Issue Report Classification

Mudita Shakya
Università degli studi dell'Aquila, Italy
Email: mudita.shakya@student.univaq.it

Academic year: 2023–2024

1 Problem Description

Issue tracking plays a significant role in the software development life cycle. It extends beyond maintaining a list of defects encountered in a project. It also includes documenting technical debt, refactoring, enhancements, and feature requests made throughout the development of a software system. Development teams can use issue-tracking systems as a framework for managing and organizing the development work to improve the system and address the needs of the project stakeholders.

Software version control and management tools like GitHub and GitLab provide built-in issue-tracking systems. Labels can be assigned to categorize issues based on their priority, type, or any criteria specific to the company or development team. Still, these need to be manually added by the project developers. Manual labeling is tedious, labor-intensive, and error-prone, as it relies on human input. Automatic classification of issue reports is therefore key to improving prioritization and achieving effective issue management in the software development process.

Different techniques have been explored for implementing a good automatic issue report classifier. Earlier studies used traditional machine learning (ML) techniques, but over the last few years, the focus has shifted to using deep-learning algorithms, more specifically pre-trained language models, and experimenting with different training methods for the implementation. These techniques are effective at improving the performance over the traditional ML methods.

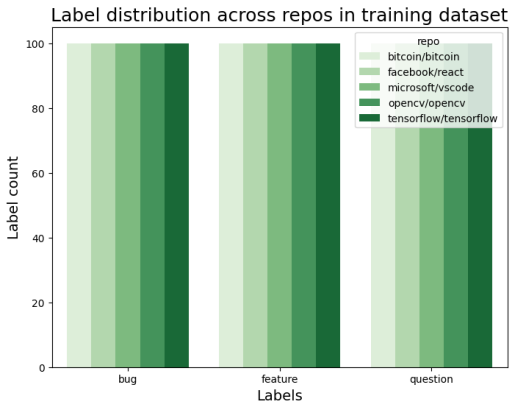
This project is based on the NLBSE 2024 Issue Report Classification competition. The competition involves building and assessing a set of multi-class classifiers to classify issue reports based on the type of information they convey. The competition provides training and test datasets with issue reports collected from 5 different GitHub repositories. The objective is to train a multi-class classifier for each of those repositories, which results in 5 different multi-class classification models. This project explores the different techniques to train multi-class issue-label classification models and compares the performance of the various techniques against one another.

2 Dataset

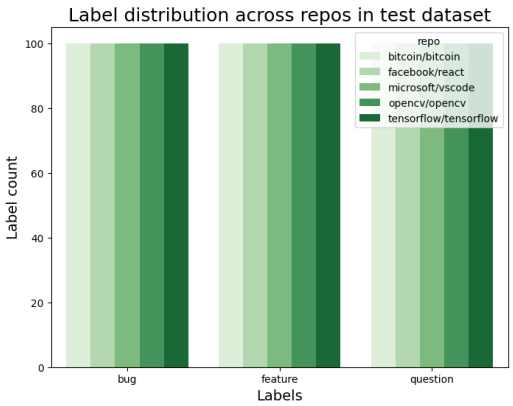
2.1 Data source

The dataset for this project is from the NLBSE 2024 Issue Report Classification competition. For this year, the competition focused on few-shot learning with a SentenceTransformer as a baseline. The dataset provided contains **3000** labeled issue reports extracted from **5** real open-source GitHub projects which are as follows:

1. bitcoin/bitcoin
2. facebook/react
3. microsoft/vscode
4. opencv/opencv
5. tensorflow/tensorflow



(a) Training set



(b) Test set

Figure 1: Data distribution in training and test sets across labels and repositories

Since the focus this year was on few-shot learning, the dataset was much smaller (3000) than the one provided in previous iterations (1.4 million) of the competition. The dataset was available to download from the associated GitHub repository and was already split into a training set (50%) and a test set (50%). Each issue was labeled with one of the following classes:

1. bug
2. feature
3. question

2.2 Data exploration

The training set and test set were both found to have balanced counts for the different label classes, with each class having 500 data points in the training and test sets. The number of issues from the different repositories was also found to be balanced, with each repository having 300 data points. Each issue report includes its *repo*, *created at*, *label*, *title*, and *body*. The label column indicates the assigned class of the issue. This data is a filtered version of the data used in the previous iteration of the competition, so the multi-labeled issue reports have been removed, and the requirement has been set to have the classifier assign one label to the issue.

Set	Bug	Feature	Question	Total
Training set	500	500	500	1500
Test set	500	500	500	1500

Table 1: Distribution of data in the training and test sets across classes

2.3 Data Preprocessing

Before training the model, it is necessary to clean and preprocess the data. The preprocessing steps followed are discussed as follows:

2.3.1 Text Cleaning and Feature Extraction

The dataset contains two columns: **title** and **body,** which together make up the complete GitHub issue report. For simplification, the two column values were combined using string concatenation to have a new column: **issue text**. Then, simple steps were applied to clean the text: removing multiple white spaces from the text, changing all characters to lowercase, and identifying and cleaning out unnecessary parts of the text using regex expressions. The data classes were then encoded so that the labels could have numerical representations instead. For this, a simple mapping was used: bug: 0, feature: 1, and question: 2. Ekphrasis [1] provides

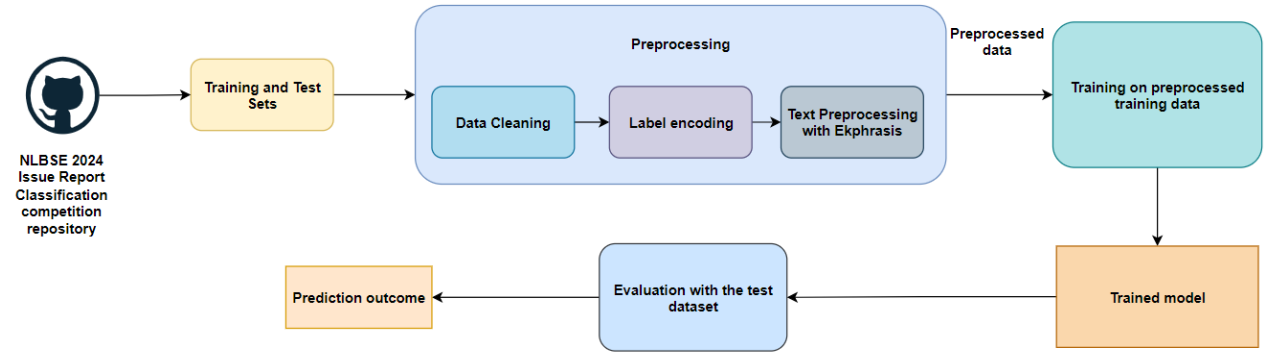


Figure 2: Project Pipeline

a text preprocessing pipeline with different functionalities including social tokenizer, word segmentation, spell correction, and customization. It was used to perform the additional text preprocessing steps including text normalization and identification and replacement of URLs, emails, timestamps, special characters such as emojis, and symbols with specific tokens. It was also used to perform word segmentation, and spelling correction using the Twitter corpus and tokenization. Both the training set and test set were passed through these steps to prepare the preprocessed training and test datasets.

3 Methodology

Different techniques for implementing a good multi-class classification model were explored throughout this project. Due to the limited number of samples in the provided dataset, observations were also made on the impact of the dataset size on the performance of the different approaches. Figure 2 shows the complete pipeline of the project.

The techniques explored include:

1. Traditional ML methods like XGBoost and Random Forest for multi-class classification
2. BERT-based classification method
3. Few-shot Learning with Sentence Transformer

3.1 Traditional Machine Learning methods

Traditional supervised machine learning methods have been effective in multi-class classification problems, especially in cases when a large amount of data is not available.

XGBoost (eXtreme Gradient Boosting) is a popular machine-learning algorithm that works under the Gradient Boosting framework. It provides the benefit of parallel tree boosting that allows to perform analytics in a fast and accurate way. XGBoost provides two objective functions: multi:softmax and multi:softprob for multi-class classification problems. The multi:softmax function only returns the class with the highest probability, so it was used.

Random Forest classifier [2] combines a forest of decision trees grown on random input vectors and splits nodes on a random subset of features for multi-class classification. Random Forest classification models are known to be robust and competent, with a capacity to cope with huge feature spaces.

3.2 BERT-based classification method

BERT-based classifiers are effective at performing multi-class classification, as demonstrated by the winners [3] of NLBSE 2022 Issue Report Classification competition. They used BERT pre-trained model and fine-tuned it using the GitHub issue report dataset. The approach outperformed the FastText approach, which was the baseline at the time.

DistilBERT is a smaller transformer model which was pretrained using the BERT base model as a teacher. DistilBERT has about half of the total number of parameters of BERT base, but still retains 95% of BERT's

performance on language understanding benchmark GLUE. DistilBERT tokenizer was used to tokenize the **issue text** column. The tokenizer uses the encode_plus method to perform tokenization and generate the ids and attention mask. To prepare the DistilBERT language model for classification, a dropout layer followed by a linear layer was added to the network. Cross-entropy was used as the loss function, and the Adam optimizer was used to update the weights of the network to improve its performance. For fine-tuning the model, the model is trained on the preprocessed training set. The training details for fine-tuning the DistilBERT model were as follows:

- **Model used:** `distil-bert-uncased`
- **MAX_LEN:** 512
- **TRAIN_BATCH_SIZE:** 4
- **VALID_BATCH_SIZE:** 2
- **EPOCHS:** 1
- **LEARNING_RATE:** 1e-05
- **tokenizer:** DistilBertTokenizer
- **Loss function:** Cross-entropy
- **Optimizer:** Adam

The model gets the data based on the batch size and the output from the model and the actual classes are compared to compute the loss. The calculated loss is used to optimize the weights of the neurons in the network. The fine-tuned model is then used to predict the classes for the test set.

3.3 Few-shot Learning with Sentence Transformer

Few-shot learning with pretrained language models is a promising solution to the problem of having limited labeled data. Sentence Transformer Finetuning (SetFit) is a framework for efficient few-shot fine-tuning of Sentence Transformers. SetFit has been found to achieve high accuracy with little labeled data. The winners of the NLBSE 2023 Issue Report Classification competition [4] used a supervised approach for training the *all-mpnet-base-v2* sentence transformer using the SetFit framework and achieved an F1-micro score of 0.8321.

SetFit first fine-tunes a Sentence Transformer model on a few labeled examples. It then trains a classifier head on the embeddings generated from the fine-tuned sentence transformer. SetFit allows for achieving high accuracy with smaller models, and the few-shot learning method is suitable for cases where labeled data is scarce and difficult to obtain.

For this project, `bge-base-en-v1.5`, which is a sentence transformer model in the top-10 ranking in the MTEB (Massive Text Embedding Benchmark) leaderboard, was used. The training details are as follows:

- **Model used:** `BAAI/bge-base-en-v1.5`
- **Loss Class:** Cosine Similarity Loss
- **Batch Size:** 4
- **Number of epochs:** 1
- **Number of iterations:** 20

4 Evaluation metrics

The different techniques explored in the project are evaluated using the following metrics

- **Precision:** Precision is a measure of how many of the positive predictions made are correct. It is computed by dividing the number of correctly predicted positive labels by the total number of positive predicted observations for a class.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

	XGBoost			Random Forest			BERT-based classifier		
Label	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Bug	0.6673	0.75	0.7027	0.6732	0.756	0.7112	0.2915	0.58	0.3773
Feature	0.7022	0.6539	0.6717	0.6729	0.6599	0.6637	0.2813	0.384	0.2482
Question	0.6403	0.592	0.6092	0.6248	0.556	0.5859	0.1218	0.2319	0.1405
Average	0.6699	0.6653	0.6612	0.6569	0.6573	0.6536	0.2315	0.3987	0.2553

	Few-shot Learning with Sentence Transformer			State-of-the-Art		
Label	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Bug	0.8177	0.794	0.8049	0.8464	0.8399	0.8426
Feature	0.8103	0.8539	0.8309	0.8448	0.8699	0.8557
Question	0.7705	0.744	0.755	0.8001	0.7699	0.7827
Average	0.7995	0.7973	0.797	0.8304	0.8266	0.827

Table 2: Comparison of the performance of different methods

- **Recall:** Recall gives the proportion of actual positive cases that were correctly identified. It is computed by dividing the number of successfully predicted observations by the total number of observations in that class.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-score:** The F1-score is the harmonic mean of precision and recall values for a classification problem.

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The precision, recall, and F1-score computed for each class are micro-averaged as the cross-class aggregation method to compute the global scores.

5 Results and Discussion

Table 2 shows the comparison of the results obtained by training classifiers based on different techniques. The results from the experiments are compared against the baseline results provided in the competition repository. The baseline results have been computed using a Sentence Transformer fine-tuned using SetFit. Although a similar approach is used in this project as well, the preprocessing steps applied along with the base model used for the fine-tuning are different.

From the results, it can be observed that the Few-shot training method with SetFit provides the best overall results and there is a definite improvement from the traditional machine learning methods. The lackluster performance of the fine-tuned DistilBERT classifier can be attributed to the lack of training data since language models like BERT typically require a large dataset for pretraining and fine-tuning purposes. This result confirms the effectiveness of the Few-shot learning approach with sentence transformer, especially in the case when there is limited labeled data.

The `bge-base-en-v1.5` sentence transformer provides an overall F1-score of 79% which is close to the SOTA F1-score value of 82%.

6 Conclusion

This project explored different techniques for training and fine-tuning models for multi-class issue report classification. Various models ranging from traditional machine learning methods with Random Forest and XGBoost to fine-tuning of pretrained language models were trained and evaluated on the provided training and test set. A complete machine learning pipeline starting from data cleaning, preprocessing, training, and inference was carried out for all the investigated approaches. While the best-performing model based on Few-shot learning with Sentence Transformer did not outperform the baseline, the model achieved a good performance on the

evaluated metrics. The experimental results confirm the notion that few-shot learning approaches are highly effective with small, high-quality datasets, and modern methods with pretrained language models tend to outperform ones based on traditional machine learning.

References

[1] Kevin Gimpel, Nathan Schneider, Brendan O'connor, Dipanjan Das, Daniel P Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, 2011.

[2] Anita Prinzie and Dirk Van den Poel. Random forests for multiclass classification: Random multinomial logit. *Expert systems with Applications*, 34(3):1721–1732, 2008.

[3] Mohammed Latif Siddiq and Joanna CS Santos. Bert-based github issue report classification. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, pages 33–36, 2022.

[4] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. Few-shot learning for issue report classification. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, pages 16–19, 2023.