Group 17 - Auto-Labelling of GitHub Issues using Natural Language Processing

Authors: Larry Law (e0325467@u.nus.edu), Liu Zechu (e0323879@u.nus.edu),
Ng Tek In (e0175456@u.nus.edu), Zhuang Xinjie (e0202855@u.nus.edu)

Abstract

On popular GitHub repositories with thousands of open issues, auto-labelling issues can save moderators significant time. In contrast to most regex-based labellers, we propose two NLP-based approaches: 1) Feature-based classifier that leverages *static word embeddings* and corpus-level information and 2) Fine-tuned BERT Sequence Classifier with pretrained *contextual word embeddings*. Given that machine information (such as URLs, logs and code blocks) can contain natural language vocabulary which may not have the same semantic meaning (e.g. docs.**rust**-lang.org), we filtered out such natural-language-like machine information. Both classifiers outperformed the regex baseline in labelling issues (into one of Bug, Doc, or Feature classes) on our hold-out set of repositories for F1 score (0.8356 vs 0.8723 vs 0.3634) and accuracy (0.8362 vs 0.8752 vs 0.5267). In addition, we empirically show that filtering out machine information helped the NLP model generalise better. Our code is available as a Github action in the marketplace[1].

1 Introduction

In a popular public GitHub repository, it is very common to see thousands of open issues. As manually labelling them is tedious, an auto-labeller is a highly demanded functionality by moderators.

Most auto-labellers are regex-based. However, regex-based approaches typically require the user to specify the regex, thus they tend not to generalise to other repositories well yet requiring engineering effort to design them. To mitigate these limitations, Issue-Label-Bot [4] offered a plug-and-play NLP model that learns from the abundance of labelled data in open source repositories. However, Issue-Label-Bot used a simple model architecture without consideration of machine information[2], such as code blocks, Uniform Resource Locators (URLs) and logs.

The limitations of existing works motivated the following research questions: How effective is NLP-based approach relative to traditional regex approach in auto-labelling GitHub issues? How should the NLP-based labeller handle machine information? We propose two NLP-based approaches: 1) Feature-based classifier that leverages *static word embeddings* and corpus-level information and 2) Fine-tuned BERT Sequence Classifier with pretrained *contextual word embeddings*. Given that machine information can contain natural language vocabulary which may not have the same semantic meaning (e.g. docs.**rust**-lang.org), we filtered out such natural-language-like machine information. Both classifiers significantly outperformed the regex baseline in labelling issues (into one of Bug, Doc and Feature classes) on our hold-out set of repositories in terms of F1 score and accuracy. In addition, we empirically show that filtering out machine information helped the NLP model generalise better.

[1] https://github.com/marketplace/actions/auto-github-issue-labeller

[2] Here, we define *machine information* as textual information that does not form coherent natural language sentences but is used to facilitate technical discussion on information technology (IT) related knowledge, such as code blocks, URLs and logs.

2 Related Work

2.1 GitHub Issues Auto-Labeller

Approach	Generalises well?	Minimal Engineering?	Accounts for Machine Info?
Ours	Yes	Yes	Yes
Regex	No	No	No
Issue-Label-Bot	Yes	Yes	No

Fig 2.1. Comparison of different approaches to automatically labelling GitHub issues.

Common approach adopted by most open-source issue labellers is regex-based keyword categorization. The engineering work is on repository moderators to set up repository-specific regular expressions to extract only predefined keywords to assign a label to each issue. This means that every repository needs a unique issue format guideline for such labellers to work well. Thus, this approach tends to generalise very poorly.

We realised existing solutions such as Issue-Label-Bot classify issues by using a simple Gated Recurrent Unit (GRU)-based model using training data from popular and established GitHub repositories. This tends to generalise better and requires no extra repository-specific engineering work. However, we believe that GRU is not sophisticated enough to capture the nuances in issue text. Also, their approach does not utilise the constituents of issue text such as different types of machine information mixed within natural language descriptions.

For our approach, we leverage on modern NLP techniques such as word embeddings and transformers. We expect our model to learn from the abundance of labelled data in open source repositories, thus no extra engineering effort is needed to tune it specifically for a repository. Moreover, unlike Issue-Label-Bot, our models take into account machine information (see Fig. 2.1).

2.2 Approaches in NLP

With the introduction of Recurrent Neural Networks (RNN), we can create a language model that accounts for sequential long distance dependencies[3]. Further refining RNN to handle long distance dependencies better, we obtain the Long Short Term Memory (LSTM) network. To utilise context from both directions, we specialize LSTM into Bidirectional LSTM (Bi-LSTM). Next, to represent the context in a machine friendly way, we incorporate Encoder-Decoder, which allows us to encode natural language into vectors and vice versa. Following that, attention was put forth to allow the model to focus on the important segments in the context.[4] This prompted the invention of transformers which showed that we can handle long distance dependencies even without LSTMs while allowing parallelization of training. Lastly, BERT leverages transformers to pretrain deep bidirectional representations by jointly conditioning on both left and right context in all layers (Fig.

[3] CS4248 lecture 7

[4] CS4248 Lecture 8

2.2). We fine-tuned BERT for our work, and justify the motivation of using BERT in section 3.2[5].

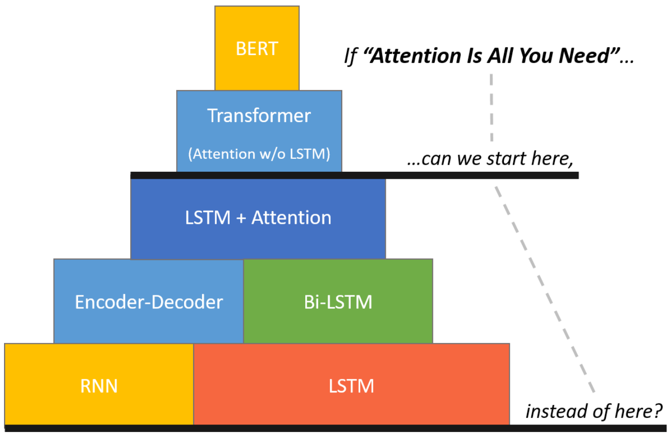


Fig 2.2: "BERT Mountain" - NLP knowledge leading up to BERT [9].

3 Method

3.1 Feature-based Classifier

Feature-based classification techniques are widely used in various text classification tasks in NLP. By transforming each GitHub issue into a feature vector using the same method that applies to all repositories, end-users do not have to define separate rules for different repositories, minimising human involvement in deciding the labelling policy. Moreover, a feature-based approach to classification gives us more flexibility in designing features and experimenting with what combinations of features would result in the best performing model.

We considered three types of features (summarised in Fig. 3.1.1):

(1) We first trained word embeddings for the vocabulary found in the tokenized training corpus, comprising text from issue titles and bodies. The embedding training algorithm used was Gensim's Word2Vec Continuous Bag of Words (CBOW), with an embedding size of 300 and window size of 5. We then generated a feature vector of length 300 for each GitHub issue by averaging the normalised word embeddings of all the tokens in the <u>body text</u> of the issue. Note that we used self-trained instead of pre-trained word embeddings since many words in the technical context may have very different semantics from those in ordinary English corpora, which most pre-trained models have been trained on.

(2) We generated a list of vocabulary which comprises N most frequent words from each class (Bug, Feature, and Doc respectively), excluding M most frequent words in the combined corpus of all the sentences. This is to capture words that are informative in the identification of each category. N and M are parameters to be adjusted. For each GitHub issue's body text, we generated a word count vector consisting of the number of occurrences of each "important" word in this shortlisted vocabulary. We set N = 150 and M = 50.

(3) Similar to (1), we used the same set of static word embeddings trained on the same training corpus, but instead for tokens in the <u>title</u> of the issue to generate a feature vector of length 300.

[5] CS4248 Lecture 12

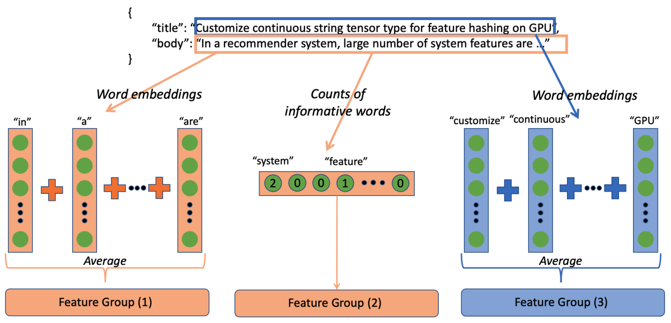

{
"title": "Customize continuous string tensor type for feature hashing on GPU",
"body": "In a recommender system, large number of system features are ..."
}
Word embeddings
"in"
"a"
"are"
Average
Counts of informative words
"system"
"feature"
2 0 0 1 ••• 0
Word embeddings
"customize"
"continuous"
"GPU"
Average
Feature Group (1)
Feature Group (2)
Feature Group (3)

Fig 3.1.1: Feature groups used in feature-based classifier

Afterwards, various combinations of these three groups of features were concatenated and classified using multinomial logistic regression (LR) from ScikitLearn and feedforward neural network (NN) from Tensorflow. The architecture of the NN is shown in Fig. 3.1.2. We applied a dropout layer to mitigate overfitting.

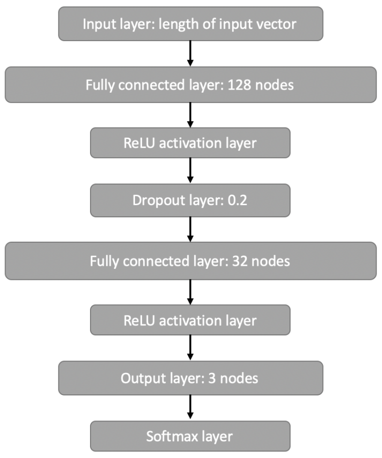

Input layer: length of input vector
Fully connected layer: 128 nodes
ReLU activation layer
Dropout layer: 0.2
Fully connected layer: 32 nodes
ReLU activation layer
Output layer: 3 nodes
Softmax layer

Fig 3.1.2: Neural network architecture for feature-based classifier

We also set a confidence threshold on the classification softmax output, such that the model does not have to make a decision if it is not sufficiently confident. Specifically, given a classification output vector with softmax values, the largest value is compared against the confidence threshold. If the largest value is greater, then the corresponding label will be the classification output. Otherwise, no classification decision will be made. The motivation for introducing a confidence threshold is for our model to also handle unseen issue labels that it was not trained on, since our model was only trained on Bug, Feature and Doc classes. Also, from the end-user's perspective, it may be administratively easier to leave a small number of ambiguous issues to human labelling as compared to giving wrong labels to the ambiguous ones (and moderators have to manually correct them afterwards).

3.2 Fine-Tuned BERT Sequence Classifier

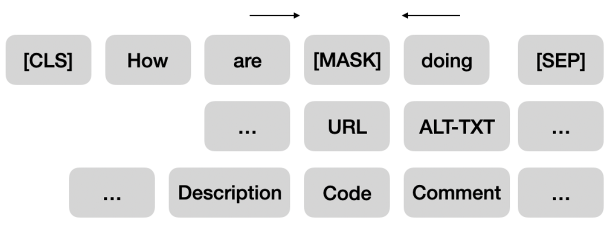


Fig 3.2: BERT's Masked Language Modeling pre-training objective (first row) and our task of GitHub issue labelling (second and third row). In both tasks, bidirectional contextual information can benefit the model.

We also fine-tuned Bidirectional Encoder Representations from Transformers (BERT) [5] for our issue classification task. By pretraining on the "Masked-Language-Modelling" objective, BERT offers deep bidirectional representations that are jointly conditioned on both the left and right context in all layers.

Why use BERT? First, this representation is fully dependent on the context, BERT's *pretrained contextualised embeddings* contrast with the *static word embeddings*[6] used in the feature-based classifier, wherein the embedding is a global representation of the word's meaning independent of context. Moreover, as descriptions of machine information can come both before and after the information (as seen in Fig 3.2), conditioning on context from both directions can be important. Lastly, in classifying issues based on the aggregate sequence representation, BERT may discover useful information within the sequence that our hand-designed features missed out. Empirically, BERT's fine-tuning based representation achieved state-of-the-art (SOTA) results on eleven NLP tasks, thus we are hopeful that fine-tuning BERT for our task of issue labelling will work similarly well.

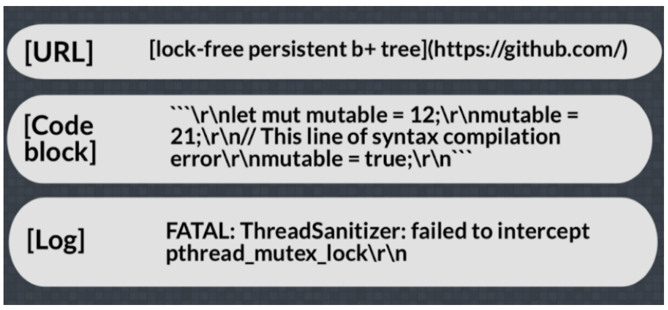
The input representation to BERT is the sequence of concatenated title and body text of the GitHub issue. No markdown preprocessing was done as the WordPiece Tokenizer used by BERT already splits the sequence by punctuation. Similar to our feature-based classifier, we allow the user to set a confidence threshold wherein the model does not make a prediction if its confidence is below this threshold.

3.3 Filtering out natural-language-like machine information

When analysing information present in each GitHub issue's title and body text, we noticed that this information can be grouped into two types: (1) natural language information, which is ordinary sentences that describe the issue, and (2) machine information, which refers to text such as code blocks, URLs, logs etc. Machine information can contain natural language vocabulary which may not have the same semantic meaning (e.g. "for" and "while" loops, "docs.rust-lang.org"). Given that such instances may interfere with the model, we decided to filter out machine information and examine whether our model will benefit from such exclusions.

We further separated machine information into 3 types (see Fig. 3.3 for examples):

(1) **URLs**, which contain natural language words in the URL text itself or its alt-text in markdown.
(2) **Code blocks**, which contain comments and keywords that may be unrelated to the label of the issue.
(3) **Logs**, which sometimes contain useful error description messages, but other times contain repository-specific or program-specific details that may interfere with our model.


[URL]
[Code block]
[Log]

```
[lock-free persistent b+ tree](https://github.com/)
```

~~~
```\r\nlet mut mutable = 12;\r\nmutable = 21;\r\n// This line of syntax compilation error\r\nmutable = true;\r\n```
~~~

```
FATAL: ThreadSanitizer: failed to intercept pthread_mutex_lock\r\n
```

Fig. 3.3 Examples of machine information in GitHub issues

[6] CS4248 Lecture 6

We processed each GitHub issue to include and exclude each type of machine information, and assessed their performance separately using our model.

4. Evaluation

4.1 Datasets

For the training set, we used 80% of the issues in the Tensorflow, Rust, Kubernetes repositories [2], which we will collectively term as "seen repositories". For the test set, we used 1) 20% of the issues in the seen repositories and 2), all issues in Flutter, OhMyZsh, and Electron [2], which we term as "unseen repositories". The first serves as a sanity check while the second tests the model's ability to generalise. To examine how filtering out machine information affects the model, we then test each change on the subset of the test sets that only contains the specific machine information. For example, we will compare the performance metrics of our model before and after removing URLs on only the subset of the test sets which contains URLs.

We focused on three labels commonly used in GitHub repositories: Feature, Bug, Doc (documentation). As each repository has a different naming convention for each label, we set up a consistent mapping policy for all repositories. Issues labelled with more than one category were dropped so as to confine our task to a multiclass single-label classification problem. The number of such cases are few, occupying 2.38% of our dataset.

From exploratory data analysis, we notice that the distribution of labels between the seen and unseen repositories are very similar as seen in Figure 4.1.1. The similar distribution shows that most repositories have similar distribution of these three labels, thus our model will not overfit on the distribution of the training set.

	Feature	Bug	Doc
Seen Repositories	0.3139	0.6164	0.06971
Unseen Repositories	0.3944	0.5267	0.07900

Fig 4.1.1 Statistics of relative proportions of issues in seen itories(Tensorflow, Rust, Kubernetes) and unseen repositories (Flutter, OhMyZsh, and Electron).

In addition, the proportion of machine information in both the seen and unseen repos is also rather significant as shown in Fig 4.1.2, which motivates our work to explore how our NLP model should effectively handle machine information. However, note that Fig 4.1.2 shows the boolean presence of machine information but does not show the proportion of it over the entire text in terms of character length or number of tokens; if the proportion is small, machine information may not be as important as Fig 4.1.2 shows. We leave this analysis for future work.

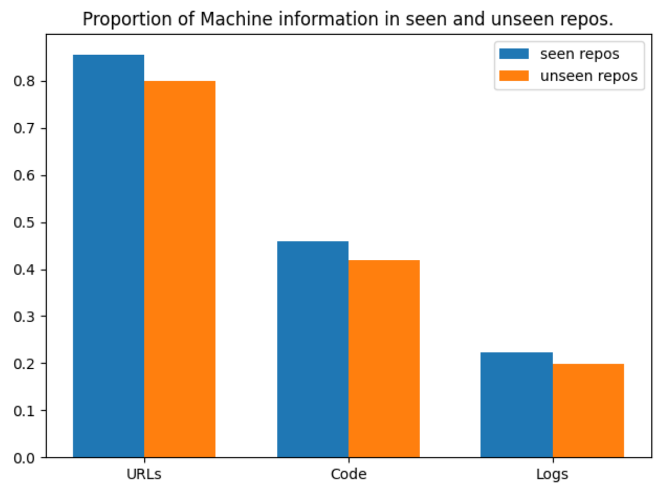


Fig 4.1.2: Proportion of machine information (URL, Code, Logs) in seen and unseen repos.

4.2 Regex-based Classifier Baseline

We chose the most common GitHub labeller on the market, regex classifier, as a baseline for comparison for our models. This allows us to compare the performance of our model against the most popular labeller that is available publicly. For this project, we set up the regex-based classifier by selecting from top 100 non-overlapping keywords in each label category in the training data through exploratory issue data analysis. We then use this classifier as the baseline to compare the performance on both seen and unseen repositories in order to showcase the extent to which our models can generalise better than current solutions.

4.3 Experiment Results

We prioritise *accuracy* as a metric when the model is forced to label all the issues, regardless of confidence level. Accuracy is indicative of the amount of work the moderators of the repository have to undertake in an event of an incorrect labelling. In the case where we allow the model to refrain from labelling when it is not confident, we prioritise the *weighted F1 score* such that it accounts for the work the moderators need to perform in the cases of non-labelling (recall)), as well as the wrong labelling (precision). Weighted F1 score is used to account for the imbalance dataset, by weighing the metric of each class based on its frequency in the dataset.

When the model is forced to label all issues, the feature-based classifier that obtained the best results in unseen repositories uses feature groups (1) and (3) (as defined under section 3.1) with NN. The BERT model that obtained the best results in Fig 4.3.1 was trained on both title and body text. All empirical results can be found in Fig A.1, A.2, and A.3 in the Appendix.

	Seen Repositories		Unseen Repositories	
Approach	F1	Accuracy	F1	Accuracy
Feature-Based	0.8504	0.8529	0.8356	0.8362
BERT	0.9001	**0.9005**	0.8723	**0.8752**
Regex	0.4815	0.6256	0.3634	0.5267

Fig 4.3.1: Best results of each classifier when forced to make a classification on the test sets of seen and unseen repositories.

To test the performance when the model refrains from labelling when unconfident, we experimented with three different confidence thresholds. Interestingly, both the feature-based classifier and BERT classifier did not obtain a better F1 score when allowed to refrain from labelling. We hypothesise the reason being that our test set does not contain any unfamiliar issues (all issues are either Bug, Doc, or Feature), thus not accurately demonstrating the efficacy of not labelling when the model is not confident. We leave the inclusion of unfamiliar issues in the test set to future work. All empirical results can be found in Fig. A.4 in the Appendix.

	Seen Repositories		Unseen Repositories	
Approach	F1	Accuracy	F1	Accuracy
Feature -based	0.8458	0.8714	0.8243	0.8656
BERT	**0.8958**	0.9170	**0.8644**	0.9050
Regex	0.4815	0.6256	0.3634	0.5267

Fig 4.3.2 Best results of each classifier when not forced to label all issues.

Given that BERT outperformed the feature-based classifier, we used BERT to further our experiments on machine information. For simplicity, we will only consider the case wherein we force the model to make a classification. We tested how filtering out machine information affects BERT's performance on the subset of test examples that contain those specific information.

	Seen Repositories		Unseen Repositories	
Approach	Avg %Δ in F1 (10^-3)	Avg %Δ in acc (10^-3)	Avg %Δ in F1 (10^-3)	Avg %Δ in acc (10^-3)
Filter Code	-2.126	-1.341	2.475	2.562
Filter URL	-2.191	-2.300	6.080	5.944
Filter Log	-0.1041	-0.5185	2.304	0.2166

Fig 4.3.3: Results of how filtering out each type of machine information affected BERT's performance on the test examples with the information. %Δ := (performance from filtering X on test sets with X - original performance) / original performance, where X is either code blocks, URLs, or logs.

5 Discussion

5.1 Performance against baseline

Comparing the performance of the regex-based issue labeller under Section 4.2 with the evaluation results of our NLP approach under Section 4.3, we can conclude that the NLP approach is significantly more effective than the regex approach in auto-labelling Github issues. Both the feature-based classifier and the sequence classifier using transfer learning performed better than the baseline regex classifier by a large margin, both on testing sets of seen repositories and unseen repositories. This observation shows that auto-labelling GitHub issues is a ripe problem for NLP machine learning models, given the abundance of labelled data and the fact that NLP machine learning models have much higher expressive power to approximate the underlying language patterns that distinguish between different issue categories when fed enough data.

5.2 Effects of removal of machine information

We also conclude that removing machine information when using the fine-tuned BERT model (as outlined in Section 3.3) led to better generalisation. From Section 4.3, we see that removal of machine information including code blocks, URLs and logs resulted in reductions in both F1 score and accuracy for all seen repositories. In contrast, the removal led to an increase in both metrics for unseen repositories. The experimental results confirmed all our hypotheses that removing machine information will benefit the model. This suggests that our sequence classifier does not handle machine information well when generalising to unseen repositories. This result may be attributed to the fact that machine information is usually more repository-specific, resulting in overfitting of the model to specific repositories that it has been trained on. Also, as shown in Fig. 3.3, machine information generally contains words that have vastly different semantics from the context in which they have been trained in ordinary English (e.g. “for”, “switch”, “this” and “python”). Thus, treating machine information as natural language text confuses the model. In particular, BERT uses WordPiece tokenization which is a variant of Byte-pair Encoding (BPE). Thus, machine information that does not exist in the training vocabulary will not be ignored (i.e. “tensorflow” can be broken down into the subwords “tensor” and “flow”) and hence misinterpreted by the model.

In addition, we also noticed that removing logs led to the smallest percentage improvement. This observation can be explained by the fact that, while some logs contain noisy information (e.g. framework-specific errors) that may cause overfitting to the training repositories, other logs contain useful information that can be generalised to other repositories to determine the label of an issue. Thus, removing logs may not have a strong overall positive effect on the model.

5.3 Model comparison

Furthermore, our fine-tuned BERT sequence classifier outperformed the feature-based one in terms of both weighted F1 score (0.8723 vs 0.8356) and accuracy (0.8752 vs 0.8362), as seen in Fig. 4.3.1. This observation may be because, in the feature-based approach, sequential information in language was not taken into account. All the feature groups used in the feature-based classifier were order-independent. For example, static word embeddings were simply normalised and averaged to form a feature vector for an issue’s body text, which means that even if the relative order of words is changed, the feature vector output will be the same. Moreover, BERT can capture hidden language patterns useful for classification which our handcrafted features might have missed out. Hence, the sequence classifier achieved better results as compared to the feature-based model.

5.4 Ablation Study

Finally, we perform ablation studies on our BERT model fine-tuned on both title and body text, wherein code, URL, and logs machine information have been filtered out.

	Seen Repositories		Unseen Repositories	
Approach	F1	Accuracy	F1	Accuracy
All	**0.8997**	**0.9003**	0.8750	0.8784
All w/o Title	0.8689	0.8700	0.8234	0.8301
All w/o Body	0.8445	0.8455	0.8379	0.8407
All w/o Filtering Code	0.8982	0.8986	0.8738	0.8760
All w/o Filtering URL	0.8986	0.8992	0.8741	0.8772
All w/o Filtering Logs	0.8975	0.8981	**0.8758**	**0.8785**

Fig 5.4: Ablation study on the BERT model trained on both title and body text, wherein code, URL, and logs machine information have been filtered out.

Our ablation study yielded two observations. *First, natural language description mattered more than machine information.* Removing either title and body information worsened F1 and accuracy scores an order more than the change in performance from removing our filter functions. We hypothesise the reason being that natural language description is semantically more informative for the model in determining the issue label. We also observe that title content mattered more than body content, which is possibly because the title typically is a concise summary of the issue and therefore carries denser semantic information.

Second*, filtering log information led to poorer overall performance despite improving performance on the subset of test examples that contains log information.* This contradicting performance underscores our explanation that log information can contain both useful error description and repository-specific/programme-specific details that may interfere with our model.

6 Conclusion

To automatically label GitHub issues, we propose two NLP-based approaches: 1) Feature-based classifier that leverages *static word embeddings* and corpus-level information and 2) Fine-tuned BERT Sequence Classifier with pretrained *contextual word embeddings*. As machine information can contain natural language vocabulary which may not have the same semantic meaning, we filtered out such natural-language-like machine information. Our experiments show that both classifiers significantly outperformed the regex baseline in labelling issues in terms of F1 score and accuracy. In addition, we empirically show that filtering out machine information helped the NLP model generalise better. However, our work has several limitations. Firstly, our work did not explore how we can process machine information as a feature to be included in our classification. Secondly, our model can only classify issues into one of the three predefined labels (Bug, Feature, or Doc), but is unable to cover other types of labels or even issues with multiple labels. For future work, we intend to 1) augment BERT with corpus level (machine) information which has shown to improve BERT’s performance [10], 2) combine our hand-crafted features with BERT’s representation and 3) use multilingual BERT to apply zero-shot transfer learning to repositories of other languages (such as Chinese) so that these repositories can benefit from our work too.

7 Acknowledgments

We would like to specially thank CS4248 Natural Language Processing course tutor Rahul Baid and professor Min-Yen Kan for their valuable advice in the consultation session and answers to our queries throughout the process of this project. We are thankful for this open-ended opportunity to do something *fun*!

8 References

[1] GitHub REST API used for scraping raw data: https://docs.github.com/en/rest/reference/issues

[2] Open-source repositories data used in our project:

- Tensorflow: https://github.com/tensorflow/tensorflow
- Kubernetes: https://github.com/kubernetes/kubernetes
- Rust: https://github.com/rust-lang/rust
- Flutter: https://github.com/flutter/flutter
- Node: https://github.com/nodejs/node
- Electron: https://github.com/electron/electron

[3] S. (n.d.). Issue-Labeller (regex-based labeller). Retrieved from https://github.com/github/issue-labeler

[4] H. (n.d.). Issue Label Bot (NLP-based labeller). Retrieved from https://github.com/machine-learning-apps/Issue-Label-Bot

[5] BERT:
Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[6] R. (n.d.). Auto-Label (regex-based labeller). Retrieved from https://github.com/marketplace/actions/auto-label

[7] M. K. (n.d.). *NUS CS4248 Natural Language Processing*. Lecture. (Kan Min-Yen, 2021)

[8] Tomas Mikolov, Kai Chen, Greg Corrado, & Jeffrey Dean. (2013). Efficient Estimation of Word Representations in Vector Space.

[9] McCormick, C. (n.d.). BERT Research - Ep. 1 - Key Concepts &amp; Sources. https://mccormickml.com/2019/11/11/bert-research-ep-1-key-concepts-and-sources/.

[10] Wah Meng Lim, & Harish Tayyar Madabushi. (2020). UoB at SemEval-2020 Task 12: Boosting BERT with Corpus Level Information.

9 Appendix

Full experimental results for Transfer Learning from BERT

	Seen Repositories		Unseen Repositories	
Approach	Avg F1	Avg Acc	Avg F1	Avg Acc
Title	0.8426	0.8442	0.8372	0.8392
Body	0.8749	0.8757	0.8260	0.8325
Title+Body	**0.9001**	**0.9005**	**0.8723**	**0.8752**

Fig A.1: Results of BERT with different input content on the test sets of seen and unseen repositories.

Full experimental results for Feature-based classifier (with confidence threshold)

Key (for first columns of Fig. A.2 and Fig. A.3):
1: body text embeddings
2: important word count vectors
3: title embeddings
LR: multinomial logistic regression
NN: neural network with dropout

Accuracy (considering only those with a decision)

	Seen Repositories			Unseen Repositories		
Threshold	0.33 (argmax)	0.4	0.5	0.33 (argmax)	0.4	0.5
1 (LR)	0.8016	0.8041	0.8191	0.8072	0.8099	0.8326
2 (LR)	0.7858	0.7907	0.8332	0.7582	0.7651	0.8386
3 (LR)	0.7712	0.7731	0.7897	0.7722	0.7754	0.8006
1 + 2 (LR)	0.8124	0.8154	0.8352	0.7945	0.8000	0.8260
1 + 3 (LR)	0.8337	0.8351	0.8456	**0.8380**	0.8406	0.8600
2 + 3 (LR)	0.8327	0.8344	0.8504	0.8183	0.8214	0.8438
1 + 2 + 3 (LR)	0.8466	0.8477	0.8606	0.8277	0.8307	0.8484
1 + 3 (NN)	**0.8529**	**0.8714**	0.9388	0.8362	**0.8656**	**0.9401**
1 + 2 + 3 (NN)	0.8528	0.8712	**0.9427**	0.8134	0.8418	0.9138

Fig A.2 Accuracy scores of the feature-based classifier on different combinations of features and confidence threshold.

Weighted F-score

	Seen Repositories			Unseen Repositories		
Threshold	0.33 (argmax)	0.4	0.5	0.33 (argmax)	0.4	0.5
1 (LR)	0.7937	0.7941	0.7887	0.7830	0.7822	0.7694
2 (LR)	0.7784	0.7775	0.7634	0.7332	0.7292	0.6985
3 (LR)	0.7623	0.7620	0.7562	0.7616	0.7610	0.7533
1 + 2 (LR)	0.8077	0.8083	0.8044	0.7762	0.7755	0.7623

1 + 3 (LR)	0.8299	0.8296	0.8273	0.8283	**0.8280**	**0.8221**
2 + 3 (LR)	0.8301	0.8299	0.8267	0.8124	0.8116	0.8030
1 + 2 + 3 (LR)	0.8444	0.8435	**0.8418**	0.8219	0.8217	0.8147
1 + 3 (NN)	0.8504	0.8458	0.7763	**0.8356**	0.8243	0.7115
1 + 2 + 3 (NN)	**0.8521**	**0.8471**	0.7787	0.8114	0.8038	0.7084

Fig A.3 Weighted F1 scores of the feature-based classifier on different combinations of features and confidence threshold

	Seen Repositories		Unseen Repositories	
Approach (confidence)	Avg F1	Avg Acc	Avg F1	Avg Acc
BERT (argmax)	**0.9001**	**0.9005**	**0.8723**	**0.8752**
BERT (2)	0.8958	0.9170	0.8644	0.9050
BERT (3)	0.8647	0.9442	0.8162	0.9366
Feature (argmax)	0.8504	0.8529	0.8356	0.8362
Feature (0.4)	0.8458	0.8714	0.8243	0.8656
Feature (0.5)	0.7763	0.9388	0.7115	0.9401

Fig A.4 Best results of each classifier when not forced to label all issues. Appended numbers denote the confidence threshold for model prediction to be accepted. The higher the confidence threshold, the more confident the model will need to be before classifying. For the feature classifier, the confidence threshold is applied on the probability. For transfer learning, the number is applied on the logit prediction.