An Industrial Study on Predicting Crash Report Log Types Using Large Language Models

Heba Aburish

A Thesis

in the Department of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical and Computer Engineering)

at

Concordia University

Montreal, Quebec, Canada

April 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis was prepared

By: Heba Aburish

Entitled: An Industrial Study on Predicting Crash Report Log Types Using Large Language Models

submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. ______________________________ Chair

Dr. ______________________________ Examiner

Dr. ______________________________ Examiner

Dr. Abdelwahab Hamou-Lhadj______________________ Supervisor

Approved by: ______________________________

Dr. __________ 202 ____________________

Dr. Mourad Debbabi, Dean, Gina Cody School of Engineering and Computer Science

Abstract

Software crashes and failures take a fair amount of effort and time to resolve. Software developers use information submitted in crash reports (CRs) to conduct root cause analysis of faults. The problem is that CRs often lack all the information required. Automatic prediction of CR fields can therefore reduce the crash resolution process time. In this thesis, we use CR headings and descriptions to predict the type of log files that should be attached to a CR. Our approach is to use multilabel learning algorithms to train a machine learning model using a dataset from Ericsson's CR database to predict the type of log files based on CR headings and descriptions. We use three different pre-trained language models Bert, Telecom Bert, and Word2Vector to extract feature vectors from CR headings and descriptions and then feed these vectors to three different multilabel learning algorithms, namely Binary Relevance (BR), Classifier Chain (CC), and Neural Network (NN). Then, we compare the performance of different feature sets. We found that the use of headings alone with pre-trained language models Bert and Telecom Bert results in the best average AUC (0.70). The use of descriptions and headings and descriptions together as features resulted in an average AUC varying from 0.65 to 0.70. In general, the algorithms showed no significant difference in their performances, but the choice of features impacts the performance. Also, the performance of predicting each type of log is influenced by the use of keywords in headings and descriptions that describe these files. We found that log types with a clear definition such as Key Performance Indicators (KPI) Logs, Post-mortem Dumps (PMD), and execution traces can be predicted with higher accuracy.

Acknowledgments

I would like to express my deepest appreciation and gratitude to my supervisor Dr. Wahab Hamou-Lhadj for giving me this great opportunity to work on this research. I appreciate his guidance and support. I am also grateful that he was always patient with my questions and always provided me with constructive feedback.

I would like to extend my sincere gratitude to the entire team at Ericsson Global AI Accelerator (GAIA) in Montreal for their valuable input and contributions to this project. I had the pleasure of working with them. More particularly, I would like to acknowledge the great support I received from Salman Memon, Zhongwen Zhu, and Pragash Krishnamoorthy. A special thanks to Salam Memon for his exceptional support. I could not have undertaken this journey without his invaluable assistance.

This work was funded and supported by Ericsson - Global Artificial Intelligence Accelerator in Montreal and Mitacs Accelerate Grant IT15986.

I would also thank the Gina Cody School of Engineering and Computer Science at Concordia University for all the support.

No words could describe my profound gratitude to my parents, siblings, and friends, for their unconditional love and support. Thank you for supporting me through my life and always believing in me, I can’t thank you enough.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Problem Motivation

Many software companies use a crash reporting system to document and track system crashes from the time they occur until they are resolved. At Ericsson, the company in which this research study was conducted, the process of handling system crashes involves three lines of support. When a crash occurs, a crash report (CR) is sent to the first line of support, a group of network engineers. Their primary task is to troubleshoot and verify configuration to obtain information about the crash that can aid in its resolution. The crash is then relayed to the second line of support, a team of network operator support engineers who review the CR, perform root cause analysis to understand the causes of the crash, and provide a solution if the crash does not involve changes to the source code. The third level of support consists of software developers who may need to modify the source code and provide patches to fix the crash.

There may be many interactions among the lines of support when more information is needed to diagnose the problem. For instance, software engineers may want additional logs to conduct root cause analysis, prompting support engineers to alter the system to generate the required data. The interactions among the line of support often result in delays and inefficiencies.

This research study aims to reduce such delays by automatically predicting the type of information that should be attached to CRs when the CR is submitted. More precisely, we focus in this paper on predicting log files. This is because logs have been shown to be useful in fault diagnosis and root cause analysis tasks [6]. In addition, Panchal et al. [1] showed in prior work involving Ericsson systems that CRs that contain log files tend to be fixed faster than those that do not have log file attachments.

Six different log types are used at Ericsson: node dumps (NDs), user-defined logs (Logs), post-mortem dumps (PMDs), key performance indicators (KPIs), mail threads, and execution traces (Traces). A node dump contains a set of commands executed on the node and a snapshot of the node state that is affected by the fault. It helps network operators diagnose configuration problems at the node. A KPI file containing performance counters provides information about system performance, used to assess the performance degradation of the network. Execution traces show the flow of execution of a system's artifacts such as function calls, inter-process communication, service invocation, etc. User-defined logs are generated from logging statements inserted by developers into the code to help them later to perform root case analysis. A logging statement typically contains a timestamp, a process id, a verbosity level, a logging function, a log message, and variables. Post-mortem dumps (PMDs) contain whatever data is available in memory when a system crashes, including processes, data exchanged between processes, etc. PMDs are rarely structured and may contain data related to multiple processes or even components. And finally, the Mail threads contain email data that is sent by the system when crashes happen it usually contains data in a table format, plots, and summaries. Table 1 summarizes the log types used in this study.

Table 1: Type of logs used in this study.

Log Type	**Description**
Node Dump (ND)	Node state and commands executed on the node.
Post Modern Dump (PMD)	Whatever data are in the memory when the system fails
Key Performance Indicators (KPI)	Information about performance metrics counters
User-Define Logs (Log)	Produced by logging statements that developers added to the code
Execution Traces	The system's execution flow in form of function calls
Mail thread	Emails sent by the system when the crash occurred

A CR may contain one or more file types depending on the nature of the crash. Predicting the file type that should be attached to a CR can significantly improve the crash resolution process by reducing the interactions among the various lines of support. In this paper, we conducted several experiments with different machine learning multilabel classification algorithms: Binary Relevance [32], Classifier Chain [31], and Neural Networks [33]. We used a different set of features extracted from CR headings and descriptions using various language model embedding techniques, namely Word2Vec [30], Bert [29], and Telecom Bert [28].

1.2 Problem Statement and Thesis Contributions

Most research that has been conducted on predicting CR fields to reduce the overhead of reassignment of CR fields during the crash handling process focuses on predicting CR priority, severity, and product and component fields. Moreover, most studies have used open-source datasets that may not reflect the crash reports used in industrial systems. Since logs and stack traces play a major role in the crash resolution and root cause analysis, in this thesis, we investigate the use of machine learning methods to build a machine learning model able to predict the type of logs attached to CR with the ultimate objective to reduce the overhead of the interaction between support lines to request more logs.

In this study, we will answer the following research questions:

- RQ1: What is the overall performance of predicting the type of logs attached to a CR?
- RQ2: What is the effect of different language models for feature extraction on the accuracy of the algorithms?
- RQ3: What is the effect of using different algorithms on accuracy?
- RQ4: What is the accuracy of predicting each type of log (i.e., label) individually?

Our results show that we can predict the type of logs that should be attached to CRs with an accuracy of 0.711 as the best result Area Under the Curve (AUC) scores using the Binary Relevance model with features extracted from CR headings using the Telecom Bert language model. Also, our result achieves an AUC score of 0.71 with the Binary Relevance model with features extracted from CR headings using Bert as the best result feature extracted with Bert scored and achieves an AUC score of 0.70 with the Neural Network model with features extracted from CR headings and description as the best result using Word2Vec.

1.3 Thesis Outline

The organization of the thesis is as the following:

Chapter 2 - Background and Related Work

In this chapter, we introduce a brief background knowledge about crash report systems and crash report structure in the first section. Then in the second section, we summarize previous studies from the m literature about crash report information quality, issues, predicting crash report fields using machine learning algorithms, and the importance of log and stack trace in the crash handling process.

Chapter 3 - Study Setup

In this chapter, we will go through the study setup including the dataset used in the experiments, the feature extraction, the learning algorithms, the training, and testing steps, as well as the evaluation metrics we used to evaluate the trained models.

Chapter 4 – Results and Discussion

In this chapter, we present the results of different experiments conducted with different algorithms and different feature sets, and then we compare performance. Afterward, we discuss the result and answer the research questions.

Chapter 5 - Conclusion and Future Work

In this chapter, we summarize the results, conclude the study, and then present some future research opportunities.

Chapter 2. Background and Related Work

2.1 Background

Software companies use crash tracking systems to track the crash resolution process. Bugzilla[1] and JIRA[2] are two popular examples of crash (and bug) tracking systems. Crash tracking systems use a database to store CR information [2][3][5]. Crash tracking systems are used by developers to monitor the process of crash resolution. They are also used by developers to exchange information that is useful to fix the crash.

A typical CR contains a crash identifier, a heading, a summary, a description, a severity, a priority, the affected components and products, the status, the operating system, the assignee, the report, and log files [2][5][6].

- The Identifier is a unique reference for the CR in the CR database.
- The severity describes the impact of the crash on system functionality and how serious the fault is.

[1]https://www.bugzilla.org/

[2]https://www.atlassian.com/software/jira

- The priority describes the importance of the CR and can be classified as critical, major, or minor.
- The summary is a short description that provides an overview of the crash.
- The description is a detailed description of the crash, and it may contain embedded stack traces or logs.
- The component and product fields provide information about the product and component where the crash occurred.
- The status is the status of the crash during its life cycle (i.e., registered, assigned, canceled, finished, etc.).
- The reporter and assignee refer to the persons who register the crash in the system and the developer who has been assigned to solve the crash.
- The operating system field contains information about all operating systems where the crash happened.
- The log files consist of user-defined logs, traces, mail threads, and other runtime data collected during the crash.

Typically, after the crash is registered in the crash tracking system it goes through different stages during the resolution process. Figure 1 shows a typical CR life cycle. First, the registered CR is assigned to a developer to tackle the problem. The developer starts by applying a root cause analysis to figure out the causes of the fault. During the troubleshooting stage the CR might be reassigned to another development team. The developers propose a solution by making the required modifications to the system, and then testing the new changes. If the problem is fixed properly the CR status is changed to “fixed” and the CR is closed. If testing the new changes fail, the CR is reopened again.

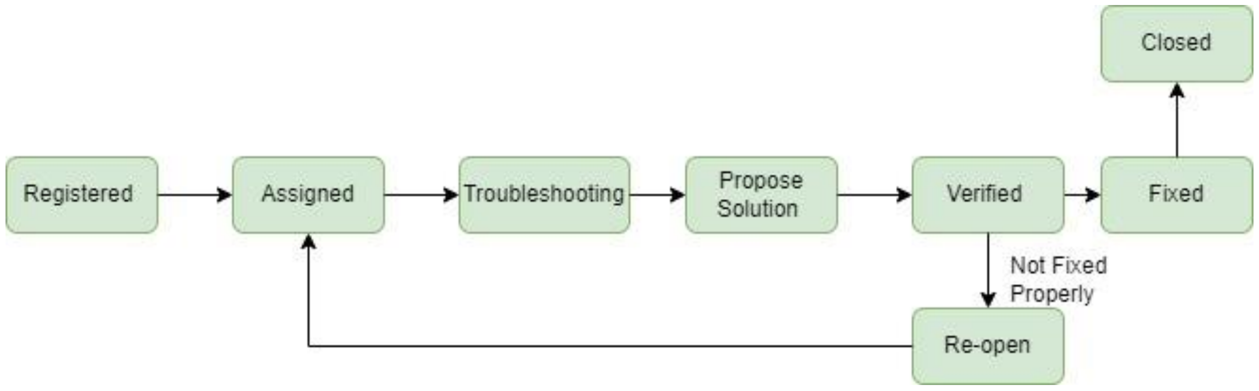

Figure 1: The flow of a crash report

Figure 2 shows the typical CR process flow at Ericson taken from [1]. The Level 1 support line, which consists of network operators, reconfigures, and restarts the system to collect run-time data and other information that can help understand the cause of the crash. It is at this stage that the CR is created with basic information such as heading, description, severity, etc. After registering the CRs, the first line of support notifies the Level 2 support line, which is composed of network operator support engineers. The engineers analyze the problem to uncover the root causes. At this point, Level 2 support could determine that the information provided in the CR is not sufficient to troubleshoot the system, in which case they request back from Level 1 support more information such as generating additional log data. This cycle of interactions is repeated until Level 2 support identifies the root cause and proposes a potential solution. If the problem requires changes to the source code, the CR is relayed to the third line of support who is composed of software developers. Similar to the previous cycle, software developers would engage in a series of interactions with the Level 1 support line to request more information until the problem adequately being solved.

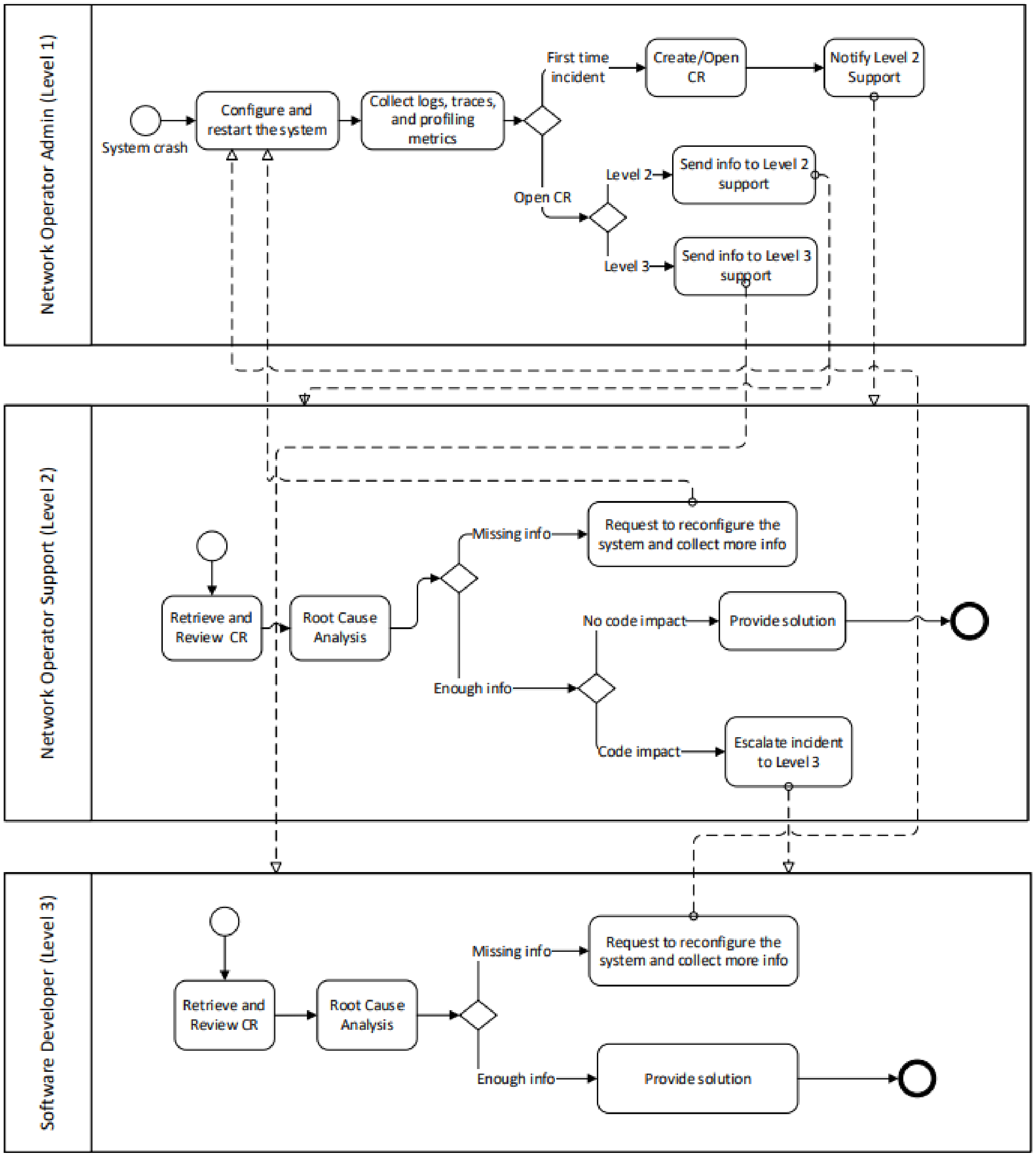
Network Operator Admin (Level 1)
System crash
Configure and restart the system
Collect logs, traces, and profiling metrics
First time incident
Create/Open CR
Notify Level 2 Support
Open CR
Level 2
Send info to Level 2 support
Level 3
Send info to Level 3 support
Network Operator Support (Level 2)
Retrieve and Review CR
Root Cause Analysis
Missing info
Request to reconfigure the system and collect more info
No code impact
Provide solution
Enough info
Code impact
Escalate incident to Level 3
Software Developer (Level 3)
Retrieve and Review CR
Root Cause Analysis
Missing info
Request to reconfigure the system and collect more info
Enough info
Provide solution

Figure 2: Typical process for handling crashes at Ericsson [1]

2.2 Related Work

The analysis of CRs has been the topic of many research studies. There is a large body of knowledge of studies that aim to improve the crash tracking process, the quality of the information in CRs and its impact on the CR resolution time [1][2][3][4][6]. Another collection of studies covers the problems and challenges related to mining CRs [3][4][5][7][8]. Many studies use machine learning to predict various CR fields (e.g., [10][11][12][13]).

Zimmermann et al. [2] conducted a survey to determine the quality of CRs from a developer's perspective. The survey showed that the most helpful information in CRs is the steps to reproduce the crash and the stack traces. Also, the most serious issues that developers run into during the crash handling process include errors in the steps to reproduce the crash, incompleteness of CR information, and incorrect observed behavior that led to incorrect information in CR. The authors emphasized the importance of having CRs with correct information to accelerate the crash resolution process.

Xia et al. [8] analyzed thousands of CRS from OpenOffice, NetBeans, Eclipse, and Mozilla projects and found that approximately 80% of the CRs have their field reassigned. They also showed that CRs that undergo multiple reassignments and refinements take more time to be fixed [4][8]. In a follow-up study, Xia, et al. [9] developed an approach using multilabel learning to predict the CR field (i.e., product, component, severity, priority, OS, version, fixer, status reassignment) that can most likely be reassigned. The authors proposed MLComposer, an ensemble approach that combines three multilabel classifiers that use ML.KNN [25]. They trained the classifiers on different features including meta-features, text features, and a mix of both. The MLComposer considered the imbalance data by introducing a threshold for each label (i.e., Im-ML.KNN). The proposed method achieves an average F1 score of 56% to 62% when it was evaluated with different open-source datasets including OpenOffice, NetBeans, Eclipse, and Mozilla. The study showed that Im-ML.KNN, on average, improves the average F1 score of ML.KNN by 9.11%.

Pushpalatha et al. [14] used the bagging ensemble method to predict the severity of bug reports. They considered the product, component, summary, and the importance of bug reports as features. The method achieved an accuracy of 81.27%, which outperformed the C4.5 classifier[27] performance.

Mondreti et al. [10] studied the accuracy of predicting the severity of crash reports using the XGBoost framework [26]. Three different tree-based models were built using XGBoot that differ in their depth, learning rate, and the number of trees, then the final prediction was determined using majority voting. They conducted experiments on the dataset from the Bugzilla crash report repository using different feature sets including CR summary, priority, and components fields. They also experimented with and without data balancing approaches. The best accuracy achieved was 73.87% using summary, priority, and component as features without applying any data balancing techniques. They showed that the selection of features impacts the performance of the prediction.

Jia et al. [11] used the EKD-BSP approach that uses word embedding of the CR summary and keywords extracted from the CR description using the Text Rank [15] algorithm to predict CR severity and compared the result with different baseline approaches that rely solely on the CR summary. Even though, the EKD-BSP approach showed improvement in the performance of F1-measure by up to 5.19% after being compared with the baselines, the method can only predict if the crash is severe or not severe, without providing any measurement of the severity.

Another study by Zhou et al. [13] investigated the possibility to use source code features such as sentiment features extracted using Convolutional Neuron Networks (CNN) from the Abstract Syntax Trees (AST) version of source code files and relational features learned using CNN from the co-occurrence network between source files to predict severity and priority. The results show that the source code file feature sets do not perform as well as the typical textual description features extracted using CNN from CR descriptions.

A study by Lamkanfi et al. [18] conducted with a dataset obtained from Eclipse showed that for each product 8.3%-30.1% of CR had their component reassigned. They also showed that CRs that had incorrect components during submission took longer time to be fixed. The same study showed that it is feasible to predict whether a component is correct or not using a Naïve Bayes algorithm using the summary and some categorical attributes such as the reporter, OS, version, and severity as features.

The primary objective of Sabor et al.'s work [16] was to investigate if it is possible to predict the component and product fields of CR using neural networks and stack traces. Feature vectors were extracted from stack traces and then weighted based on the appearance of functions in stack traces using frequency-inverse document frequency (IF-IDF). These vectors are then fed to a three-layer neural network model. The approach outperformed the KNN approach by about 85% and 73% for predicting components and products, respectively. However, the fact that only a small percentage of CRs contain stack traces (about 10%) limits the generalizability of the study. Another study by the same authors Sabor et al. [17] proposed to use of a linear combination of stack traces and categorical features (system version, severity, and platform) using a cost sensitive KNN algorithm to predict the CR product and component fields. The study showed that the use of stack traces and categorical features provided better accuracy compared to the use of CR descriptions.

A recent study by Chawla et al. [19] proposed the use of an approach based on fuzzy similarity to identify the correct component field of a bug report as well as to predict the possibility of component reassignment. Three open-source projects from Eclipse’s bug reports are used in the experimentation. In two of the three datasets, the experimental results demonstrate that the fuzzy similarity technique outperforms state-of-the-art approaches in terms of precision.

Alenezi et al. [22] investigated the prediction of CR priority using different classification algorithms including Naïve Bayes, Decision Trees, and Random Forest with two different sets of features. The first one is based on CR summary whereas the second one is based on CR categorical features (component, severity, and OS). Their work showed that the use of categorical features outperforms the use of CR summaries. They also showed that Random Forest and Decision Trees provide better results than Naïve Bayes. An interesting work by Umer et al. [23] proposed CR priority prediction by assigning a sentiment value to each CR and combining these values with feature words extracted from CR descriptions. The authors used SentiWordNet to give each word in the description a sentiment value and then sum up those values to obtain the CR sentiment value. The study showed that there is a strong positive correlation between emotion implied in words of CR description and the priority of CRs by calculating the correlation between them using the the Pearson correlation coefficient. The proposed method outperformed DRONE (PreDicting PRiority via Multi-Faceted FactOrANalysEs) method [21] where it improved the F1 score by about 6%. The DRONE method extracts feature from multiple factors of the CR (e.g., author, related-report, severity, and product) then fed them to a linear regression classifier that introduced thresholds to handle imbalanced data [21]. The latest study by Rathnayake et al. [20] proposed a CNN-based model to predict CR priority. After extracting textual features from the CR descriptions using a bag of words method, the resulting feature vectors are fed to CNN to predict CR priority. The approach achieves 71% accuracy compared to 63% and 48% obtained using Support Vector Machine (SVM) and Temporal Convolutional Networks (TCN), respectively.

Many CRs contain logs and execution traces either embedded into CR descriptions and comments or attached separately to the CRs. Logs and stack traces in CRs provide a view of the system execution when the crash occurs. This information is useful for developers to diagnose and fix the reported crashes [6]. Zimmermann et al. [2] provided empirical evidence that supports the finding that the lifetime of CRs with stack traces is less than those without stack traces. Schroter et al. [24] showed that up to 60% of fixed CRs that contained stack traces involved modifications to one of the stack frames. On the other hand, Chen et al. [6] conducted an empirical study using 10 large-scale open-source systems (ActiveMQ, AspectJ, Hadoop Common, HDFS, MapReduce, YARN, Hive, PDE, Storm, and Zookeeper) and showed a different result. Compared to CR without logs, CR with logs takes more resolution time. They justified this result by the fact that it is common for developers to ask for more logs while analyzing the root causes of the crash/bug. These cycles of interaction increase the bug resolution time.

A recent empirical study by Panchal et al. [1][51] was conducted in Ericsson to analyze the proportion of various types of log files (traces, performance logs, user-defined logs, etc.) in Ericsson's CR database. The study also examines the effect of CR attached logs on crash fixing time and crash severity [1][51]. The study shows that most severe crashes have user-predefined logs as the most important log files to diagnose the crash. In addition, the study shows that the type of attached logs could significantly affect the crash fixing time. Another part of the same study [1] conducted an experiment to predict if a CR would have an attached log or not using the categorical data of CR as features then fed them to different machine learning algorithms. The study results in an F1-score of 84% using the Random Forest algorithm. However, it is only predicted whether certain CRs will need more logs to be resolved. The study did not cover the prediction of the log type of the CRS. In this thesis, we continue this work by predicting the type of logs attached to CRs.

Chapter 3. Study Setup

3.1. Goal and Research Questions

The goal of our study is to build a machine learning model able to predict the type of log file to be attached to a CR with the objective to reduce the interaction between support lines during the crash report handling process in order to reduce the fixing time. We investigate the usefulness of using CRs headings and descriptions as features and obtain feature vectors from them using different language models and then compare features performance with different machine learning algorithms. The study focuses on answering the following research questions:

RQ1: What is the overall performance of predicting the type of logs attached to a CR?

As we mentioned earlier, requesting more files in the later stages of the crash resolving process results in overhead and cost. So, by doing many experiments using different machine learning algorithms that leverage CRs heading and descriptions to predict the type of log and evaluate their performance we can take a glance at how much we could rely on the machine learning model to speed up crash resolution. Also, we can investigate the difference in the performance of using CR headings and descriptions and compare them to know how to improve their structure and the information they contain to be more useful.

RQ2: What is the effect of different language models for feature extraction on the accuracy of the algorithms?

The answer to this question will help us understand the best language model to use to improve prediction accuracy. We answer this question by comparing the performance of the algorithms when using features extracted with different language models.

RQ3: What is the effect of using different algorithms on accuracy?

There are many machine learning algorithms that can be used to predict log types. In this question, we compare the performance of using three algorithms, namely binary relevance, chain of classifiers, and neural networks. We examine the ability of the algorithms to tackle the correlation between labels (i.e., type of logs) as well as the performance of the algorithm using different feature sets.

RQ4: What is the accuracy of predicting each type of log (i.e., label) individually?

The objective of answering this question is to see how the performance of different log types is different from each other's and then investigate the reason behind this performance. This will help us understand which log file types are easier to predict and why. Lessons learned can be used to make recommendations on how to improve the prediction of other log types.

3.2. The Dataset

The dataset used in this study is a set of CRs from the Ericsson crash report database system (for confidential reasons, we cannot reveal either the number of CRs or the system to which they are related). Each CR has a heading, description, various attributes (product, component, severity, etc.), and a set of log files attached to it. The heading consists of a short description of the fault whereas the description contains detailed information about the crash including the impact of the crash on the system, a description of the nature of the crash, and a set of actions to be taken to remedy the problem. Each log file has a name, a short description, and a registration date when the file was attached to the CR. Since each CR could have one or more log file types, we formulate the domain of the experiments as a multilabel classification problem where each CR is assigned to a set of labels (log types) and these labels are not mutually exclusive. As a preprocessing step to obtain the target vector for each CR, a script was written to loop over the CR, then assign "0" for the label to represent the absence of the file type on the CR enclosures set and "1" to represent the presence of the file type. Afterward, different multilabel classifier models were trained and tested with the given dataset with different combinations of features. The trained models that were learned from the previous CRs dataset could be used to predict the type of logs.

3.3. Feature Extraction

We use as features the headings and description of CRs. We run experiments using headings, descriptions, and a combination of both. Since the heading and description are written in natural language, we used natural language processing techniques to extract features. For this, we follow two steps: a pre-processing step, and the generation of embeddings.

For the pre-preprocessing step, we run a pipeline to clean the heading and description texts. The cleaning pipeline starts by splitting each text into word tokens and then cleaning punctuation and digit in the text by replacing punctuation and digits with white spaces. We eliminated the noise tokens such as stop words some symbols and digits or tokens that violate some predefined conditions such as word length. Another important pre-preprocessing step is to replace acronyms that appear in headings and descriptions with their full forms. To achieve this, we had access to a glossary list of acronyms used at Ericsson. Finally, we apply lemmatization for sentence tokens, which is the process where we take individual tokens and reduce them to their base form. For this, we use NLTK tools [34], such as Wordnet, PorterStemmer, and WordNetLemmatizer.

The next step is to generate embeddings from the pre-processed headings and descriptions, which consists of mapping each CR to a feature vector that encodes the heading, description, or the heading and description together. Feature extraction from text is typically done in two steps: word (token) embedding and feature aggregation [35]. The word embedding is to represent each word (i.e., token) in the text as a vector that carries the semantics of the word. Feature aggregation is simply to convert the embedding vectors of tokens into one embedding vector representing the entire text meaning (i.e., CR) by adding a new layer on top of the language model that performs some aggregating function (i.e., sum, max, mean, etc..). In our case, for each CR we averaged the tokens embedding vectors to get one vector feature to represent the CR.

In this study, we extract features for each CR based on three language models, Word2Vec [30], Bert [29], and Telecom Bert [28]. Word2vec is a language model technique composed of shallow two-layer neural networks; one input layer, one hidden layer, and one output layer are used to produce word embeddings based on a list of vocabulary extracted from the words in the training corpus. Word2vec maps each word in the text, which is covered in the vocabulary, into a vector that captures the meaning of the words, i.e., static embedding. Hence representing word embedding based on semantic similarity between words. The size of the embedding vector could be any number [30]. Many studies leverage Word2Vec in different domain knowledge tasks. A. Caliskan et.al [45], without any human labeling or supervision, showed the ability to encode scientific knowledge in the literature as information-dense word embedding that captures complex materials science concepts such as the underlying structure of the periodic table and structure properties of chemical compounds. They trained Word2Vec on roughly 3.3 million abstracts from scientific papers published between 1922 and 2018. They demonstrated how Word2Vec can assist in discovering new chemical compounds with specific properties such as thermoelectric materials by calculating cosine similarities of material embeddings with the embedding of the word 'thermoelectric'. Thus, by studying embedding relations they could look at the relations and analogies among chemical compounds.

Another use case for using Word2Vec in knowledge representation is the work of Gligorijevic et al. [46] used Electronic Health Record (EHR) and word2vec model to discover relationships between diseases and disease-genes associations by applying trivial K-nearest neighbor searches on the diseases and genes common embedding representation space.

Bert is a Bidirectional encoder representation from the transformer that has 12 layers in the Encoder stack with an embedding dimension of length 768. It is a transformer-based general language model used for generating dynamic contextual embedding. Thus, Bert allows different representations for a word based on the given context, since it is pre-trained bidirectionally to generate word vector representations based on the word position in the text [29].

Bert pre-trained on a large corpus extracted from Toronto Book Corpus and Wikipedia. Usually, Bert is used with any general NLP task such as Question Answering, Named Entity Recognition, Auto Summarization, etc. In general, pre-trained models can be used as a features extractor and then used the result embedding vectors as features input to other Machine learning models to perform certain tasks usually classification. Also, pre-trained model parameters can be directly fine-tuned on a specific task. To customize Bert to adopt a specific vocabulary related to specific domain knowledge (e.g., Biology, Telecom, chemistries, etc.), Bert should be pre-trained from scratch on a new corpus related to this domain knowledge. Therefore, it could generate a better encoding representation of the concepts from the domain knowledge point of view [49]. Examples of this are SciBERT [48] and BioBERT [47]. SciBERT is focused on scientific NLP related tasks instead of general language models. It was developed using random 1.4 million papers from semantic scholars. The corpus consists of 18% papers on computer science and 82% the from broad biomedical domain. SciBERT outperforms BERT based on several scientific and medical NLP tasks. BioBERT is a domain-specific language representation model pre-trained on large-scale biomedical corpora.

The Telecom Bert is a DistilRoBERTa-based model which is a distilled version of the RoBERTa-Base model [36]. DistilRoBERTa follows the same training procedure as DistilBERT [37] that was proposed. The Telecom Bert pre-trained on telecom data, and telecom documentation, to create a domain-specific language model to be used in a variety of tasks within the telecom business, such as crash report classification, question-answering for support engineering, and test case generation. The model was used in a previously published study done at Ericson [28]. Telecom Bert was fine-tuned and evaluated for question-answering dataset within the telecom domain, the model showed an f1 score of 77.35 that outperforms the performance of the original DistilRoBERTa model which achieved an f1 score of 69.77 with the same dataset.

In total, we generated nine different feature sets using the three mentioned language models with different combinations of headings and descriptions. For pre-trained models Bert and Telecom Bert the embedding dimensionality is 768 (i.e., features size) which is the size of the last hidden state, and it is fixed in Bert model architectures. On the other hand, for Word2Vec we tried different embedding dimensionalities 10, 50,100, and 300 then we chose 50 since performance doesn’t improve that much after with 100, and 300.

3.4. Algorithms

As discussed earlier, the problem of predicting log file types is a multilabel classification problem because each CR may have more than one log file type. It is therefore important to select classification algorithms that are suited to multilabel classification. For this purpose, we chose to experiment with three different multilabel learning techniques: binary relevance [32], classifier chains [31], and neural networks for multilabel classification [33].

3.3.1 Binary Relevance

Binary relevance (BR) considers each label as one independent binary classification problem. It transforms any multilabel classification problem into one binary classification problem for each label and then uses a binary classification algorithm (e.g., Random Forest, Decision Tree, Logistic Regression, etc.) to train a classifier that predicts the value for each label. In other words, it runs N binary classifiers in parallel, where N is the number of labels, then concatenates the output for each of them as one vector per sample [32]. In our experiments, the binary relevance uses six binary classifiers; each one responsible to predict a type of log (i.e., ND, log, PMD, KPI, Mail thread, and Traces). In our experiments, we used the Random Forest (RF) classification algorithm for each binary classifier [43]. RF is an ensemble machine learning method for classification that combines the output of multiple decision trees to obtain a single output. We chose the RF algorithm since it works great with high dimensional data, handles unbalance in data, and reduces the risk of overfitting [43]. Figure 3 shows an example of how this approach works. The BR approach assumes there is no correlation between labels i.e., predicting NDs file does not affect predicting Trace file and so on. The BR is used in different applications with multilabel tasks like Movie genre classification [39] and Arabic text classification [40].

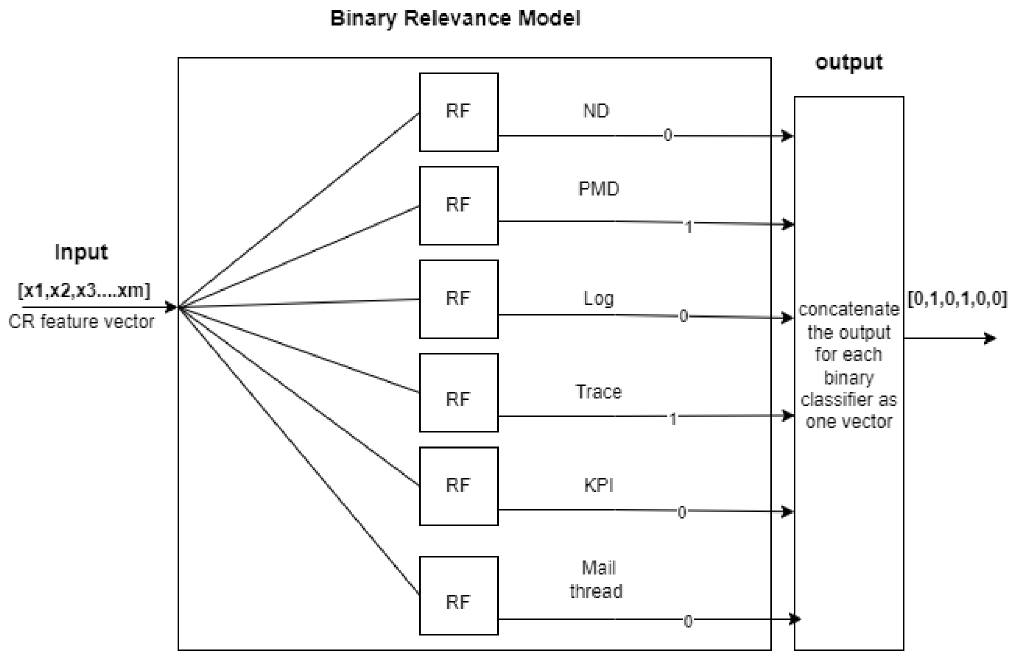

Figure 3: Example of how the Binary Relevance Model works.

3.3.2 Classifier Chains

Classifier chains (CC) are a chain of binary classifiers where each classifier uses the prediction of the previous classifiers. CC runs N single-label classifiers where N is the number of labels, but instead of running them in parallel, they run in sequence (chain of classifiers) where the resulting label of each classifier is passed along with input features to the next classifiers in the chain. Our classifier chain has six binary classifiers to represent the six types of log files. For example, after the classifier responsible for predicting ND logs is executed, the predicted output will be concatenated to the input features and then fed to the next classifier, and so on. For the base binary classifier, we can use any binary classification Algorithm. In our case, we used a Random Forest classifier. CC takes label correlation into account i.e., predicting the type of file would be related to predicting other log types. Also, the order of classifiers in the classification chain would affect the classification performance. Thus, with the help of some specialists from Ericsson, we choose an order to reflect how log files can depend on each other. Figure 4 shows an example of how the classifier chain will work.

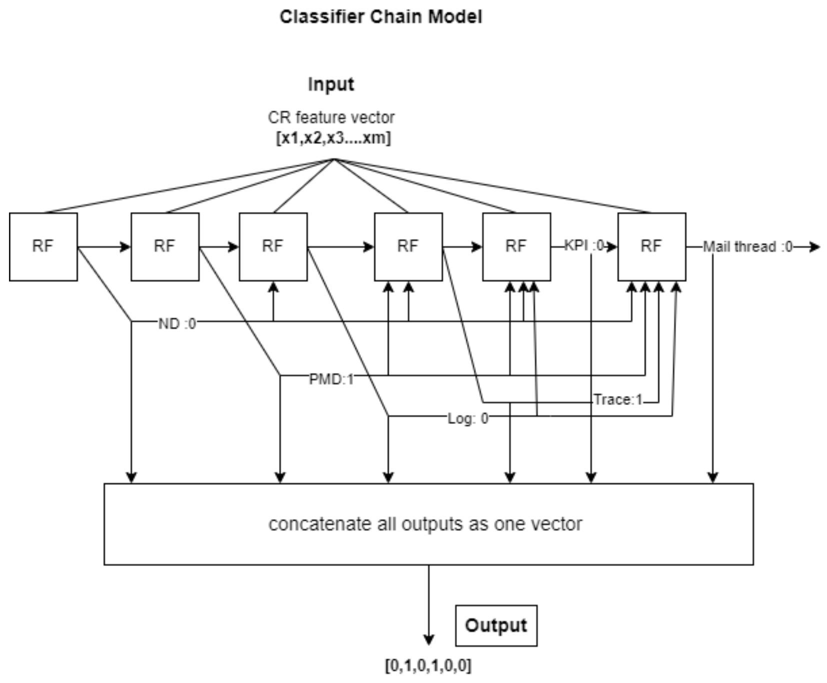

Figure 4: Example of how the Classifier Chain model will work.

3.3.2 Neural Network for Multilabel Classification

Neural network for multilabel classification (NN): NN can support multilabel classification by specifying the number of nodes in the output layer as the number of target labels (i.e., in our case 6 output nodes). We then train it on top of the loss function that can handle multilabel classification. Since it proved that NN with one hidden layer can approximate any function if have enough neurons [50] and we gained only around 0.1% improvements in the performance when trying to add more than one layer in our experiments. Therefore, since the improvement in the performance does not worth increasing the complexity of our NN, our NN model is simply a neural network with an input layer and one hidden layer then we apply binary cross entropy (BCE) loss function on top of the output layer which consists of 6 nodes each give the prediction for one of the log types. The NN also considers label correlations [33]. The NN is used in different applications with multilabel tasks like text categorization [38].

3.4 Training and Testing

We trained each machine-learning model using the technique mentioned in the previous section and applied it to the nine feature sets shown in Figure 5. In total, we trained 27 machine-learning models. For each trained model, we used a 5-fold cross-validation approach where each fold consists of 80% of the dataset used for training and 20% used for testing. We take the average performance of the 5-fold as the final performance of the model as the final accuracy of the model.

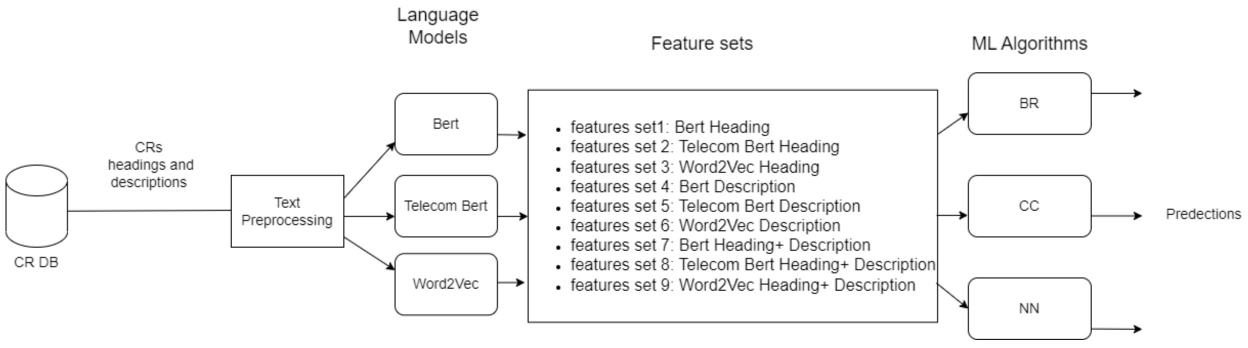
Language Models
Feature sets
ML Algorithms
CRs headings and descriptions
CR DB
Text Preprocessing
Bert
Telecom Bert
Word2Vec
features set1: Bert Heading
features set 2: Telecom Bert Heading
features set 3: Word2Vec Heading
features set 4: Bert Description
features set 5: Telecom Bert Description
features set 6: Word2Vec Description
features set 7: Bert Heading+ Description
features set 8: Telecom Bert Heading+ Description
features set 9: Word2Vec Heading+ Description
BR
CC
NN
Predections

Figure 5: Overview of our approach

3.5 Evaluation Metrics

For multilabel classification, there are three possibilities for each prediction Fully correct prediction, partially correct prediction, and fully incorrect prediction. Partial predictions occur when we predict one log type but not the other logs that should be attached to a CR. For example, for a given CR, the model predicts that a user-defined log file is needed but misses the fact the CR also requires a PMD file. To evaluate the accuracy of the algorithm, we use the Area Under the

Curve (AUC) of the Receiver Operating Characteristic (ROC) curve [42]. The ROC curve plots true positive rates (TRP) against false positive rates (FPR) at different thresholds. The AUC reflects the ability of the classifier to distinguish between classes, the higher the AUC, the better the classier [42]. With multilabel classification usually different AUC values are calculated for each label. The final accuracy of the classifier is calculated as the average AUC across all labels [44].

In addition to AUC, we use the Hamming Loss measure [41], which is the average fraction of the labels that are incorrectly predicted and reflects how many times the relevance of an example to a label class is incorrectly predicted on average. For example, if the Hamming Loss is 0.32 when trying to predict labels (types of logs) of 100 CR would mean that the model predicted incorrectly 32% of the independent labels. Hamming Loss is computed according to Equation (1) [41].

$$Hamming\ Loss = \frac{1}{NL}\sum_{i=0}^{N}\sum_{j=1}^{L} I\left(y_i^j \neq \hat{y}_i^j\right) \qquad (1)$$

where N is the number of samples, L is the number of labels, y is the actual output, $\hat{y}$ is the prediction output, and I is the indicator function. Ideally, we expect a Hamming loss, HL = 0, which would imply no error. The smaller the value of the hamming loss, the better the performance of the learning algorithm.

Chapter 4. Results and Discussion

In this section, we answer the four research questions introduced in the study setup. We also discuss the key findings, followed by a summary of the findings, and threats to validity.

4.1. RQ1: What is the overall performance of predicting the type of logs attached to a CR?

We present the detailed results of the experiment in Table A1 of Appendix A. Overall; the best average accuracy is obtained when using the headings alone with pre-trained language models Bert and Telecom Bert average AUC in average = 0.70 with a Hamming loss between 0.21 and 0.24. The figure shows that using the heading alone provides the best accuracy. The use of descriptions and headings and descriptions together as features resulted in an average AUC varying from 0.65 to 0.70 with a Hamming Loss varying between 0.21 to 0.30.

A closer look at these results is presented in Figure 6, which shows a boxplot that depicts the average prediction accuracy considering the features for all pre-trained language models and classification algorithms.

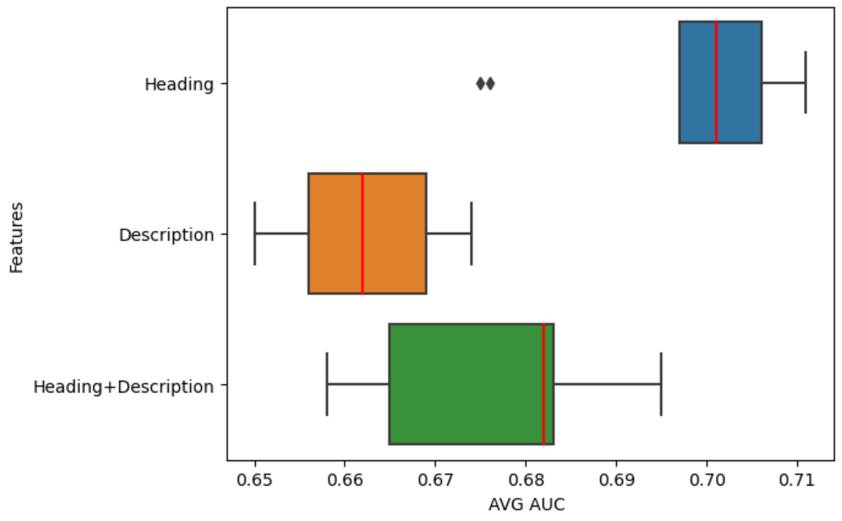

Figure 6:Auc score values concerning Features.

With the assistance of Ericsson domain experts, we investigated several CR headings and descriptions to determine the underlying causes. We found that it is common for a CR heading to contain sufficient information regarding the faulty product, node information, and a brief description of the effect of the fault. It is typical for Ericsson engineers to provide this information methodically, despite the absence of defined guidelines. The problem with CR descriptions is that they are written as free text, and hence suffer from quality problems due to noise in the data and the ambiguity and imprecision associated with the use of natural language. We also find many CR descriptions that contain a significant amount of technical information in the form of text, tables, and logs, which mislead the classifiers. This could be due to a lack of precise guidelines for writing descriptions. As a result, the descriptions included no new information that might be used to improve the prediction accuracy. On the contrary, as shown in Figure 6, combining descriptions with headings reduces accuracy.

4.2. RQ2: What is the effect of different language models for feature extraction on the accuracy of the algorithms?

Figure 7 shows a boxplot of the average accuracy of the classifiers organized by pre-trained language models. We found that the pre-trained language models Bert and Telecom Bert perform on average slightly better than Word2Vec. This is because pre-trained language models take context into account, by embedding tokens based on their meaning in the text [29][34]. Word2Vec provides static embedding that represents the word semantics without considering the complete context of the tokens [30]. In addition, the pretrained models adopt the word piece tokenization, which enhances their ability to learn new words from a given context. On the other hand, Word2Vec uses word tokenization as it considers each word as one token and then builds a specific vocabulary list based on words on the training dataset.

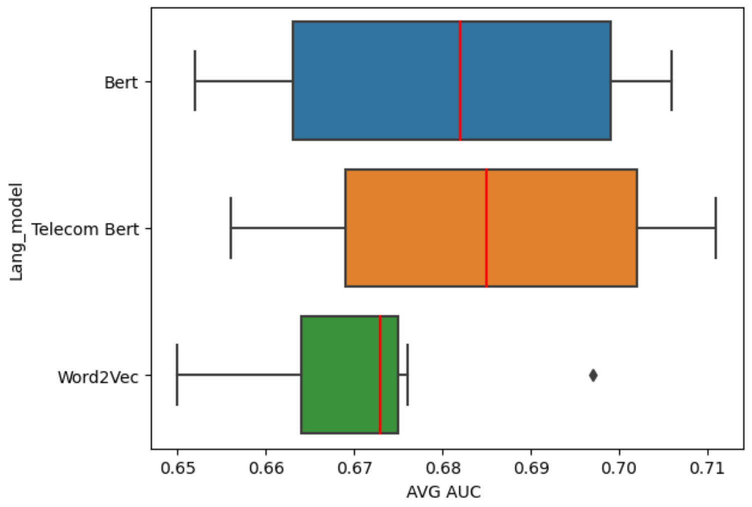

Figure 7: AUC score values concerning the language model.

An example of this would be the prediction of KPIs logs. At Ericsson, each KPI performance metric counter used to measure a certain performance aspect of the network follows specific format IndicatorName where the Indicator is fixed for all counter names where each counter has specific name. For the Word2vec language modeling approach, the embedding values of the token will be close to the embedding value of tokens commonly surrounding it given a certain window size. Therefore, when a counter name is mentioned one time within the window size of related keywords that indicate KPI log such as “degradation, increase, KPI”, it is not enough to associate their occurrence and give them similar embedding values. In the case of Word2Vec, the embedding values for counter names will not be helpful to infer KPI logs. On the contrary, the pre-trained model is better at associating the tokens’ meanings from the text context because the counter names embedding values will be closer to the embedding values of the keywords indicates KPI.

Take for example a CR heading that contains the keywords KPI and the counter name. This is an example of CR that requires a KPI log. For this CR, the embedding vectors of the keyword KPI and the counter name in the context of the CR heading, as well as the entire heading of this CR using the three language models are plotted in a 2D space as shown in Figure 8 to compare their similarity for the given language model. We expect the counter name embedding vector to be closer to the embedding vector of the keyword ‘KPI’ and the entire CR heading embedding vector using the pre-trained language models Bert and Telecom Bert than when usingWord2Vec. To compare the precise similarity, we calculate the cosine similarity between the counter name embedding vector with both the word ‘KPI’ and the CR heading embeddings for each language model as shown in Table 2. According to the cosine similarity, the smaller the angle between two vectors the higher the cosine of the angle, meaning the higher the similarity.

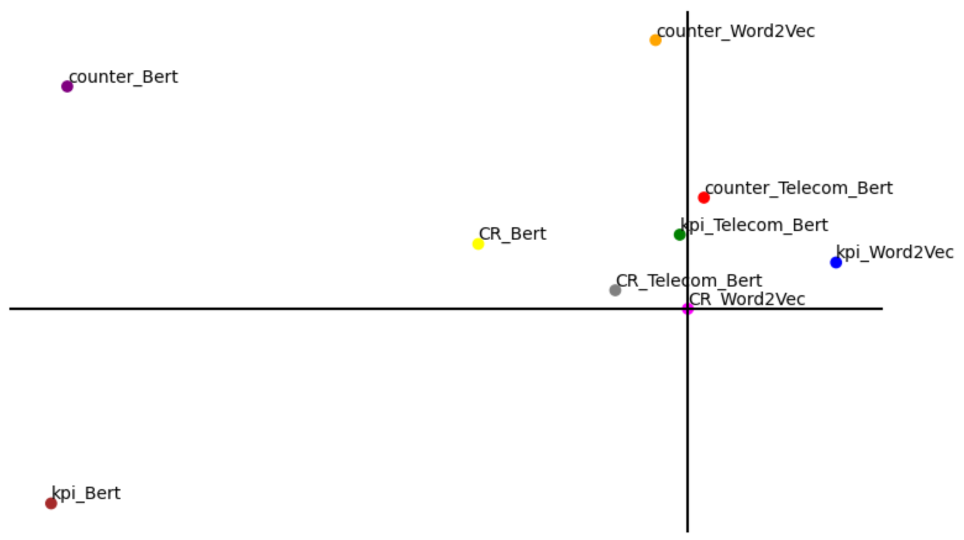

Figure 8: Embedding vectors in 2D space of a counter name, the word 'KPI' and CR heading the contains them for Bert, Telecom Bert and Word2Vec.

Table 2: Cosine similarity between counter name embedding vector with both the word 'KPI' and the entire CR heading embedding vectors for all language models.

Language Model	Cosine similarity between counter name and the word KPI	Cosine similarity between the counter name and the CR heading
Bert	0.636	0.832
Telecom Bert	0.925	0.973
Word2Vec	0.156	0.004

We can see as we excepted the counter name embedding vector is more similar to the context and related keywords (such as KPI) with Telecom Bert and Bert than when using Word2Vec.

Moreover, using Word2Vec, each counter name will be a token and since each counter has a different name the of names new counters or counters not mentioned in the dataset at least once will not be included in the vocabulary list, i.e., they will not be learned. The pre-trained models, on the other hand, can learn the names of the counters by context using the power of word piece tokenization. Take an example the following counter name: “IndicatorName”. Assume that the counter name is new. Word2VEc only learned counter names mentioned in the dataset. Therefore, it will not be able to give an embedding value of the new counter or give it a random value. On the other hand, the pre-trained models will tokenize this counter name as [‘Identifier’, ‘##Name]. Thus, they will have the ability to learn the counter names from the context because usually the token “Indicator” will be associated with keywords that indicate KPI so the remaining piece tokens will have similar embeddings values close to context.

Moreover, the use of Bert and Telecom Bert shows no significant difference in their performances. Surprisingly, Telecom Bert, which is trained in publicly available 3GPP documentation, did not introduce any knowledge that would improve the quality of the embedding features as we expected. We believe that this can be attributed to two factors. The first one may be because Telecom Bert was trained in publicly available 3GPP documentation which does not necessarily reflect the vocabulary used at Ericsson to describe crashes and failures. An interesting future work would be to fine-tune Telecom Bert by enhancing its training base with Ericsson-specific terms found in CR headings and descriptions. The second reason could be that our CRs are related to specific systems. Including more CRs that cover a wider range of Ericsson products may lead to different results.

Figure 9 shows the average AUC using the pre-trained languages Bert and Telecom Bert used with each type of feature, i.e., headings, descriptions, and headings+descriptions. We can see that Bert and Telecom Bert perform better than Word2Vec for headings and headings+descriptions. On the other hand, Word2Vec shows better accuracy when using descriptions alone. As discussed in RQ1, this could be caused by the fact that descriptions contain noise and non-textual content, failing to exhibit sufficient context that can be used by pre-trained language models.

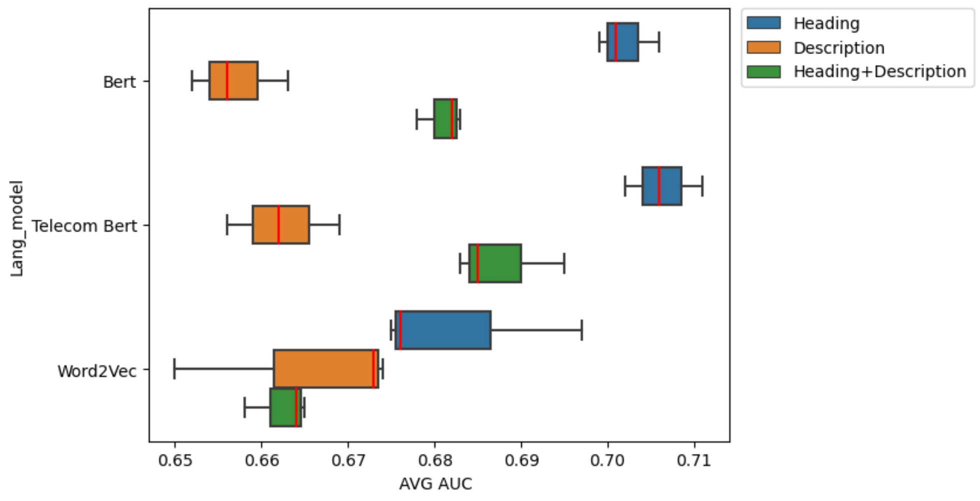

Figure 9: Average AUC using language models concerning features.

4.3. RQ3: What is the effect of using different algorithms on accuracy?

Figure 10 shows how the different algorithms compare to each other. Our findings show that there is no significant difference in the performance of the algorithms~~,~~ with a slightly better accuracy obtained using Neural Networks. However, we observe that the interquartile range (the width of the box) for Neural Network algorithm exhibits a relatively high variability of the data compared to Binary Relevance and Classifier Chains. This suggests that the accuracy of the Neural Network algorithm is more sensitive to the selected features and pre-trained language models than when using the other two algorithms.

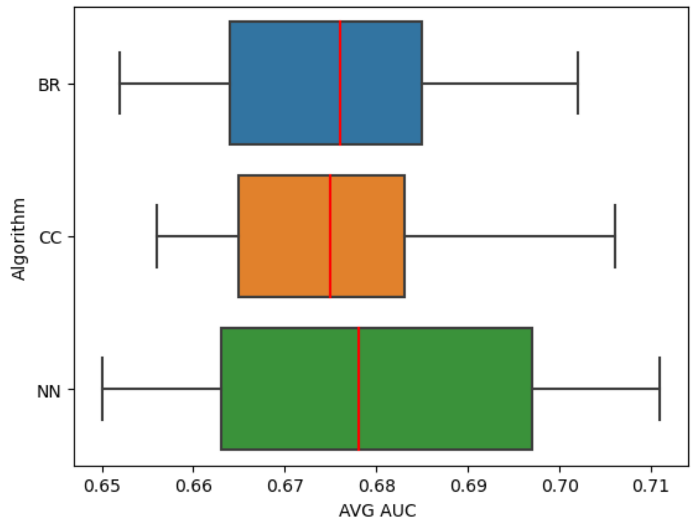

Figure 10: AUC values concern Algorithms

Even though there is no obvious correlation between labels that would give a privilege for multilabel algorithm over another, individual labels performance changes more clearly with different algorithms. Figure 11 shows the accuracy for each label across the various algorithms. We can see how PMD and KPI, for example, work better with NN. On the other hand, ND and Logs work better with BR and CR.

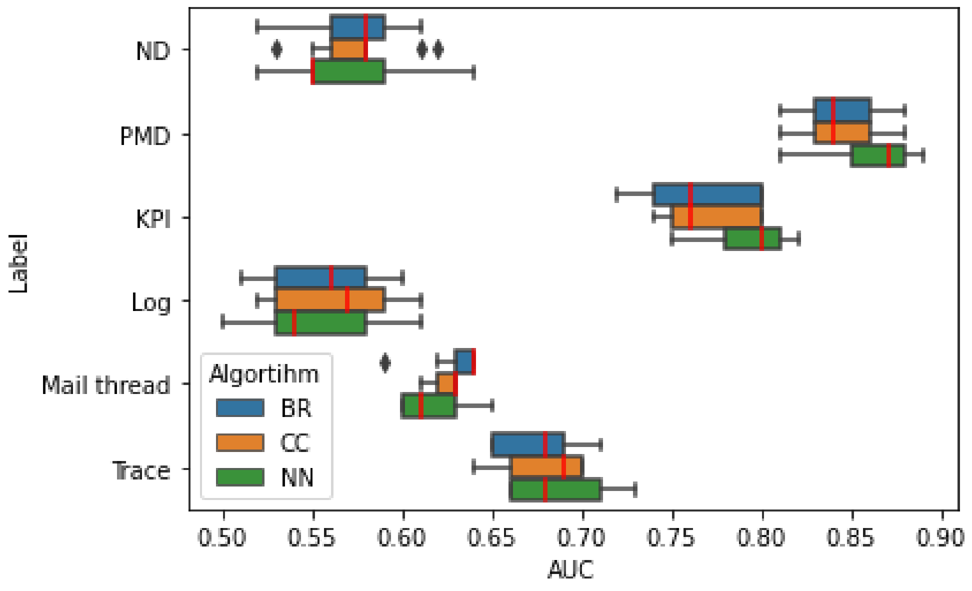

Figure 11: ACU for each label concerning the different algorithm.

4.4. RQ4: What is the accuracy of predicting each type of log (i.e., label) individually?

In this section, we discuss the performance of the algorithms and pre-trained language models for each type of log (i.e., ND, PMD, KPI, Log, Trace, and Mail Thread). Figure 12 shows a boxplot of the average AUC for each type of log for all features and algorithms. The figure shows best results are obtained when predicting PMDs, KPIs, and Traces, which an overall average AUC of 0.86, 0.78, and 0.68, respectively. To investigate the difference in performance in predicting the type of logs, we constructed a word cloud for each log type from CRs headings and descriptions to understand the keywords that may have influenced the algorithms toward the prediction of a certain type of logs. For example, it should be easier to predict that a KPI log is needed for a CR if this CR's heading, and description contain names of performance counters. We analyzed the cloud words for each type of log using Ericsson internal documentation with the assistance of

Ericsson domain experts. Many of the terms in the headings and descriptions relate to Ericsson networks.

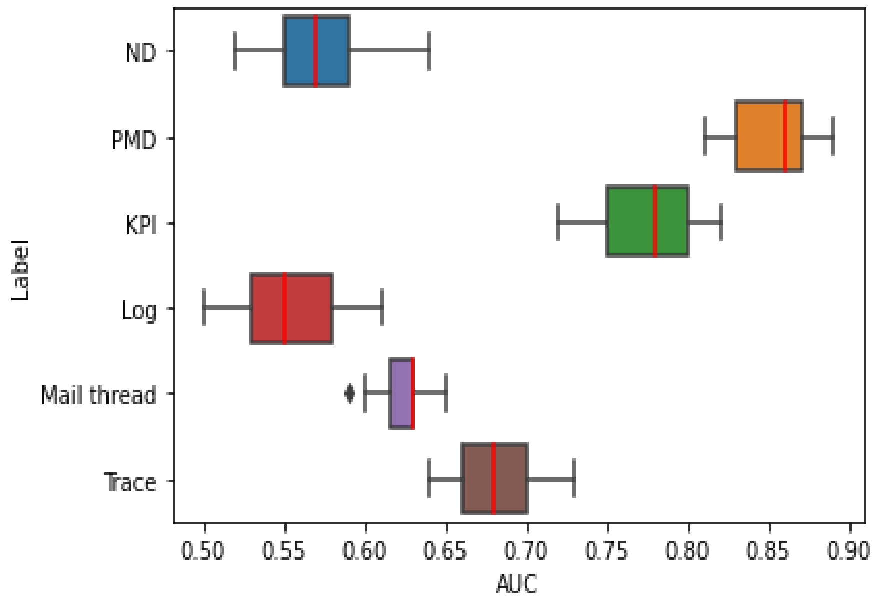

Figure 12: AUC values concern the type of log.

Typically, PMDs are associated with crashes that have a major impact on the system (e.g., complete failure of the system). We found many cases where the headings and descriptions of CRs with PMD logs contain keywords that refer excellently to different aspects of system crashes. Examples of these keywords include the terms “crash”, “post dump crash”, “system crash”, “system restart”, and so on. This may have contributed significantly to the high accuracy of up to 87% that is obtained when predicting this type of log.

The same applied to KPI logs. A KPI log is used to describe a crash associated with changes in observable performance metric counters. Usually, the CRs’ headings and descriptions of this kind of crash use references in specific performance counter names. Also, we found many cases where the degradation of performance related to performance counters is mentioned in CRs headings and descriptions. Typical keywords used in CR headings and descriptions to indicate performance degradation include Ericsson-specific performance counter names, abnormal rate, performance degradation, change, and traffic, as well as the term KPI itself which is referred to in many CR headings and descriptions.

Traces files are well labeled in comparison to NDs and Logs. Traces are usually used with errors that minorly affect the performance system. From the word cloud, we noticed the major keywords that Trace relies on conflict with keywords of other types, especially KPI. It is common to use traces in combination with KPI logs to allow engineers to perform root cause analysis of performance faults. Examples of traces include stack traces, which contain function calls that occur during a system crash. Stack traces are used by developers to troubleshoot the system to provide a fix.

The average AUC for predicting mail threads is around 63%, which is higher than the average AUC for predicting logs, and Node Dumps. The reason is that mail threads are often used to explain the changes in the performance metrics and to help diagnose the faults. This association between mail threads and KPIs may explain the acceptable accuracy when predicting mail threads.

Figure 8 shows that worse performance in terms of average AUC is obtained when predicting NDs and user-defined logs. According to Table A1, the AUC for predicting NDs and user-defined logs is between 52% and 64% in the case of NDs and between 51% and 61% for user-defined logs. According to Ericsson experts who helped us interpret these results, user-defined logs are more general than PMDs, KPIs, and Traces in the sense that they can contain anything that goes on in the system when a crash occurs including OS system calls, internal variables, functions, etc. It is challenging to know which user-defined logs are useful to fix a given crash. User-defined logs are used when other types of logs (PMDs, traces, and KPIs) are not sufficient to perform root cause analysis. NDs are also too general as they contain commands that are used to run the failed nodes to collect other types of logs. Therefore, the information in the NDs is not used to describe the crashes themselves.

Chapter 5. Conclusion and Future Work

5.1. Conclusion

In this thesis, we proposed an approach to predict crash report log file types that leverages CR headings and descriptions as main features. In our approach, we have used various language models including Bert, Telecom Bert and Word2Vec to obtain feature vectors from CR headings and descriptions. These vectors are fed t to three multilabel machine learning algorithms, namely binary relevance, classifier chains, and deep neural networks to train a model able to predict log file types. We used a confidential dataset of CRs from Ericsson's CR database.

Our results showed that given the CR heading as a feature regardless of the language model or the multilabel classification algorithm, we can predict the log file types with an average AUC of 0.70. In addition, we found that the log types PMDs, KPIs, and Traces are the ones that are predicted with high accuracy. This may be attributed to the fact the headings of CRs with these logs usually have words (e.g., performance counters for KPI logs) that can help the classifier recognize the log file type. We found that the CR descriptions do not provide more information that would enhance the accuracy, probably because of the amount of noise CR descriptions tend to contain. Moreover, we found that the pre-trained language models Bert and Telecom Bert provide a better representation of the context of CR headings than Word2Vec, which improves the performance of the models that use features extracted using Bert and Telecom Bert. The performance of the individual type of log varies based on the keywords associated with this log type and the correlation between log types which makes the prediction of some log file types, namely PMDs, KPIs, and Traces better than the prediction of NDs and user-defined logs.

5.2. Limitations and Future Work

A. Experimenting with more datasets: Despite our effort, we cannot claim generalizability of the result because we experimented with CRs generated from a specific Ericsson system. As part of future work, we need to experiment with more datasets from open source or industrial systems. The challenge, however, is to find open-source systems with crash reports that contain log data. This is not common because of the overhead of storing and managing large log files [52].

B. Experimenting with more machine learning algorithms: As we have seen, the different multilabel methods did not show a significant difference in the performance. In the future, we could investigate some imbalance multilabel algorithms to handle the imbalance distribution of the labels within the multilabel scope. Also, we could try more multilabel algorithms such as ML-KKN [25].

C. Improving the language model part: We expected Telecom Bert to provide features that outperform the use of Bert and Word2Vec features because Telecom Bert is trained in the Telecom domain similar to that of Ericsson. In the future, we should work towards fine-tuning or pre-training Telecom Bert and Bert language models on the language that is used in the CRs headings and descriptions. Doing this will require domain knowledge and expertise that can be obtained by working closely with Ericsson experts.

D. Improving the definition of log types: There is no clear definition of the various log types used in this study. We relied mainly on domain expertise to classify log files based on the six types discussed in this thesis (ND, traces, KPIs, etc.). We also believe that this is a common problem in other companies and not only at Ericsson. There is a need to define better logging and tracing guidelines that clearly distinguish among various types of log files generated at Ericsson.

E. Integrating this work with existing tools: In the future, we intend to integrate this work in the crash report handling system of Ericsson and work with analysts to understand how they will use these techniques in practice. We can also add humans in the loop to improve the recommendation made by the log type prediction tool.

References

[1] K. Panchal, “An Empirical Study of Runtime Files Attached to Crash Reports,” *Master’s Thesis, Concordia University, Montreal, Quebec, Canada,* 2022.

[2] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, “What Makes a Good Bug Report?” *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[3] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu, “Improving bug tracking systems,” in *Proc. of the 31st International Conference on Software Engineering (Companion Volume)*, 2009, pp. 247–250.

[4] R. K. Saha, S. Khurshid, and D. E. Perry, “An empirical study of long-lived bugs,” *in Proc. of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 144–153.

[5] L. Hribar and D. Duka, “Reporting and removing faults in telecommunication software,” in *Proc. of the 34th International Convention MIPRO*, 2011.

[6] A. R. Chen, T.-H. (Peter) Chen, and S. Wang, “Demystifying the challenges and benefits of analyzing user-reported logs in bug reports,” *Empirical Software Engineering*, vol. 26, no. 1, 2021.

[7] L. An, F. Khomh, “Challenges and Issues of Mining Crash Reports,” in *Proc. of the IEEE 1st*

International Workshop on Software Analytics (SWAN), 2015, pp. 5–8.

[8] X. Xia, D. Lo, M. Wen, E. Shihab, and B. Zhou, “An empirical study of bug report field reassignment,” in *Proc. of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 174–183.

[9] X. Xia, D. Lo, E. Shihab, and X. Wang, “Automated Bug Report Field Reassignment and Refinement Prediction,” *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1094–1113, 2016.

[10] V. Mondreti and C. J. Satish, “Bug Severity Prediction System Using XGBoost Framework,” *in Proc. of the 2020 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT’20)*, 2020.

[11] Y. Jia, X. Chen, S. Xu, G. Yang, and J. Cao, “EKD-BSP: Bug Report Severity Prediction by Extracting Keywords from Description,” *in Proc. of the 8th International Conference on Dependable Systems and Their Applications (DSA’21)*, 2021, pp. 42–53.

[12] J. Kim, G. Yang, “Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm,” *IEEE Access*, 2023, pp. 94643 - 94651.

[13] C. Y. Zhou, C. Zeng, and P. He, “An Exploratory Study of Bug Prioritization and Severity Prediction based on Source Code Features,” *in Proc. of the 34th International Conference on Software Engineering and Knowledge Engineering* (SEKE’22), 2022, pp. 178–183.

[14] M. N. Pushpalatha and M. Mrunalini, “Predicting the severity of bug reports using classification algorithms,” in *Proc. of the 2016 International Conference on Circuits, Controls, Communications and Computing (I4C),* 2016, pp. 1–4.

[15] R. Mihalcea and P. Tarau, “Textrank: Bringing order into text,” in *Proc. of the 2004 Conference on Empirical Methods in Natural Language Processing,* 2004, pp. 404–411.

[16] K. K. Sabor, M. Nayrolles, A. Trabelsi, and A. Hamou-Lhadj, “An Approach for Predicting Bug Report Fields Using a Neural Network Learning Model,” in *Proc. of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* 2018, pp. 232–236.

[17] K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi, and J. Hassine, “Predicting bug report fields using stack traces and categorical attributes,” in *Proc. of the 29th Annual International Conference on Computer Science and Software Engineering,* 2019, pp. 224–233.

[18] A. Lamkanfi and S. Demeyer, “Predicting Reassignments of Bug Reports - An Exploratory Investigation,*”* in *Proc. of the 17th European Conference on Software Maintenance and Reengineering,* 2013, pp. 327–330.

[19] I. Chawla and S. K. Singh, “Improving bug report quality by predicting correct component in bug reports,” *International Journal of Computational Intelligence Studies,* vol. 8, no. 1–2, 2019, pp. 143–57.

[20] R. M. D. S. Rathnayake, B. T. G. S. Kumara, and E. M. U. W. J. B. Ekanayake, “CNN - Based Priority Prediction of Bug Reports,” *in 2021 International Conference on Decision*

Aid Sciences and Application (DASA), Dec. 2021, pp. 299–303. doi: 10.1109/DASA53625.2021.9682284.

[21] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis," in *Proc. of the IEEE International Conference on Software Maintenance,* 2013, pp. 200–209.

[22] M. Alenezi and S. Banitaan, "Bug Reports Prioritization: Which Features and Classifier to Use*?"* in *Proc. of the 12th International Conference on Machine Learning and Applications*, vol. 2, 2013, pp. 112–116.

[23] Q. Umer, H. Liu, and Y. Sultan, "Emotion Based Automated Priority Prediction for Bug Reports," *IEEE Access*, vol. 6, 2018, pp. 35743–35752.

[24] A. Schroter, A. Schröter, N. Bettenburg, and R. Premraj, "Do stack traces help developers fix bugs?" in *Proc. of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 118–121.

[25] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognition,* vol. 40, no. 7, 2007, pp. 2038–2048.

[26] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[27] S. L. Salzberg, "C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993," *Mach Learn*, vol. 16, no. 3, pp. 235–240, Sep. 1994, doi:

10.1007/BF00993309.

[28] "Adopting neural language models for telecom," Available online : https://www.ericsson.com/en/blog/2022/1/neural-language-models-telecom-domain.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv*, 2019. Available online: https://arxiv.org/abs/1810.04805

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." *arXiv*, 2013. Available online: https://arxiv.org/abs/1301.3781

[31] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach Learn*, vol. 85, no. 3, pp. 333–359, Dec. 2011, doi: 10.1007/s10994-011-5256-5.

[32] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: an overview," *Frontiers of Computer Science*, vol. 12, no. 2, 2018, pp. 191–202.

[33] L. Jia, J. Fan, D. Sun, Q. Gao, and Y. Lu, "Research on multi-label classification problems based on neural networks and label correlation," in *Proc. of the 41st Chinese Control Conference (CCC)*, 2022, pp. 7298–7302.

[34] S. Bird, E. Klein, and E. Loper. Natural Language Processing with Python. Available online: https://www.nltk.org/book/.

[35] D. Jurafsky and J. H. Martin, “Speech and Language Processing,” Available online: https://web.stanford.edu/~jurafsky/slp3/

[36] Y. Liu, M. Ott, *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *arXiv*, 2019. Available online: https://arxiv.org/abs/1907.11692

[37] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv,* 2020. Available online: https://arxiv.org/abs/1910.01108

[38] M-L. Zhang and Z.-H. Zhou, “Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, 2006, pp. 1338–1351.

[39] S. Kumar, N. Kumar, A. Dev, and S. Naorem, “Movie genre classification using binary relevance, label powerset, and machine learning classifiers,” *Multimedia Tools and Applications*, vol. 82, no. 1, 2023, pp. 945–968.

[40] A. Y. Taha and S. Tiun, “Binary relevance (BR) method classifier of multi-label classification for arabic text,” *Journal of Theoretical and Applied Information Technology,* 2005.

[41] M. S. Sorower, “A Literature Survey on Algorithms for Multi-label Learning,” Oregon State University, 2010

[42] T. Fawcett, “An Introduction to ROC Analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, 2006, pp. 861–874.

[43] T. K. Ho, "Random decision forests," in *Proc. of the 3rd International Conference on Document Analysis and Recognition*, pp. 278-282, 1995.

[44] D. J. Hand, R. J. Til, “A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems,” *Machine Learning*, 45(2), 2001, pp. 171-186.

[45] A. Caliskan, J. J. Bryson, A. Narayanan, “Semantics derived automatically from language corpora contain human-like biases,” *Science, Vol 356, Issue 6334,* 2017, pp. 183-18.

[46] D. Gligorijevic, J. Stojanovic, N. Djuric, V. Radosavljevic, M. Grbovic, R. J. Kulathinal, Z. Obradovic, “Large-Scale Discovery of Disease-Disease and Disease-Gene Associations,” *Sci. Rep.*, vol. 6, no 32404, 2016.

[47] J.Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, J. Kang, “BioBERT: a pre-trained biomedical language representation model for biomedical text mining,” *Bioinformatics,* vol. 36, no. 4, 2020. pp. 1234–1240.

[48] I. Beltagy, K. Lo, and A. Cohan, “SciBERT: A Pretrained Language Model for Scientific Text.” arXiv, 2019. Available online: http://arxiv.org/abs/1903.10676

[49] L. Tunstall, L. von Werra, T. Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022.

[50] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning.* MIT Press, 2016.

[51] K. Panchal, F. Ait-Mahammed, A. Hamou-Lhadj, Z. Zhu, S. Memon, A. Isac, P. Krishnamoorthy, "A Study on the Use of Runtime Files in Handling Crash Reports in a Large Telecom Company," *in Proc. of the IEEE Future Networks World Forum*, 2022.

[52] A. V. Miranskyy, A. Hamou-Lhadj, E. Cialini, A. Larsson, "Operational-Log Analysis for Big Data Systems: Challenges and Solutions," *IEEE Software 33(2),* pp. 52-59, 2016.

Appendix A.

Table A1. Detailed Results of the Experiments

Features	**Algorithm**	**Av AUC**	**HL**	**ND**	**PMD**	**KPI**	**Log**	**MT**	**Trace**
Bert-Heading	BR	0.701	0.23	0.61	0.86	0.8	0.58	0.64	0.71
	CC	0.699	0.23	0.61	0.86	0.8	0.59	0.62	0.7
	NN	0.706	0.22	0.6	0.88	0.81	0.61	0.6	0.72
Telecom Bert - Heading	BR	0.702	0.24	0.61	0.87	0.8	0.6	0.64	0.69
	CC	0.706	0.24	0.62	0.86	0.8	0.61	0.63	0.7
	NN	**0.711**	0.21	0.64	0.88	0.81	0.58	0.62	0.71
Word2Vec-Headign	BR	0.676	0.29	0.58	0.84	0.74	0.58	0.64	0.65
	CC	0.675	0.3	0.58	0.83	0.75	0.59	0.63	0.65
	NN	0.697	0.22	0.56	0.85	0.8	0.58	0.65	0.73
Bert-Description	BR	0.652	0.25	0.55	0.83	0.73	0.51	0.62	0.65
	CC	0.656	0.25	0.55	0.83	0.74	0.52	0.62	0.66
	NN	0.663	0.22	0.55	0.86	0.78	0.52	0.6	0.66
Telecom Bert - Description	BR	0.656	0.24	0.58	0.84	0.72	0.53	0.59	0.65
	CC	0.662	0.24	0.58	0.84	0.75	0.53	0.61	0.64
	NN	0.669	0.22	0.55	0.87	0.79	0.53	0.6	0.66
Word2Vec-Description	**BR**	**0.674**	0.29	0.56	0.81	0.76	0.56	0.63	0.7
	CC	0.673	0.29	0.56	0.81	0.76	0.58	0.63	0.7
	NN	0.650	0.22	0.53	0.81	0.77	0.5	0.61	0.66
Bert-Heading +Description	BR	0.683	0.23	0.57	0.86	0.78	0.54	0.64	0.69
	CC	0.682	0.24	0.56	0.86	0.79	0.54	0.63	0.69
	NN	0.678	0.22	0.55	0.88	0.81	0.53	0.6	0.68
Telecom Bert - Heading +Description	BR	0.685	0.23	0.59	0.88	0.8	0.52	0.63	0.68
	CC	0.683	0.23	0.58	0.88	0.8	0.52	0.63	0.68
	NN	**0.695**	0.21	0.59	0.89	0.82	0.54	0.63	0.68
Word2Vec-Heading +Description	BR	0.664	0.3	0.52	0.81	0.75	0.57	0.64	0.68
	CC	0.665	0.3	0.53	0.81	0.76	0.57	0.62	0.69
	NN	0.658	0.22	0.52	0.82	0.75	0.55	0.63	0.67