., 2017, May. Bayesian multinomial Naïve Bayes classifier to text classification. In International Conference on Multimedia and Ubiquitous Engineering (pp. 347-352). Singapore: Springer Singapore.

[17] Charbuty, B. and Abdulazeez, A., 2021. Classification based on decision tree algorithm for machine learning. Journal of applied science and technology trends, 2(01), pp.20-28.

[18] Maruf, S., Javed, K. and Babri, H.A., 2016. Improving text classification performance with random forests-based feature selection. Arabian Journal for Science and Engineering, 41(3), pp.951-964.

[19] Nair, A.R., Singh, R.P., Gupta, D. and Kumar, P., 2024. Evaluating the impact of text data augmentation on text classification tasks using DistilBERT. Procedia Computer Science, 235, pp.102-111.

[20] Berezovskiy, V., Gorodilova, A., Trofimova, E. and Ustyuzhanin, A., 2023. Machine learning code snippets semantic classification. PeerJ Computer Science, 9, p.e1654.

[21] T. Basu et al., A novel framework to expedite systematic reviews by automatically building information extraction training corpora, arXiv preprint arXiv:1606.06424 (2016).

[22] T. Basu, C. Murthy, A supervised term selection technique for effective text categorization, International Journal of Machine Learning and Cybernetics 7 (2016) 877–892.