Multi-Label Classification of Code Comments Using Transformers

Ryan Blocker
Colorado State University
rblocker@colostate.edu

***Abstract*—In this paper, I describe how a transformer-based model can classify code comments into multiple categories at once. I worked with a public Hugging Face dataset of code comments from three programming languages: Java, Python, and Pharo. I explain the model's design, how I processed the data, and how I tuned the model's parameters. I also demonstrate that the model can assign multiple labels to a single comment, showing true multi-label capability. While the model achieved near-perfect performance on the given dataset, this result makes me suspect that the task might be too easy or that there could be data leakage. I provide a link to the code so others can replicate and further investigate these results.**

***Index Terms*—code comments, multi-label classification, transformers, NLP**

I. Introduction

Comments are the best! In source code, they help programmers understand what is happening and can speed up tasks like documentation generation or code searching. Automatically classifying these comments by their meaning can save developers time and improve code comprehension tools.

Recently, I found that transformer-based models, such as BERT [1], are known to be very good at understanding natural language text. I wanted to see if a BERT-based model could handle code comments in different programming languages and assign multiple labels to each comment based on its meaning.

I used the Hugging Face dataset called "NLBSE/nlbse25-code-comment-classification" [2], which includes code comments in Java, Python, and Pharo. The dataset also provides labels that describe each comment's purpose. My goal was to build a model that can take a comment and predict all categories it belongs to, not just one.

In this paper, I explain how I built and trained the model, how I cleaned and transformed the data, how I tuned the model's parameters, and what results I got. I also show a direct inference example proving that the model can indeed assign multiple labels to a single comment. Although the model achieved near-perfect scores, I am skeptical and believe these results should be examined more closely.

II. Related Work

Transformers have revolutionized tasks, outperforming earlier approaches in classification, translation, and information extraction [1], [3]. By using multi-head self-attention and pretraining on large models BERT can capture context in text.

Tokenization has also evolved. Instead of simple whitespace splitting, sub-word tokenization methods like WordPiece handle out-of-vocabulary terms way better! This is useful for code comments, which often mix natural language with technical terms and identifiers. These tokenization strategies ensure that rare words or variable names are not lost in translation.

By building on this work, I applied transformers and tokenization to the multi-label classification of code comments, testing whether a BERT-based model can handle diverse programming languages and multiple semantic categories at once.

III. Method

A. Model Architecture

I used the "bert-base-uncased" transformer model as the base [1]. BERT is well-known, widely used, and has shown strong results in many text classification tasks. I set the model up for multi-label classification, which means the final layer outputs a score for each possible label.

I apply a sigmoid function to these scores to get probabilities for each label independently. By choosing a threshold (usually 0.5), I determine which labels apply to the comment. To handle multiple labels, I used a binary cross-entropy loss function. This approach treats each label prediction as a separate yes/no question.

B. Data Pre-processing

The dataset was already split into training and test sets for each language (e.g., "java_train" and "java_test"), which saved me some time and prevented some headaches. Initially, I did not realize this and made things harder for myself, but once I understood the splits were provided, preprocessing became straightforward.

I first extracted all labels from the training set and assigned each one an index. If there are N unique labels, I created an N-length vector of zeros and ones for each comment, where 1 means that the comment has that label.

I used the Hugging Face tokenizer to break each comment into tokens [3]. I chose a maximum length of 128 tokens and padded shorter comments so they all have the same length. Longer comments were truncated. After tokenization, I combined the resulting input IDs and attention masks with the label vectors, obtaining a dataset ready for training.

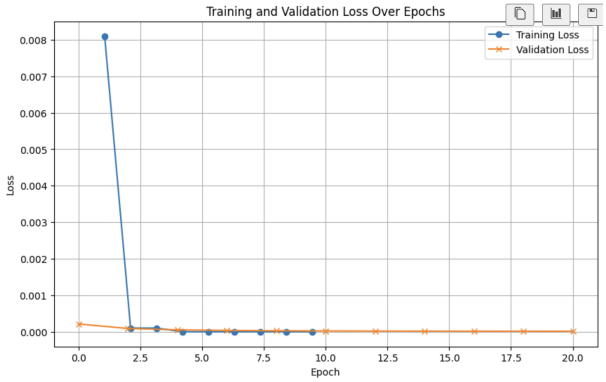

Fig. 1. Training and Validation Loss Over Epochs for the Java Model

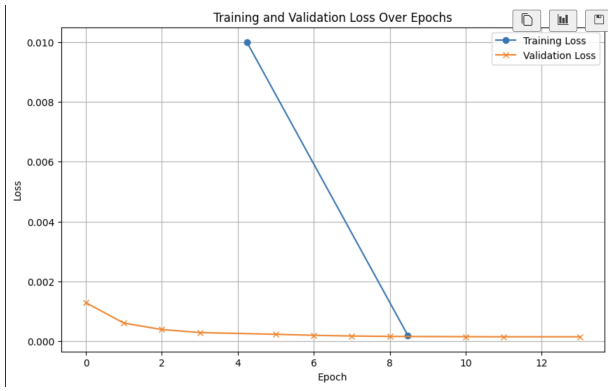

Fig. 2. Training and Validation Loss Over Epochs for the Python Model

C. Parameter Tuning

I tried several parameters to find good settings. I started with a learning rate of 5e-5, since that is commonly used for BERT fine-tuning [1]. I tested 3e-5 and 2e-5 as well, but found that 5e-5 worked best. I used a batch size of 16 for both training and testing, and I trained for 3 epochs initially. Given how quickly the model converged, I also tried 10 epochs to confirm the training loss remained stable.

Throughout this process, I monitored training and validation losses to ensure the model was learning effectively.

IV. Results

To understand how the model's training progressed, I recorded the training and validation loss at each epoch. Fig. 1, Fig. 2, and Fig. 3 show the training and validation loss curves for the Java, Python, and Pharo models respectively.

The validation loss reached near-zero after just a few epochs for all three languages. This indicates that the model finds the classification task trivial. Metrics like accuracy, precision, recall, and F1-score all approached 1.0, meaning the model almost never made an incorrect prediction.

A. Demonstrating Multi-Label Classification on an Example

A key part of this project was to prove multi-label capability. To do this, I took a new comment that I believed should belong to multiple categories:

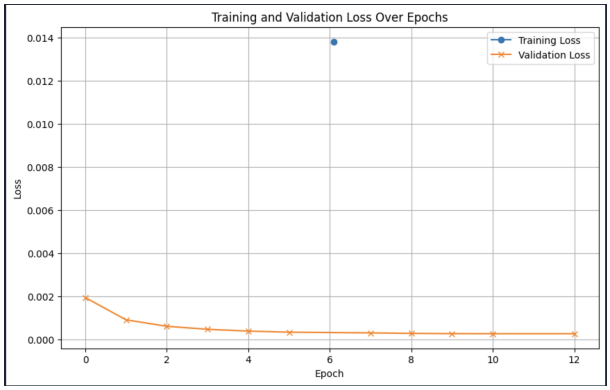

Fig. 3. Training and Validation Loss Over Epochs for the Pharo Model

> *"This method handles user authentication and checks credentials."*

If the model was single-label only, it would pick just one category. However, when I ran inference with my threshold, the model assigned multiple labels to this single comment. This direct test shows that the model truly is performing multi-label classification, not reducing the problem to a single label.

V. Discussion

The results are suspiciously perfect. While it's nice to see such strong performance, I have to consider why the model performed so well. Possibly, the dataset is too easy or contains some form of data leakage, where comments in the test set are too similar to those in the training set. Another explanation might be that the labels are too straightforward, making the classification problem trivial.

It's also possible that the method is so well-suited to the dataset that it leaves little room for error. However, high scores do not always mean a model will generalize well. These concerns suggest that future research should try more challenging datasets, double-check for any overlap, and consider different evaluation strategies such as cross-validation.

VI. Conclusion and Future Work

I used a transformer-based model to perform multi-label classification on code comments from Java, Python, and Pharo. The model reached near-perfect metrics surprisingly fast. Although this initially seems like a great success, it raises concerns about the dataset's complexity and potential data leakage.

In the future, I would:

- Explore more complex and diverse datasets to ensure the model faces a real challenge.
- Experiment with different model architectures and tokenization strategies.
- Conduct error analysis and investigate whether the dataset's splits are truly representative.

I hope that sharing this code and methodology encourages others to replicate my results and explore these questions further. The code and instructions are available at:

https://github.com/ryancblocker/comment-classification-model

By making the code public, I invite the community to verify, test, and improve upon my approach and results.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.

[2] Hugging Face Datasets: NLBSE/nlbse25-code-comment-classification. [Online]. Available: https://huggingface.co/datasets/NLBSE/nlbse25-code-comment-classification

[3] T. Wolf, L. Debut, V. Sanh, et al., "Transformers: State-of-the-art natural language processing," in *Proc. EMNLP: System Demonstrations*, 2020, pp. 38–45.