Dicle University
Journal of Engineering
https://dergipark.org.tr/tr/pub/dumf
duje.dicle.edu.tr

Research Article

Classifying Release Notes Explanations using BERT: An Initial Step to Automatic Versioning in Distributed Software Development

Abdulkadir ŞEKER[1*]

[1] Sivas Cumhuriyet University, Computer Engineering Department, aseker@cumhuriyet.edu.tr, Orcid No: 0000-0002-4552-2676

ARTICLE INFO

Article history:

Received 18 August 2023
Received in revised form 11 October 2023
Accepted 21 October 2023
Available online 31 December 2023

Keywords:

Release note, Software versioning, Explainable AI, BERT, Software development process.

Doi: 10.24012/dumf.1345893

* Corresponding author

ABSTRACT

Distributed Software Development is the practice of developing software with a team in different locations. The process of software versioning is crucial in distributed development as it helps in keeping track of the various software versions that are being developed and maintaining projects. In transition of each new version, the development team present release notes that inform all team members and stakeholders are aware of changes and provide tracking project progresses. Release notes consist information about the features, bug fixes, and other changes included in a new software release. Generating release notes and determining the release transition timing for new software versions can be costly. Despite of there are some papers about generating release notes in the literature, there is not any study about automatic versioning. In this context, the aim of this paper is to predict the development types in release notes as the first phase of an automated versioning tool that is planned to be built in future work. We used BERT which is one of the popular transformers to classify developments of release notes and our model has 86% accuracy rate on our own public dataset. Additionally, we presented insights on the model's decision-making process in the context of explainable AI using ELI5 library.

Introduction

Distributed software development (DSD) refers to the practice of developing software using a distributed team, where team members are located in different geographic locations [1]. There are several benefits to using a distributed team for software development includes accessing to a larger pool of talent, cost savings, improved work-life balance. DSD allows organizations to tap into a global pool of talent, allowing them to find the best developers for their projects regardless of location. It also allows for greater flexibility in scheduling and the ability to work around the clock, as teams in different time zones can take over development tasks [2].

One of the important processes of DSD is software versioning. Software versioning is the process of assigning unique version numbers to unique states of computer software [3]. These version numbers are used to identify and track changes to the software as it is developed and released. Versioning helps teams to coordinate the development and release of software, and it helps users to understand the state and evolution of the software [4].

There are many different conventions and strategies for versioning software, and the specific approach used can vary depending on the needs of the software and the development team. In general, most of projects used semantic versioning [5]. In this approach, software versioning involves a combination of a major version number, a minor version number, and a patch number, with each number representing a different level of change to the software. Major releases are typically more significant and may include significant new features or major changes to the software. Minor releases, on the other hand, are typically smaller and may include bug fixes, security updates, and small new features or improvements. The frequency of minor releases is often higher than the frequency of major releases. The frequency of software releases can vary greatly depending on the software and the development team behind it. Some software is released on a very frequent basis, with new versions or updates being released every few weeks or even every few days. Other software may only be released once or twice a year, or even less frequently.

Each new version of software has release notes which explain the development of related release. Release notes are documents that provide information about the features, bug fixes, and other changes included in a new software release. They are typically written by the development team and are intended to help users understand what has changed in the latest version of the software [6]. Release notes may include a list of new features, a summary of bug fixes and performance improvements, and any known issues or limitations. They may also provide guidance on how to upgrade to the new release, as well as information about compatibility with other software or systems. Release notes are an important resource for users, as they help them stay informed about the latest updates and ensure that they are using the most current version of the software. In the context of distributed software development, software release notes are particularly important for several reasons [7]. They help ensure that all team members and stakeholders are aware of the changes that have been made to the software. This is especially important in a distributed environment, where team members may be working in different locations and time zones. They can serve as a reference point for future development, allowing team members to easily look back and understand the reasoning behind certain decisions that were made. Moreover, they can also be used to help track the progress of the development project and identify any potential issues or delays. Overall, software release notes play a vital role in the successful development and distribution of software, helping to ensure that all team members are informed and aligned on the direction of the project. There are several types of developments that may be included in software release notes, such as new features, improvements, bug fixes, security updates, performance improvements and deprecations, etc.

The cost of generating release notes will depend on a number of factors, including the complexity of the software being released, the size of the team working on the project, and the tools and processes being used [8]. Development teams may feel that the process of generating release notes is an extra time burden [9]. Nevertheless, it requires them to document the changes that have been made to the software. However, release notes serve an important purpose in communicating the changes that have been made to stakeholders, such as customers, users, and other team members. Without clear and concise release notes, it can be difficult for these stakeholders to understand the impact of the changes and how they should be using the updated software. There are a number of ways that teams can make the process of generating release notes more efficient, such as using automated tools to generate draft release notes, establishing clear guidelines and templates for writing release notes, and involving the appropriate team members in the process.

In our empirical study, we firstly analyzed the release notes of a popular open source project and predicting the development types of release notes in order to automatically generate them in the next studies. To conduct this study, we collected some release notes from four open-source project. After that, we use the transformers that is the one of popular deep learning models to predict the development types. By predicting the development types of release notes, we may be able to automate the process of generating release notes for future releases. Thanks to automate this process, development team may be able to save time, resources and improve the efficiency.

Other parts of the study are organized as follows. In the second section, the data set used are explained in detail. In the third section, the literature that guided the study was examined. The research methodology is presented and the findings are given in detail in the next sections. In the last section, the results of the study are presented and future studies are expressed.

Related Works

Classifying and labeling issues in software projects is a significant challenge. While developers may occasionally label issues, the practice is not widely used [10]. Different methods are being explored to address this challenge. In one study, the authors examined the cost of inaccurately labeling issues [11]. In another study, a naive bayes method was used to classify and analyze 4000 issues extracted from JIRA [12].

Despite some studies on issue labeling and classification, there is limited research in the literature on classifying developments within release notes. The one of the studies about this problem, the authors used classic machine learning techniques. The most successful model is based on SVM with the %77 accuracy rate [13].

Some other studies focus on generating release notes. In a study, the authors develop a framework named as ARENA, for automatically generating release notes [9]. Another study found that release notes generated using this method were more accurate than those created manually [14].

Text classification serves as a foundational domain within the realm of natural language processing, garnering significant attention from researchers and practitioners alike [15]. In recent years, deep learning models have emerged as powerful tools for tackling this challenge, with transformers gaining widespread prominence [16]. Notably, Soyalp et al. introduced an expanded transformer model designed specifically for text classification tasks, achieving superior performance when compared to earlier state-of-the-art models like LSTM and CNN [17]. In another study, a BERT-based CNN model was leveraged to classify Chinese news articles, exhibiting exceptional accuracy [18]. These studies underscore the transformative impact of deep learning and transformer-based approaches in advancing text classification techniques.

Materials and Methods

The diagram below illustrates the flowchart utilised in this study (Figure 1).

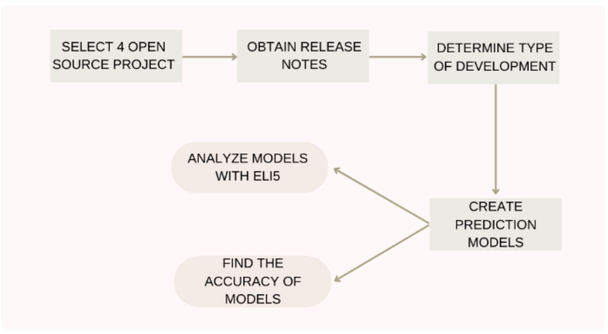

SELECT 4 OPEN SOURCE PROJECT
OBTAIN RELEASE NOTES
DETERMINE TYPE OF DEVELOPMENT
CREATE PREDICTION MODELS
ANALYZE MODELS WITH ELI5
FIND THE ACCURACY OF MODELS

Figure 1. The flowchart of the study.

Collecting Data with Labels

We selected four popular open-source projects in different domain to conduct our study. As we review projects, we determined according to their release consistency schedule and the presence of comprehensive release notes. Thus, we create a dataset from the release notes of Mozilla Firefox [19], Thunderbird [20], Slack [21], and OBSProject [22]. In our dataset, we collected 333 release notes published after each version's migration (Table 1).

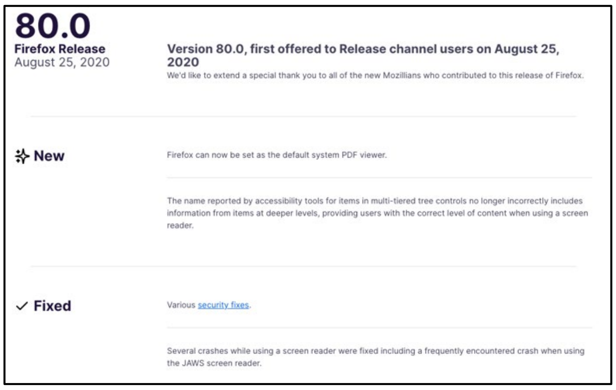
To crawl this data, we used selenium framework. Our dataset is shared on a GitHub repository. With this framework, we navigate the web pages of each release. Thereby we use the unique features (id, class, type) of web components, we reach the context of release notes developments. Then, we record them with labels into an excel file. In the release notes, developments and changing labeled such as new, fixed, changed, unresolved, developer, enterprise, web platform, improvements, addition, etc.

The labels introduce the development types of release note explanations in related release. Some of them are given below.

- **New**: It typically refers to the introduction of new features or functionality that has been added to the software or system being released.
- **Changed**: It typically refers to modifications or updates that have been made to existing features or functionality of the software or system being released. It could also refer to changes in the behavior of the software or system. These changes could be bug fixes, performance improvements, or changes in the user interface or user experience.
- **Fixed**: It typically refers to a correction of a problem or error that existed in a previous version of the software or system. This could include resolving issues such as bugs, glitches, or crashes that have been reported by users or identified by the development team. These fixes are done to improve the stability, performance and correct the problem that was previously identified in the software.
- **Unresolved**: It typically refers to issues, bugs or problems that have been reported but have not yet been fixed or addressed in the current release. These are known issues that developers are aware of and are working to resolve in future releases. The unresolved issues may also be reported in the bug tracking system. This information is usually provided in the release notes to inform the users of the current state of the software and to set expectations for the known issues that are yet to be fixed.

As observed, the definitions of labels are often similar to one another, which makes the classification of them a challenging task.

As seen Figure 1, each release note consists of some development types of titles (new, fixed, development, etc.) and explanations about changing part of the application. While we were creating the dataset, we record each part of notes as an individual development.


80.0
Firefox Release
August 25, 2020

Version 80.0, first offered to Release channel users on August 25, 2020
We'd like to extend a special thank you to all of the new Mozillians who contributed to this release of Firefox.

New

Firefox can now be set as the default system PDF viewer.

The name reported by accessibility tools for items in multi-tiered tree controls no longer incorrectly includes information from items at deeper levels, providing users with the correct level of content when using a screen reader.

Fixed

Various security fixes.

Several crashes while using a screen reader were fixed including a frequently encountered crash when using the JAWS screen reader.

Figure 2. An example of a release notes of The Mozilla Firefox Project.

For example, in Figure 2, there are two changes that are titled with "New". In this way, in our dataset there are 800 total rows with labels (Table 1).

Table 1. The dataset information.

	Release Count	Total Changes
Chrome	74	212
Slack	134	74
ThunderBird	67	196
OBSProject	58	318

In our dataset, some of the labels (classes) are used rarely. Because of that we bring together some labels into "other" label. In Table 2, it is given that the number of labels in our dataset.

Table 2. The label counts of changes in our dataset.

Initial Status		**Status After Merge**	
labels	**count**	**labels**	**count**
new	*115*	*new*	*115*
fixed	*295*	*fixed*	*295*
changed	*226*	*changed*	*226*
unresolved	*63*	*unresolved*	*63*
developer	*30*		
enterprise	*32*		
web platform	*2*	*other*	*101*
improvements	*34*		
changed	*2*		
addition	*1*		

Pre-processing

Pre-processing is an important step in natural language processing (NLP). It involves preparing the raw text data for further processing and analysis. The specific steps in the preprocessing stage can vary depending on the task and the specific needs of the project, but common steps include;

- Tokenization: This involves splitting the text into individual words or smaller pieces called tokens.
- Lowercasing/uppercasing: This involves converting all text to either lowercase or uppercase.

- Removing punctuation: This involves removing any punctuation marks from the text.
- Removing numbers: This involves removing any numerical values from the text.
- Removing stop words: This involves removing common words that do not add significant meaning to the text, such as "the" "and" and "but".
- Stemming: This involves reducing words to their base form to reduce the dimensionality of the data. For example, "jumping" "jumps" and "jumped" would all be reduced to the base form "jump".

We applied pre-processing steps to our dataset, except for stemming. We attempted to use four different stemmer libraries/framework such as TextHero, Lancaster, Porter and Snowball. As seen in Table 3, we found that TextHero library has the most accurate and fastest stemming process among four, and noticed that it did not change the words too much. As a result, there was no significant impact on the prediction model, so we did not apply stemming to dataset.

Creating Classification Model with Transformers

Transformers are a type of neural network architecture that was introduced in the paper of Vaswani et al [23]. Since then, they have become one of the most popular and effective models in natural language processing (NLP) tasks, such as machine translation and language modeling.

One of the key innovations of the Transformer architecture is the use of self-attention mechanisms. In a traditional neural network, the input is processed sequentially through a series of layers, with each layer using the output of the previous layer as input. In contrast, the Transformer uses self-attention mechanisms, which allow the model to consider the entire input sequence simultaneously when processing a particular element in the sequence [23].

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art language processing model developed by Google [24]. It is trained with approximately 4 million words extracted from Wikipedia and BooksCorpus to understand the context of a word based on the words that come before and after it—a technique called contextual modeling. BERT is designed to preprocess text data for natural language processing tasks such as question answering, classification, and language translation, by encoding the context of words in a sentence. DistilBERT, on the other hand, is a smaller and faster version of BERT. It is a distilled version of BERT, which means that it has been trained to have a smaller architecture while still maintaining a similar performance to BERT. DistilBERT can be fine-tuned for the same NLP tasks as BERT but with less computational resources.

To predict the development types of release notes changes, we used two different BERT models, "DistilBERT-base-uncased" and "BERT-base-uncased".

We divided the dataset into 80% for training and 20% for testing. Before training, we shuffled data with the Pandas library. In one of the iterations, the distribution of labels in the training and testing datasets is shown in Table 4.

Table 3. The comparison of the stemming libraries.

	WORD	TextHero	Lancaster	Porter	Snowball
SAMPLE WORDS	information	information	inform	inform	inform
	developer	developer	develop	develop	develop
	various	various	vary	variou	various
	fixed	fix	fix	fix	fix
	images	images	im	imag	imag
	capturing	capture	capt	captur	captur
	activities	activities	act	activ	activ
	stories	stories	story	stori	stori
	automatic	automatic	autom	automat	automat
Running Time		75ms	576ms	856ms	702ms

Table 4. The distribution of the test-train dataset parts.

Label	Label_num	Type	Record
changed	4	train	180
		test	45
fixed	0	train	232
		test	58
new	2	train	90
		test	23
other	1	train	78
		test	20
unresolved	3	train	51
		test	12

We develop our models in the Google Colaboratory platform with NVIDIA-SMI GPU. We trained our models with ktrain library which is a lightweight wrapper library for TensorFlow Keras. The hyper parameters of our models are given Table 5.

Table 5. The hyper parameters of our models.

	Bert	**DistilBert**
model name	BERT_base_uncased	DistilBERT_base_uncased
epoch size	5	5
learning rate	5e-5	5e-5
batch size	6	6
maxlen	500	500

Results

Model Results

Our model provides class-specific scores and overall evaluation metrics. Classification problems are commonly evaluated using accuracy, precision, recall, and F1 score metrics. In a multi-class problem, the calculation of these metrics is specific to each class [25]. The terms of metrics (True Positives, True Negatives, False Positives, False Negatives) are different to binary classification. In figure 3, it is seen that the difference between them.

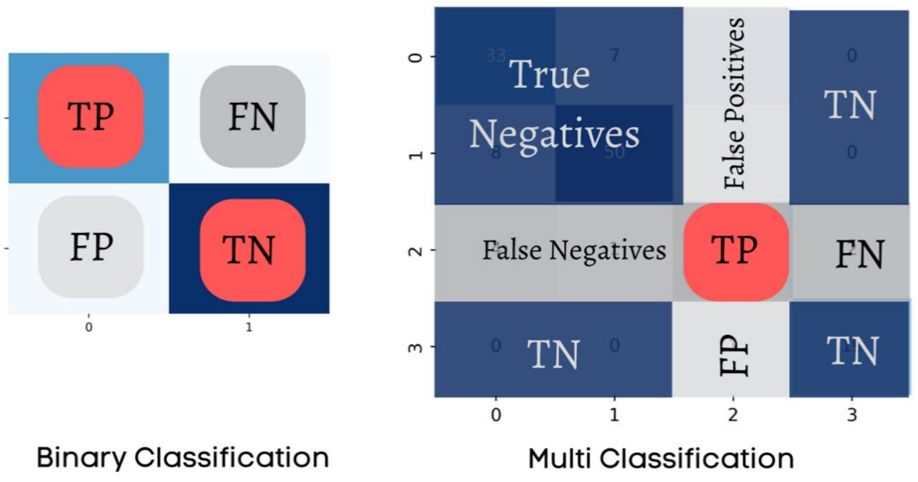


Figure 3. The comparison of between binary-class and multi-class classification.

Precision is referred to the proportion of correct predictions among all predictions for a particular class. Recall is referred to the proportion of examples of a specific class that have been predicted by the model as belonging to that class. F1 Score is the harmonic mean of precision and recall.

$$\text{precision}_{class_i} = \frac{TP_{class_i}}{TP_{class_i} + FP_{class_i}} \quad (1)$$

$$\text{recall}_{class_i} = \frac{TP_{class_i}}{TP_{class_i} + FN_{class_i}} \quad (2)$$

$$\text{f1score}_{class_i} = 2 \text{ x } \frac{\text{precision}_{class_i} * \text{recalll}_{class_i}}{\text{precision}_{class_i} + \text{recalll}_{class_i}} \quad (3)$$

These metrics should be used with caution as it can be misleading, especially when the class distribution is imbalanced. Because of that we also give macro and averaged metrics. The macro approach calculates the average measure for each class without considering class size. In contrast, the weighted method takes into account the number of samples per class and calculates a weighted average measure. For example, weighted and macro precisions are calculated with Equation 4 and Equation 5.

$$precision_{weight} = \frac{\sum(precision_{class_i} * number\ of\ sample_{class_i}}{\sum(number\ of\ sample_{class_i})} \quad (4)$$

$$precision_{macro} = \frac{\sum(precision_{class_i})}{number\ of\ classes} \quad (5)$$

In these equations above, “i” is the class index and the sum ($\sum$) is over all classes.

In addition to these evaluation metrics, the confusion matrices of the models is also provided. The results of above metrics and confusion matrices for both models are presented below.

Table 6 and 7 present the precision, recall, and f1-scores for each class of the dataset. The last column shows the number of samples associated with each label. Additionally, the overall macro and weighted metrics of the models can be found at the bottom of the table (italic). Lastly, the last row of the table shows the overall accuracy score of the model. In the tables, the top two scores in each column are displayed in bold font.

Table 6. The evaluation metrics of the “DistilBERT” model.

	precision	**recall**	**f1-score**	**support**
changed	0.73	0.73	0.73	45
fixed	0.83	0.85	0.84	59
new	0.67	0.70	0.68	23
other	0.90	0.95	0.93	20
unresolved	1.00	0.77	0.87	13
macro avg	0.83	0.80	0.81	160
weighted avg	0.80	0.80	0.80	160
accuracy		**0.80**		**160**

The results of the first model “DistilBERT” are given Table 6. According to results, the model is given overall 0.80 accuracy. The confusion matrix of DistilBERT model is given Figure 4.

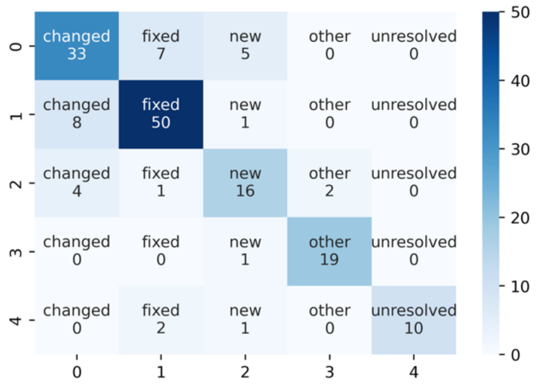


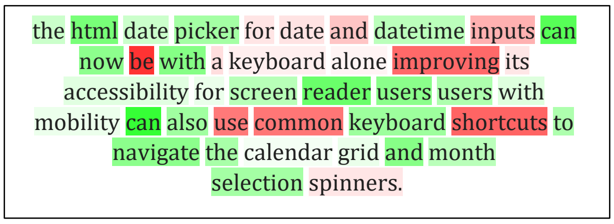
Figure 4. The confusion matrix of “DistilBERT” model.

The model confused between “changed” and “fixed” label. Indeed, these two classes are very similar in definition and content.

Table 1. The evaluation metrics of the “BERT” model.

	precision	**recall**	**f1-score**	**support**
changed	0.79	0.84	0.82	45
fixed	0.88	0.85	0.86	59
new	0.90	0.83	0.86	23
other	1.00	1.00	1.00	20
unresolved	0.79	0.85	0.81	13
macro avg	0.87	0.87	0.87	160
weighted avg	0.87	0.86	0.86	160
accuracy	**0.86**	**160**		

Table 7 presents the BERT model’s results. The BERT model has higher scores because it contains more comprehensive data than the DistilBERT model. In our dataset, we combined rare labels under the "other" category. Because of the model has labeled all data it could not decide on as "other.", the highest scores are seen in the "other" label. The overall accuracy of this model is 0.86.


the html date picker for date and datetime inputs can now be with a keyboard alone improving its accessibility for screen reader users users with mobility can also use common keyboard shortcuts to navigate the calendar grid and month selection spinners.

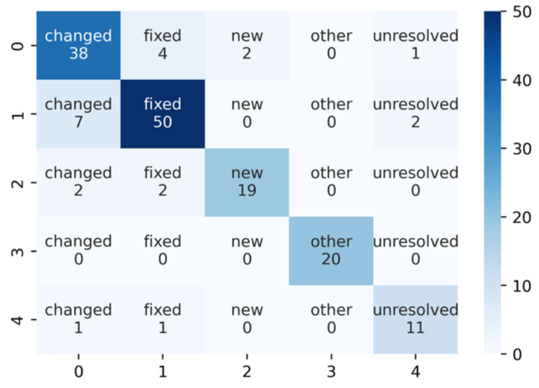


Figure 5. The confusion matrix of “BERT” model.

When looking at each class individually (Figure 5).

- ✓ The “changed” class is most often confused with the “fixed” class.
- ✓ The “fixed” class is most often confused with the “changed” class.
- ✓ The “new” class is most often confused with the “fixed” and “changed” classes.
- ✓ The “unresolved” class is most often confused with the “fixed” and “changed classes.

This situation also shows that the "changed" and “fixed” classes are the most difficult to distinguish.

After seeing these results, we wanted to perform another analysis to understand how the model makes decisions based on the classes.

The explanation of AI

We analyze the BERT model's explainable AI aspect in order to gain insight into the model's decision-making process. With this purpose, we use ELI5 (explain like I'm five) that is a Python library that provides an easy-to-use interface for interpreting the predictions of machine learning models [26]. It allows users to inspect the features of a model that are most important for making a prediction and to understand the reasoning behind a model's predictions. Additionally, it has some visualization tools which can be used to understand the predictions. It helps non-technical people or people who are not experts in machine learning to understand how a model works and why it makes certain predictions.

We gathered release notes from recent versions of Mozilla Firefox and observed which words in the release notes are prominent to classifying process. Besides, after removing these words from the sentence, we also investigate whether the decision of the model changes or not when we ask for a prediction again.

Our model correctly predicted the label for the first below sample, which was labeled as “new”.

SAMPLE 1: “the html date picker for date and datetime inputs can now be with a keyboard alone improving its accessibility for screen reader users with mobility can also use common keyboard shortcuts to navigate the calendar grid and month selection spinners”

ELI5 is a library that helps to explain the predictions made by the model. It produces a colorful output that shows which parts of the input were most important in the model's decision-making process. This can clarify how a model is making its predictions and identifying any potential issues or biases in the model's decision-making. In our sample the output of ELI5 explain() methods is given below.

In addition to the explanations, ELI5 also provides a table that shows the weights of the words in the input text during the decision-making process. The table can be used to identify any patterns or relationships in the input text that the model is using to make its predictions. In Table 8, it is seen that the words have negative impact on decision such as “improving, common, shortcuts and use”.

Table 2. The weights of words used by ELI5 to decision-making process in the Sample-1.

y=changed top features		**y=fixed** top features		**y=new** top features		**y=other** top features		**y=unresolved** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+0.664	improving	+1.226	shortcuts	+0.419	mobility can	+0.471	mobility can	*… 35 more positive …*	
+0.550	now be	+0.650	a keyboard	+0.310	now	+0.451	month	*… 24 more negative …*	
+0.492	be	*… 29 more positive …*		+0.294	navigate	+0.413	shortcuts to	-0.369	keyboard
+0.410	now	*… 23 more negative …*		+0.270	can	*… 34 more positive …*		-0.371	datetime
… 26 more positive …		-0.440	also	*… 38 more positive …*		*… 22 more negative …*		-0.392	improving
… 34 more negative …		-0.484	for	*… 17 more negative …*		-0.375	be	-0.399	spinners
-0.461	month	-0.488	the	-0.344	mobility	-0.400	with	-0.406	the
-0.482	accessibility	-0.566	datetime	-0.416	common	-0.425	calendar	-0.408	can
-0.539	mobility can	-0.588	reader	-0.507	be	-0.480	picker	-0.451	shortcuts
-0.590	navigate	-0.746	to	-0.507	<BIAS>	-0.520	shortcuts	-0.460	for
-0.679	<BIAS>	-0.790	keyboard	-0.642	shortcuts	-0.542	keyboard	-0.511	<BIAS>
-0.705	users	-1.187	now	-0.711	improving	-1.023	now	-0.568	now

Then, we chose a description that our model is unable to predict correctly. Although, the correct label is “fixed”, the model predicted as “new”.

SAMPLE 2: “when using a screen reader on windows pressing enter to activate an element no longer fails or clicks the wrong element and or another application window for those blind or with very limited vision this technology reads out loud what is on the screen and users can adapt them to their needs now on our platform without errors.”

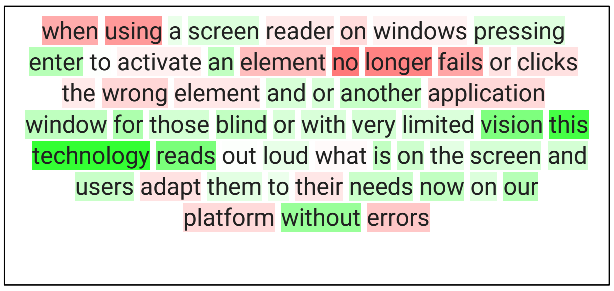

when using a screen reader on windows pressing enter to activate an element no longer fails or clicks the wrong element and or another application window for those blind or with very limited vision this technology reads out loud what is on the screen and users adapt them to their needs now on our platform without errors

Table 3. The weights of words used by ELI5 to decision-making process in the Sample-2.

y=changed top features		**y=fixed** top features		**y=new** top features		**y=other** top features		**y=unresolved** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+1.241	without errors	+2.690	errors	+1.495	this technology	+1.196	this technology	+0.362	reads out
+0.842	this technology	+0.935	fails	+1.452	without errors	+0.961	vision this	+0.351	enter to
+0.534	longer	+0.835	when	+0.960	vision this	+0.863	without errors	+0.339	fails or
+0.474	for those	+0.669	wrong	+0.744	technology reads	+0.673	technology reads	*… 38 more positive …*	
… 34 more positive …		+0.540	using	*… 37 more positive …*		+0.624	activate an	*… 42 more negative …*	
… 41 more negative …		*… 39 more positive …*		*… 36 more negative …*		*… 41 more positive …*		-0.323	what
-0.500	users	*… 40 more negative …*		-0.493	without	*… 41 more negative …*		-0.328	clicks
-0.538	our	-0.670	vision this	-0.567	using	-0.469	platform	-0.370	using
-0.545	those	-0.683	for those	-0.580	<BIAS>	-0.477	limited	-0.405	pressing
-0.567	when	-0.692	technology reads	-0.648	fails	-0.564	activate	-0.447	a
-0.622	fails	-1.686	this technology	-0.668	no longer	-0.655	without	-0.515	<BIAS>
-1.728	errors	-2.010	without errors	-1.782	errors	-0.944	errors	-0.636	errors

As seen from Table 9, the phrase that cause the model to make mistakes can be given as " vision this technology reads ".

The last selected sample is one that our model was unable to predict correctly. The real label is “changed”, the model predicted as “new”.

SAMPLE 3: "firefox has a new focus indicator for links which replaces the old dotted outline with a solid blue outline this change unifies the focus indicators across form fields and links which makes it easier to identify the focused link especially for users with low vision."

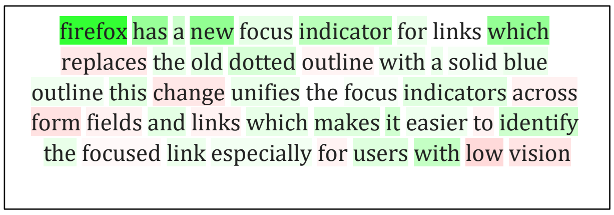

firefox has a new focus indicator for links which replaces the old dotted outline with a solid blue outline this change unifies the focus indicators across form fields and links which makes it easier to identify the focused link especially for users with low vision

Table 4. The weights of words used by ELI5 to decision-making process in the Sample-3.

y=changed top features		**y=fixed** top features		**y=new** top features		**y=other** top features		**y=unresolved** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+1.385	replaces	*… 16 more positive …*		+1.291	firefox has	+0.927	new	+0.441	links which
+1.130	has	*… 19 more negative …*		+0.732	which replaces	+0.700	which replaces	*… 38 more positive …*	
+0.589	low	-0.399	low	+0.722	new	+0.674	firefox has	*… 28 more negative …*	
+0.540	has a	-0.429	blue	+0.662	firefox	*… 35 more positive …*		-0.369	low
+0.480	blue	-0.473	indicator	+0.259	users with	*… 25 more negative …*		-0.375	new
… 22 more positive …		-0.478	replaces	+0.258	old dotted	-0.504	low	-0.381	the
… 39 more negative …		-0.484	easier	*… 32 more positive …*		-0.514	<BIAS>	-0.403	which
-0.463	indicator	-0.492	vision	*… 24 more negative …*		-0.615	old	-0.406	focus
-0.781	firefox	-0.509	outline	-0.294	change	-0.639	has a	-0.421	a
-1.087	which replaces	-0.525	focus	-0.605	has	-0.679	blue	-0.463	links
-1.185	new	-0.777	new	-0.890	replaces	-0.701	replaces	-0.484	outline
-1.779	firefox has	-0.838	firefox	-0.950	<BIAS>	-1.121	has	-0.492	<BIAS>

The words "replace" and "new" have negative impact to the decision of the model.

As can be seen from the examples, an AI model considers some words more when deciding for text classification. Particularly effective words here are "replace, fix, new, this, etc".

Evaluating Model with Unseen Data

Lastly, we used data not seen by the model to evaluate our best model. The results for the 10 enhancements taken from the version notes of the new versions that are not in the dataset are given below (Table 5). In this context, it is also seen that the dataset is %84 successful.

Conclusion

In this paper, we analyzed the release notes of four open-source projects. In these release notes, we aimed to predict the development types such as new, fixed or changed. Contrast to previous papers, we used for classification BERT that is one of the popular transformer models. We developed models with two different pre-trained models of BERT.

In the text classification model development process, there's a pre-processing step applied to text data. However, after trying 4 different stemming libraries, we found that stemming did not have a significant impact on the model. Because of that, we apply other process except stemming.

As results of models are given that some common metrics such as accuracy, recall, precision, and f1-score. We presented the scores per class and the average scores calculated using various methods while presenting results. We achieved better results compared to a study using traditional machine learning methods.

Besides, according to our conclusions from the confusion matrices, the model experiences the most confusion in the "changed" and "fixed" classes. It is also estimated that these two classes are similar to each other when considering the developments made in version transitions.

Lastly, we investigated how the model made its classification decision using explainable AI concept and presented the prominent parameters (words) in this context.

In this research paper, we employ text classification techniques, a fundamental aspect of Natural Language Processing (NLP), to address a pertinent challenge within the realm of software engineering. Our objective is to extend our investigations in future work by analyzing commit notes within open-source projects. Specifically, we aim to ascertain the nature of the development being undertaken within these projects. Furthermore, we intend to provide recommendations for version transitions and their corresponding significance levels based on the degree of change identified in these commit notes. This research bridges the gap between NLP and software engineering, offering valuable insights into the management and evolution of open-source software projects.

Table 5. Unseen data scores of the best model

	Text	**Actual Class**	**Predicted Class**
1	You'll encounter less website breakage in Private Browsing and Strict Enhanced Tracking Protection with SmartBlock, which provides stand-in scripts so tha…	new	new
2	Now, you can set a default app to open a file type. Choose the application you want to use to open files of a specific type in your Firefox settings	**fixed**	**new**
3	The native HTML date picker for date and datetime inputs can now be used with a keyboard alone, improving its accessibility for screen reader users. Users …	**new**	**changed**
4	Firefox builds in the Spanish from Spain (es-ES) and Spanish from Argentina (es-AR) locales now come with a built-in dictionary for the Firefox spellchecker	new	new
5	Fixes the default search engine being reset on upgrade for profiles which were previously copied from a different location.	fixed	fixed
6	You can now pin private windows to your Windows taskbar on Window 10 and Windows 11 for simpler access. Also, private windows have been redesigned…	new	new
7	Removed a configuration option to allow SHA-1 signatures in certificates: SHA-1 signatures in certificates—long since determined to no longerbe secure …	changed	changed
8	Power profiling — visualizing performance data recorded from web browsers — is now also supported on Linux and Mac with Intel CPUs, …	new	new
9	When using a screen reader on Windows, pressing enter to activate an element no longer fails or clicks the wrong element and/or another application window.	fixed	fixed
10	Removed subject common name fallback support from certificate validation. This fallback mode was previously enabled only for manually	changed	changed

References

[1] A. B. Marques, R. Rodrigues, and T. Conte, 'Systematic literature reviews in distributed software development: A tertiary study', in IEEE 7th International Conference on Global Software Engineering, ICGSE 2012, 2012, pp. 134–143. doi: 10.1109/ICGSE.2012.29.

[2] L. Linsbauer, F. Schwägerl, T. Berger, and P. Grünbacher, 'Concepts of variation control systems', Journal of Systems and Software, vol. 171, p. 110796, Jan. 2021, doi: 10.1016/J.JSS.2020.110796.

[3] A. M. Aytekin, 'Release Management with Continuous Delivery: A Case Study', Release Management with Continuous Delivery: A Case Study, vol. 8, no. 9, 2014, Accessed: Jan. 08, 2023. [Online]. Available: https://publications.waset.org/9999440/release-management-with-continuous-delivery-a-case-study

[4] L. Layman, L. Williams, D. Damian, and H. Bures, 'Essential communication practices for Extreme Programming in a global software development team', Inf Softw Technol, vol. 48, no. 9, pp. 781–794, Sep. 2006, doi: 10.1016/J.INFSOF.2006.01.004.

[5] T. Preston-Werner, 'Semantic Versioning 2.0.0 | Semantic Versioning'. Accessed: Jan. 10, 2023. [Online]. Available: https://semver.org/

[6] G. Karsai and D. Balasubramanian, 'Assurance Provenance: The Next Challenge in Software Documentation', in Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering, vol. 13702, T. Margaria and B. Steffen, Eds., Springer, Cham, 2022, pp. 90–104. doi: 10.1007/978-3-031-19756-7_6.

[7] A. C. B. G. da Silva, G. de F. Carneiro, F. Brito e Abreu, and M. P. Monteiro, 'Frequent Releases in Open Source Software: A Systematic Review', Information 2017, Vol. 8, Page 109, vol. 8, no. 3, p. 109, Sep. 2017, doi: 10.3390/INFO8030109.

[8] S. S. Nath and B. Roy, 'Automatically Generating Release Notes with Content Classification Models', International Journal of Software Engineering and Knowledge Engineering, vol. 31, no. 11–12, pp. 1721–1740, Jan. 2022, doi: 10.1142/S0218194021400192.

[9] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, 'ARENA: An Approach for the Automated Generation of Release Notes', IEEE Transactions on Software Engineering, vol. 43, no. 2, pp. 106–127, Feb. 2017, doi: 10.1109/TSE.2016.2591536.

[10] A. Şeker, B. Diri, and H. Arslan, ‘Using Open Source Distributed Code Development Features on GitHub: A Real-World Example’, in 2nd International Eurasian Conference on Science, Engineering and Technology, 2020, pp. 518–525.

[11] K. Herzig, S. Just, and A. Zeller, ‘It’s not a bug, it’s a feature: How misclassification impacts bug prediction’, in Proceedings - International Conference on Software Engineering, 2013, pp. 392–401. doi: 10.1109/ICSE.2013.6606585.

[12] M. Ohira et al., ‘A dataset of high impact bugs: Manually-classified issue reports’, in IEEE International Working Conference on Mining Software Repositories, IEEE Computer Society, Aug. 2015, pp. 518–521. doi: 10.1109/MSR.2015.78.

[13] A. Şeker, S. Yeşilyurt, İ. Can Ardahan, and B. Çınar, ‘Prediction of Development Types from Release Notes for Automatic Versioning of OSS Projects’, in Smart Applications with Advanced Machine Learning and Human-Centred Problem Design, Springer International Publishing, 2023, pp. 399–407. doi: 10.1007/978-3-031-09753-9_28.

[14] M. Ali, A. Aftab, and W. H. Buttt, ‘Automatic Release Notes Generation’, Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, vol. 2020-October, pp. 76–81, Oct. 2020, doi: 10.1109/ICSESS49938.2020.9237671.

[15] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, ‘Deep Learning--based Text Classification: A Comprehensive Review’, ACM Computing Surveys (CSUR), vol. 54, no. 3, Apr. 2021, doi: 10.1145/3439726.

[16] A. Gasparetto, M. Marcuzzo, A. Zangari, and A. Albarelli, ‘A Survey on Text Classification Algorithms: From Text to Predictions’, Information 2022, Vol. 13, Page 83, vol. 13, no. 2, p. 83, Feb. 2022, doi: 10.3390/INFO13020083.

[17] G. Soyalp, A. Alar, K. Ozkanli, and B. Yildiz, ‘Improving Text Classification with Transformer’, Proceedings - 6th International Conference on Computer Science and Engineering, UBMK 2021, pp. 707–712, 2021, doi: 10.1109/UBMK52708.2021.9558906.

[18] X. Chen, P. Cong, and S. Lv, ‘A Long-Text Classification Method of Chinese News Based on BERT and CNN’, IEEE Access, vol. 10, pp. 34046–34057, 2022, doi: 10.1109/ACCESS.2022.3162614.

[19] ‘Mozilla Firefox Release Notes’. Accessed: Jan. 16, 2023. [Online]. Available: https://www.mozilla.org/en-US/firefox/releases/

[20] ‘Thunderbird Release Notes — Thunderbird’. Accessed: Jan. 16, 2023. [Online]. Available: https://www.thunderbird.net/en-US/thunderbird/releases/

[21] ‘Slack for Windows - Release Notes | Slack’. Accessed: Jan. 16, 2023. [Online]. Available: https://slack.com/release-notes/windows

[22] ‘Releases · obsproject/obs-studio’. Accessed: Jan. 16, 2023. [Online]. Available: https://github.com/obsproject/obs-studio/releases

[23] A. Vaswani et al., ‘Attention is All you Need’, in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2017.

[24] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, ‘Bert: Pre-training of deep bidirectional transformers for language understanding’, in Proceedings of NAACL-HLT 2019, Minnesota, 2019, pp. 4171–4186. Accessed: Jan. 17, 2023. [Online]. Available: https://arxiv.org/abs/1810.04805

[25] M. Grandini, E. Bagli, and G. Visani, ‘Metrics for Multi-Class Classification: an Overview’, arXiv preprint arXiv:2008.05756, Aug. 2020.

[26] ‘ELI5 ’. Accessed: Jan. 24, 2023. [Online]. Available: https://eli5.readthedocs.io/en/latest/overview.html