Clustering the stack traces of ML/DL applications using pre-trained and topic models

Amin Ghadesi
Student
Computer and Software Engineering
Polytechnique Montreal
amin.ghadesi@polymtl.ca

Roozbeh Aghili
Student
Computer and Software Engineering
Polytechnique Montreal
roozbeh.aghili@polymtl.ca

Abstract

Bug reports are among the most valuable sources of information for debugging a system. They have different sections, including a stack trace part. A stack trace is formed of several log lines, indicating a list of method calls when an error has occurred. In this study, we aim to cluster these stack traces. By clustering them, a critical step of bug-triage workload will be reduced. We use Stack Overflow as our source of stack trace information, focusing on the stack traces of machine learning and deep learning applications. We train our models using various pre-trained and topic models. We then evaluate our models to understand which model works better with our input data. Our results show that pre-trained models can achieve more promising results than topic models, having acceptable results on all our evaluation metrics. To the best of our knowledge, this study is among the first attempts to cluster stack traces in the topics of machine learning and deep learning. We have published our dataset, code, and results. [1]

1 Introduction

Bug reports are the most valuable source of information that are heavily used by triaging and development teams during debugging software systems. The problem is that using bug reports to fix the issues is not straightforward (Bhattacharya et al., 2012; Valdivia Garcia and Shihab, 2014). After submitting a bug report, the triaging team should analyze it to check if it describes a significant problem or improvement and then assigns it to suitable developers for addressing (Reis and de Mattos Fortes, 2002). Moreover, the number of bug reports in large and medium software projects is enormous. According to published records (Guo et al., 2010; Uddin et al., 2017), Mozilla and Eclipse received 170 and 120 new bug reports each day in 2009. Manually handling all the bug reports and assigning them to the proper developers demands excessive effort.

It is also important to note that debugging a system is counted as one of the most time- and energy-consuming parts of development. It is anticipated that the cost of performance assurance via debugging, testing, and verification activities ranges from 50 to 75 percent of the total development cost (Hailpern and Santhanam, 2002).

Generally, a bug report contains multiple attributes including bug-id, summary, priority, severity, execution time, and a stack trace part. A stack trace is a list of the method calls that the application was executing when an exception has occurred (Medeiros et al., 2020). Unfortunately, stack traces are often lengthy and messy, so understanding them will be time-consuming, which leads to more development costs. One of the most valuable sections of a stack trace is its related error message. An error message is usually written at the end of a stack trace. Fig 1 shows a stack trace with its error message.

Also, Machine-learning (ML) and Deep-learning (DL) techniques are among the most rapidly evolving approaches in computer science, which have been very popular recently (Jordan and Mitchell, 2015; LeCun et al., 2015). These techniques are designed to imitate human intelligence by acquiring information from the surrounding environment (El Naqa and Murphy, 2015).

In this work, we extract the stack traces with a tag of ML or DL from Stack Overflow (SO) [2], extract the error messages, and then cluster them using pre-trained and topic models. We focus on some of the most popular frameworks and libraries in ML and DL for extracting the stack traces, as suggested by (Islam et al., 2019): Ten-

[1] https://github.com/roozbehaghili/NLU_final_project_winter_22

[2] https://stackoverflow.com/

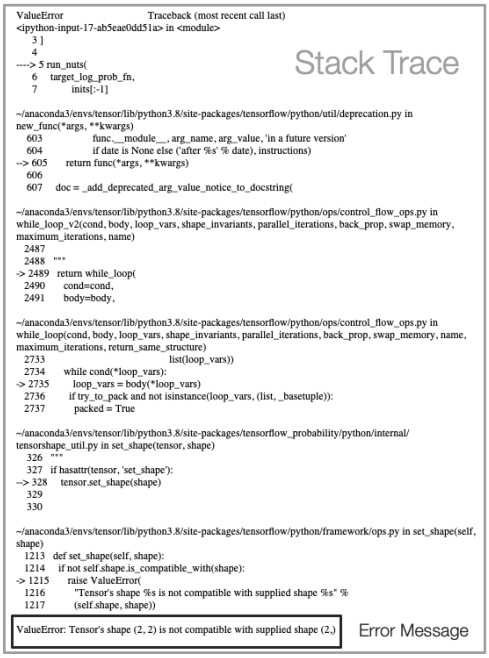

Figure 1: An example of a stack trace (shortened)

sorflow [3], Pytorch [4], Scikit-learn [5], and Keras [6].

As previous work (Islam et al., 2019) studies the types of bugs in DL software using classification, we intend to categorise the types of errors by using clustering. Clustering has a significant advantage regarding classification, which is not requiring the labeled data. In this way, we can use all the available data without the necessity of labeling them. Fig 2 illustrates the structure of our proposed approach.

We aim to use various pre-trained and topic models. We evaluate our models to understand which one best works with our dataset. We anticipate our approach can achieve a better result than the previous study (Islam et al., 2019) as we try to cluster the stack traces and do not limit ourselves to the labeled tags.

2 Related work

Bug characteristics of ML/DL applications

(Thung et al., 2012) study three machine learning systems, Apache Mahout, Lucene, and OpenNLP, and manually classify their bugs into different categories. The center point of their study is finding the severity of bugs, bug frequencies, bug types, and the needed time and effort to fix them.

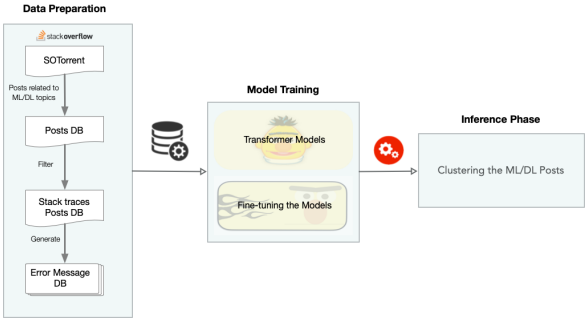

Figure 2: Overview of our method

(Zhang et al., 2018) investigate bugs from deep learning applications, focusing only on the Tensorflow library. They extract 500 SO posts and narrow them to 87 by filtering. They also select 88 bugs from 11 Github projects. They find 7 types of bugs and root causes and 4 types of impacts and symptoms.

(Islam et al., 2019) manually study 2716 SO posts, and 500 Github bug fix commits to recognize three aspects of these bugs: bug types, root causes of bugs, and the effects of these bugs in the usage of deep learning libraries. They consider five popular deep learning libraries: Caffe, Keras, Tensorflow, Theano, and Torch. using 555 bugs from GitHub and 415 Stack Overflow posts. They extend Zheng et al. results to 11 bug types, 10 root causes, and 7 effects.

In contrast to all the attempts mentioned above that investigate the bugs manually, we try to cluster the bugs without any labels, using Natural Language Processing (NLP) approaches. We also expand the dataset and look into both ML and DL applications that utilize Tensorflow, Pytorch, Scikit-learn, and Keras.

Logs and NLP

We use the error messages of stack traces as our dataset, and these error messages are basically log lines. Prior research shows that we can assume log files as a natural language since the logs are written by humans at work (Hindle et al., 2016). Also, similar articles illustrate that software systems are even more predictable and repetitive than human languages, such as English (Tu et al., 2014; Allamanis and Sutton, 2014). Consequently, language models outperform software engineering tasks rather than common English.

(He et al., 2018) characterize logging state-

[3] https://www.tensorflow.org/
[4] https://pytorch.org/
[5] https://scikit-learn.org/
[6] https://keras.io/

ments using NLP. They find that there exists repetitiveness in logging descriptions. They also demonstrate this repetitiveness is capturable using n-gram models.

Some research have focused on the anomaly detection of logs using NLP approaches. (Bertero et al., 2017) considers logs as regular text and utilizes the word embedding technique word2vec. They use 660 logfiles, train their model, and try to find the anomalous ones. Their result shows a robust predictive performance with around 90% accuracy using three classifiers. The results of (Gholamian and Ward, 2021) also confirms the regularity between log statements. Based on this knowledge, they create an anomaly detection tool called ANALOG that is built upon NLP features.

Following these researches, we plan to behave the error messages of stack traces as a natural language. Despite the mentioned works, we endeavor to cluster the error messages using NLP approaches.

Pre-trained models and fine-tuning

With the emerge of NLP, several pre-trained models have been developed recently. BERT (Devlin et al., 2018), XLNet (Yang et al., 2019), and RoBERTa (Liu et al., 2019) are among them, which are all based on Transformer (Vaswani et al., 2017). In addition to these main models, many fine-tuned ones exist in the community of hugging face [7]. BERT and its variations are among the most common models, which we also use. As suggested by (Wu and Dredze, 2020), monolingual BERT works better than mBERT on high-resource languages like English. As our logging statements are in English, we will use the classic BERT.

Pre-training these language models on an enormous amount of unlabeled data then fine-tuning has become a new schema for NLP tasks. Since these models usually have a large set of parameters, training them is time-consuming, even using GPUs and TPUs. Different previous researches use the knowledge distillation approach (Hinton et al., 2015) to reduce the model size while retraining the accuracy. TinyBERT (Jiao et al., 2019) and DistilBERT (Sanh et al., 2019) are among them.

(Xu et al., 2020) employs self-ensemble and self-distillation mechanisms for fine-tuning BERT without using external knowledge. Also, (Sun et al., 2019) investigate different approaches for fine-tuning BERT for the text classification task. They find that the top layer of BERT is the most useful layer for text classification. They also demonstrate that in-domain pre-training can significantly boost the model performance.

We will use the results of the mentioned articles to increase the accuracy of our model by fine-tuning the core BERT.

Topic Models

One of the ultimate aims of data analytics is to determine data features. When working with text data, the goal is to understand the concepts of a document. While this information might be trivial for a human reader, the program needs to go through additional steps. Topic modeling has been created to solve this problem.

Topic modeling is an unsupervised technique that is able to scan a set of documents, find hidden patterns between words and phrases, and cluster them into groups that represent the whole documents. Though topic models have been used for social (Hong and Davison, 2010) and environmental data (Girdhar et al., 2013), they work best with text data.

After early attempts at modeling text corpora (Baeza-Yates et al., 1999; Papadimitriou et al., 2000), Latent Dirichlet Allocation (LDA) (Blei et al., 2003) was introduced in 2003. LDA is probably the most frequently utilized topic modeling method. However, this model can not perform well on documents that do not have sufficient length and is not able to find more advanced data relationships (Vayansky and Kumar, 2020).

As indicated by (Vayansky and Kumar, 2020) and (Blei, 2012), LDA is not the only topic modeling technique. With the emergence of deep-learning techniques, more topic models rely on this approach. As a fair comparison with our pre-trained models, we decide to choose deep-learning-based topic models such as AVITM (Srivastava and Sutton, 2017), ETM (Dieng et al., 2020), and ZeroShotTm (Bianchi et al., 2020b).

3 Models

This section presents an overview of all the models we use. In order to comprehensively compare different models and suggest the best one for working with software engineering error messages, we use 6 models, namely, BERT (Devlin et al., 2018), CodeBERT (Feng et al., 2020), LDA

[7] https://huggingface.co//

(Blei et al., 2003), AVITM (Srivastava and Sutton, 2017), ETM (Dieng et al., 2020), and ZeroShotTm (Bianchi et al., 2020b). The first two are language pre-trained models, and the four laters are topic models.

3.1 Language Pre-trained Models

3.1.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) was introduced in 2018 and has been used heavily by researchers afterward. BERT is a multi-layer bidirectional Transformer encoder in which the base form model has 12 layers, 768 hidden layers, and 110M parameters. It was pre-trained using BooksCorpus with 800M words and English Wikipedia with 2,500M words. The BERT model was pre-trained on two tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). BERT results showed it outperforms all the previous models in different benchmarks.

3.1.2 CodeBERT

CodeBERT is another pre-trained model with the same structure as BERT but is trained with codes extracted from Github in 6 different languages, Go, Java, JavaScript, PHP, Python, and Ruby. The architecture of CodeBERT is identical to RoBERTa-base (Liu et al., 2019), with total parameters of 125M. The CodeBERT model has two objectives for training, namely MLM and Replaced Token Detection (RTD). We anticipate our clustering task would work better with the CodeBERT model since it is specifically trained with code lines. CodeBERT was presented in 2020.

3.2 Topic Models

3.2.1 LDA

Published in 2003, LDA (Latent Dirichlet Allocation) is a generative probabilistic model for topic modeling. Most of the later research in the area of topic modeling has introduced their approaches based on LDA. LDA is a three-level hierarchical Bayesian model that represents each document as a mixture of the topics.

3.2.2 AVITM

Autoencoding Variational Inference for Topic Models or AVITM was introduced in 2017 and is the first model that has combined the AEVB (AutoEncoding Variational Bayes) based inference method for LDA. AVITM is a neural-based approach for topic modeling and a black-box model: it does not require strict mathematical extractions to handle changes in the model. Also, because it only needs one forward pass through the neural network, it is much faster than traditional approaches.

3.2.3 ETM

Embedded Topic Model (ETM) is a generative model of documents that marries LDA with word embeddings. Same as LDA, ETM is a generative probabilistic model in which each document is a mixture of topics and words. Unlike LDA, it can discover interpretable topics even with large vocabularies that have rare and stop words. ETM uses 20Newsgroups and New York Times corpus as its corpora. This model was introduced in 2020.

3.2.4 ZeroShotTm

Published in 2020, ZeroShotTm (Zero-Show Topic Model) is a zero-shot cross-lingual topic model. In other words, this model learns topics in one language and predicts them for unseen documents in languages other than the one in the training data. This model also uses a neural network: it replaces the input bag of words document representations with multilingual contextualized embeddings. It is built upon AVITM and uses English Wikipedia abstracts as its dataset.

4 Dataset and Evaluation Metrics

This section describes our dataset and evaluation metrics for comparing the models.

4.1 Dataset

We will use the stack traces of the question section of posts on stack overflow that have related machine learning or deep learning tags, Tensorflow, Pytorch, Scikit-learn, and Keras. We extract stack overflow questions from SOTorrent (Baltes et al., 2018, 2019), an open dataset based on the official SO data dump. SOTorrent provides posts' texts and code blocks.

We use the last published version (2020-12-31) of SOTorrent available on BigQuery [8], containing all the version history of all questions and

[8] https://console.cloud.google.com/bigquery?project=sotorrent-org&p=sotorrent-org&d=2020_12_31&page=dataset

answers in the official SO data dump from its first post on July 31, 2008 until 2020-12-31.

We only consider posts that have stack trace on their code blocks. So we extract the code blocks of posts that meet two criteria; I. The post should have at least one of the mentioned ML/DL tags, and II. The post should have at least one stack trace on its code blocks.

After extracting the stack traces, we will take out the error messages, usually located at the end of a stack trace. We will use these error messages as our dataset, feed them into our models. Table 1 illustrates four sample error messages in our dataset.

1	tensorflow.python.framework.errors_impl.InternalError: Failed to run py callback pyfunc_4: see error log.
2	ValueError: Cannot evaluate tensor using eval(): No default session is registered.
3	ImportError: DLL load failed:
4	RuntimeError: save_for_backward can only save input or output tensors, but argument 0 doesn't satisfy this condition

Table 1: A sample of four error messages extracted from stack traces in SO

We decide to use the error messages of stack traces since they are the most critical part of a stack trace; developers and debuggers pay more attention to them for debugging purposes. We also decide to extract the stack traces from stack overflow since it is the most popular question-answering platform for software developers, providing a significant amount of code snippets (Baltes et al., 2019). Future work can also consider extracting the error messages from other public sources such as GitHub issues section.

Filtering all the posts with ML/DL tags, we have 124K posts. Some of these posts have multiple tags, resulting in duplications. After deleting the duplicated posts, 71K remains. 58K of them contain code snippets, and 8.9K contain at least one stack trace. 1.5k of these posts have stack trace but lack the error messages, so we delete them. Since some posts can have multiple stack traces and multiple error messages, the final amount of error messages would be more: 8271 lines of error messages. Some of these lines are identical, and we delete them. Finally, we delete all the lines with less than 10 characters since they can not provide useful information to the model. After all the mentioned curations, we have 7823 lines of error messages. A detailed schema of the steps we have done to create our dataset is shown on Fig 3.

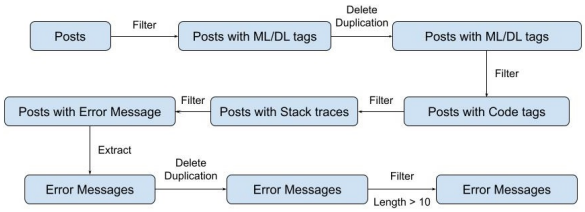

Figure 3: The steps performed to create our dataset

We divide our dataset into training, validation, and testing sets, with 80, 10, and 10 percent. Since we delete the error messages with the length less than 10, the error messages of the dataset have a length between 11 and 3229. The detail of the lengths of the error messages is shown in table 2.

mean	72.17
std	78.06
min	11
25%	38
50%	55
75%	87
max	3229

Table 2: Detail of dataset length

All the error messages have an error tag inside them, such as “ValueError” and “KeyError” (Look at table 1 to see the real examples). These tags can be helpful for validating the results of clustering. Table 3 demonstrates the top 10 most frequent errors.

Error Type	**Count**
ValueError	2584
TypeError	1329
ImportError	1084
AttributeError%	890
InvalidArgumentError	423
RuntimeError	416
KeyError	206
IndexError	191
ModuleNotFoundError	163
OSError	129

Table 3: Top 10 most frequent error message types

4.2 Evaluation Metrics

The performance of topic modeling algorithms and, in general, clustering algorithms can be evaluated using different metrics. Over the years, different metrics such as topic coherence, topic sig-

nificance, diversity, and similarity have been introduced. For the purpose of this study, we focus on three metrics; coherence, diversity, and similarity.

4.2.1 Coherence

In topic modeling, the topic coherence measure aims to compute how the top-k words of a topic are related to each other. In other words, it estimates the semantic interpretability of topics. Different approaches exist to calculate this coherence, as discussed by (Lau et al., 2014) and (Röder et al., 2015). We briefly discuss C_uci, C_npmi, and C_v.

C_uci measure, one of the primary methods for calculating the coherence, is based on a sliding window and the word pairs' pointwise mutual information (PMI). It is calculated as follows,

$$C_{UCI} = \frac{2}{N.(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} PMI(w_i, w_j) \quad (1)$$

where

$$PMI(w_i, w_j) = log \frac{P(w_i, w_j) + \varepsilon}{P(w_i).P(w_j)} \quad (2)$$

A more accepted approach is normalized PMI (NPMI) (Bouma, 2009) introduced by (Lau et al., 2014). It is computed as follows.

$$C_{NPMI}(w_i) = \sum_{j}^{N-1} \frac{log \frac{P(w_i, w_j)}{P(w_i)P(w_j)}}{-logP(w_i, w_j)} \quad (3)$$

As suggested by (Röder et al., 2015), we use C_v as our primary coherence measure since it correlates better with human interpretation. C_v measure is based on a sliding window and combines the NPMI with the indirect cosine similarity.

4.2.2 Diversity

Topic diversity is another essential criterion to compare topic models. It is defined as the percentage of unique words in the top words of all topics. Diversity close to 0 indicates redundant topics, and diversity near to 1 represents more varied topics.

Like coherence, several techniques are used for calculating diversity. The most well-known approaches are topic diversity (Dieng et al., 2020) and inversed ranked-biased overlap (Bianchi et al., 2020a). Since we use ranked-biased overlap as our third metric, we use topic diversity as our main diversity measure.

Topic diversity measure looks at the top 25 words of all topics and computes their uniqueness.

4.2.3 Similarity

As our last measure, we choose the topic similarity. Topic similarity aims to find how closely the topics have been generated. For example, we expect a topic about basketball is more similar to a topic about football rather than global finance (Aletras and Stevenson, 2014).

As our metric, we use RBO (Rank-Biased Overlap) introduced by (Webber et al., 2010), the first well-known metric for computing similarity in topic models. RBO falls in the range of 0 to 1, where 0 means dissociated and 1 indicates identical topics. The below equation describes the RBO of two infinite ranked lists, S and T,

$$RBO(S, T, p) = (1-p) \sum_{d=1}^{\infty} p^{d-1} A_d \quad (4)$$

Where d is the depth of the ranking which is examined, A_d represents the overlap intersection between the lists S and T, and p is the tunable parameter that determines the contribution of the top d ranks in the final score.

5 Experimental Setup

This section represents the details of our experimental setup. We first fix all our models' hyperparameters and then compare the models using different metrics. We base our experiments on two recently-introduced frameworks, OCTIS (Terragni et al., 2021) and BERTopic (Grootendorst, 2022). The earliest is for modeling the topic models, and the latest is for modeling the language pre-trained models.

5.1 Language Pre-trained Models

For running our pre-trained models, we set BERTopic parameters, namely top_n_words, min_topic_size, and number_of_topics.

top_n_words refers to the number of words per topic we want to extract for further analysis, e.g., measuring the evaluation metrics. We set this value to 10 for all our pre-trained and topic models.

The following parameter, min_topic_size, is used to specify the minimum size of each topic. The lower this value, the more topics are created. We choose to have a minimum of 50 pairs in each cluster after trial and error.

We set the number_of_topics to auto, so the model itself can find the best value for each model.

5.2 Topic Models

For hyperparameter optimization, we run all models for 50 iterations. This number is a trade-off between time and performance, where more iterations typically mean higher performance. Furthermore, each time a new setting is trained, the algorithms are initialized with random weights. To get robust results, we run each model 5 times to have reasonable confidence about our outcome.

We evaluate our hyperparameter settings based on the coherence score (Look at section 4.2.1 for details). The optimization is based on the Bayesian Optimization strategy (Snoek et al., 2012; Archetti and Candelieri, 2019) to find the optimal hyperparameter values of each model based on the dataset and the specific evaluation metric.

We divide our topic models into three categories and optimize the parameters based on that.

- *LDA:* For LDA, we optimize three parameters; number of topics, alpha, and beta. The number of topics represents the final clusters we will have, alpha represents document-topic density, and beta represents topic-word density. Higher the alpha value, documents will contain more topics, and higher the beta value, topics will have a larger number of words.

- *ETM:* We optimize the number of topics and dropout for this model.

- *AVITM, and ZeroShotTm:* We optimize four parameters for these neural models; number of topics, number of layers, number of neurons in each layer, and dropout.

Table 4 demonstrates our hyperparameters for each topic model. We discuss the best number of topics in the next section. After finding the hyperparameters, we run each model for 100 epochs. We follow the same settings prescribed for training the models on OCTIS. Then we compute the values of coherence, topic diversity, and topic significance.

	LDA	AVITM	ETM	ZeroShotTm
alpha	0.31	-	-	-
eta	1.61	-	-	-
number of layers	-	1	-	1
number of neurons	-	200	-	200
dropout	-	0.31	0.22	0.78

Table 4: Our selected hyperparameters for different topic models

We have shown the results of our hyperparameter tuning for 50 iterations on the topic models in Fig 4.

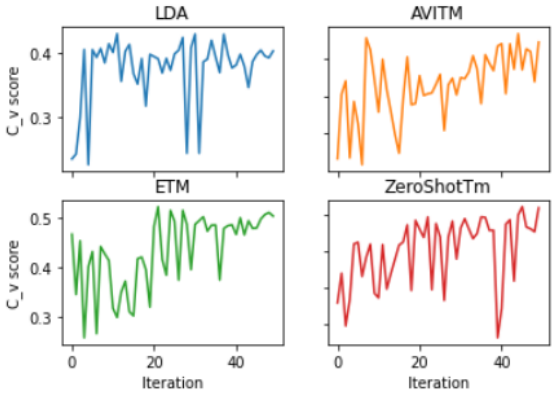

Figure 4: Hyperparameter tuning for 50 iterations

6 Restuls and Discussion

We compare topic and pre-trained models for clustering our curated dataset using three metrics. We run each model for 10 times and present our main evaluation results on Table 5.

The results indicate that even though topic models work relatively well for the coherence metric, both pre-trained models outperform them. Another insight is the improvement of coherence regarding new topic models; compare LDA introduced in 2003 to ZeroShotTm introduced in 2020. Our best-seen value for C_v coherence is linked with the BERT-based model.

We can observe the same pattern among topic models for the diversity metric. Topic models show promising results in this metric, and ZeroShotTm beats other models.

Regarding similarity metric, models perform likewise. We can achieve the similarity of 0.01

for our BERT-based, CodeBERT-based, and ZeroShotTm models, which indicate that the generated topics are well separated and do not have overlaps.

Considering all evaluation metrics, we can conclude that pre-trained models work better for the purpose of clustering the stack traces. However, unlike our initial conjecture, the CodeBert-based model could not outperform the BERT-based one. Although the difference between these two models is not a lot, we estimate this is due to the fact that error messages are not pure code. While error messages have some parts of code, they also have some human-language text, which CodeBERT can not represent well. Based on the results, we can conclude the human-language text sections of error messages are more important than the code parts.

Model	Coherence		Diversity		Similiarity		#Topics
	avg	sd	avg	sd	avg	sd	
Pre-trained models							
Bert	**0.67**	0.01	0.68	0.01	**0.01**	0.00	58
CodeBert	0.66	0.00	0.63	0.01	**0.01**	0.00	64
Topic models							
LDA	0.40	0.01	0.58	0.01	0.05	0.01	34
AVITM	0.49	0.01	0.51	0.04	0.10	0.02	40
ETM	0.48	0.03	0.70	0.08	0.10	0.05	9
ZeroShotTm	0.53	0.03	**0.74**	0.05	**0.01**	0.04	6

Table 5: The main results of our models evaluated by three metrics. We present the average and standard deviation of each model after 10 runs. The highest scores among the models are highlighted.

Another interesting insight about the models is the generated number of topics. The results (the rightest column of Table 5) show that pre-trained models tend to create more topics. On the other hand, ETM and ZeroShotTm create low but large topics.

We also embed the representations of the BERT-based and CodeBERT-based topics in 2D and visualize the dimensions. This visualization is shown in Fig 5

Finally, for a better understanding of the clustered groups of our models, we compare the BERT-based model with ZeroShotTm as our best pre-trained and topic models in Table 6. Even without relying on the evaluation metrics, BERT topic words seem more accurate than ZeroShotTm most frequent words.

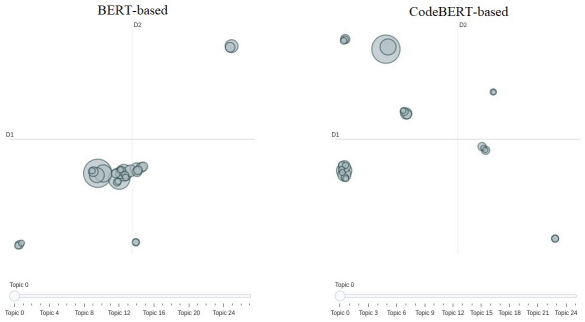

Figure 5: The topic representations of BERT and CodeBERT models

	BERT	ZeroShotTm
topic 1	tensorflow attribute module python name	python tensorflow fail site c
topic 2	dll load specify fail could	tensor dtype float shape expect
topic 3	expect array get check error	object correct incoming jun ne

Table 6: A comparison between the top 5 words of the first 3 topics of BERT and ZeroShotTm models

7 Conclusion

In this work, we cluster machine learning (ML) and deep learning (DL) stack traces to help bug-triage and development teams during debugging time. We first create a dataset of error messages extracted from Stack Overflow. For the purpose of this study, we only focus on ML/DL tags, i.e., tensorflow, pytorch, scikit-learn, and keras.

Then, We compare different approaches for clustering this dataset. We select two pre-trained models, namely BERT and CodeBERT, and four topic models, LDA, AVITM, ETM, and ZeroShotTm.

By comparing the models based on different criteria, we find that pre-trained models perform better than topic models. Our best model, BERT-based, achieves admissible results on coherence, diversity, and similarity metrics.

Our approach and results show that clustering these bug messages is possible, and it can reduce the workload of bug-triage teams by grouping similar bug reports and sending them to proper developers for handling.

Regarding future work, more models can be compared based on various evaluation metrics. For example, topic significance metrics (Al-Sumait et al., 2009; Terragni et al., 2020) can also be added to the evaluations. We also plan to implement our setup on a running system and cluster their real-world error messages on a daily basis.

8 Contributions

Amin has extracted the error messages from Stack Overflow. Amin and Roozbeh have done the preprocessing steps together. Amin has done the experiments with pre-trained models, and Roozbeh has done the topic models' experiments. Amin and Roozbeh have analyzed the results together, and Roozbeh has written the report.

References

Nikolaos Aletras and Mark Stevenson. 2014. Measuring the similarity between automatically generated topics. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 22–27.

Miltiadis Allamanis and Charles Sutton. 2014. Mining idioms from source code. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, pages 472–483.

Loulwah AlSumait, Daniel Barbará, James Gentle, and Carlotta Domeniconi. 2009. Topic significance ranking of lda generative models. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 67–82. Springer.

Francesco Archetti and Antonio Candelieri. 2019. *Bayesian optimization and data science*. Springer.

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*, volume 463. ACM press New York.

Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. 2018. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th international conference on mining software repositories*, pages 319–330.

Sebastian Baltes, Christoph Treude, and Stephan Diehl. 2019. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 191–194. IEEE.

Christophe Bertero, Matthieu Roy, Carla Sauvanaud, and Gilles Trédan. 2017. Experience report: Log mining using natural language processing and application to anomaly detection. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 351–360. IEEE.

Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, and Michalis Faloutsos. 2012. Graph-based analysis and prediction for software evolution. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 419–429. IEEE.

Federico Bianchi, Silvia Terragni, and Dirk Hovy. 2020a. Pre-training is a hot topic: Contextualized document embeddings improve topic coherence. *arXiv preprint arXiv:2004.03974*.

Federico Bianchi, Silvia Terragni, Dirk Hovy, Debora Nozza, and Elisabetta Fersini. 2020b. Cross-lingual contextualized topic models with zero-shot learning. *arXiv preprint arXiv:2004.07737*.

David M Blei. 2012. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of GSCL*, 30:31–40.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Adji B Dieng, Francisco JR Ruiz, and David M Blei. 2020. Topic modeling in embedding spaces. *Transactions of the Association for Computational Linguistics*, 8:439–453.

Issam El Naqa and Martin J Murphy. 2015. What is machine learning? In *machine learning in radiation oncology*, pages 3–11. Springer.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Sina Gholamian and Paul AS Ward. 2021. On the naturalness and localness of software logs. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 155–166. IEEE.

Yogesh Girdhar, Philippe Giguere, and Gregory Dudek. 2013. Autonomous adaptive underwater exploration using online topic modeling. In *Experimental Robotics*, pages 789–802. Springer.

Maarten Grootendorst. 2022. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*.

Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 495–504.

Brent Hailpern and Padmanabhan Santhanam. 2002. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12.

Pinjia He, Zhuangbin Chen, Shilin He, and Michael R Lyu. 2018. Characterizing the natural language descriptions in software logging statements. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 178–189.

Abram Hindle, Earl T Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Communications of the ACM*, 59(5):122–131.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Liangjie Hong and Brian D Davison. 2010. Empirical study of topic modeling in twitter. In *Proceedings of the first workshop on social media analytics*, pages 80–88.

Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A comprehensive study on deep learning bug characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 510–520.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260.

Jey Han Lau, David Newman, and Timothy Baldwin. 2014. Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 530–539.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*, 521(7553):436–444.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Marcos Medeiros, Uirá Kulesza, Rodrigo Bonifacio, Eiji Adachi, and Roberta Coelho. 2020. Improving bug localization by mining crash reports: An industrial study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 766–775. IEEE.

Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. 2000. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235.

Christian Robottom Reis and Renata Pontin de Mattos Fortes. 2002. An overview of the software engineering process and tools in the mozilla project. In *Proceedings of the Open Source Software Development Workshop*, figure 1, pages 1–21.

Michael Röder, Andreas Both, and Alexander Hinneburg. 2015. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Akash Srivastava and Charles Sutton. 2017. Autoencoding variational inference for topic models. *arXiv preprint arXiv:1703.01488*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *China national conference on Chinese computational linguistics*, pages 194–206. Springer.

Silvia Terragni, Elisabetta Fersini, Bruno Giovanni Galuzzi, Pietro Tropeano, and Antonio Candelieri. 2021. Octis: Comparing and optimizing topic models is simple! In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 263–270.

Silvia Terragni, Debora Nozza, Elisabetta Fersini, and Messina Enza. 2020. Which matters most? comparing the impact of concept and document relationships in topic models. In *Proceedings of the First Workshop on Insights from Negative Results in NLP*, pages 32–40.

Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. An empirical study of bugs in machine learning systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, pages 271–280. IEEE.

Zhaopeng Tu, Zhendong Su, and Premkumar Devanbu. 2014. On the localness of software. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 269–280.

Jamal Uddin, Rozaida Ghazali, Mustafa Mat Deris, Rashid Naseem, and Habib Shah. 2017. A survey on bug prioritization. *Artificial Intelligence Review*, 47(2):145–180.

Harold Valdivia Garcia and Emad Shihab. 2014. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th working*

conference on mining software repositories, pages 72–81.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ike Vayansky and Sathish AP Kumar. 2020. A review of topic modeling methods. *Information Systems*, 94:101582.

William Webber, Alistair Moffat, and Justin Zobel. 2010. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):1–38.

Shijie Wu and Mark Dredze. 2020. Are all languages created equal in multilingual bert? *arXiv preprint arXiv:2005.09093*.

Yige Xu, Xipeng Qiu, Ligao Zhou, and Xuanjing Huang. 2020. Improving bert fine-tuning via self-ensemble and self-distillation. *arXiv preprint arXiv:2002.10345*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 129–140.