CS 440 Final Project
Team Name: Ctrl + Alt + Elite
Members: Dureke Sanchez, Daniel Oh

Abstract

This project will be based on the NL-Based Software Engineering competition based in Ontario, Canada April 27-28, 2025: https://nlbse2025.github.io/tools/. Based on code comments from Java, Python, and Pharo projects, we will use a pre-processing technique to remove stop words from our data. Furthermore, we will be using a pre-trained model, “paraphrase-MiniLM-L6-v2”, alongside the few-shot learning algorithm SetFit to classify the multi-label classification problem on the dataset provided by the competition. We also attempted to use BERT for text classification. We will then use the F1 score, runtime, and flops of the predictions to determine performance. Ultimately, our additions to the baseline have dropped performance instead of increasing it. Our code can be found here: https://github.com/Dureke/CS-440-Final-Project

Introduction

This competition aims to categorize code comments into predefined categories based on which programming language the code comments are for. The labels for each language can be found in Figure 1.1.

Since code comments can contain multiple labels, this becomes a multi-label classification problem. For example, if a code comment is for Python, it can have multiple categories like: *summary, parameters, usage, development notes, and expand.*

We also attempt to use BERT for classifying this problem. BERT–Bidirectional Encoder Representations from Transformers–is a natural language processing model (NLP) created by Google. Unlike SetFit, it is not a few-shot algorithm and thus the training of the data takes significantly less time than SetFit does.

Related Work

Figure 1.1: Labels used to classify the code comments of each language.

Java	Python	Pharo
• Summary • Ownership • Expand • Usage • Pointer • Deprecation • Rational	• Usage • Paramets • DevelopmentNotes • Expand • Summary	• Keyimplementationpoints • Example • Responsibilities • Classreferences • Intent • Keymessages • Collaborators

1. **NLSBSE'25 Tool Competition:** This competition is what our Final Project is based on. It is a competition using NLP models to classify code comments and obtain the highest score or perform better than the benchmark provided.
2. **How to Identify Class Comment Types? A Multi-Language Approach for Class Comment Classification:** This paper provides insight towards classifying code comments using predefined categories.
3. **Classifying Code Comments in Java Open-Source Software Systems:** Paper about classifying code comments on open-source Java projects. It goes into details about classification strategies, performance evaluation that can inform approach to classification.

Method

We will be using the SetFit multi-label classification model and use pre-trained sentence transformers to encode text into embeddings. We will have training data and and evaluation data to measure performance. Each sentence has five columns: ***class***, ***comment_sentence***, ***partition***, ***combo***, and ***labels***.

The following are the labels for each programming language of java, python, and pharo, as seen in Figure 1:

- **`java`: [`summary`, `Ownership`, `Expand`, `usage`, `Pointer`, `deprecation`, `rational`],**
- **`python`: [`Usage`, `Parameters`, `DevelopmentNotes`, `Expand`, `Summary`],**
- **`pharo`: [`Keyimplementationpoints`, `Example`, `Responsibilities`, `Classreferences`, `Intent`, `Keymessages`, `Collaborators`]**

Here are the general rules and guidelines for this competition:

1. We can only use the sentence text and the class name for the classification. No external sources can be used to train/ fine-tune the classifiers.
2. Classification can be pre-trained on other datasets but must be fine-tuned on the provided dataset.
3. Participants may preprocess, sample, select subsets of the attributes, split training set into fine-tuning validation set.

Classification performance is measured by $F1$ score achieved by each classifier for each language.

Figure 2.1: Formula for defining metrics, to evaluate performance of the algorithm.

These metrics are defined as follows:

$$P_c = \frac{TP_c}{TP_c + FP_c} \qquad R_c = \frac{TP_c}{TP_c + FN_c} \qquad F_{1,c} = 2 \cdot \frac{P_c \cdot R_c}{P_c + R_c}$$

- TP, FP, FN represent True Positive, False Positive, and False Negative.

- P, R are Preceicion for each category, and Recall for each category.
- F1 scores are the harmonic mean between P and R for each category.

Below is how the competition will be scored:

Figure 2.2: Submission score formula to determine the overall effectiveness of an algorithm.

To rank the competition submissions and determine a winner, we will use the following formula:

$$submission\_score(model) = 0.60 \times avg.\ F_1 + 0.2 \times \frac{(max\_avg\_runtime - measured\_avg\_runtime)}{max\_avg\_runtime)} + 0.2 \times \frac{(max\_avg\_GFLOPS - measured\_avg\_GFLOPS)}{max\_avg\_GFLOPS)}$$

Notable things to consider are:

- avg. F1 is the average score achieved by classifier (SVM, BERT, etc) across 19 categories.
- max_avg_GFLOPS indicate the max average compute in Giga FLOating Point operations per Second.

The competition provides a dataset of 14,875 code comment sentences. Our mission was to outperform the baseline classifier by fine-tuning ML models. Each dataset entry consists of comment_sentence, labels (which are binary vectors for which category the sentence belongs to), and class (name of class the comment comes from).

For preprocessing the dataset, we decided on only one technique of removing stop words, such as “the” and “in”, which do not provide the overall understanding of the sentence. This might also improve training times, as there is less data to iterate over. We utilize the python package “nltk” to import a comprehensive set of English stopwords.

The pre-trained model used to tune the classifiers on the training sets was paraphrase-MiniLM-L6-v2, which is 3 versions more than the baseline of the competition. We attempted a lot of different models, but a majority of the attempted models did not improve performance or actively hindered it.

The platform we ran our code on was Visual Studio Code, in a Jupiter Notebook, in which all relevant commands are present to run. This code can be found the following repository: https://github.com/Dureke/CS-440-Final-Project

We also attempted to use BERT, using an identical pre-processing technique of removing the stopwords. Furthermore, our model used “bert-base-uncased” as our tokenizer and our model. Furthermore, we used the transformers BertForSequenceClassification for classification. We created a train/val split of 80%-20% with the built-in test dataset to test the results.

Other considerations we made include preprocessing using **nltk** in order to improve the mode’s ability to generalize by grouping similar words together. We used a pre-trained BERT model and using the tokenizers converted the data to PyTorch tensors and create DataLoader instances for training and

testing. We use a F1 micro score to consider all true positives, false positives, and false negatives. F1 macro is for then averaging the scores.

Results

In our SetFit algorithm, we achieved a GFLOPs total of 1997.27, with an average runtime of 0.74 seconds per prediction. For the precision, recall, and f1 scores, we obtained these results, as seen below in Figure 3.1.

Figure 3.1: SetFit prediction results for the Model "paraphrase-MiniLM-L6-v2".

	lan	cat	precision	recall	f1
0	java	summary	0.884746	0.877803	0.881261
1	java	Ownership	1.000000	1.000000	1.000000
2	java	Expand	0.364341	0.460784	0.406926
3	java	usage	0.933333	0.812065	0.868486
4	java	Pointer	0.790393	0.983696	0.876513
5	java	deprecation	0.714286	0.666667	0.689655
6	java	rational	0.255319	0.352941	0.296296
7	python	Usage	0.761905	0.661157	0.707965
8	python	Parameters	0.785185	0.828125	0.806084
9	python	DevelopmentNotes	0.233333	0.512195	0.320611
10	python	Expand	0.422414	0.765625	0.544444
11	python	Summary	0.666667	0.536585	0.594595
12	pharo	Keyimplementationpoints	0.500000	0.837209	0.626087
13	pharo	Example	0.875000	0.882353	0.878661
14	pharo	Responsibilities	0.441176	0.288462	0.348837
15	pharo	Classreferences	0.031250	0.250000	0.055556
16	pharo	Intent	0.809524	0.566667	0.666667
17	pharo	Keymessages	0.653061	0.744186	0.695652
18	pharo	Collaborators	0.120000	0.600000	0.200000

After running the averages through the score function, which computes the submission score based off of Figure 2.2. Thus, for this model, our submission score is 0.65.

For our BERT algorithm, we failed to fully obtain a result. While the model does tune the classifiers, it fails to predict, ultimately leaving an undesired submission score of 0.

Discussion (Code and Tool)

First using nltk, we did text preprocessing to remove "stop words" which are common words like "the", "and", "is". The main driver for this is to improve overall model performance and for more efficient filtering for our specific text classification process. We then did a tag count and a word count, and returned the 10 most common sentences and words for sanity check, after removing the stop words. By doing this, we can then figure out which high-frequency words contribute most to predictive tasks like text classification. This also allows us to measure a way to detect anomalies, for further preprocessing data. As seen in Figures 4.1 and 4.2, removing stop words had a significant contribution to removing redundant and common words throughout the code comments.

Figure 4.1: Most common tags and words, before removing stop words

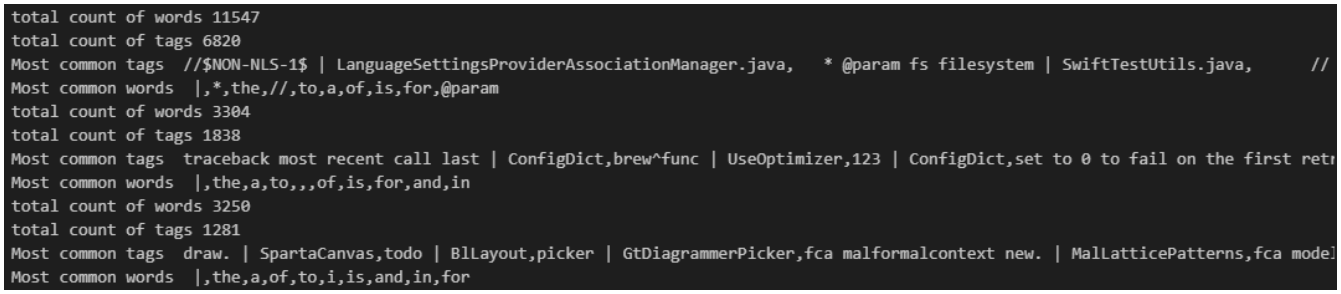
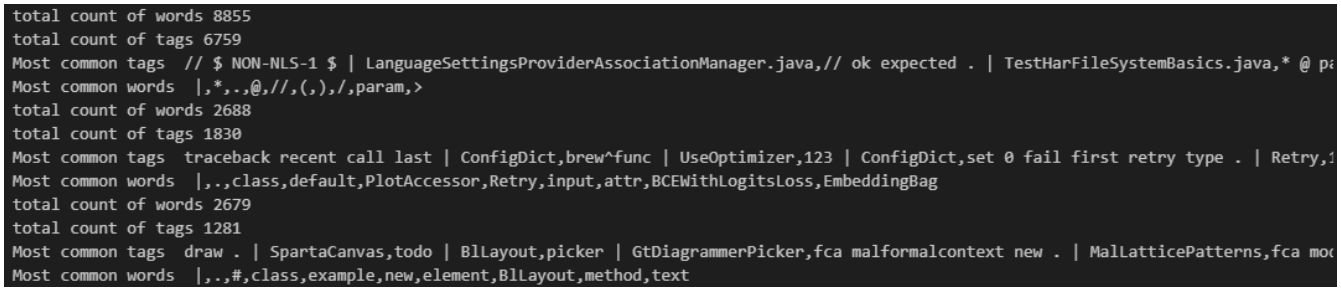


Figure 4.2: Most common tags and words, after removing stop words



Our idea behind removing the stop words was to first reduce the complexity by removing redundant data. Common words such as “the” offer no insight to an algorithm, and may confuse a model by providing a common link between two code comments that have completely different labels. Thus, we hoped to remove such tokens from the equation. Furthermore, we intended to reduce training times by significantly reducing the number of words to be evaluated. The number of words went from 11547 to 8855 words in Java alone–a 23% reduction of data. Ultimately, this failed to provide meaningful improvement towards the baseline.

In our BERT algorithm, while data is properly trained, it fails to utilize the pre-existing code to evaluate the trained results. As a result, we eventually discarded it and focused on our SetFit model instead.

Conclusion

While updating the version of the sentence transformer used from the baseline had subtle improvements over the submission score, our attempt at removing stop words from the dataset has hindered the training algorithm. It is possible that the pre-trained model used relied on those stopwords and thus the removal of them hindered its ability to tune the classifiers. Thus, our modifications to the baseline have worsened the overall few-shot algorithm.

Future Work

Further tuning with the BERT algorithm is an option to create a vastly different approach to the multi-label classification of the code comment dataset. Furthermore, a pre-trained model that trained on datasets with removed stopwords could be a potential avenue of further exploring the pre-processing technique.

References

"Colab Notebook for Code Comment Classification." *Google Colab*, https://colab.research.google.com/drive/1GhpyzTYcRs8SGzOMH3Xb6rLfdFVUBN0P#scrollTo=S5oC0iUlxHzz. Accessed 7 Dec. 2024.

Hugging Face. "SetFit: Efficient Few-Shot Learning with Sentence Transformers." *GitHub*, https://github.com/huggingface/setfit. Accessed 7 Dec. 2024.

Hugging Face. sentence-transformers/paraphrase-MiniLM-L6-v2. Hugging Face, 2021, https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2. Accessed 8 Dec. 2024.

NLBSE 2025. "NLBSE 25 Code Comment Classification." *GitHub*, https://github.com/nlbse2025/code-comment-classification. Accessed 7 Dec. 2024.

NLBSE. "Tools." *NLBSE 2025*, https://nlbse2025.github.io/tools/. Accessed 7 Dec. 2024.