Multi-Label Classification of Code Comments for Java, Python, and Pharo

JT Higgins
Computer Science Department
Colorado State University
Fort Collins, Colorado
jthiggin@colostate.edu

***Abstract*—Code comment classification can help developers understand, manage, and prioritize documentation scattered throughout source code. In this work, we describe our approach to the NLBSE'25 tool competition on multi-label code comment classification. We fine-tune a pre-trained language model (DistilBERT) for each of three programming languages—Java, Python, and Pharo—classifying sentences from class comments into multiple categories. Our experiments yield competitive results against the provided baselines, with distinct performance characteristics for each language and category. We detail our preprocessing, modeling, and evaluation strategies, and release our code and instructions for full reproducibility.**

Keywords—multi-label classification, code comments, software engineering, NLP

I. Introduction

Modern software engineering involves dealing not only with code but also large volumes of textual information, including comments, commit messages and issues. Classifying code comments into categories (summary, usage, parameters) can streamline documentation maintenance, improve code comprehension, and aid in automated documentation generation.

The NLBSE'25 tool competition provides a dataset of 14,875 comment sentences spanning three languages (Java, Pharo, Python) and multiple categories per language. Our goal is to build a multi-label classifier that can automatically assign one or more relevant categories to each comment sentence.

We adopt a pre-trained Transformer-based model (DistilBERT) and fine-tune it separately for each language. We report per-category precision, recall, F1 scores, and micro-averages, as requested by the competition, and compare our results to the given baselines.

II. Related work

Previous studies have addressed code comment classification (Pascarella and Bacchelli 2017; Rani et al. 2021) and have shown that NLP-based techniques are effective in identifying the nature of comments. The competition baselines (STACC by Al-Kaswan et al. 2023) rely on Sentence Transformers. In contrast, we leverage a pre-trained DistilBERT model to achieve a good trade-off between performance and computational efficiency.

III. Dataset and preprocessing

The provided dataset (NLBSE'25 competition dataset) includes training and test splits for three languages:

- **Java:** 7 categories (summary, Ownership, Expand, usage, Pointer, deprecation, rational)
- **Python:** 5 categories (Usage, Parameters, DevelopmentNotes, Expand, Summary)
- **Pharo:** 7 categories (Keyimplementationpoints, Example, Responsibilities, Classreferences, Intent, Keymessages, Collaborators)

Each instance is a single comment sentence with a multi-hot label vector indicating the categories it belongs to. We use only the provided training set for fine-tuning. No external data is introduced.

We tokenize sentences using the DistilBERT tokenizer, apply dynamic padding to handle varying sentence lengths, and convert the provided multi-label annotations into binary vectors. We do not perform additional filtering beyond removing unused columns (index, class name, etc.).

IV. Model and tuning procedure

We fine-tune DistilBERT (distilbert-base-uncased) with a classification head for multi-label classification. The model outputs logits for each category, which we convert to probabilities via a sigmoid function. We assign a category if its probability exceeds 0.5.

Hyperparameters:

- Learning rate: 5e-5
- Weight decay: 0.01
- Batch size: 8 for training and evaluation
- Epochs: 1 (for demonstration; could be tuned further)

Due to time constraints and demonstration purposes, we run each language classifier separately for one epoch. Future work could involve hyperparameter tuning, early stopping, or validation splits.

V. Evaluation metrics

As per the competition instructions, we compute precision, recall, and F1 for each category individually, and also compute a micro-averaged F1 across all categories. We use zero_division=0 to handle categories with no predicted positives. After training, we evaluate on the test set provided for each language.

VI. Results

Python:

- For Python, we achieve a micro-F1 of 0.4502. We see moderate performance on categories like *Usage* (F1=0.5625) and *Summary* (F1=0.6622), but we fail to predict positive instances for *DevelopmentNotes* and *Expand*, resulting in 0.0 F1 for these categories.

Java:

- Our micro-F1 for Java is 0.8415, with strong performance on categories like *Ownership* (F1=1.0) and *summary* (F1=0.9054). However, categories like *Expand* and *rational* remain challenging, with low F1 scores (0.1053 for Expand, 0.05 for rational).

Pharo:

- Our Pharo model attains a micro-F1 of 0.5033. Categories like *Example* (F1=0.768) and *Intent* (F1=0.5714) show decent performance, while *Keyimplementationpoints*, *Classreferences*, *Keymessages*, and *Collaborators* are not predicted at all by our model, yielding F1=0.0 for those categories.

In general, we observe that certain categories (e.g., “DevelopmentNotes” in Python and “Expand” in Java) are harder for the model to predict, often receiving zero or near-zero F1 scores. The model performs moderately well on more common or less ambiguous categories (e.g., “Usage” or “Summary”).

Comparing with the baseline (STACC), our approach matches or outperforms in some categories but may lag behind in others. Detailed comparisons can be made once baseline metrics are fully integrated.

VII. Discussion

Our approach, while not outperforming the baseline on all categories, has a key advantage: it achieves these results using a lightweight training configuration. We fine-tuned the DistilBERT model for only one epoch and performed all training and evaluation on a standard CPU environment. This choice significantly reduces computational overhead, training time, and hardware requirements, making the approach more accessible to a wider range of practitioners and researchers who may not have access to high-end GPUs.

With just a single epoch of training, our model already demonstrates moderate success on common categories like *Summary* in Python and *Ownership* in Java. Although more complex categories remain challenging, this initial showing suggests that even minimal fine-tuning on a CPU can yield usable classification performance. Researchers or developers working in resource-constrained environments, such as educational contexts or organizations with limited computational resources, can still attempt code comment classification without substantial time or cost investments.

VIII. Conclusion and future work

This trade-off between accuracy and training complexity is worth exploring further. Future work could investigate how additional epochs, hyperparameter tuning, or lightweight data augmentation strategies might improve performance, potentially closing the gap with the baseline models—still maintaining a relatively fast and accessible training pipeline.

We presented a multi-label classification approach for code comments in Java, Python, and Pharo. Using a DistilBERT-based model and a straightforward fine-tuning strategy, we successfully produced per-category and micro-averaged metrics. Future work includes deeper hyperparameter tuning, improving performance on challenging categories, and leveraging code-contextual features.

We provide our code and instructions on our GitHub repository. This should enable easy replication of our results and further experimentation by the research community.

[1] Rani, P., Panichella, S., Leuenberger, M., Di Sorbo, A., & Nierstrasz, O. (2021). How to identify class comment types? A multi-language approach for class comment classification. *Journal of Systems and Software*, 181, 111047. https://doi.org/10.1016/j.jss.2021.111047Al-Kaswan, A. et al. (2023). STACC: Code Comment Classification using SentenceTransformers. NLBSE 2023 ...

[2] Al-Kaswan, A., Izadi, M., & Van Deursen, A. (2023). STACC: Code Comment Classification using SentenceTransformers. *arXiv:2302.13149*. https://arxiv.org/abs/2302.13149

[3] Pascarella, L. & Bacchelli, A. (2017). Classifying code comments in Java open-source software systems. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR 2017)*, 227–237. https://lucapascarella.com/articles/2017/Pascarella_MSR_2017.pdf

[4] Hugging Face. 2024. *DistilBERT model documentation.* from https://huggingface.co/docs/transformers/en/model_doc/distilbert.

[5] NLBSE 2025 Code Comment Classification Dataset. 2025. *NLBSE 2025 Tool Competition – Code Comment Classification.* GitHub repository. from https://github.com/nlbse2025/code-comment-classification.