Analysis and Detection of Information Types of Open-Source Software Issue Discussions Based on Deep Learning & Logistic Regression

Than Than Swe

Department of Computer Science and Engineering Seoul
National University of Science and Technology

***Abstract*—Due to the presence of valuable information about the software project, extracting relevant information from discussion threads in Issue Tracking Systems (ITS) is an important task to satisfy the diverse needs of Open-Source Software (OSS) stakeholders. However, it can be difficult to navigate this information due to their length and vast number of issues. In** [1]**, the authors have conducted qualitative content analysis of 15 complex issue threads from three projects on GitHub and identified 16 different types of information present in these discussions. Additionally, to address the challenges mentioned, the authors explored supervised learning classification techniques namely, Logistic Regression and Random Forest to detect information types. Our work is an extension to** [1] **by collecting additional data and reporting on a series of experiments with Artificial Neural Networks namely, Convolutional Neural Networks and transfer learning using the pre-trained language model Bidirectional Encoder Representations from Transformers (BERT) in addition to Logistic Regression. We show that with additional data and balancing the classes in the dataset via oversampling of the minority class(es), a simple classifier can outperform the previous work in** [1] **by a huge margin.**

***Index Terms*—Issue tracking system, collaborative software engineering, logistic regression, convolutional neural network (CNN), transfer learning.**

I. INTRODUCTION

Issue Tracking Systems (ITSs) are used to manage various aspects of the software development process, including bug reports, feature requests, documentation updates, and general tasks. ITSs serve as a central platform for handling these issues and play a crucial role in supporting software engineering activities such as bug triaging, impact analysis, and release planning. In the context of Open-Source Software (OSS) projects, ITSs such as the *Issues* feature of GitHub facilitate engagement and collaboration among different stakeholders involved in OSS projects throughout the project life cycle. Not to mention these ITSs serve as valuable repositories of information about the software project, benefiting developers, users, and other stakeholders by providing insights and updates on project-related matters.

Deep learning has achieved remarkable results in speech recognition in recent years. In the field of natural language processing (NLP), deep learning methods have been primarily focused on learning word vector representations using neural language models. These models aim to encode semantic features of words by projecting them from a sparse, 1-of-V encoding (where V is the vocabulary size) to a lower-dimensional vector space through a hidden layer. Word vectors serve as feature extractors and provide dense representations of words, where words with similar meanings are closer to each other in terms of Euclidean or cosine distance in the lower-dimensional vector space. This approach enables effective composition over the learned word vectors for classification tasks. Overall, deep learning models have demonstrated their ability to capture and leverage semantic information from words, leading to significant advancements in natural language processing. CNN which utilizes layers of convolving filters applied to local features have been shown to be effective for NLP and have achieve excellent results in semantic parsing [2] search query retrieval [3], sentence modelling, and other traditional NLP tasks. In our work, we train a simple CNN with one layer of convolution on top of word vectors. Moreover, we utilize the BERT Transformer [4], a model architecture that eliminates the utilization of recurrence and instead heavily relies on an attention mechanism to establish comprehensive connections between input and output.

II. RELATED WORKS

Our work is an extension of the previous studies done in [1], [5] focusing on the classification of software-related texts as a natural language processing (NLP) issue. A number of such classification related work has been done in the past. In [6], Ko and Chilana have conducted the identification of topics involved in discussion threads in bug reports. But their purpose is to understand the focus and the dynamics of such discussions. Our goal which is on par with [1] is to identify useful information types. Viviani et al. [7] have analyzed the design topics presented in the discussion of pull requests at the paragraph level in order to explore the types of design-related information. While we follow a similar manual annotation approach, our work focused on a sentence-level analysis of a wider range of information that goes beyond a design perspective.

Our interest is in identifying information types which will help reduce the amount of time a user would take to navigate through discussion threads and retrieve the information relevant to their situation. In this work we mainly focus on evaluation of the performances of supervised learning models for the classification purpose. For this we employ a simple Logistic Regression similar to [1]. We also explore the performance of Artificial Neural Networks particularly CNN and the Transformer model, BERT. In [5], classification of issue

discussions has been performed on the same dataset as used in [1] with a number of state-of-the-art language models such as the transformer model BERT.

III. Issue Comment Dataset Analysis

In this section, we discuss the process of data acquisition, manual annotation/feature extraction, preprocessing, and balancing training data.

In our study, we collected an additional 3400 sentences from five libraries: Tensorflow, scikit-learn, spaCy, Keras, and PyTorch. These sentences were extracted from closed issue comments that had more than 40 comments. To ensure the quality of the data, we manually labeled the collected sentences.

Following the preprocessing steps outlined in the original paper [1], we performed several transformations on the data. Firstly, we replaced all code blocks with the token "code" to remove any code-specific information. Additionally, we replaced URLs surrounded by brackets to further generalize the data.

Solution Discussion	2952
Social Conversation	795
Investigation and Exploration	749
Contribution and Commitment	655
Bug Reproduction	618
Usage	368
Motivation	352
Observed Bug Behaviour	303
Potential New Issues and Requests	253
Workarounds	228
Task Progress	179
Expected Behaviour	156
Social Convesation	137
Action on Issue	61
Testing	34
Future Plan	20
Solution Usage	10

Name: Code, dtype: int64

Table I: Distribution of the percentage of the identified information types in the selected issues.

After preprocessing the data, we observed that the labels of the code columns were imbalanced. To address this issue, we decided to remove certain features that were contributing to the imbalance. These features included Testing, Future Plan, Solution Usage, and Solution Discussion.

To mitigate the class imbalance problem, we employed the RandomOverSampler technique. This technique is commonly used to oversample the minority class or classes in an imbalanced dataset. It works by randomly duplicating examples from the minority class(es) until a more balanced distribution is achieved.

By applying the RandomOverSampler technique, we aimed to create a more balanced dataset for our analysis. This approach helps to ensure that the classification model can learn from sufficient examples of both the majority and minority classes, thereby reducing any bias towards the majority class. Note that these steps were taken as part of our data preprocessing and balancing strategy to address the class imbalance issue in our dataset (Figure 1).

Next, we performed stemming, a natural language processing step where words are reduced to their root or base form called a stem, by removing any suffixes or prefixes. Stemming is an important technique in NLP because it reduces word variations, helps in standardizing and normalizing for consistency in text representation, vocabulary reduction, improving information retrieval and so on.

IV. Learning Based Classifiers

A. *Logistic Regression*

This model estimates the probability of an event or class based on a linear combination of the input features. It is essentially a binary classifier, but it can be generalized to a multi-class logistic regression classifier using a one-vs-rest scheme in which the model treats each label as a binary classification problem (i.e., the samples with the target label being one class and the samples with all the other labels as being another class). We created a pipeline using the Pipeline class from scikit-learn. The pipeline consists of two components: TF-IDF (Term Frequency-Inverse Document Frequency) vectorization and Logistic Regression. The first step in this model is TF-IDF vectorization where every word in the text is represented as a numerical word vector capturing the importance of words and word combinations. The ngram_range was set to (1, 2). This parameter specifies that both single words and two-word combinations (bigrams) should be considered as features. The second step which is Logistic Regression, then uses this vectorized data to train a classification model. For this model, the maximum number of iterations was set to 1000 and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (lbfgs) algorithm was set as the solver. After this, we performed grid search for hyperparameter tuning. The the maximum document frequency threshold for the TF-IDF vectorization parameter "max_df", was set to three different values: [0.8, 0.9, 1.0]. The parameter represents the maximum proportion of documents in which a term can appear to be considered relevant. The Logistic Regression parameter is the regularization strength. Three different values [0.1, 1, 10] are specified. Cross-validation was performed on each combination to select the best hyperparameter configuration based on accuracy metric. The optimal vectorization parameter was found to be 0.8 and regularization strength was 10.

B. *Convolutional Neural Network Model*

The model architecture is a derivation of the CNN architecture of Yoon Kim [5]. Every word in the comment is embedded into a word vector. Let $\boldsymbol{x}_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to the i-th word in the sentence. A sentence with n words is therefore represented as

$$\boldsymbol{x}_{1:n} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \ldots \oplus \boldsymbol{x}_1$$

where $\oplus$ is the concatenation operator.

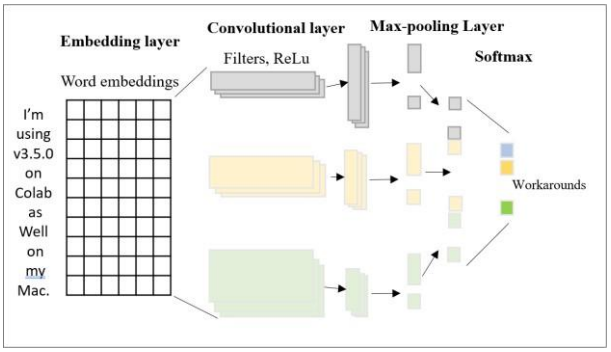

Figure 1: CNN Model architecture.

Generally, $\boldsymbol{x}_{i:i+j}$ represents the concatenation of words $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+j}$. The convolutional operator involves filter $\boldsymbol{w} \in \mathbb{R}^{hk}$ is applied to window of h words to produce new feature

$$c_i = f(w \cdot \boldsymbol{x}_{i:i+h-1} + b)$$

where, $f = \text{ReLu}$, $b = \text{bias term}$. This feature is applied to each possible window of words in the sentence $\{\boldsymbol{x}_{1:h}, \boldsymbol{x}_{2:h+1}, \ldots, \boldsymbol{x}_{n-h:n}\}$ to produce a *feature map*

$$\boldsymbol{c} = [c_1, c_2, \ldots, c_{n-h+1}],\ c \in \mathrm{R}^{\mathrm{n\text{-}k+1}}$$

We then apply GlobalMaxPooling over the feature map and take the maximum value max{c} as the feature corresponding to this feature. We use multiple filters on the same window size to obtain multiple features. These filters form the penultimate layer and are passed to fully connected softmax layer whose output is the probability distribution over labels. For regularization, we employ dropout on the penultimate layer. Given our current dataset, a dropout layer of rate 0.3, filter window of size 4 and 128 feature maps gives the highest accuracy. We additionally included a dense layer with 64 units and "ReLu" as activation function. "Categorical cross entropy" and "adam" optimizer were used as loss function and optimizer respectively.

C. *Bidirectional Encoder Representations from Transformers*

The model architecture is a multi-layer bidirectional Transformer encoder in [9] based on the original implementation described in [4]. BERT is just a collection of stacked Transformer encoders.

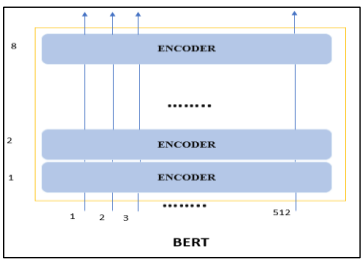

Figure 1: The Transformer model – a foundation concept for BERT.

The pre-trained BERT model can be fine-tuned with just additional one output layer to create state-of-the-art model. By using BERT as a starting point in transfer learning, we leverage the knowledge it has gained during pre-training and fine-tune the model on our labeled dataset.

V. EVALUATION OF THE CLASSIFIERS

A. *Performance Metrics*

We use the following metrics for evaluation of our classifiers: (1) *Precision*, measures the accuracy of the classifier in correctly categorizing instances of a specific class. It is calculated as the ratio of the number of correct categorizations of a class to the total number of categorizations made of that class. (2) *Recall* measures the completeness of the classifier's categorizations for a specific class. It is calculated as the ratio of the number of correct categorizations of a class to the total number of data points in that class in the golden test set. (3) *F1-Score* is a combined metric that considers both Precision and Recall. It is calculated as the harmonic mean of Precision and Recall, providing a single measure that balances both aspects.

To assess the overall performance of the model, the weighted average of the metrics for all classes (information types) is calculated for each fold. The weighting is determined by the Support, which represents the frequency of each class in the test set. By using these evaluation metrics, the performance of the classifier can be assessed in terms of precision, recall, and overall quality across different classes. By using these evaluation metrics, the performance of the classifier can be assessed in terms of precision, recall, and overall quality across different classes.

B. *Results and Discussion*

	precision	recall	f1-score	support
Action on Issue	1.00	1.00	1.00	146
Bug Reproduction	0.75	0.66	0.70	145
Contribution and Commitment	0.92	0.80	0.86	173
Expected Behaviour	0.88	0.99	0.93	155
Investigation and Exploration	0.63	0.47	0.54	152
Motivation	0.75	0.93	0.83	142
Observed Bug Behaviour	0.88	0.89	0.88	149
Potential New Issues and Requests	0.86	0.90	0.88	141
Social Conversation	0.76	0.68	0.72	152
Social Convesation	0.89	0.93	0.91	147
Task Progress	0.88	0.94	0.91	160
Usage	0.80	0.92	0.85	131
Workarounds	0.95	0.92	0.93	155
accuracy			0.85	1948
macro avg	0.84	0.85	0.84	1948
weighted avg	0.84	0.85	0.84	1948

Table II. Detailed results for each information type from Logistic Regression.

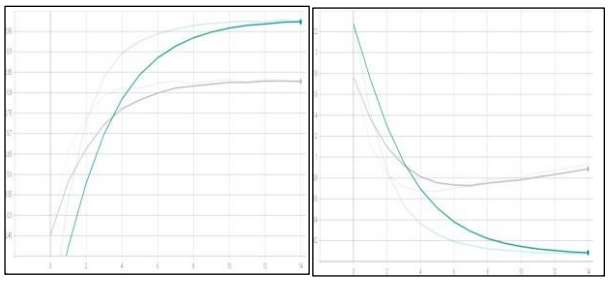

Figure 2: Plot of Epoch Accuracy and Loss on training (green curve) and test (gray curve) set. The vertical axis (ranging from 0 to 1 for accuracy and 0.7 to 1.5 for loss) represents scale of accuracy and loss. The horizontal axis represents epochs.

	precision	recall	f1-score	support
0	0.99	1.00	0.99	156
1	0.79	0.69	0.74	153
2	0.88	0.78	0.83	147
3	0.90	0.99	0.94	151
4	0.53	0.45	0.48	140
5	0.83	0.86	0.84	161
6	0.84	0.79	0.82	150
7	0.86	0.92	0.89	159
8	0.75	0.69	0.72	151
9	0.90	0.91	0.90	132
10	0.90	0.97	0.93	144
11	0.79	0.87	0.83	151
12	0.86	0.93	0.89	153
accuracy			0.84	1948
macro avg	0.83	0.83	0.83	1948
weighted avg	0.83	0.84	0.83	1948

Table III. Detailed results for each information type from CNN.

Metric	**LR**	**CNN**	**BERT**
Accuracy	0.8475	0.8367	0.8634

Table IV. Performance comparison between the three Machine Learning Models.

The Precision, Recall and F1-score of Logistic Regression and CNN on each information type are summarized in Table II and Table III. We observed that the classification achieved good results on most of the information types, especially on *Action on Issue* with a precision of 1.00 and a recall of 1.00 which means that the classifier has achieved perfect performance for this class. Overall, the BERT model outperforms the other two models given our current dataset.

VI. CONCLUSION

In the present work we have demonstrated a series of experiments with learning-based classifier models. Our work represents a first step towards tools and techniques for identifying and obtaining the rich information recorded in ITSs.

VII. ACKNOWLEDGEMENTS

We would like to thank Professor Ji-Hyeong Han for her helpful feedback and suggestions during the course of this project.

REFERENCES

[1] W. W. J. L. G. J. C. D. Arya, "Analysis and Detection of Information Types of Open Source Software Issue Discussions," in *International Conference on Software Engineering*, Montreal, 2019.

[2] X. H. C. M. Wen-tau Yih, "Semantic Parsing for Single-Relation Question Answering," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Baltimore, 2014.

[3] e. a. Y. Shen, "Learning semantic representations using convolutional neural networks for web search," in *In Proceedings*, 2014.

[4] A. S. N. P. N. U. J. J. L. G. A. N. .. &. P. Vaswani, "Attention is all you need. Advances in neural information processing systems," 2017.

[5] F. B. A. S¸ evval Mehder, "Classification of Issue Discussions in Open Source Projects Using Deep Language Models," in *IEEE 30th International Requirements Engineering Conference Workshops (REW)*, Melbourne, 2022.

[6] A. J. K. a. P. K. Chilana, "Design, discussion, and dissent in open bug," in *Proceedings of the 2011 iConference on - iConference '11*, New York, 2011.

[7] C. J.-J. M. F. X. X. a. G. C. M. G. Viviani, "What design topics do developers discuss?," in *Proceedings of the 26th Conference on Program Comprehension, ICPC '18,*, Gothenburg, 2018.

[8] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, 2014.

[9] M.-W. C. K. L. K. T. Jacob Devlin, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, 2019.