DUPLICATE BUG REPORT DETECTION USING PRE-TRAINED LANGUAGE MODELS

K.A.Udeshika Sewwandi

(199363T)

Degree of Master of Science in Computer Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

July 2022

DUPLICATE BUG REPORT DETECTION USING PRE-TRAINED LANGUAGE MODELS

Kahandawala Arachchige Udeshika Sewwandi

(199363T)

Thesis/Dissertation was submitted in partial fulfillment of the requirements for the degree MSc in Computer Science specializing in Data Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

July 2022

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:..... Date:……

The above candidate has carried out research for the PhD/MPhil/Masters thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of the Supervisor: Dr. Surangika Ranathunga

Signature of the supervisor: ..... Date:……

ACKNOWLEDGEMENTS

I would like to show my gratitude to Dr. Surangika Ranathunga for guiding me to initiate and find a better methodology and try out different novel approaches to conduct the research. Her supervision greatly helped me in setting goals and engaging in the research.

I would like to express our greatest gratitude to the Department of Computer Science and Engineering, the University of Moratuwa for providing the support to overcome this effort.

Last but not least, my heartfelt gratitude goes to my parents, husband, and friends who supported me throughout this effort.

ABSTRACT

Software testing and defect reporting are significant factors of software development and maintenance. Defects are identified and reported in a bug tracking system like JIRA, or Bugzilla. Those reported defects are further triaged by an expert who has an understanding of the repository, system, and developers and assigns them to the developers to fix them. During this defect reporting there can be duplicate bugs reported and identifying duplicate bugs is a crucial task. Manual labeling of duplicate defects is time-consuming, may identify defects as duplicate bug reports, and also increases the cost of software maintenance. Therefore automated duplicate bug report detection is very significant. This research proposes a duplicate bug report classification methodology that leverages the Pre-trained language models BERT and XLNet with Multi-Layer Perceptron as the Deep Learning classifier for duplicate bug detection. We tested on publicly available datasets related to Eclipse, NetBeans, and OpenOffice bug reporting datasets. The selected models were shown to outperform the previously proposed systems for the same task. Among them, the approach used with BERT embeddings has shown the best results. Further experiments showed that BERT is capable of domain adaptation –meaning that even when the BERT was fine-tuned with different bug report datasets, it is still capable of detecting duplicate bugs in an unseen dataset. Finally, a multi-stage classification was done using a Convolutional Neural Network model and a BERT model using Eclipse and NetBeans datasets and a combined dataset of Eclipse and NetBeans. The approach used with the combined dataset has outperformed the baseline approach.

Keywords: Duplicate Bug detection, BERT, XLNet, MLP, CNN, Domain Adaptation, Multi-Stage Classification

Table of Contents

List of figures

List of tables

1. CHAPTER 1: INTRODUCTION

1.1 Background

Software testing and defect reporting are key factors of software development and maintenance. Software testing starts from the design phase and moves until the testing phase in SDLC. In all these phases, defects are identified and the cost of fixing the bugs varies with the phase transition. The structure of a bug is illustrated in Figure 1.1.

Tag Name	Content
Bug ID	11112
Product	Platform
Component	SWT
Summary	Multi monitors not correctly supported
Status	RESOLVED
Resolution	DUPLICATE
Duplicates	7232
Priority	P3
Severity	major
Version	2.0
Created	2002-03-11 13:53:00 -0500
Modified	2002-05-15 12:07:18 -0400
Description	Situation: Windows XP with two monitors. If you run eclipse on your secondary monitor, code completion and comments are displayed on the primary monitor.

Figure 1.1: Sample bug report document. Source [9]

The System Sciences Institute at IBM has stated that it is six times more costly to fix a bug in the implementation phase rather than to fix it in the design phase [1]. They further stated that it is 15 times more costly to fix a bug in the testing phase rather than fixing it in the design phase. The defects are identified and reported in a bug tracking system like JIRA, or Bugzilla, and those reported defects are further triaged by an expert who has an understanding of the repository, system, and developers and assigns to the developers to fix them. During this defect reporting there can be duplicate bugs reported and identifying duplicate bugs is a crucial task. There can be two causes for reporting duplicate bugs. 1) There are a large number of users for a popular software system and users may come across a similar issue in several different ways. This results in duplicate bug reporting by different users. 2) Software systems have different versions with the evolution of the software. Different users

may install different versions of the software. An issue fixed in a higher version may be still available in a lower version and it will result in duplicate bug reporting [2].

Duplicate bugs may have information captured in different ways which may be very helpful for developers to solve the bug with that concatenated information from all duplicate bugs. That will relieve the stress of developers in fixing the particular bug [3]. The duplicate bugs can be assigned to the same developer as well as the same team or different developers as well as different teams. If it is assigned to the same developer or team, the bug count for the particular developer or team rises unnecessarily with duplicate bugs. If it is assigned to different developers or different teams, many will attend to the same bug, and the debugging cost and time consumption increases.

Most software systems use bug tracking systems for bug reporting. They are useful in many instances by providing a range of functionalities to make it easy to report bugs with detailed descriptions and search bugs with queries. But still, there are some limitations in bug tracking systems where automatic labeling of duplicate bugs cannot be done and only manual labeling can be performed.

1.2 Research Statement

How to improve duplicate bug report detection using pre-trained language models.

1.3 Motivation

Duplicate bug detection reduces the cost of software maintenance, unnecessary time wastage, and debugging cost by developers. Two methods have been commonly used for automatic bug detection: Information retrieval (IR) and classification.
The most frequently used method is classification. In this technique, a bug report document is classified into one of the two classes whether it is a duplicate or non-duplicate. Previous classification systems have employed several Machine Learning algorithms such as Support Vector Machine (SVM) [5], K-Nearest Neighbours (KNN) [22], Zero-R [12], C4.5 [12], and Random Forest[13]. Although many of these methods have made the life of the triager easier still the performance and accuracy of the developed models are not to the expected level. Deep Learning techniques such as Multi-Layer Perceptron (MLP) [10], Long Short Term Memory (LSTM) [14], and Convolutional Neural Networks (CNN) [2], [9] which have shown better results than the traditional Machine Learning algorithms.

Currently, usage of pre-trained language models (e.g.Bidirectional Encoder Representation from Transformers (BERT) and XLNet) for contextual information capturing has shown state-of-the-art(SOTA) performance in various classification tasks [37]. But very little of the previous research has used BERT or XLNet based networks to capture contextual information related to duplicate bug report detection

and still there's room for improvement [39], [40]. Bug reports have a description of the bug, code snippets, a summary of the bug, and product and component like features which are helpful to capture the context easily. Thus, the classification problem is different from other classification problems such as topic or sentiment classification.

Another problem of duplicate bug detection is the domain-specific nature of the problem. In other words, the information reported in a bug may depend on the corresponding software system, and a model trained on one data from one software system may not generalize to bug reports of another system. However, previous research has not studied this problem.

1.4 Objectives

The main objective of this research is to solve the problem of duplicate bug report detection. To achieve this there are sub-objectives to be accomplished,

1. Develop a system that classifies a bug report pair as a duplicate or non-duplicate more accurately than existing systems using pre-trained language models.
2. Explore the possibility of using pre-trained language models for domain-adaptation

1.5 Thesis Structure

The thesis contains four chapters ahead namely Literature Review as Chapter 2, Methodology as Chapter 3, Evaluation as Chapter 4, and Conclusion as Chapter 5. The Literature Review chapter will elaborate on the fundamental components of this research such as BERT-based contextual information extraction, Text classification using Deep Learning techniques, and more literature review on Information Retrieval, Text classification using machine learning techniques. Also, it will further elaborate on the specific dataset that will be used in the research, the evaluation metrics, and the baseline research that has much room to improve for the selected data set.

The methodology chapter covers the approaches suggested in this research like text classification with MLP, domain adaptation, text classification with CNN, and Multi stage classification. It further elaborates what techniques were used to successfully carry out the work, how the work was accomplished and the architectural diagrams related to the approaches suggested.

The evaluation section illustrates the results obtained on each suggested approach, a comparison of those approaches to the baseline research, a comparison of the

suggested approaches, and shows which approaches work the best. Further, it discusses which approaches work better than others and why it happened.

The last section is the conclusion section which describes what is found from the research and the implications of the research.

2. CHAPTER 2: LITERATURE REVIEW

2.1 Overview

Research on automating duplicate bug report detection started around the year 2004 [11]. Since then many researchers have taken forward the research on automatic bug report detection and similar research disciplines such as duplicate Community Question Answering and question search etc [3], [4].
Duplicate detection is not an easy task to perform, because it solely depends on the text which is user-specific in this research. There are many problems in bug reports such as syntactically wrong sentences, a semantically ambiguous text, usage of user-specific abbreviations, etc in text written by users which makes it harder to distinguish the text properly. An instance of the description and title of a duplicate bug report pair from the esmall mongo dump is represented below in Figure 2.1.

Tag Name	Content	
Bug ID	4524	9002
Product	Math	Math
Component	UI	Code
Summary	space between a vector and its arrow too large.	formating of font attributes.
Description	Hi, The space between a vector and its arrow is too large making the formula too high. It doesn't matter much when the formula is a paragraph of its own but it looks clumsy when placed among the text of a paragraph. To make myself clear, copy this text in a sxw file then isert in the middle of the previous paragraph the formula "vec u" or the formula "**widevec** AB". Compare with what you'd get inserting the formula "overline AB". Thanks	The attibutes: hat, grave, tilde, check, bar, vector, and so on are too far removed from the font. Seems to be a problem with the font definitions used. Workaround are **widevec**, widehat, widebar etc. Unfortunatelly the 'wide' version does not exist for all attributes. Also,'bold' in formulae is tranlated into some sort of arial font with poor spacing wthin characters. It is unfortunate that this has changed from SOv5 which used the more conventional mathematical notation of Times bold for that, which incidentally has better character kerning.

Figure 2.1: Duplicate bug report pair. Source [9]

The literature in this research can be categorized under several criteria. Basic text similarity related techniques have evolved from lexical similarity to syntax and from syntactic similarity to the more advanced semantic similarity techniques. The most popular techniques for duplicate bug report detection are text classification, Information retrieval, and query reformulation.

2.2 Text Classification

In text classification, a bug report pair is classified as duplicates or non-duplicates by the means of machine learning techniques in some research and Deep Learning techniques in some research. This section will describe the literature on text classification that is based on traditional machine learning and Deep Learning techniques and contextual information extraction based on pre-trained language models such as BERT and XLNet.

2.2.1 Text Classification using traditional machine learning techniques

Capturing contextual information is also a crucial thing in duplicate bug report detection as per Alipour et al. [12]. They have incorporated the textual and categorical comparison method of Sun et al. [5] and concentrated on how and what contextual information can be captured from bug reports. They have considered five contextual word collections as Labeled-LDA, LDA, NFR, Android architecture, and English random words. They have also leveraged BM25F to measure textual similarity and 0-R, Logistic Regression, Naive Bayes, C4.5, and K-NN as machine learning algorithms. The approach has improved the accuracy of duplicate bug report detection by 11.55% than Sun et al. [6].

Lazar et al. [13] have later conducted different research based on duplicate bug report detection based on textual similarity and classification. The datasets used were Eclipse, Open Office, and Mozilla which are the datasets generated from their previous research on generating duplicate bug datasets. The features considered were bug id, title, description, product, component, type, priority, version, and open date. The classification was done using K-NN, Linear SVM, RBF SVM, Decision Tree, Random Forest, and Naive Bayes machine learning algorithms. This approach has improved duplicate bug report detection by 9.53% over Alipour et al. [12].

2.2.2 Text Classification using Deep Learning techniques

Deshmukh et al. [14] are one of the first people to apply Deep Learning techniques for the duplicate bug report detection problem. The researchers proposed a model based on IR and classification. They have created a model based on Siamese Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM) with high accuracy of detection and retrieval of duplicate and similar bugs. They have shown promising results in duplicate bug report detection using Deep Learning.

Xie et al. [2] have considered domain-specific features as a novel technique. The researchers have exploited word embeddings and CNN like Deep Learning techniques to compute the similarity between a pair of bug reports. After preprocessing, word embeddings were created for the words, and the training dataset is trained on the CNN model. The highest accuracy is obtained for the hdfs dataset and it is 6% higher than the accuracy obtained by Deshmukh et al. [14].

Poddar et al. [10] proposed an architecture that identifies if a bug report pair is a duplicate and create latent topics from them. They have designed a loss function to perform duplicate detection and create latent topics and also used an attention layer for duplicate detection. MLP is used to train the model.

He et al. [9] have proposed a novel approach to explore Dual-Channel Convolutional Neural Networks (DC-CNN) on duplicate bug report detection. They have presented

a novel method to represent bug report pairs. That is done by concatenating two single-channel matrices representing bug reports to form a dual-channel matrix. First, they have preprocessed the text by performing tokenization, stemming, stop words removal, and case conversion. The continuous Bag of Words (CBOW) model and word2vec were used to generate word embeddings. They compared it with the single-channel matrix approach and have improved accuracy by 3.03%.

Isotani et al. [39] have proposed a method based on BERT based sentence embeddings. They have vectorized the bug reports using sentence embeddings and calculated the similarity of the report using the vectors. They considered a Japanese dataset and a Japanese BERT model for the research and obtained a precision of 0.829.

Rocha et al. [40] have proposed an approach that leverages individual bugs as well as bug clusters. They have considered contextual and semantic information on structured and unstructured data. They carried out both classification and IR tasks and obtained 84% AUROC and Recall@25 mean of 85%.

2.3 Datasets for Duplicate Bug Detection

When considering the past research, they had to extract datasets from open source large datasets to carry out the research. Lazar et al. [7] have identified this and have taken steps to generate duplicate bug datasets for future research. They have created datasets for Eclipse, Open Office, NetBeans, and Mozilla by extracting data. In the dataset, all initial data and cleansed data are stored separately in MongoDB. The dataset contains pairs of all duplicate bug reports and randomly selected pairs of non-duplicate bugs.

2.4 Information Retrieval for Duplicate Bug Report Detection

In information retrieval top-k relevant bug reports are fetched and ranked for a given bug report document based on a similarity measurement like cosine similarity [15], Manhattan distance [19], etc. For query reformulation, the query inserted by the user to the system is again reformulated by employing different strategies to make sure the most relevant documents are retrieved.
Sun et al. [5] created a discriminative model for duplicate bug report detection using Support Vector Machines (SVM) which can be used for classification in information retrieval. To perform the classification they used a probability value. They have used Firefox, Eclipse, and Open Office datasets [8] and have achieved a recall rate of around 70% for all three datasets when k=20 where k is the top k list size.

By taking the previous research [5] as the baseline Sun et al. [6] have done the next research by proposing a retrieval function (REP). Here they have extended BM25

which is a text similarity function. That was done by accounting for the term weights in terms. This REP retrieval function is optimized with the means of gradient descent. Here also they have used the same datasets and have shown that this novel approach has improved by 22.99%, 19.91% 17.20% for OpenOffice, Mozilla, and Eclipse datasets.

Rocha et al. [38] have evaluated NextBug, which is a recommender for duplicate bug report detection. They have shown that bug component and short description works well by evaluating NextBug on 69 projects in the Mozilla ecosystem. NextBug has shown better results than REP proposed by Sun et al. [6].

To find similar bugs Yang et al. [18] have proposed an approach combining word embeddings with information retrieval. They have four components: TF-IDF component, word embedding component, product & component information component, and combination component. In the combination component, they have combined the three similarity scores of the first 3 components into one score. Similar bugs to the querying bug were found by taking the pending bugs with the highest final similarity score. This approach has shown it has outperformed NextBug of Rocha et al. [38] in similar bug recommendations.

Budhiraja et al. [15] have explored a novel approach using word embeddings to retrieve top k similar reports for a given bug report. Their approach is to generate all bug reports as dense vectors unlike considering just the word vectors as in most of the previous research. First, the words were represented as vectors and the vectors are averaged according to the presence of the words in the bug report to create the document vectors. They have used OpenOffice and Mozilla datasets and their approach outperformed the baseline approaches including BM25F and LDA approaches and worked best and have the highest recall rate when k=20.

2.5 Pre-Trained Language Models

2.5.1 BERT

BERT is a pre-trained language model developed by Google [23]. It exploits publicly available data on the web like Wikipedia and Google Books [24]. The significant factor here is that it captures contextual word embeddings unlike Word2Vec like word embeddings.
BERT architecture is based on transformers architecture. BERT is based on the Auto Encoder(AE) Language Model which aims to construct original data from corrupted data or masked input. Older language models only could read text inputs in order. But as the name implies BERT learns from bidirectional sides and from the first layer to the last layer. BERT is mainly designed for two tasks namely Masked Language Modelling(MLM) and Next Sentence Prediction(NSP). In MLM one word

is replaced with a [MASK] token and tries to predict that word based on the word's context. In NSP, as the name implies when two sentences are given namely A and B we predict if B is the next sentence or is it just a random sentence when sentence A is given.
There are two pre-trained BERT models namely BERT Base and BERT Large. BERT Base has 12 layers, 12 attention heads, and 110mn parameters. BERT Large has 24 layers, 16 attention heads, and 340mn parameters. Here in this research, I used the BERT Base model to create embeddings for the bug reports.

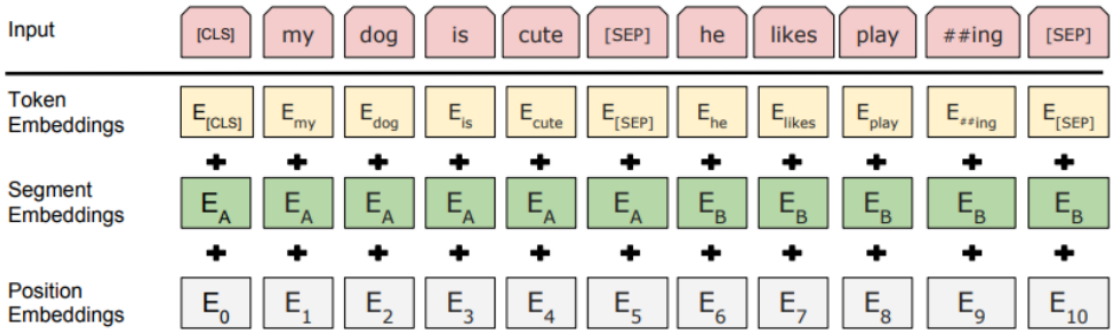

Figure 2.2: BERT text processing. Source [33]

Figure 2.2 shows how text processing happens in BERT to create word and sentence embeddings. Here [CLS] is a special classification token used to identify the last hidden state of BERT corresponding to a token. [SEP] token is used to separate inputs. That is to identify the end of one input and the start of another input.

As shown in Figure 3.4, there are three types of embeddings created with BERT such as token embeddings, segment embeddings, and position embeddings. Specific tokens from WordPiece vocabulary are used to learn token embeddings. Segment embeddings are uniquely learned for sentences to distinguish between them. Position embeddings are a very valuable type of embedding that captures the position of a word in a sentence which has overcome the limitations of Transformers which are unable to capture the order information.

The contextual word embeddings in BERT can be identified by considering the following sentences in Lund et al.[22].

- “I went to the bank last week.”
- “The river bank is magnificent.”

When considering the two sentences, the word “bank” has a different semantic meaning there. In the first sentence, the bank means a financial company and in the second sentence, the bank means the river shore. So if we use the same vector

representation for the word bank, it affects the semantic meaning of the text and will be represented incorrectly in one instance.

2.5.2 XLNet

XLNet is a generalized autoregressive pre-training method that,
1. Learns from left to right and vice versa by considering permutations
2. Addressed the restrictions of BERT with the help of autoregressive formulation

XLNet leverages the techniques of Transformer-XL, autoregressive model as per Yang et al. [28]. XLNet is a pre-trained model like BERT which is an enhancement to BERT. XLNet has addressed the issues in BERT and has outperformed in many NLP tasks. In Auto-Regressive Language Modeling, a context word is used to predict the next word and this is done both forward and backward. An example of an AR Language Model is illustrated in Figure 2.3.

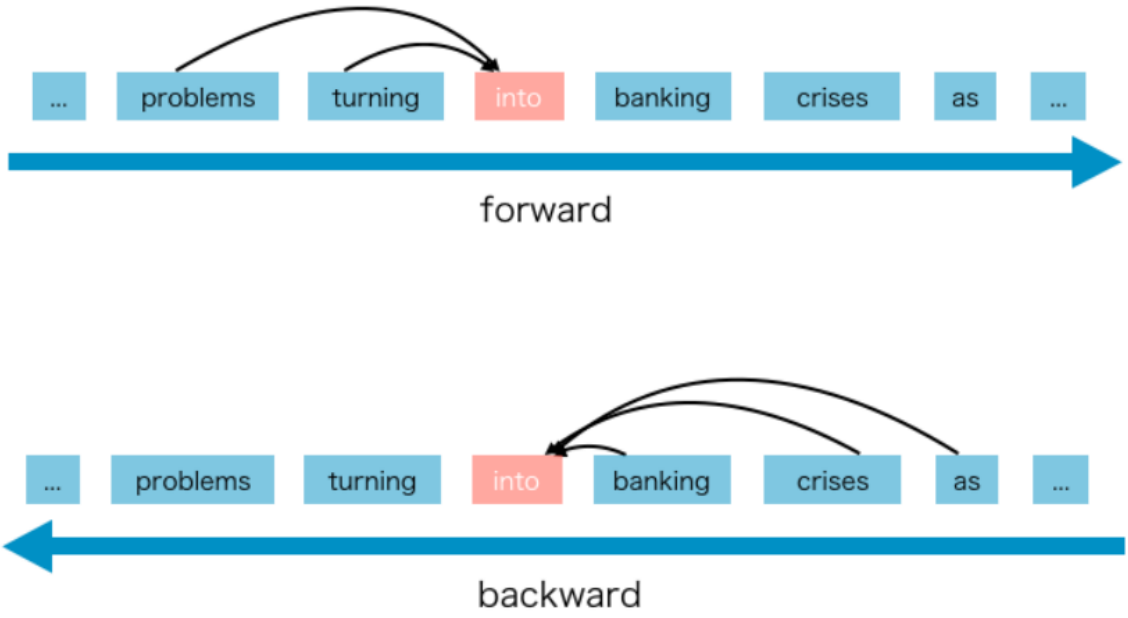

Figure 2.3: Auto Regression Language Model. Source [34]

XLNet has introduced permutation language modeling, where all tokens are predicted but in random order, whereas in BERT MLM only 15% of the masked tokens are predicted. In BERT [MASK] token is used in pre-training, but in fine-tuning that symbol is unavailable in real-world datasets which has the assumption that masked tokens are independent of each other given unmasked tokens. But this is not the case since the masked tokens can be dependent on each other. For instance, let us consider the sentence "I want to take a housing loan from the bank". If "loan" and "bank" words are masked, BERT tries to predict mask tokens independently given the unmasked tokens by ignoring the relationship between the two words. XLNet has overcome this masking issue and outperforms BERT in many NLP tasks.

This in turn implies XLNet is anyway better than other traditional language models which does the token prediction sequentially.

XLNet produces three types of embeddings such as token embeddings, mask word embedding, and segment embeddings. Token embeddings are learned from the tokens in the dataset. Segment embeddings are learned for sentences to distinguish between them. Mask word embeddings are created for masked tokens. In this research XLNet, the base-case pre-trained model is considered to create the embeddings.

Since this is a novel technique, very few researchers in the NLP domain have adapted this technique in their research.

2.5.3 Comparison between BERT and XLNet

XLNet has outperformed BERT in many approaches [27]. To depict the behaviour of XLNet let us take the example [27] "I like cats more than dogs.". Language models like BERT predict the tokens in the sequence "I", "like", "cats", "more", "than", "dogs". But in a permutation language model like XLNet, the order of tokens is a random order like "dogs", "I", "than", "like", "cats", "like" where "dogs" would be conditioned on seeing "than" and so on.

2.5.4 BERT and XLNet based contextual information extraction

BERT has been used for duplicate question detection(a task related to duplicate bug detection) by Ruckle et al. [21] and have gained promising results. There they have proposed two novel approaches to generate duplicate questions and to perform weak supervision on the title and body(WS-TB) of a question. They have shown that both approaches outperform although labeled data is not present. There they have incorporated fine-tuned BERT-base(uncased) models for supervised training, WS-TB on Android, Apple, AskUbuntu, and SuperUser question datasets and have obtained promising results being the most effective method with 0.9pp on average.

2.6 Domain Adaptation

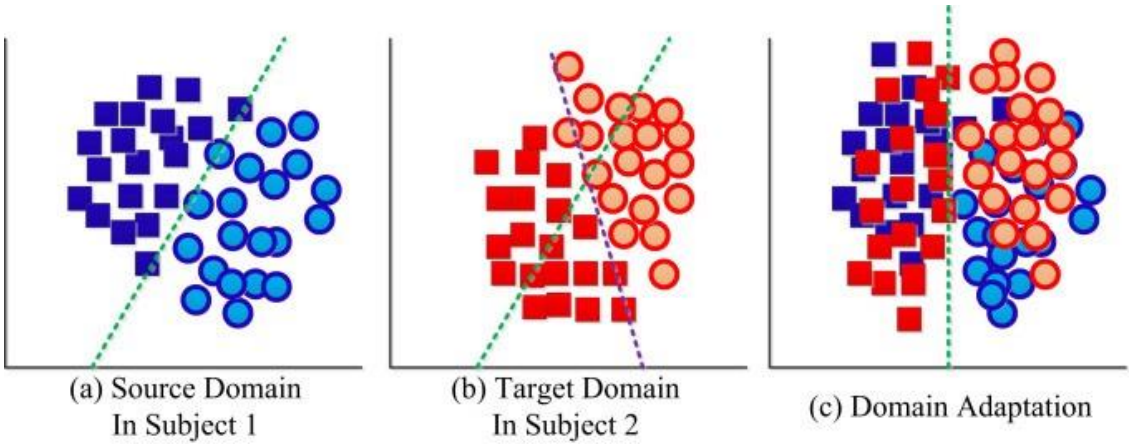

Figure 2.4: Domain Adaptation. Source [42]

Domain adaptation is a technique associated with Machine Learning which is used to train a model on one or more source datasets and use the model with a target dataset in the same domain or context to solve similar or different tasks. When the relatedness of source distribution and target distribution increases, the domain adaptation becomes useful [36]. Multi-stage classification is also a part of domain adaptation where we fine-tune the model with multiple datasets. When data are absent for a particular learning task, we can use a related dataset of the same domain with the same distribution that has labeled data.

Domain adaptation is three types: unsupervised domain adaptation, semi-supervised domain adaptation, and supervised domain adaptation [41]. Unsupervised domain adaptation means training from a labeled source dataset and testing from an unlabelled target dataset. Semi-supervised domain adaptation means the source dataset is large and it has a small subset of labeled data and a large subset of unlabelled data. In supervised domain adaptation, both source and target datasets are labeled datasets.

2.7 Datasets

Lazar et al. [7] have researched to generate duplicate bug report datasets for future research by contributing a dataset of Eclipse, Open Office, NetBeans, and Mozilla [8], [17] by extracting data from open-source systems. This dataset is the well-known msr14 dataset and it has MongoDB dumps that need to be extracted to be used for the research. This dataset has become very popular among researchers and the whole dataset or a part of the dataset is being used as the dataset for most of the research. I leveraged Eclipse, Netbeans, and OpenOffice datasets from the msr14 dataset in this research. The distribution of the whole dataset across all the types and duplicate and non-duplicate classes is shown the Table 2.1.

Table 2.1: Data distribution of msr14 dataset

Dataset	**Duplicate**	**Non Duplicate**
OpenOffice	57340	41751
Eclipse	86385	160917
NetBeans	95066	89988
Combined (Eclipse + NetBeans)	181451	250905

2.8 Evaluation

To evaluate research that incorporates text classification the following metrics can be defined.
The following short forms will be used in representing the equations for the metrics.
TP - True Positive
FP - False Positive
TN - True Negative
FN - False Negative

2.8.1 Accuracy

Accuracy is the ratio of correctly predicted observations to the total observations.
$(TP + TN)/(TP + FP + TN + FN)$

2.8.2 Precision

Precision is the ratio of positives to the total predicted positives.
$TP/(TP + FP)$

2.8.3 Recall (Sensitivity)

Recall is the ratio of positives to all actual positives.
$TP/(TP + FN)$

2.8.4 F1 score

F1 Score is the weighted average of Precision and Recall.
$2{*}Precision * Recall/(Precision + Recall)$

Using these metrics classification models can be evaluated accurately.

3. CHAPTER 3: METHODOLOGY

In this research, duplicate bug report detection will be considered as a classification task to identify a pair of bug reports as duplicates or non-duplicates. The main approach was based on text classification using pre-trained language models and the MLP model. Further, instead of MLP, a CNN model was considered to see if there is room for improvement. Then domain adaptation capabilities of the pre-trained language models were tested against the dataset along with multi-stage text classification.

There were several past research based on the above techniques as mentioned in sections 2.2.2 and 2.5.4 that have shown promising results. But most attempts have been made to improve the methods used for classification or IR. Very little attention has been made in the recent past to integrate novel techniques related to contextual similarity of bug reports. Therefore in this research, I am trying to fill this gap by classifying bug reports as duplicates or not with higher accuracy than the baseline research by considering contextual information capturing related novel techniques such as pre-trained language models. Here in this research, I have used pre-trained BERT and XLNet models as pre-trained language models. They were used to create word/sentence embeddings from the dataset and feed that as an embedding layer to train the MLP Neural Network.

The proposed solution is shown from two viewpoints as a flow chart for the entire system and system architecture as shown below.

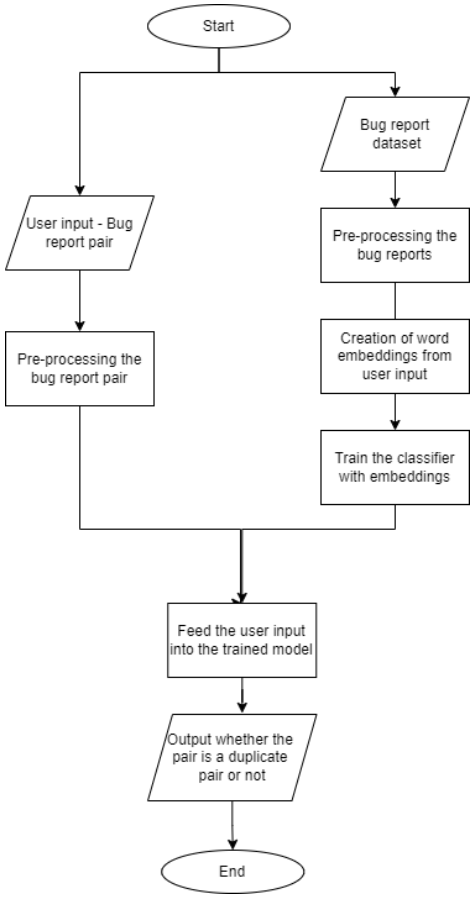

Figure 3.1: Flow Chart for the entire system

Figure 3.1 shows the flow chart of a user reporting a bug and finding the duplicates to the reported bug. Initially, the user input is preprocessed by tokenizing, stop word removal, and lower casing the text. Then same preprocessing techniques are applied to the bug reports from the bug report dataset. Then the user input is paired with the bug reports from the dataset and word/sentence embeddings are created for both the user input and the bug reports from the dataset. In the end, the bug report pairs are fed to the trained model to predict whether the given pair of bug reports are duplicates or non-duplicates.

3.1 Text Classification with BERT and XLNet

The system architectures of the proposed approaches are illustrated in Figure 3.2 and Figure 3.3.

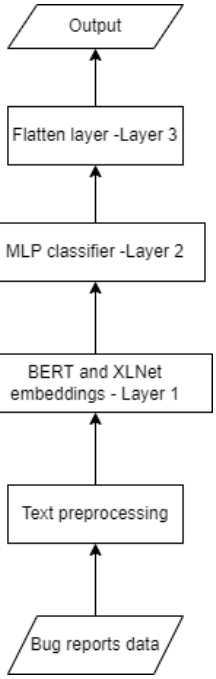

Figure 3.2: Proposed Approach for BERT or XLNet with MLP classifier

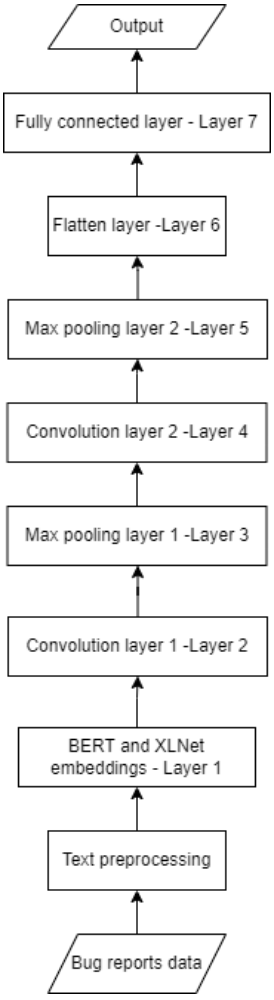

Figure 3.3: Proposed Approach BERT or XLNet with CNN classifier

The steps carried out in the research were feature selection, feature extraction, pre-processing, creation of embeddings, training the classifier, and evaluation. The next section will further elaborate on each component of the architecture in detail.

3.1.1 Feature Selection

In this step, necessary features were selected out of the features in the dataset. The columns in the original dataset were bug_id, product, dup_id, description, short_description, bug_severity, version, component, bug_status, and resolution. In most of the research related to duplicate bug report detection, textual features such as long, and short descriptions and domain-specific features such as product and component were used. So bug_id, product, dup_id, description, short_description, component, and resolution were considered for the research. Among those columns description, short_description, product, and component were selected as features.

3.1.2 Feature Extraction

To make use of the selected features a new feature was formulated to train the model along with labeling the dataset. The feature was created by combining all structured and unstructured features such as description, short_description, product, and component.

The labeling of the dataset was done by considering the resolution status of the bug reports. When the resolution of a bug report is DUPLICATE, that bug report is combined with the corresponding duplicate bug report to form a duplicate pair. If the resolution status is not DUPLICATE that bug report is paired randomly with another bug report with the resolution status not DUPLICATE. Bug report pairing is shown in Figure 3.4.

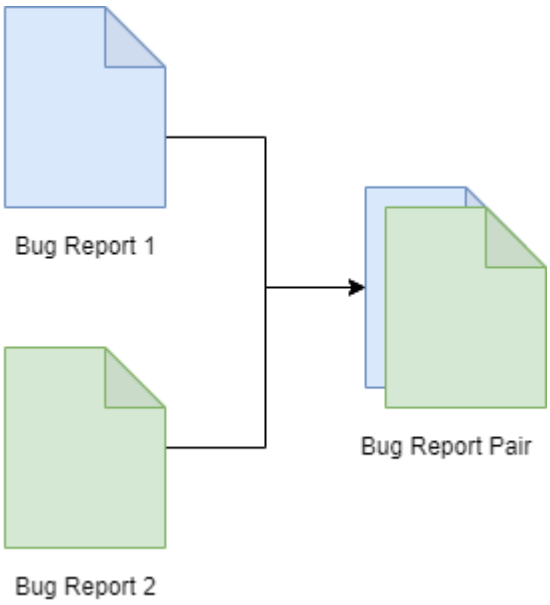

Figure 3.4: Bug report pairing

3.1.3 Preprocessing

The bug reports were preprocessed using preprocessing techniques like parsing tokens, stop word removal, and lower casing the text. When parsing tokens, a tokenizer specific to the language model had to be used. As stop words standard English stop word set provided by NLTK was used.

3.1.4 Classifier Layer

The below sections will elaborate more on the Deep Learning models used in the classifier layer. The first section will describe the MLP model and the next section will describe the CNN model.

3.1.4.1 MLP Model

MLP is a type of feed-forward Artificial Neural Network. It is a basic type of ANN that expects to at least have an input layer, a hidden layer, and an output layer. Except for the input layer, other layers need to have a non-linear activation function. Data is sent to the input layer as any ANN and data propagates through other layers to the output layer which is the visible layer that provides an output. MLP exploits a supervised classification technique called backpropagation and tries to minimize the errors occurring in the training phase by adjusting weights accordingly. MLPs are mainly suitable for supervised classification problems [35].

In this research to create the MLP, the above type of model is created. The model contained an input layer with three inputs for both BERT and XLNet. That is to give the token embeddings, segment embeddings, and position embeddings or mask embeddings as inputs. Then the pre-trained BERT model or XLNet model is used as the next layer. BERT and XLNet models have several built-in layers themselves. Then the output is captured with a sigmoid activation function. The proposed model is depicted in Figure 3.6.

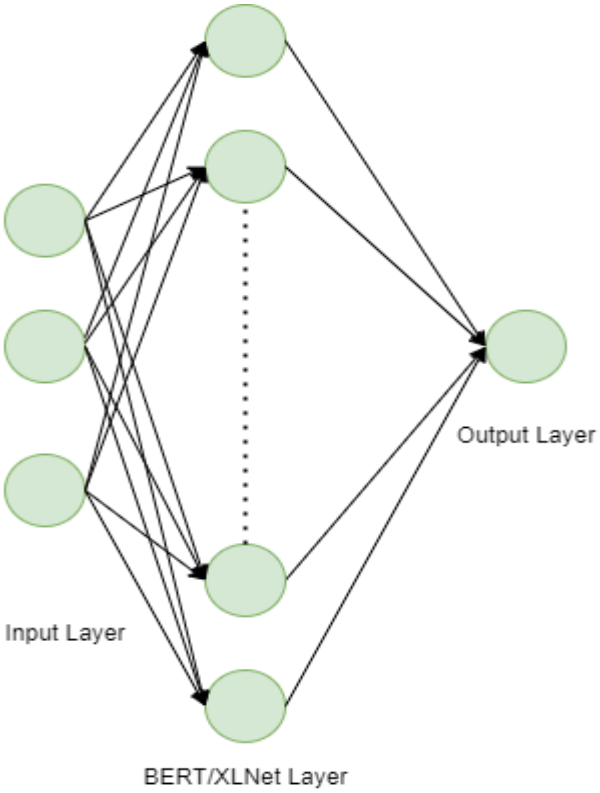

Figure 3.6: Model used in the research

The trained model was evaluated using Stratified fivefold cross-validation to generate the evaluation metrics for further comparison. This will be further discussed in the Evaluation chapter.

3.1.4.2 CNN Model

To see if CNN works better than MLP this classification task was carried out at this stage. Instead of using an MLP classifier, I have used a Convolutional Neural Network(CNN) classifier to train the model. With this change the overall architecture has not changed, instead, the model and the layers of the neural network have changed as shown in Figure 3.7.

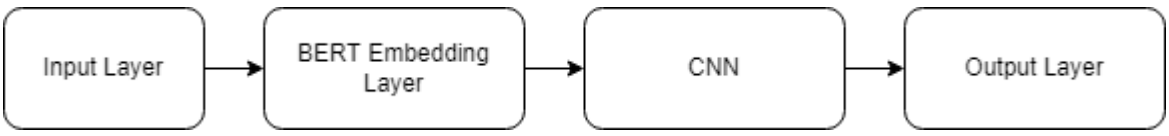

Figure 3.7: Model with CNN

The CNN model that I exploited in the proposed approach is depicted in Figure 3.8.

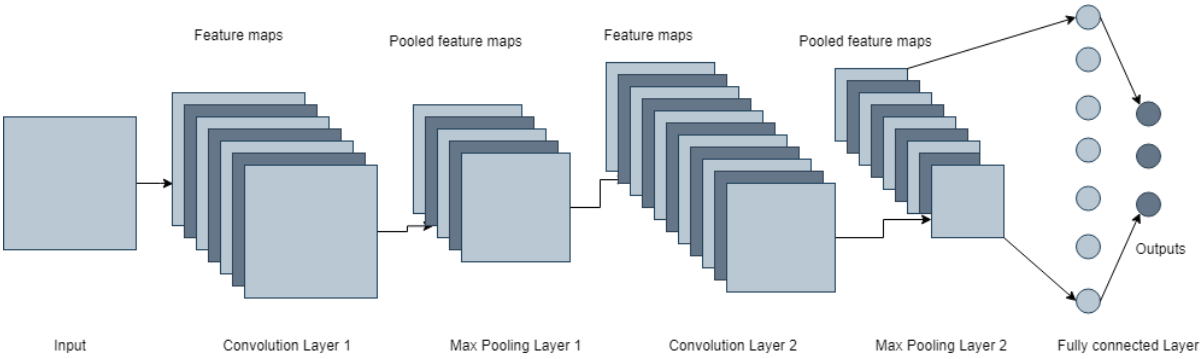

Figure 3.8: CNN Model architecture

The input is fed into a Convolution 1D layer that has 32 filters of size 3x3. Rectified Linear Unit (ReLU) activation function is used in the convolution layer and the 'same' padding technique is used to generate output size same as input size. Then in the max-pooling layer that reduces the size of the vectors. Another Convolution 1D layer and a max-pooling layer are added to the network. Finally, the result is aggregated by the fully connected layer and presented as the output of the model training.

4. CHAPTER 4: Evaluation

4.1 Introduction

This chapter illustrates the bug report dataset, baseline used, hyper parameters, and experimental results for classification and domain adaptation tasks in detail. Then it will discuss the results and possible root causes for them.

Several experiments were carried out to see how the datasets perform and to see what techniques perform well for the datasets. The first classification task was performed on OpenOffice, NetBeans, and Eclipse datasets. This task was based on an MLP classifier along with BERT and XLNet based embeddings that capture the contextual information better from the data than other embedding methods. The results were generated based on stratified fivefold cross-validation.

Then the next experiment was a domain adaptation task based on OpenOffice, Eclipse, and NetBeans datasets. There also the same text sequence length was considered. Here only BERT embeddings were used to generate the model because BERT seems to be best performing in the classification task as shown in Table 4.3. OpenOffice and Eclipse datasets were used to train the model and adapted that model to the NetBeans dataset to see whether domain adaptation works for Eclipse and NetBeans datasets.

Further multi-stage classification tasks were performed on Eclipse and NetBeans to see how it impacts the classification results. Here a CNN model was used instead of an MLP to compare how the change of classifier and use of multi-stage classification changes affects the classification results.

Then few experiments were carried out with a combined dataset of Eclipse and NetBeans and trained on a CNN classifier along with only BERT embeddings to create the model. Then tested the combined model against Eclipse and NetBeans datasets separately to check which method performs the best. The next experiment was carried out to fine-tune the combined model with Eclipse and NetBeans separately and tested against both Eclipse and NetBeans separately.

4.2 Experiment Setup

4.2.1 Bug Report Dataset

The bug report dataset used for experiments was the same as the datasets discussed in section 2.7.

4.3 Baseline

The most recent research based on duplicate bug report detection has been done with classification as the major technique and has shown promising results. In this regard, I considered duplicate bug report detection as a classification problem of classifying a given pair of bug reports as duplicates or non-duplicates. I selected He et al. [9] as my which is the most recent research that has leveraged the msr14 dataset. This has shown really good performance in terms of duplicate bug report classification.
The continuous Bag of Words (CBOW) model with word2vec was used on the corpus to obtain the word vector of each word. The single-channel matrices of bug reports are then paired to form dual-channel matrices which are a novel bug report pair representation they have proposed. By exploiting this novel approach they have gained promising results on the msr14 dataset. The classification accuracy achieved for Open Office, Eclipse, and NetBeans datasets is 0.9429, 0.9685, and 0.9534 respectively.

4.4 Evaluation

The results for the main approach were obtained by adapting the stratified fivefold cross-validation technique to be more precise by reducing generalization error and be in line with the testing strategy followed by the baseline research. Here the dataset is randomly divided into five folds using fivefold cross-validation. In Stratified cross-validation, each fold takes duplicates and non-duplicates in the same proportion as the original dataset. There one fold is kept as the test set and the other four folds are taken as train sets in five-fold cross-validation. This is done five times and each fold becomes a test set in one of the iterations. The results for other approaches were obtained by taking the train-test split ratio to 0.3.

4.5 Hyper Parameters

With regards to the implementation, during the model training, some hyper parameters were optimized while training the model. In that case, text sequence length was tried out with 160 and 220 and 160 seems to be the best fit for text sequence length.

To improve the model training process, some word/sentence embeddings like BERT and XLNet were used. There were different embedding sizes in the pre-trained models of BERT and XLNet that were used in the research. For BERT, the BERT-large-uncased model is used and the embedding size is 1024. For XLNet, the XLNet-base-cased model was used and the embedding size is 768.

In the MLP model, the optimization algorithm used was Adam optimizer, the activation function used in the Dense layer was Sigmoid, five epochs were used as the number of training iterations and the batch size varied among 10, 20, and 30 for different training approaches. In the CNN model apart from the above parameters, ReLU was used as the activation function in convolution layers, the padding strategy was set to the same and the two convolution layers have 32 filters of size 3.

4.6 Text Classification with BERT and XLNet

The task carried out is classifying pairs of bug reports as duplicates or non-duplicates. The experiments were done by considering the baseline set by He et al. [9] as mentioned in section 4.3. Therefore the same msr14 dataset was considered and prepared a dataset compatible with the dataset they used.

The experiment was done by fine-tuning BERT and XLNet with OpenOffice, NetBeans, and Eclipse datasets along with MLP as the classifier. The results obtained for this experiment in terms of accuracy, precision, recall, and F1-score are illustrated in Table 4.1.

Table 4.1: Classification results of our approach and baseline

Dataset	**Approach**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
OpenOffice	Baseline	94.29	93.65	96.70	95.15
	BERT	**98.74**	**99.23**	**98.22**	**98.71**
	XLNet	95.23	93.76	96.99	95.32
NetBeans	Baseline	95.34	95.76	95.13	95.44
	BERT	96.77	94.83	98.95	96.84
	XLNet	95.67	93.12	98.67	95.80
Eclipse	Baseline	96.85	96.80	94.09	95.43
	BERT	96.91	96.00	94.74	95.33
	XLNet	96.54	92.16	97.97	94.97

As shown in Table 4.1, BERT has outperformed in all the three datasets in this experiment. XLNet has shown better results for only OpenOffice and NetBeans datasets, but for the Eclipse dataset, it has slightly less accuracy. The reason for slightly less accuracy may be the use of text sequence length to create embeddings. The average text length of Eclipse is closer to 220 and we have considered the text sequence length as 160. We have used the BERT-large-uncased BERT model with 1024 as the embedding size, but the XLNet-base-cased XLNet model was used with

768 as the embedding size. The reason for BERT to outperform XLNet may be that BERT has used a favourable embedding size for the datasets.

Since BERT has performed the best in the above experiment, several more experiments were carried out by fine-tuning the BERT model using Eclipse and NetBeans datasets along with CNN as the machine learning model. In the first experiment, a combined dataset was created using Eclipse and NetBeans datasets to train the model. The trained model on the combined dataset was again tested with Eclipse and NetBeans separately. The dataset sizes of Eclipse and NetBeans datasets are similar to those that were used in the above text classification experiments and the dataset size of the combined model is an aggregation of individual sizes.

Here the evaluation was done by taking the train-test split ratio as 0.3 and reporting the results. The baseline considered was the first approach. The results are listed in Table 4.2.

Table 4.2: Results of Classification with CNN approach

Test Dataset	**Approach**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Eclipse	Baseline	96.91	96.00	94.74	95.33
	BERT	96.30	96.11	92.58	94.32
NetBeans	Baseline	96.77	94.83	98.95	96.84
	BERT	94.66	95.77	93.52	94.63

Here both datasets have shown slightly less accuracy than the accuracy obtained in the baseline. The reason behind these may be because the filter size or number of convolution, and pooling layers used in the CNN model are not the best values for the above scenario. Also, the dataset sizes used may not be the best.

4.7 Domain Adaptation Results

In this research, domain adaptation was done by fine-tuning the BERT model with the Eclipse dataset and OpenOffice dataset separately as the source dataset and Netbeans dataset as the target dataset. Here in the first experiment MLP classifier was used to train the models based on Eclipse and OpenOffice. The overall results are illustrated in Table 4.3 in terms of accuracy, precision, recall, and F1-score.

Table 4.3: Domain adaptation results of our approach

Trained Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
OpenOffice	76.54	99.03	53.72	69.65
Eclipse	77.3	97.11	56.29	71.27

This experiment has shown that Eclipse, NetBeans pair shows better accuracy. The reason behind this is Eclipse, NetBeans pair is from the same domain or their domains are related to each other. But for OpenOffice, NetBeans pair, domains are not related to each other.

Since the Eclipse and NetBeans pair has shown higher accuracy, the next experiments were carried out using those datasets. As the next experiment of domain adaptation, multi-stage text classification was adapted. In this research, I used Eclipse and NetBeans datasets to perform multi-stage training. To carry out this task I also have used a CNN classifier to train the model to find if there's room to improve accuracy.

A model was fine-tuned with BERT and a combined dataset of Eclipse and NetBeans along with a CNN classifier. The trained combined model was again fine-tuned with both Eclipse and NetBeans separately and tested on the same dataset. The evaluation was done by taking the train-test split ratio as 0.3.

The second multi-stage text classification experiment was also done on Eclipse and NetBeans datasets. First, the model was fine-tuned on the Eclipse dataset and then again fine-tuned on the NetBeans dataset and tested on both datasets separately. The baseline considered was the first approach. Table 4.4 depicts the results for the evaluation based on accuracy, precision, recall, and f1-score.

Table 4.4: Multi-Stage Classification results on the combined model, Eclipse and NetBeans approach

Test Dataset	**Approach**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-score (%)**
Eclipse	Baseline	96.91	96.00	94.74	95.33
	BERT- combined dataset	97.19	96.43	95.10	95.76
	BERT	90.99	92.47	79.41	85.45
NetBeans	Baseline	96.77	94.83	98.95	96.84
	BERT- combined dataset	95.36	93.77	97.22	95.47
	BERT	95.11	93.27	97.27	95.23

The model developed with the combined dataset has shown higher accuracy than the other approach for the Eclipse dataset. The reason for this better performance may be the usage of a fairly larger dataset to train the model on multiple stages and the hyper parameters have precisely matched the scenario. Another reason may be the model has fit the datasets well, since the model is trained on the same dataset twice in this case.

A comparison between the different approaches and how the accuracies of different models vary across the approaches are depicted in Figure 4.1.

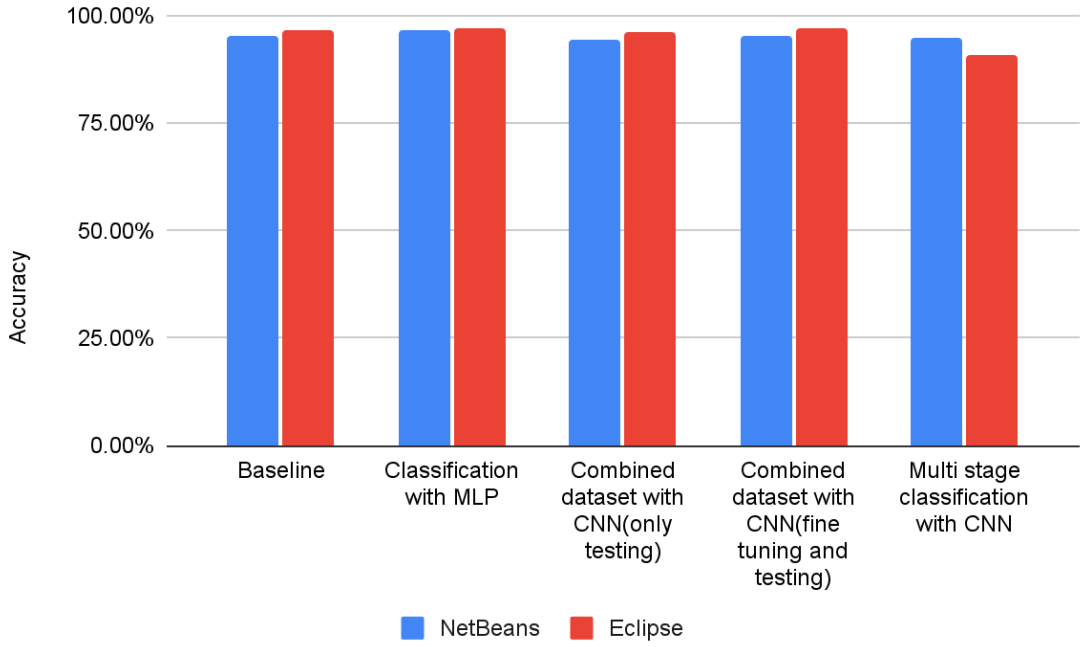

Figure 4.1: Accuracy obtained for different approaches

A comparison of different approaches on NetBeans and Eclipse datasets is shown in Figure 4.2 which is more comprehensive.

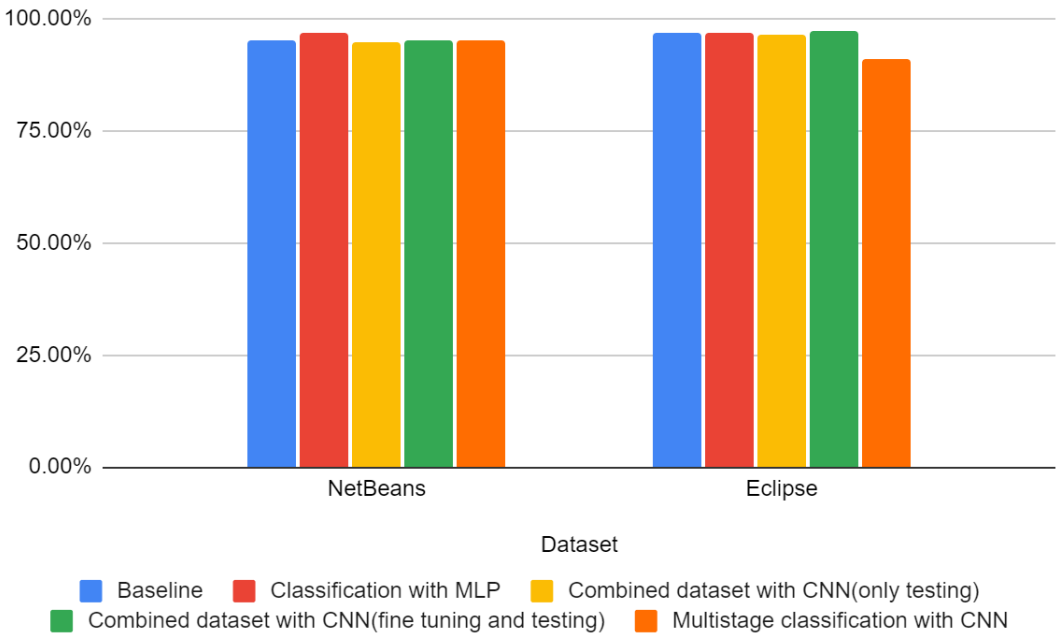

Figure 4.2: Accuracy obtained for NetBeans and Eclipse datasets on different approach

5. CHAPTER 5: Conclusion

No research has adapted pre-trained language models for duplicate bug detection. Therefore this research was carried out to utilize such novel techniques in duplicate bug report detection to improve the results.

The results have revealed that the classification accuracy has been improved for duplicate bug report detection which used the BERT model for fine-tuning along with MLP compared to the baseline [9] research that used the same msr14 dataset. XLNet has shown better results only for OpenOffice and NetBeans datasets. This implies that BERT has outperformed XLNet and baseline approaches.

According to the domain adaptation results, it has shown that a BERT model trained on the Eclipse dataset has shown better performance on the NetBeans dataset than the model trained on the OpenOffice dataset implying domain adaptation is successful in the duplicate bug report detection when similar domains are considered.

Results have shown that when using BERT embeddings for fine-tuning Eclipse and NetBeans datasets, the accuracy is slightly less than the accuracy obtained in the baseline research. Multistage classification on Eclipse and NetBeans datasets also has shown less accuracy than the baseline approach. This also implies for CNN and BERT combination fairly a large dataset needs to be used for fine-tuning and also needs to consider changing hyper parameters of CNN and using different language models.

When multistage classification is performed on a combined model and fine-tuned again with Eclipse and NetBeans separately based on a CNN classifier and BERT model, it shows better results in both datasets than the baseline research. This also supports the idea that domain adaptation works well with Eclipse and NetBeans datasets implying datasets in related domains adapt well.

For future improvements, we can incorporate approaches like pre-training BERT and XLNet with one or more of the datasets and fine-tune with another dataset that is domain related and not related and see how the models behave and the classification results. Here we need to use the whole dataset to pre-trained BERT and XLNet. Since we have only considered a part of the dataset, in the future we can experiment on the whole dataset and see how it performs. Another important aspect is we can consider more options like RoBERTa which also captures contextual information, rather than only considering BERT and XLNet.

6. REFERENCES

[1] Synopsys, “How much do bugs cost to fix during each phase of the SDLC?”, 2017.

[2] Q. Xie, Z. Wen, J. Zhu, C. Gao, Z. Zheng, “Detecting Duplicate Bug Reports with Convolutional Neural Networks” in 25th Asia-Pacific Software Engineering Conference (APSEC), 2018.

[3] J. Anvik, L. Hiew, and G. C. Murph iy, “Coping with an open bug repository,” in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange. ACM, 2005, pp. 35–39.

[4] L. Hiew. “Assisted detection of duplicate bug reports”. Ph.D. Dissertation, University of British Columbia, 2006.

[5] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, “A discriminative model approach for accurate duplicate bug report retrieval” in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 2011.

[6] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, “Towards more accurate retrieval of duplicate bug reports” in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011, pp. 253–262.

[7] A. Lazar, S. Ritchey, and B. Sharif, “Generating duplicate bug datasets” in MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 392-395.

[8] A. Lazar, “Duplicate Bug Datasets”, 2014.

[9] J. He, L. Su, M. Yan, X. Xia, Y. Lei, “Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks” in 28th IEEE/ACM International Conference on Program Comprehension (ICPC 2020), 2020.

[10] L. Poddar, L. Neves, W. Brendel, L. Marujo, S. Tulyakov, P. Karuturi, “Train One Get One Free: Partially Supervised Neural Network for Bug Report Duplicate Detection and Clustering” in Proceedings of NAACL-HLT 2019, pp 157–165.

[11] D. Cubranic, G. Murphy, “Automatic bug triage using text categorization”, 2004.

[12] A. Alipour, A. Hindle, E. Stroulia, “A contextual approach towards more accurate duplicate bug report detection” in 10th Working Conference on Mining Software Repositories (MSR), 2013.

[13] A. Lazar, S. Ritchey, B. Sharif, “Improving the accuracy of duplicate bug report detection using textual similarity measures” in MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014.

[14] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, N. Dubash, “Towards Accurate Duplicate Bug Retrieval Using Deep Learning Techniques” in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017.

[15] A. Budhiraja, K. Dutta, M. Shrivastava, R. Reddy, “Towards Word Embeddings for Improved Duplicate Bug Report Retrieval in So ware Repositories” in Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval, 2018.

[16] O. Chaparro, J. M. Florez, U. Singh, A. Marcus, “Reformulating Queries for Duplicate Bug Report Detection” in 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019.

[17] A. Lazar, S. Ritchey, B. Sharif, “Improving the Accuracy of Duplicate Bug Report Detection Using Textual Similarity Measures”, 2014.

[18] X. Yang, D. Lo, X. Xia, L. Bao, J. Sun, “Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports” in 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016.

[19] B. S. Neysiani, S. M. Babamir, "New Methodology for Contextual Features Usage in Duplicate Bug Reports Detection: Dimension Expansion based on Manhattan Distance Similarity of Topics" in 2019 5th International Conference on Web Research (ICWR), 2019.

[20] J. K. Rani, "Software Engineering Domain Knowledge to Identify Duplicate Bug Reports" in International Journal of Computer Engineering In Research Trends, 2015, Volume 3, Issue 11, November 2016, pp. 585-588.

[21] A. Ruckle, N. S. Moosavi, I. Gurevych, “Neural Duplicate Question Detection without Labeled Training Data” in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, 2019, pp 1607–1617.

[22] M. Lund, “Duplicate Detection and Text Classification on Simplified Technical English”, 2019.

[23]Github, “google-search/bert”, 2018.

[24] Medium, “Sentiment Classification with BERT Embeddings”, 2020.

[25] NLP Progress, “Tracking Progress in Natural Language Processing”.

[26] T. Mikolov, K. Chen, G. Corrado, J. Dean. “Efficient estimation of word representations in vector space”. In: arXiv preprint arXiv:1301.3781 (2013).

[27] Machine Learning Explained, “Paper Dissected: “XLNet: Generalized Autoregressive Pretraining for Language Understanding” Explained”, 2019.

[28] Z. Yang, Z. Dai, Y Yang, J. Carbonell, R. Salakhutdinov, Q.V. Le, “XLNet: Generalized Autoregressive Pre Training for Language Understanding” in 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), 2020.

[29] H. Gregory, S. Li, P. Mohammadi, N. Tarn, R. Draelos, C. Rudin, “A Transformer Approach to Contextual Sarcasm Detection in Twitter” in Proceedings of the Second Workshop on Figurative Language Processing, pp 270–275, 2020.

[30] N. Kim, S. Feng, C. Gunasekara, L. A. Lastras, “Implicit Discourse Relation Classification: We Need to Talk about Evaluation” in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp 5404–5414, 2020.

[31] W. Siblini, C. Pasqual, A. Lavielle, C. Cauchois, “Multilingual Question Answering from Formatted Text applied to Conversational Agents”, 2019.

[32] J. Devlin, M. Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018.

[33] Analytics Vidya, “Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework”, 2019.

[34] Medium, “What is XLNet and why it outperforms BERT”, 2019.

[35] Machine Learning Mastery, “When to Use MLP, CNN, and RNN Neural Networks”, 2018.

[36] Towards Data Science, “Deep Domain Adaptation In Computer Vision”, 2019.

[37] KDnuggets, “BERT: State of the Art NLP Model, Explained”, 2018.

[38] H.Rocha, M.T.Valente, H.Marques-Neto, G.C.Murphy, “An Empirical Study on Recommendations of Similar Bugs”, IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016.

[39] H.Isotani, H.Washizaki, Y.Fukazawa, T.Nomoto, S.Ouji, S.Saito, “Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning”, IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021.

[40] T.M.Rocha, A.L.D.C.Carvalho, “Siamese QAT: A Semantic Context-based Duplicate Bug Report Detection Using Replicated Cluster Information”, in IEEE Access, 2021, Volume 9, March 2021.

[41] Towards Data Science, “Understanding Domain Adaptation”, 2021.
[42] X. Chai, Q. Wang, Y. Zhao, X. Liu, O. Bai, Y. Lia, “Unsupervised domain adaptation techniques based on auto-encoder for non-stationary EEG-based emotion recognition”, 2016.