جـــامـعـــة الـــشـرق الأوسـط
MIDDLE EAST UNIVERSITY
Amman - Jordan

Detecting Bug Severity Level using Machine Learning Techniques

تعيين مستوى الخطورة للأخطاء باستخدام تقنيات التعلم الآلي

Prepared by:

Hamza AL-Jundi

Supervised by:

Dr. Sherifa Murad

A Thesis Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Science

Department of Computer Science

Faculty of Information Technology

Middle East University

Jan. 2021

Authorization

I, Hamza AL-Jundi, authorize Middle East University to supply copies of my thesis to libraries, establishments, or individuals on request, according to the University's regulations.

Name: Hamza AL-Jundi

Date: 28 / 01 / 2021.

Signature:

Committee Decision

This is to certify that the thesis titled (**Detecting Bug Severity Level using Machine Learning Techniques**) was successfully defended and approved on: / / 2021.

Examination Members	Signature
Dr. Sharefa Murad,(Supervisor)	....
Dr. Abdel-Rahman Abuarqoub,(Member)	....
Dr. Bassam Al-Shargabi,(Member)	....
Dr. Abdel-rahman Falah Al-Ghuwairi,(Member)	....

Acknowledgements

The completion of this thesis would not have been possible without the support and encouragement of several special people. Hence, I would like to take this opportunity to show my gratitude to those who have assisted me.

I would first like to express my heartfelt thanks to my supervisor Dr. Sherifa Murad. A more supportive and considerate supervisor I could not have asked for. There was many times where I had reached the 'crossroads' and each time Dr. Sherifa was there to steer me towards the right path. Her willingness to offer me so much of her time and intellect is the major reason this thesis was completed.

Finally, I would like to thank my family to whom I owe a great deal. To my late Grandfather Mohammad, thank you for showing me that the key to life is enjoyment. To my father Jamal, who makes the highest sacrifice and dedication for us thank you for your great support and effort, and finally, the one person who has made this all possible has been my mother Wafa'a. She has been a constant source of support and encouragement and has made an untold number of sacrifices for the entire family, and specifically for me to continue my schooling. She is a great inspiration to me. Hence, great appreciation and enormous thanks are due to her, for without her understanding, I am sure this thesis would never have completed. I thank you all.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CNN	Condensed Nearest Neighbour
CSV	Comma-Separated Values
HTTP	Hypertext Transfer Protocol
ID	Identifier
IG	Information Gain
ITIL	Information Technology Infrastructure Library
LSI	Latent Semantic Indexing
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perception
MLR	Multinomial Logistic Regression
MMLR	Multi-Nomial Multivariate Logistic Regression
NB	Naïve Bayes
PITS	Project And Issue Tracking System
QA	Quality Assurance
RF	Random Forest
RNG	Relative Neighbour Graph
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SRcut	Size Regularized Cut
STC	Suffix Tree Clustering
SVM	Support Vector Machine

Detection Bug Severity Level using Machine Learning Techniques

Prepared by: Hamza AL-Jundi

Supervised by: Dr. Sherifa Murad

Abstract

Software maintenance is the process of modifying a component or system after delivery, in order to correct defects, improve quality characteristics, or adapt to a changing environment (ISTQB, 2019). To reduce maintenance cost the quality assurance engineers ensure that the software meets the requirements of the software owner and the user perspective by applying some testing techniques, such as usability testing, and performance testing.

When the testing team finds a bug, the bug reported to the development team, and after the bug is resolved, the testing team should re-test the reported bugs. This process will repeat each time the quality assurance team members find any bug. Bugs report should contain all the needed information to the developers, such as the steps to reproduce the bug, the bug priority and severity, and a brief description of it.

The most common point that makes software quality tester and developers' life harder is the limitation of time and human resources, which may lead him/her to discard some of the reported bugs, to take care of bugs that are more critical. This study aims to overcome the mentioned problems, by automating the whole process of assigning the severity level on newly reported bugs to replace the manual severity assigning.

This thesis focuses on the detection of bugs severity (sever or non-sever), using machine learning approach, the features of the bugs report will be cleaned using text mining techniques such as (tokenization, stemming), and then a comparison between (LSTM and RNN) to evaluate which technique is giving the best result in assigning bugs severity.

The implementation divided into four main phases, in the first phase, the data set will extracted, then in the second step, dataset pre-processing will be done, the third phase is feature selection and in the last phase, the framework will propose a prediction and it called the prediction phase. The bug reports dataset extracted from the repository of JIRA related to closed-source projects developed by TETCO Company located in Riyadh, Saudi Arabia; the datasets mainly contain four features including project name, bug id, bug description, and the severity level of the bug. After model training, the different evaluation measures used for evaluating model performance. According to the experimental results, we achieved a better result using the LSTM neural network instead RNN.

Keywords: Bug Severity, Long Short-Term Memory, Recurrent Neural Network, Neural Network.

تعيين مستوى الخطورة للأخطاء باستخدام تقنيات التعلم الآلي

إعداد: حمزة الجندي.

إشراف الدكتورة: شريفة مراد.

الملخص

يعرف خطأ البرنامج بأنه مجموعة المشاكل التي تحدث خلال مراحل بناء المشروع والتي تؤدي إلى نتيجة غير صحيحة أو غير متوقعة. في عملية اختبار البرمجيات، تعد المرحلة الرئيسية هي التنبؤ بخطورة تقارير الأخطاء. ومع ذلك، يحتاج تصنيف تقارير الأخطاء يدويًا إلى وقت وموارد من ذوي الخبرة. مما يؤدي الى تأخير إصلاح الأخطاء ذات الأولوية العالية.

في هذه الاطروحة، تم إقتراح إطاراً لتعيين مستوى الخطورة المناسب لتقارير الأخطاء، بإسناد قيمة لخطورة تقرير الخطأ، والهدف من هذا الإطار هو تجنب استغراق الوقت المستهلك أثناء تعيين خطورة الاخطاء بشكل يدوي بالاضافة الى تحسين الدقة والفعالية في التنبؤ خطورة تقارير الأخطاء.

تم التحقق من فعالية هذا الإطار وصحته بتجربته على مجموعات بيانات مستخرجة من JIRA باستخدام لوحة معلومات شركة TETCO وهو مشروع مغلق المصدر لم يتم إستخدامه في أبحاث سابقة، ويحتوي على 2355 تقرير خطأ، للحصول على أداء أفضل وتحقيق دقة أعلى.تم إجراء التجارب على مجموعة البيانات الحقيقية من خلال التعلم العميق بإستخدام خوارزميتين وهما: الذاكرة العصبية طويلة المدى (LSTM)، و (RNN).

تشير نتائج تجربتنا الى أن إطار العمل الخاص بتعيين مستوى الشدة المناسب لتقارير الأخطاء والذي يستند الى التعلم العميق، بأنه يتنبأ بخطورة تقارير الأخطاء بدقة مرتفعة، حيث أظهرت النتائج نسبة التنبؤ بمستوى الخطورة إستناداً الى LSTM تصل الى: 0.858، أما نسبة التنبؤ بمستوى الخطورة إستناداً الى RNN تصل الى: 0.58، مما يعني أن خوارزمية LSTM كانت الأكثر دقة في التنبؤ بمستوى الخطورة المناسب لتقارير الأخطاء مقارنة بخوارزمية RNN.

الكلمات المفتاحية: خطورة الأخطاء، الذاكرة العصبية طويلة المدى، الشبكة العصبية المتكررة، الشبكة العصبية.

Chapter One

Introduction

1.1 Research Context

This thesis focuses on the detection of bugs severity (severe or non-sever). Using machine learning approach, the features of the bugs report will be cleaned using text mining techniques such as tokenization and stemming, and then the comparison between long-term memory and recurrent neural network to evaluate the technique and determine which gives the best result in assigning bugs severity.

1.2 Background

In today's world, a quick way to predict the severity of bug reports was necessary in order to fix these bugs quickly.

The idea of the proposed framework appeared due to the large increase in the submitted bug reports with limited resources, whether it was human resources such as developers, or the time consumed in determining priority.

Therefore, there was a great need to suggest an important framework in order to focus on high severity bug reports and resolve them quickly.

The framework is becoming increasingly important, as a unique dataset used to build this framework, extracted from closed source projects TETCO containing more than 2355 bug reports not used in previous research, provided by the JIRA dashboard.

1.3 Problem Statement

In today's agile world is very important to deliver the software in less time without affecting the quality of software. It is the job of bug trigger to classify the bugs based on criticality. In a personal communication, Mozilla trigger highlighted that “Every day, almost 300 bugs appear that need triaging.

This study aims to automate the bug severity detection process to replace the manual severity assigning.

1.4 Research Aims and Objectives

This thesis aims to create an efficient system that can detect the bug severity in software. To achieve this aim different set objectives listed below:

- Automate the bug severity detection process to replace the manual severity assigning.
- Exploring data pre-processing technique and choosing the best technique of pre-processing of the dataset.
- Training and testing model on training and validation dataset.
- Overcome the limitation of existing methodologies.
- Comparing the performance of both RNN and LSTM.

1.5 Research Questions

The problem in this thesis can filtered in the following questions:

- What is the performance of the proposed framework?
- What will be the accuracy of the proposed framework?

1.6 Research Methodology

In this thesis, the general study methodology used is the positivist approach. This methodology uses hypotheses and empirical finding (Iivari, Hirschheim, & Klein, 1998), so it would be appropriate for the thesis. The method used in this thesis includes several steps, including:

- Define the problem.

- Formulate a hypothesis.
- Prepare the experiment and produce the details.
- Collect the information and perform pre-processing.
- Evaluate the model.
- Interpret the conclusions from the model.

There are four phases of the adopted research methodology in this thesis as shown on in Figure 1.1 include the literature review, the literature analysis, design and modelling and performance evaluation.

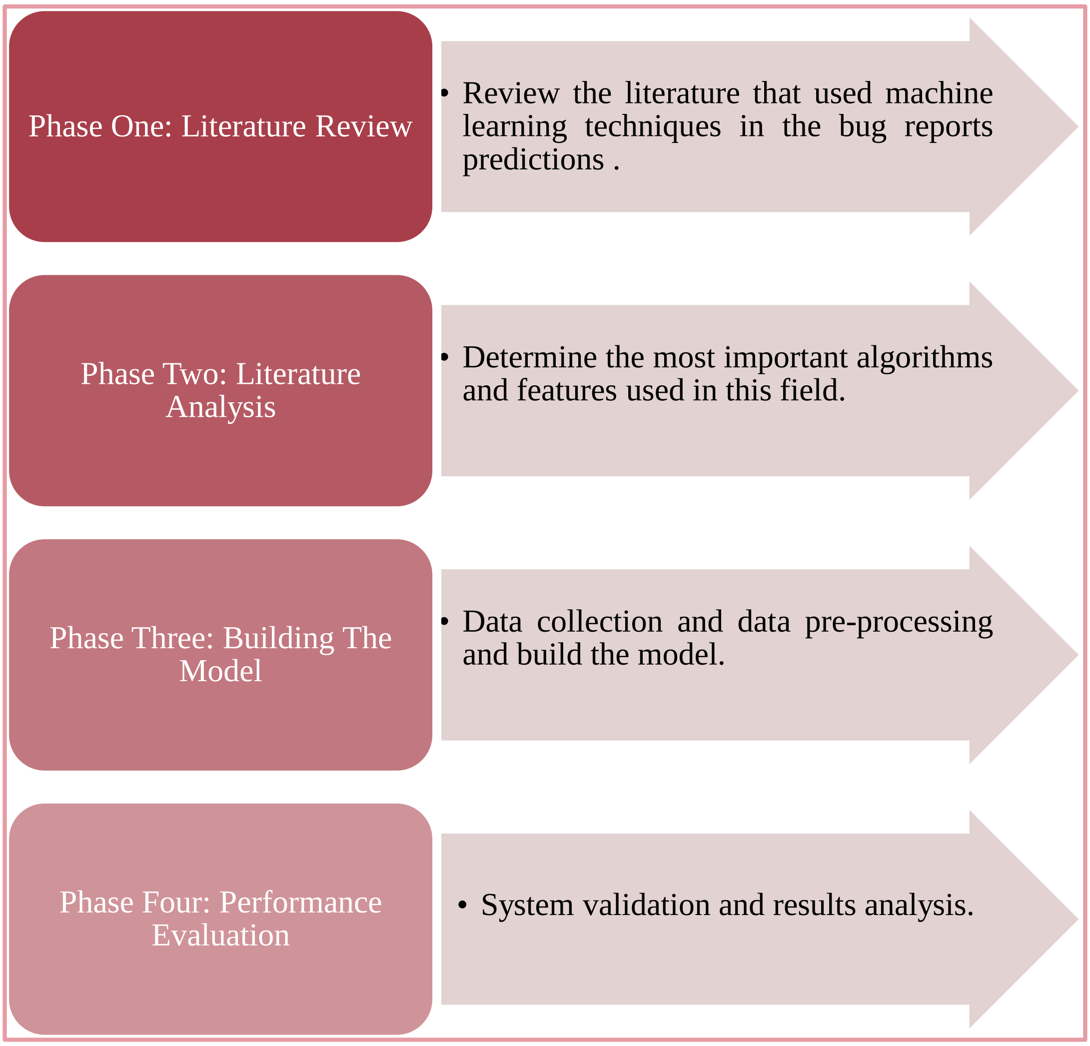


Figure 1.1: The Research Methodology.

1.7 Thesis Organization

The rest of this thesis organized in the following structure:

- Chapter Two: Reviews the theoretical background and related work of relevant research papers in the bug reports and explains the importance of predicting bug report severity. In addition, this chapter introduces the most common machine learning methods that can used to predict bug report severity.
- Chapter Three: Provides a detailed explanation of the prediction process of the bug report, and it provides an overview of the dataset used in this study and its source, in addition to a detailed presentation of the proposed framework steps that were used in this thesis.
- Chapter Four: Explains the most important results of this thesis and compares the results of the algorithms used. In addition, this chapter presents a comparison between the study that conducted in this thesis and a previous study.
- Chapter Five: Presents the conclusion and future works.

Chapter Two

Background and Related work

2.1 Overview

This chapter discusses the theoretical background for bug severity, including the definition of the bug, its life cycle, and the difference between the severity and priority of the bug. This chapter provides an overview of machine learning, in addition to giving a brief overview of many of the current research related to machine learning applied to predicting and assigning the bug severity.

2.2 What is Bug?

According to ISTQB, the bug is an imperfection or deficiency in a work product where it does not meet its requirements or specifications (ISTQB, 2019).

The Information Technology Infrastructure Library, defined the bug as: event that is not part of the standard operation of service and causes an unplanned interruption or decrease the quality of service (Dabade, 2012).

2.3 Bug Severity and Priority

Users through issue tracking systems often submit bug reports. The bug report can describe the particular case when a software bug occurs and the bug report includes bog regeneration information, a bug report contains several attributes: the bug-id, submission date, the status, the priority, the severity, the summary, and the description.

The severity is how austere a bug is, it's an important attribute of a bug report that decides how quickly it should be resolved (Kukkar, Mohana, & Kumar, 2020), also it can be used to indicate whether a bug is an enhancement request. Bug severity terms can be expressed in different ways depending on the bug tracking system that used, as follows (Bibyan, Anand, & Jaiswal, 2020):

- Bug Severity in Bugzilla indicates how severe the problem is, it can vary from trivial, minor, normal, major, and critical to blocker, the blocker means the application unusable. Priority, however, determines the urgency for repairing a bug. In Bugzilla, the combination of priority and severity defines the importance of a Bug (Bugzilla, 2021).
- Bug Severity in JIRA is referred to as a priority, it indicates how important the bug, it can vary from the highest priority which is a blocker to the lowest priority which is minor in relation to other bugs.
- Bug Severity in Google Issue Tracker is referred to as priority, which indicates how priority the bug is, and it can vary from P0 which means the bug needs to be addressed immediately to P4 which means the bug fixing can be postponed in relation to other bugs(Pandey, Hudait, Sanyal, & Sen, 2018).

Bug Severity in JIRA (JIRA, 2020) can be divided into five levels including **Blocker, Critical, Major, Minor,** and **Low**, as shown in figure 2.1.Each of these levels will be defined in detail as follows:

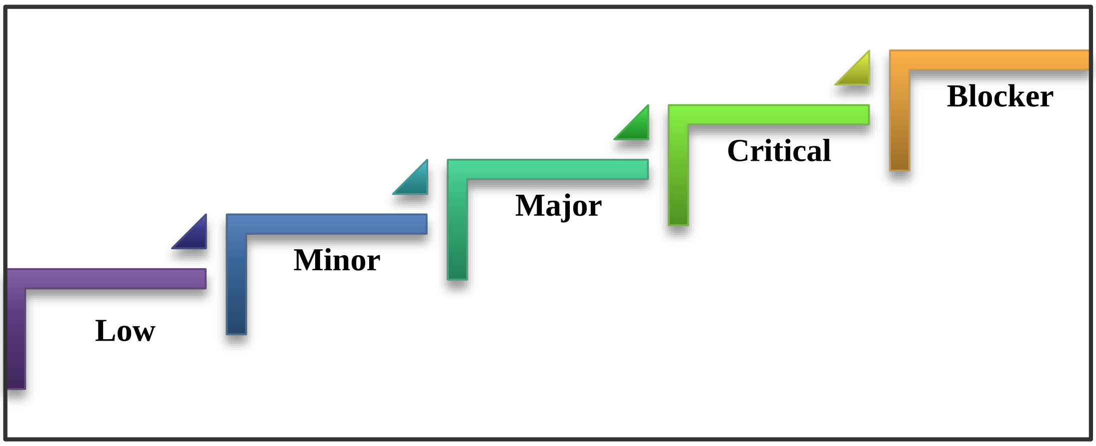


Figure 2.1: Severity Levels.

- Blocker: The bug currently makes the system or functionality unavailable.
- Critical: The bug affects sensitive or critical data and there is no way to avoid it.
- Major: The bug has a big impact on features or main data and solutions are available, but it is not clear or hard to implement.
- Minor: This bug affects minor or non-critical data and a reasonable solution is available.
- Low: The bug does not affect functions or data, nor does it affect performance or efficiency. It is only inconvenience and does not require any solution.

The priority in the bug report is how quick a system bug is. It demonstrates the urgency of handling and deleting a bug. It really is a test of the way that the bug is priority in the debugging hierarchy. Bug goals are appropriately allocated to scheduling a software development life cycle (Bibyan et al., 2020).

The priority can be divided into four levels including **Immediate, High, Medium, and Low** (JIRA, 2020), as shown in figure 2.2.

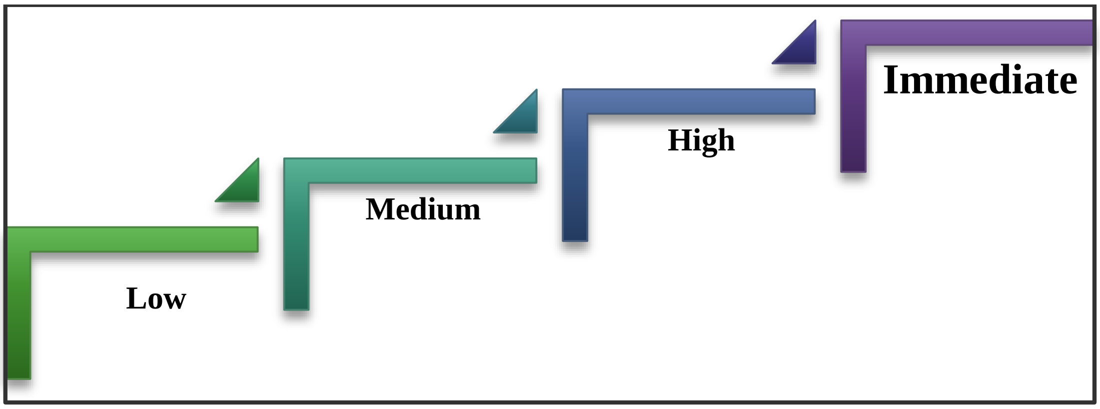


Figure 2.2: Priority Levels.

- Immediate: a bug that is of the highest priority and should fixed as soon as possible.
- High: the best bug fixed when the next build cycle occurs.

- Medium: this type of bug takes precedence over low-priority bug. It should fixed but it can placed on the next iterations or release cycle if necessary. If necessary.
- Low: fixed bugs are the lowest priority after all of the high and medium-priority bugs are fixed.

2.4 Bug Severity VS Priority

Bug severity and the bug priority in software testing are two widely used terms; usually they are synonymously use, which is wrong. The severity is related to standards and functionality of the system; whereas, the priority is related to scheduling so the severity of a bug is determined by quality analyst, test engineer; whereas, a priority of a bug is determined by the product manager or client (Ramay, Umer, Yin, Zhu, & Illahi, 2019).The difference between the two terms is shown in the following figure:

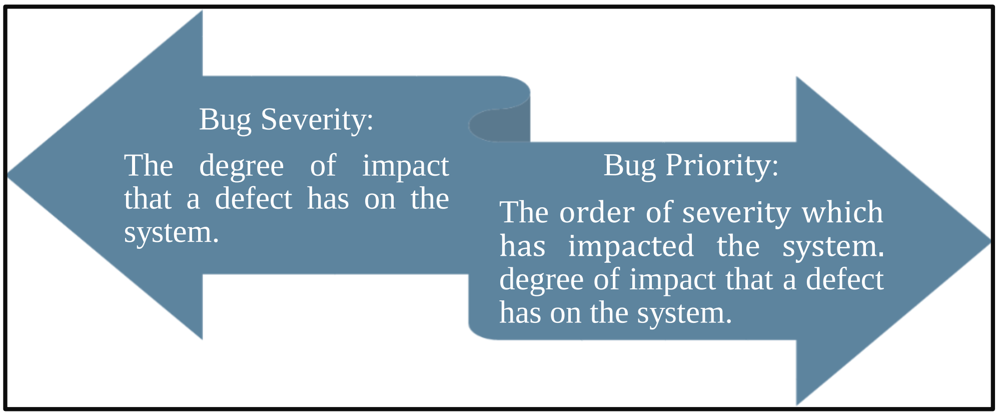


Figure 2.3: Bug Severity VS Priority.

2.1 Bug Reports Lifecycle

The lifecycle of bug reports contains the entire bug that has discovered to start through a process. Bug reports go through a series of status, this state varies from one project to another (Xie, Wen, Zhu, Gao, & Zheng, 2018). Where the bug reports begin when the bug is found and ends when the bug reports are closed (JIRA, 2020).

The life cycle of a bug contains a set of states that any detected bug goes through, and the number of these cases depends on the project itself. In this thesis, the life cycle of the bug reports has divided into five states including:

- New
- Assigned
    1. Rejected or
    2. Deferred or
    3. Duplicate
    4. Fixed
- Retested
- Verified or Re-Opened
- Closed

The figure bellow illustrates the lifecycle of bug reports from the JIRA software (JIRA, 2020).

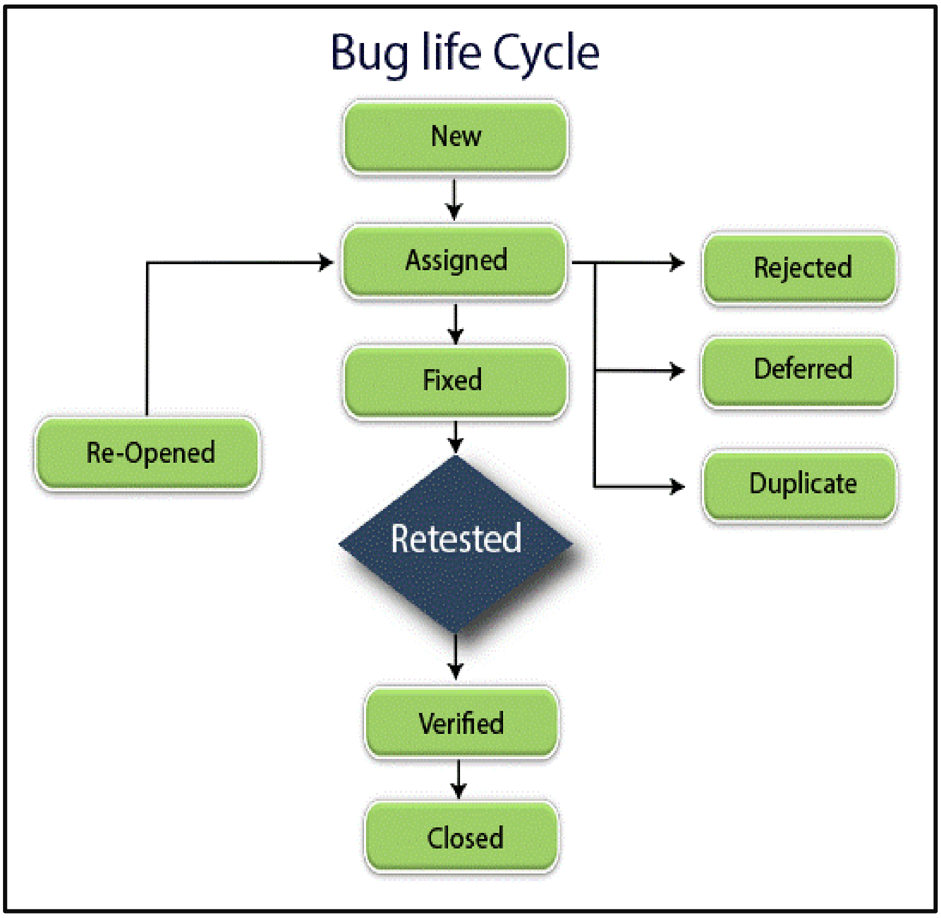


Figure 2.4: Life Cycle Of Bug Reports (JIRA, 2020).

Once the quality-assurance member opens a bug, the status of the bug is **new**, it will remain new until a lead assigns it to a developer team member, and it will be converted to **assigned**.

The assigned developer has various options for converting bug status to, if it is not a bug, the status is converted to **reject**, if the submission bug is not really that severe, the status of the bug is converted to **deferred** and resolved during future releases.

If two bugs of the same scenario are record, the developer can make the status of this bug **duplicate.**

If a new bug is resolve by a developer, it changes the status to **fixed**, and then it will go back to the qi team member to **retest** it, if **verified** and solved, the statues will **closed**, else the status will change to **re-opened**.

2.5 Bug Reports Content

The Bug reports include set of components which provide developers with knowledge to help reproduce and resolve the problem (Bugzilla, 2021), The bug reports included a combination of factors including (report id, summary, description, project name, priority, severity, attachment, status, sprint number, and reporter name), all of these factors are explained in the following table:

Table 2.1.Bug Report Content

Field	Definition
Report Id	A unique identifier.
Summary	A line of word describing a bug.
Description	More details that help the developer to reproduce the bug, such as test step, expected and actual results.
Project Name	The name of the project the reported bug relates to.

Priority	Represented in three words, low, medium, high and it says how quickly this bug should be resolved.
Severity	Represented in four common words, low, medium-high, and critically based on the impact of this bug on the system functionality.
Attachment	A screenshot or video shows the bug to the developer.
Status	The status of the bug.
Sprint Number	Shows during which sprint this bug detected.
Reporter Name	The person who filed this bug.

2.6 Machine Learning

Machine Learning (ML) is an area of study that focuses officially on the hypothesis, performance and properties of learning systems and algorithms. ML uses computer capacities by integrating calculations and data recovery to make it seem to understand and make logical choices, not only according to a particular strategy, but also to the earlier behaviour or responses (Mohri, Rostamizadeh, & Talwalkar, 2018).

The term machine learning described as a method of making a system sufficiently efficient that the different case uses can predicted accurately with experience. Machine learning algorithms allowed important "regularities" to be discover in large sets of data. It is regard as a research information-technology field rapid growth is due to developments in data analysis.

ML algorithms can divide primarily into two categories, the first category is supervised machine learning, and the purpose of supervised machine learning is to predict the right label for the newly presented information through assessments and observations, which it then classifies according to the training set.

And the second category is unsupervised machine learning, which implies that the data are not accessible for training, the aim of unsupervised machine learning is to obtain uncompelled data structures through analysing a similar approach between pairs of items, which are generally connected to the approximate density or data clustering (Kukkar et al., 2020).

Machine Learning (ML) is a discipline of AI that handles the development and analysis of a model from the information obtained from the data .The various applications of ML include classification and regression (Zhang, 2020). The ML classified into three main kinds depend on the existent of labelled samples, which include unsupervised learning, semi-supervised learning, and supervised learning. Moreover, ANNs, Naive Bayes classifier, SVMs, Logistic and Linear regression are the popular utilized ML algorithms (Burkov, 2019).

Deep Learning is a sophisticated type of ML with various levels of abstraction of data at several processing levels (Voulodimos, Doulamis, Doulamis, & Protopapadakis, 2018). Deep Learning can learn the complex distributions of entered samples via back-propagation and point out how the internal parameters updated at each level. The commonly applied deep learning comprises Recurrent Neural Networks (RNNs), CNNs, DBNs, and auto-encoders.

According to (He, Xu, Yan, Xia, & Lei, 2020), there are three significant justifications for deep learning outstanding. First, a recent increase in research on machine learning. Second, affordable computing hardware. Finally, processing capabilities (GPU) are grown sharply.

This thesis used the deep neural network algorithm including Long Short-Term Memory, and K-Nearest Neighbours to predict the severity of bug reports.

2.6.1 Recurrent Neural Network

The Recurrent Neural Network (RNN) is a type of artificial neural network that uses sequential or time-series data. These are frequently use for normal or transient issues including language translation, natural languages processing, and speech recognition, and they are used in popular applications such as Siri, speech recognition and Google Translate (Zaremba, Sutskever, & Vinyals, 2014).

There are several advantages to using RNN, including (Wang & Tax, 2016):

- It is the first algorithm that, due to the internal memory, remembers its input, which makes it well suited to machine learning problems involving sequential data.
- RNN has redundant connection in hidden state. This recurring constraint ensures that the sequential information captured in the input data. That is, the dependency between words in the text while making predictions.
- RNN has a "memory" that remembers all the information about what was calculated.
- All RNNs have feedback loops in the repeating layer. This allows them to retain information in their "memory" over time.

The RNN use training data, including Feedforward and CNNs. It can draw information from previous inputs to affect the current input and output by using its "memory." While CNNs assume that inputs and outputs are distinct, RNN output based on previous elements. Although future events can also help to evaluate the performance of the sequence, these events cannot taken into account in their predictions by unidirectional repetitive neural nets (Dyer, Kuncoro, Ballesteros, & Smith, 2016). The following figure show the RNN:

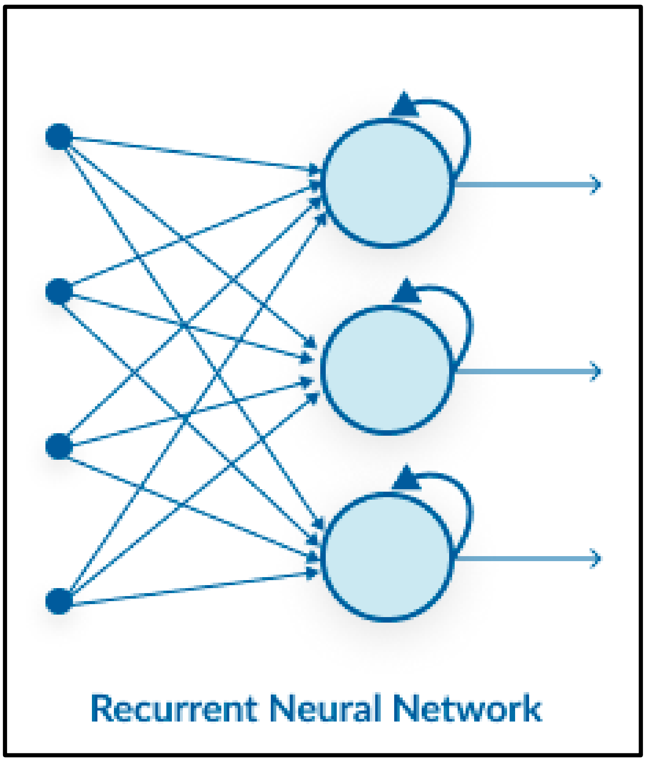


Figure 2.5: RNN.

There are several types of RNN, which are (Cui, Long, Min, Liu, & Li, 2018):

- One-to-one: image and predicate the class (NN).
- One-to-many: one input and many outputs (take an image and give a description)
- Many-to-one: take a sentence and predicate if it is positive or negative.
- Many-to-many: take much input and predicate much output (translation of a sentence from Arabic to English as an example).

2.6.2 Long Short-Term Memory

Long short -term memory (LSTM) is a form of supervised learning use for deep learning to produce bandwidth prediction using historical measurements and to remember information for long periods. Can be used in prediction problems for learning to turn input data into a preferred response (Beran, Schützner, & Ghosh, 2010). LSTM remembers historic events, which saw un-important data and forgets them. The corresponding information was select to save via different activation function layers called Internal Cell State Gates, as shown in the figure below:

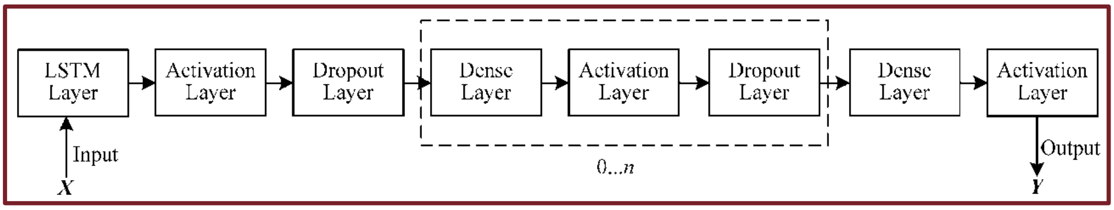


Figure 2.6: LSTM Layers.

LSTM is considered a type of RNN that uses previous events to warn future events (Tan et al., 2020). A set of gates used to control when information enters the memory, when it has output, and when it forgotten, and these gates are:

- Input gate: the input gate controls the flow of input activations into the memory cell.
- Output gate: output gate controls the output flow of cell activations into the rest of the network.
- Forget gate: scales the internal state of the cell before adding it as input to the cell through the self-recurrent connection of the cell, therefore adaptively forgetting, or resetting the cell's memory.

2.7 Related Works

The literature has included significant work on the use of machine learning to determine the severity of bug reports, and some of these works will discusses in this section.

Tim Menzies et al. study (Menzies & Marcus, 2008) is considered one of the first studies to predict the severity label of bug reports. A rule-based learning technique used to build a new tool called SEVERIS. SEVERIS relies on text mining and machine-learning techniques applied to unstructured data of the bug report unstructured data, which includes report summary and description. The automated prediction model for this study applied to the NASA-Project, and Project-Issue-Tracking-System (PITS). The results showed that the SEVERIS tool could applied to other open-source repositories such as Bugzilla, with a slight modification.

Anvik et al. (Anvik, Hiew, & Murphy, 2006) mentioned its personal communication with a Mozilla triager that impacts: “Every day, almost 300 bug appear that need triaging. This is far too much for only the Mozilla programmers to handle”. Anvik discussed the possibility to construct severity predictors from the inserted text. For data sets with more than 30 examples of high severity issues, SEVERIS always found good issue predictors with high f-measures.

Cheng-Zen et al (C.-Z. Yang, Chen, Kao, & Yang, 2014) studied the impact of four quality indicators of bug reports on severity prediction: stack traces, report length, attachments, and steps to reproduce. The authors used the Eclipse dataset in their empirical study. They concluded that examining the quality indicators in previous work could efficiently improve the prediction performance in most cases that used text information only.

In Yang et al (C.-Z. Yang, Hou, Kao, & Chen, 2012) study, they discussed the extent of the impact of specific features effectively on severity prediction. They selected their features Information Gain, Chi-Square, and Correlation Coefficient, based on the Multinomial Naive Bayes classification method. They used four open-source components in their experiment and used ROC curves to evaluate the measuring process. They concluded that selected features affect efficiency in severity prediction performance in most cases.

Meera Sharma et al (Sharma, Kumari, Singh, & Singh, 2014) developed a model to predict severity level of a reported bug based on multiple attributes namely priority, bug fix time, number of comments, number of bugs on which it is dependent, number of duplicates for it, number of members in cc list, summary weight and complexity of bug in a cross-project context. The authors used 5,859 bug reports in different open source platforms. The result shows that the proposed model can help to predict bug reports, which its historical data is not available, and provide accuracy in the range of 37.34 to 91.63%, 94.99 to 100%, 44.88 to 97.86% and 61.18 to 95.99% for different classifiers.

Imran et al (Imran, 2016) presented an approach, that combines feature extraction and, machine learning, to predict the severity of each bug, this approach depends on a keyword extraction text-mining algorithm for extracting keywords then it extracts the important keyword. The data set used in these classes included 4 different labels in every binary and multi-class, 90% refined data was used with machine learning algorithms and then the model was tested on 10% refined data and the result was better performance and higher classification precision -up to 90%-, data collection from Eclipse, Mozilla, GNOME and other systems.

In Jindal (Jindal, Malhotra, & Jain, 2017) study, A different examination was performed on four datasets of NASA's PITS using three main methods including decision tree, Multi-Nominal Multivariate Logistic Regression (MMLR) and Multi-Layer Perception (MLP) Prediction models were fed in various top-k terms, and these terms were extracted from training and testing sets using an Information Gain (IG) feature selection. The results showed that the performance of the decision tree is consider the best of all previous methods in determining the severity of bug.

Madhu Kumari et al (Kumari, Sharma, & Singh, 2018), presented a new classification approach they used five attributes for each reported bug, namely CC count, Component, Operating system, number of comments, and priority, and from those attributes, they derived two attributes called summary weight and entropy. To enhance the classification process they applied six types of classifications namely: Naïve Bayes (NB), k-Nearest Neighbours (KNN), Random Forest (RF), Relative Neighbours Graph (RNG), Condensed Nearest Neighbour (CNN), and Multinomial Logistic Regression (MLR) to make their classifier, the data sites used were collected from PITS, Mozilla and Eclipse. After applying the classifier, they initialized, the result showed an improvement in F-measure performance in comparison with previous research works.

Yang et al. (G. Yang, Min, Lee, & Lee, 2019) introduced a new technique, it's an amalgamation between similarity using KL-divergence and topic modeling using LDA to define the severity of bug reports. In their research, they used 20,000 bug reports, those reports were collected from four open-source projects (Xamarin, Eclipse, Wireshark, and Mozilla) were assembled to validate their proposed technique. The result of applying their technique showed that their model attains better performance, from an accuracy perspective than other cutting-edge studies listed in their literature.

Ramey et al. (Ramay et al., 2019) Proposed a deep neural network based automatic approach for the severity prediction of bug reports. This approach applies a deep learning model, natural language techniques, and emotion analysis on the given dataset for the severity prediction of bug reports. In addition, the approach automates the severity assessment process and helps users by subtracting the severity assignment step from bug reporting. This approach was evaluate on the history- data of open source products from Eclipse and Mozilla, and the results of cross-product show that the approach outperforms the state-of-the-art approaches, because it improves the f-measure by 7.90%.

Arvinderet al (A. Kaur & Jindal, 2019) evaluated the performance of ten different machine learning algorithms, which are naive Bayes, KNN, SVM, maximum entropy, random forest, decision tree, bagging, boosting, Glmnet and SLDA, in terms of precision, recall, and accuracy at the system-level and component-level. The evaluation conducted in thirteen Apache projects that are automatically extract by BRCS tools. The result shows that the Boosting algorithm performed best in twelve projects with an accuracy of 81% to 98% followed by a random forest of 75% to 97%, while Glmnet and SLDA achieved the most accurate results among other machine learning algorithms. In addition, the prediction of severity at component level gives better results than system-level prediction as Component's frequent terms are more specific than system-level frequent terms which in turn give better results than Inter-system level prediction.

Hamdy (Hamdy & El-Laithy, 2019) proposed a framework for predicting fine-grained severity levels which utilize a Minority Over-sampling Technique "SMOTE", to balance the severity classes, and a feature selection scheme, to reduce the data scale and select the most informative features for training a KNN classifier, which utilizes a distance-weighted voting scheme to predict the proper severity level of a newly reported bug. The effectiveness of the proposed approach has validated with two bug repositories, Eclipse and Mozilla. The result showed that their approach outperforms cutting-edge studies in predicting minority severity classes.

Chauhan et al (Chauhan & Kumar, 2020) proposed a new automated classifier, that works using bigram and TF-IDF approach to extract report features, and then they used SVM and neural network, using they found that the accuracy level of the classes is above 80, which make the approach effective and efficient.

The following table shows a summary of the literature review with limitations, methodology, data set, and feature details for several studies related to determining severity of bug reports.

Table 2.2. Summary of Related Methodology.

Ref	Methodology	Dataset	Limitation Of Study	Feature	Evaluation Matrix
(Menzies & Marcus, 2008)	SEVERIS	NASA Project, PITS	The proposed methodology has a Lack of consistency in PITS. The written conclusion is rules and is self-certifying	Textual	Precision, Recall, and F-1 score
(Sun, Song, & Jiao, 2009)	k-means, SRcut	Mozilla	The only textual features considered by the study. Some comments also reduce the model accuracy rate. Overall, the performance of the model was not so good. K means and normalization rates were no better than SRC.	Textual	Cluster purity, and Accuracy
(Nagwani & Verma, 2011)	STC	Mozilla, Jboss-Seam, MySQL	The proposed method only considered the small amount of dataset. The only features used by methodology were textual.	Textual	Purity, a total count of Clusters, and total time.
(Nagwani & Verma, 2012)	CLUBS	Androi, JBoss, Mozilla, MySql	The accuracy rate of the proposed methodology was quite low as compared to the amount of dataset	Categorically and textual	Precision, Recall, and F-1 score
(Somasundaram & Murphy, 2012)	SVM	Bugzilla Eclipse Pl	The only textual features considered by the study. Some comments also reduce the model accuracy rate. Overall, the performance of the model was not so good.	Textual	Recall
(Chawla & Singh, 2014)	TF-IDF, LSI	Google Chrome	The accuracy rate of the model was not so high.	Textual	Accuracy

Chapter Three

Methodology

3.1 Methodology Overview

The methodology approach in this thesis is experimental. The idea of the proposed framework emerged due to the increase of the submitted bug reports. Usually, developers spend a lot of time reading and analysing the description of a bug report to enhance the detection process of bug severity (Blocker, Critical, Major, Minor, and Low). Often the appropriate level of severity cannot be determined and historical records must reviewed in order to identify a relevant bug report.

3.2 Proposed Framework

This section presents the process of assigning the severity level for bug reports, it consists of two phases as shown in the figure below and these phases are:

- Phase one: Data collection and text pre-processing.
- Phase two: Feature extraction, training dataset and applied LSTM, and RNN algorithms and finally evaluation process.

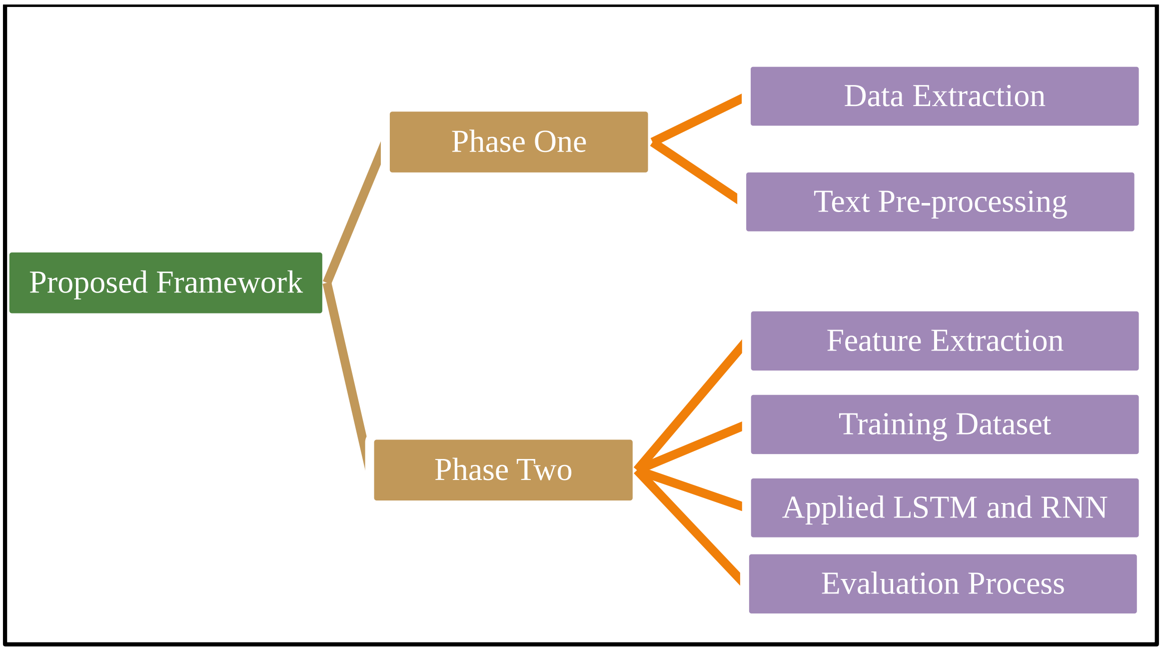


Figure 3.1: The Proposed Framework.

3.3 Phase One: Data Extraction and Text Pre-Processing

3.3.1 Dataset Extraction

The bug reports dataset was extracted from the repository of JIRA (JIRA, 2020) related to closed-source projects developed by TETCO Tatweer Educational Technologies Company (TETCO, 2020) in Riyadh, Saudi Arabia.

These data collected over a period of two and a half years, and it contains more than 2355 bug reports organized in one CSV file.

Each bug report described by set of factors such as summary, description, bug id, status, project name, project lead, priority, resolution, assignee, reporter, created date, resolved date, component, environment, sprint, attachment files and comments.

This thesis used a specific set of factors from the chosen datasets. The factors are considered as the most appropriate factors in order to predict the severity level **(Blocker, Critical, Major, Minor, and Low)** are **(Summary, Project key, Severity, Assignee, and Reporter)**.

The dataset processed in three phases including dataset extraction, pre-processing, and dataset training and testing, as shown in the figure 3.2, and in the following subsections, these phases will explained in detail.

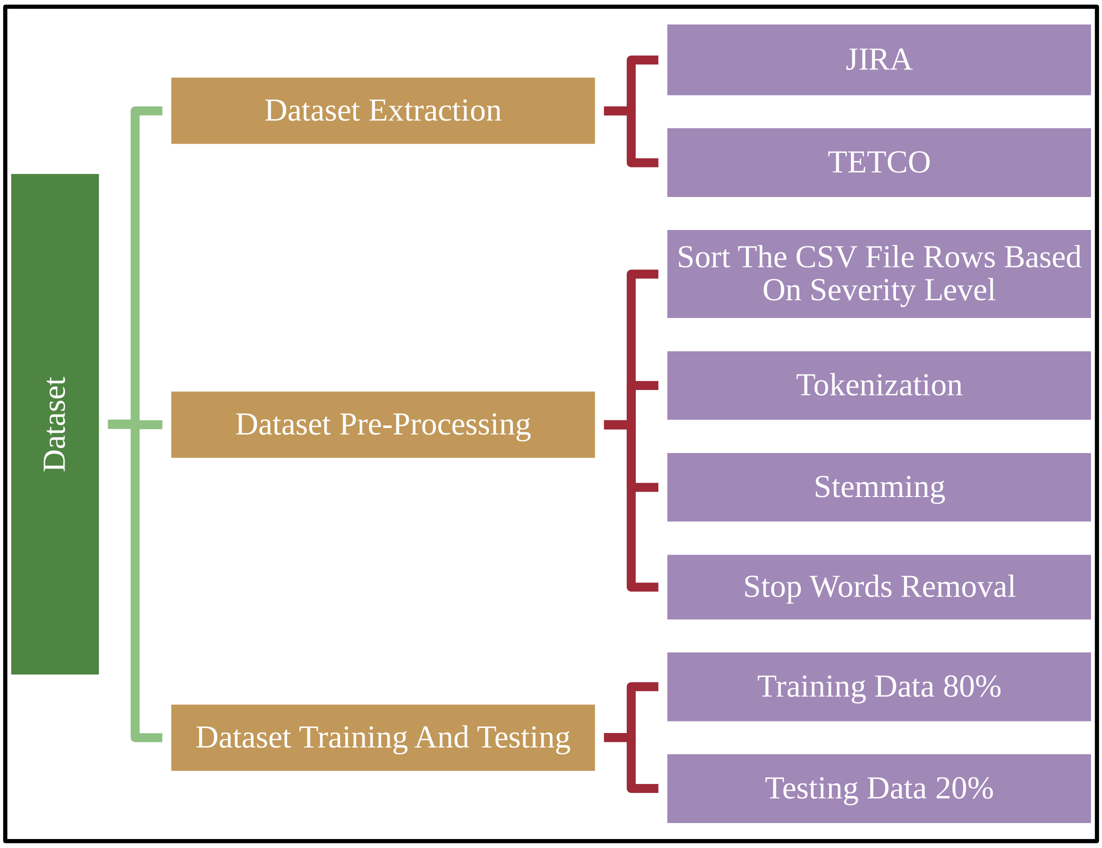


Figure 3.2: Dataset.

3.3.2 Dataset Pre-Processing

Data pre-processing is an important phase in the data mining process, as incorrect results generated by the analysis of data that has not analysed; also, it makes it easy to work with the input data. To this end, prior to the execution of the experiments, the quality and accuracy of data should first be ensured data cleaning, data integration, data transformation and data reduction are component of pre-processing activities (Dagao & Yang, 2018).

A new training package is the result of a data pre-processing task, which would create higher assignment efficiency and reduce classification time. This is due to the reduction in the size of the data, which allows for the faster and easier operation of learning algorithms (Bilalli, Abelló, Aluja-Banet, & Wrembel, 2018).

In this thesis, the Pre-processing of the TETCO dataset contains several activities as shown in the following figure below, and these activities include sorting the row, tokenization, stop-word removal, stemming, and remove the punctuation marks.

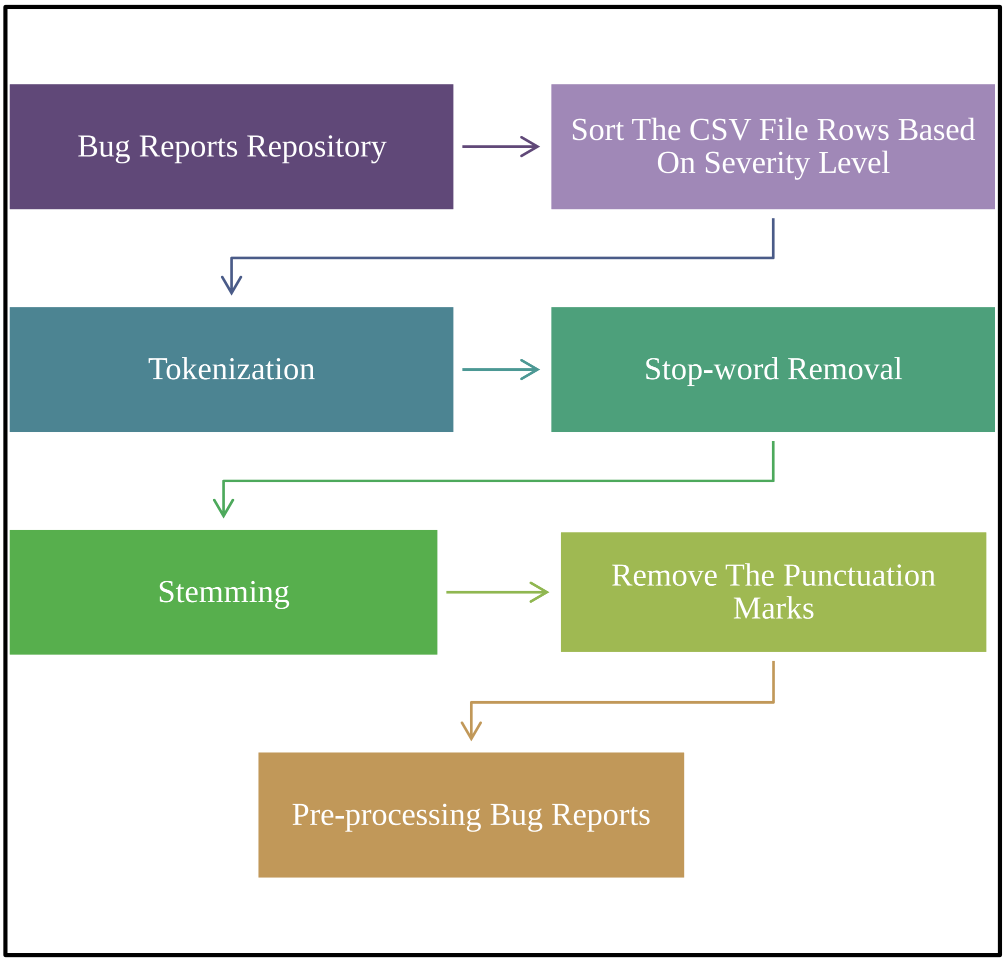


Figure 3.3: Pre-Processing Activitys.

These activities will discussed and explained in more detail in the following subsections.

3.3.2.1 Sorting

Sorting the CSV file rows based on the severity level.

3.3.2.2 Tokenization

Tokenization is the process of dividing text into words or sentences, converting it into lowercase letters, replacing punctuation marks, and removing end spaces.

3.3.2.3 Removal of Stop words

Words that are used to associate sentence flow with stop words, such as "the", "a", "on", "is", "all", while processing data these words are removed because they can make the computation complicated (J. Kaur & Buttar, 2018).

This process takes place in two steps, first the stop words extracted from the summary column of the dataset using the NLTK library (NLTK, 2017) and then the second step is to select the words and remove them from the dataset.

3.3.2.4 Applying Steaming

The steaming is a mechanism by which words are reduced to their root forms (Junior & do Carmo Nicoletti, 2019).

For example, words “send”, “sending”, and “sent” are different words and the same root word “send”. The word can be reduced to its steams and changed converted into “send”.

3.3.2.5 Punctuation Marks

Punctuation marks are symbols that add clarity to sentences(Nádvorníková, 2020). English has 14 punctuation marks including period, question mark, exclamation point, comma, semicolon, colon, dash, a hyphen, parentheses, brackets, braces, apostrophe, quotation marks, and ellipsis (“?, !., . – { },: ;).

Punctuation marks are not necessary to train the model, so this step removes punctuation marks from the data set. In addition, it is use to remove duplicate characters. Finally, the words "www", "http?:" And "//" have been removed from the dataset.

3.3.2.6 Removing Repeating Character

The repeating character removed from the dataset, as they can affect the computation complexity, time, and efficiency of the model.

3.3.2.7 The Words in Dataset

After pre-processing, the top 25 words an extracted from the dataset, as shown in the figure below:

Top 25 words in Summary

```
counter = Counter(all_words)
```

```
counter.most_common(25)
```

```
[('request', 957),
 ('apear', 847),
 ('student', 470),
 ('field', 340),
 ('scholarship', 292),
 ('mesag', 275),
 ('date', 274),
 ('valu', 272),
 ('companion', 267),
 ('wrong', 259),
 ('aprov', 235),
 ('studi', 234),
 ('nt', 218),
 ('data', 198),
 ('display', 195),
 ('chang', 184),
 ('stage', 178),
 ('incorect', 158),
 ('submit', 149),
 ('valid', 148),
 ('type', 146),
 ('s', 143),
 ('updat', 128),
 ('atach', 125),
 ('financi', 124)]
```

Figure 3.4: Top 25 Words Of The Dataset.

The frequency distribution of the top 25 words is also generate as shown in the figure below. The x-axis indicates the count of words and the y-axis indicates the top 25 words. The frequency distribution indicates that the word “request” has the most number of counts.

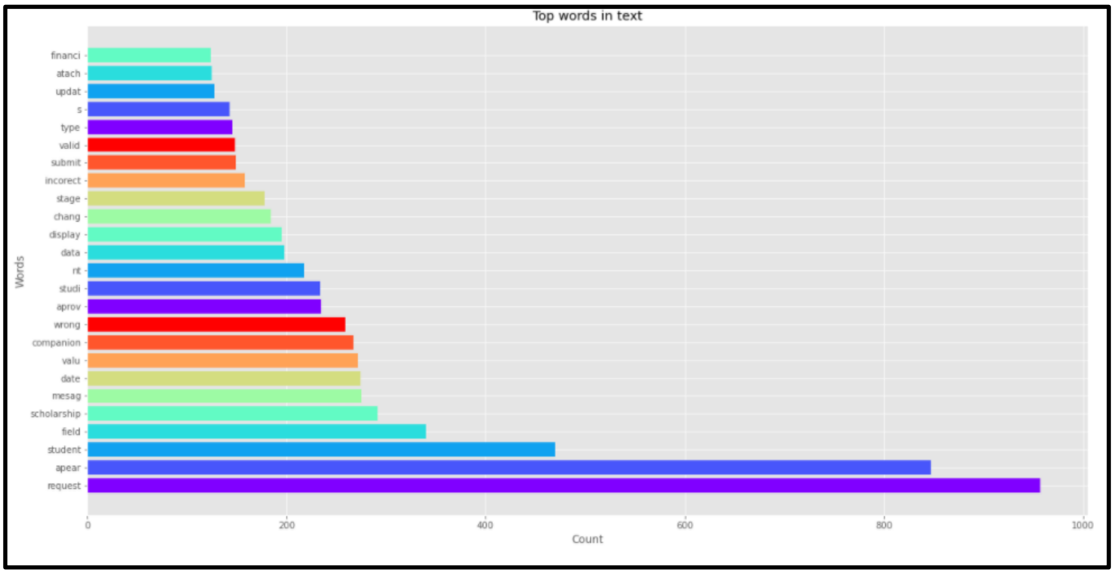


Figure 3.5: Top Words In The Text.

The table below shows the 10 most used words and indicates that the word that appeared the most frequently was "request" which appeared 955 times, while the word "wrong" was the least visible, as it appeared 255 times.

Table 3.1.The most used 10 words

Word	Count
Request	955
Appear	850
Student	469
Field	339
Scholarship	304
Mesag	283
Date	276
Valu	271
Companion	266
Wrong	255

The word cloud of severe class is shows in the figure below.

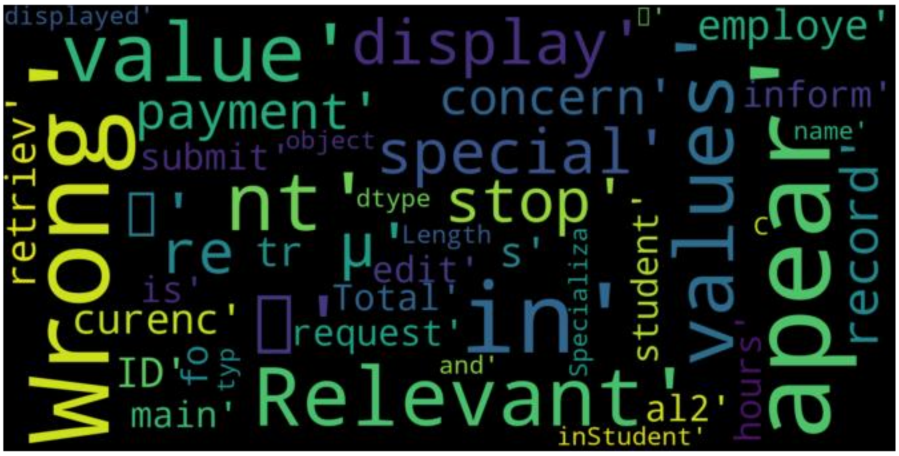


Figure 3.6: A Cloud Of Severe Class.

The table below shows the 10 most used words of severe class, it is indicates that the word that appeared the most frequently was "request" which appeared 605 times, while the word "wrong" was the least visible, as it appeared 124 times.

Table 3.2. Words Of Severe Class

Words	Counts
Request	605
Apear	535
Student	292
Field	186
Scholarship	153
Approve	145
Value	145
Companion	131
Studi	128
Mesag	127
Wrong	124

The word cloud of the non-severity class is shows in the figure below.

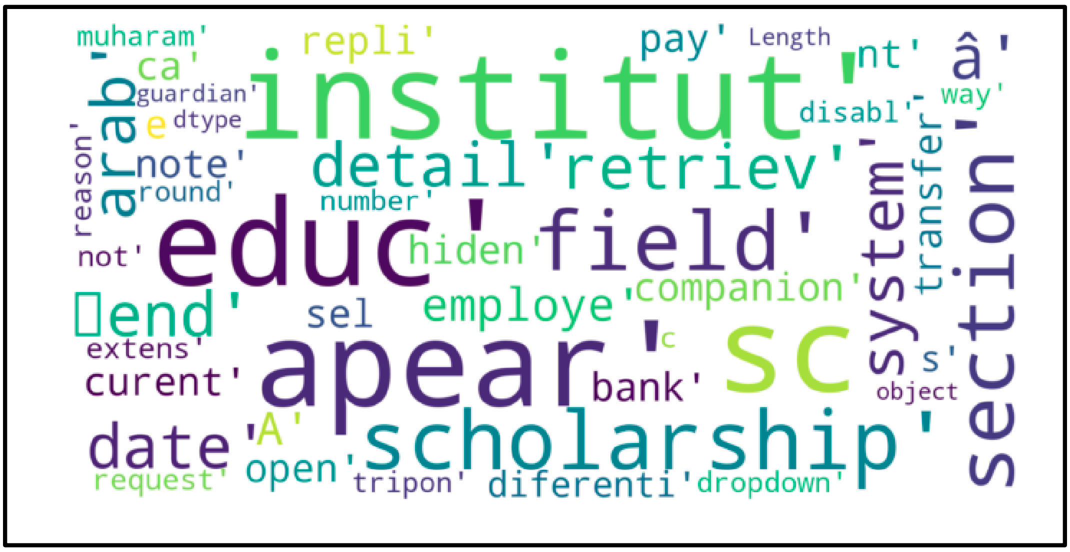


Figure 3.7: Word Cloud Of A Non-Severe Class.

The table below shows the ten most used words of non-severe class, it is indicates that the word that appeared the most frequently was "request" which appeared 350 times, while the word "Update" was the least visible, as it appeared 129 times.

Table 3.3. Ten Words Of A Non-Severe

Words	Counts
Request	350
Apear	315
Data	310
Student	177
Mesag	156
Field	153
Scholarship	151
Valid	151
Companion	135
Wrong	131
Update	129

3.4 Phase Two: Feature Extraction, Training Dataset and Applied LSTM, and RNN Algorithms and Evaluation Process

3.4.1 Feature Extraction

The next step is to extract a feature from the pre-processed dataset. First, the input and output features are extracted. The reshaping is performed with the values of (-1, 1).

Getting inpout and output features

```
X = data.Summary
Y = data.Severity
le=LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
```

Figure 3.8: Feature Extraction.

After that, the feature selection performed on the extracted feature. The maximum words are selected 1500 and the maximum length is selected to 100. The tokenization of the feature also performed.

Features Extraction and features selection from Summary

```
max_words = 1000
max_len = 500
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X)
sequences = tok.texts_to_sequences(X)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
```

Figure 3.9: Feature Selection.

After the extraction of features, the dataset divided into training and testing sets. The testing data used for the training valuation of the model and the training dataset used to train the model.

Split data into training and testing

```
X_train, X_test, y_train, y_test = train_test_split(sequences_matrix, Y, test_size=0.30, random_state=2)
```

Figure 3.10: Dataset Splitting.

3.4.2 Dataset Training and Testing

In this thesis, a Python library called Tensor flow (Tensorflow, 2015) was used to divide the data sets into training and testing sets in the ratio of 8:2, as shown in the figure below. The testing sets used to evaluate the training for the model. Specifically, 471 bug reports are used to train the model, where 1884 bug reports were trained. In addition, it is worth noting that the length of the data set has a great influence on the models, so the large length helps the dataset to be more efficient in performance.

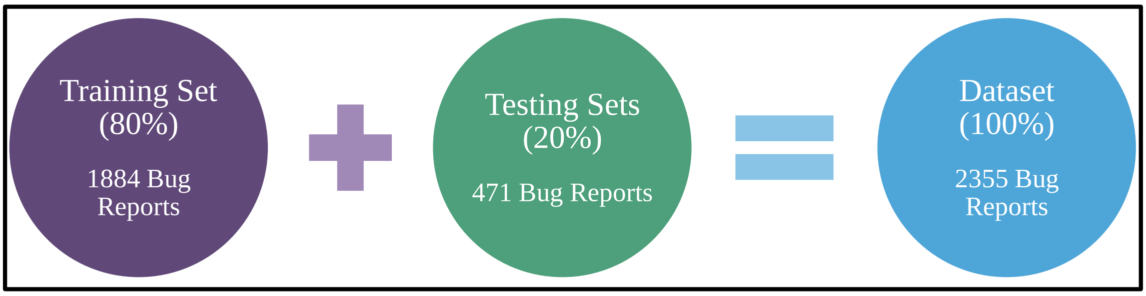


Figure 3.11: Dataset Training and Testing.

The study used three algorithms for testing and training i.e., LSTM and RNN. Each of these model training will be discussed in the following subsections in detail.

3.4.2.1 RNN Model Training

The RNN model trained on the training dataset. The RNN model used with the activation function of “sigmoid”, to get output from zero to one. In addition, the drop out is set to 0.1, and the density is set to 1 and 64.

The rectified linear function "RELU" has also been used with two activation layers, since RELU has shown great power when the input features are not independent of RELU if x>zero returns x, otherwise it returns zero. For many neural network types, it has converted into the default activation feature since a model used is easier to train and often performs better.

The model is set to sequential. The model contains eight layers. The first layer is the dense layer that followed by the dropout layer. The dropout layer followed by an activation layer. After the activation layer, again dense layer used that follows the dropout layer. The dropout layer followed by the activation layer, dense layer, and third activation layer. The model trained on 6 epochs. The batch size is set to 32 and verbose is set to one. The model split validation also performed with 0.1.

Implementing RNN model

```
RNN = Sequential()
RNN.add(Dense(64,input_shape=(max_len,)))
RNN.add(Dropout(0.1))
RNN.add(Activation('relu'))
RNN.add(Dense(64))
RNN.add(Dropout(0.1))
RNN.add(Activation('relu'))
RNN.add(Dense(1))
RNN.add(Activation('sigmoid'))
RNN.summary()
```

Figure 3.12: RNN implementation.

The figure below shows the structure of the RNN model. This model contains eight layers; the first layer is a dense layer that followed by the dropout layer. The dropout layer followed by the activation layer. After the activation layer, again dense layer used that follows the dropout layer. The dropout layer followed by the activation layer, dense layer, and third activation layer.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                64064
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
activation_3 (Activation)    (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 68,289
Trainable params: 68,289
Non-trainable params: 0
```

Figure 3.13: RNN Model Structure.

The model trained on 20 epochs. The batch size is set to 32, the model split validation also performed with 0.1.

Training and validating

```
In [259…
start = time.time()
RNN.compile(loss='hinge', optimizer='adam', metrics=['acc'])
history=RNN.fit(X_train,y_train,batch_size=32,epochs=6,validation_split=0.1,shuffle='true')
end = time.time()
RNN_time=end-start
print('Training finished !!')
```

Figure 3.14: RNN Model Training

The figure below shows the model training with 6 epochs. The loss and accuracy rate of the model quantified against each epoch. For epoch 1, the model score validation loss of 0.82, the loss rate of 0.84, the accuracy rate of 0.60, and a validation accuracy of 0.60. The accuracy rate of the model increase and decrease with increasing epoch.

```
Epoch 1/6
53/53 [==============================] - 1s 5ms/step - loss: 0.8428 - acc: 0.6009 - val_loss: 0.8217 - val_acc: 0.6085
Epoch 2/6
53/53 [==============================] - 0s 2ms/step - loss: 0.8559 - acc: 0.5938 - val_loss: 0.8249 - val_acc: 0.5979
Epoch 3/6
53/53 [==============================] - 0s 2ms/step - loss: 0.8502 - acc: 0.5850 - val_loss: 0.8427 - val_acc: 0.5661
Epoch 4/6
53/53 [==============================] - 0s 2ms/step - loss: 0.8181 - acc: 0.6175 - val_loss: 0.8215 - val_acc: 0.5926
Epoch 5/6
53/53 [==============================] - 0s 2ms/step - loss: 0.8255 - acc: 0.6036 - val_loss: 0.8164 - val_acc: 0.5979
Epoch 6/6
53/53 [==============================] - 0s 2ms/step - loss: 0.8129 - acc: 0.6165 - val_loss: 0.8224 - val_acc: 0.5979
Training finished !!
```

Figure 3.15: The Accuracy Rate Of The RNN Model

3.4.2.2 LSTM Model Training

The LSTM model trained on the training dataset. The LSTM model used with the activation function of "sigmoid" to get output from zero to one. In addition, the drop out is set to 0.5, and the density is set to 1 and 64.

The activation function "RELU" is also used since it has shown great power when the input features are not independent of RELU if x>zero returns x, otherwise it returns zero. For many neural network types, it has converted into the default activation feature since a model used is easier to train and often performs better.

The embedding also performed that takes maximum words and input data length. The model contains eight layers. The first layer is the input layer that followed by the embedding layer. The LSTM, F1, activation and dropout layers used with LSTM. The model trained on 6 epochs. The batch size is set to 32. The model split validation also performed with 0.1.

Firstly, the data was pre-processed. The model trained on extracted features. The ROC curve, F1, Precision-Recall, confusion matrix, accuracy plot, and loss plot are calculated for analysing model performance (Keras, 2019).

The LSTM model trained on the training dataset. The LSTM model used with the activation function of "sigmoid". The drop out is set to 0.5. The density is set to one. The activation function "RELU" is also used. The embedding also performed that take maximum words and input data length.

```
Implementing LSTM model

def LSTM_model():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
```

Figure 3.16: Implementation of LSTM

The figure below represent the structure of the LSTM model. The model contains eight layers. The first layer is the input layer that followed by the embedding layer. The LSTM, F1, activation, and dropout layers used with LSTM.

```
Model: "model_49"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          [(None, 500)]             0
_________________________________________________________________
embedding_67 (Embedding)     (None, 500, 50)           50000
_________________________________________________________________
lstm_65 (LSTM)               (None, 64)                29440
_________________________________________________________________
FC1 (Dense)                  (None, 256)               16640
_________________________________________________________________
activation_231 (Activation)  (None, 256)               0
_________________________________________________________________
dropout_128 (Dropout)        (None, 256)               0
_________________________________________________________________
out_layer (Dense)            (None, 1)                 257
=================================================================
Total params: 96,337
Trainable params: 96,337
```

Figure 3.17: LSTM Model Structure

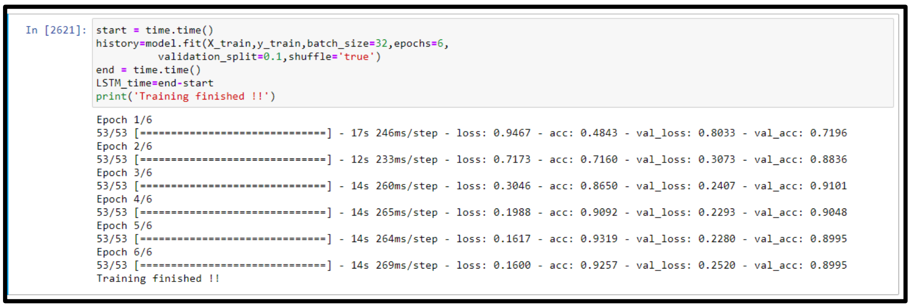
The LSTM model trained on 20 epochs. The batch size is set to 80. The model split validation also performed with 0.1.

```
Training and validating

In [2621]: start = time.time()
           history=model.fit(X_train,y_train,batch_size=32,epochs=6,
                    validation_split=0.1,shuffle='true')
           end = time.time()
           LSTM_time=end-start
           print('Training finished !!')
```

Figure 3.18: LSTM Model Training and Validating.

The figure below shows the LSTM model training with 6 epochs. The loss decrease and the accuracy increases against each epoch. For epoch one, the model score validation loss of 0.80, the loss rate of 0.94, the accuracy rate of 0.48, and the validation accuracy of 0.71. The accuracy rate of the LSTM model increase with increasing epoch.



```
In [2621]: start = time.time()
           history=model.fit(X_train,y_train,batch_size=32,epochs=6,
                     validation_split=0.1,shuffle='true')
           end = time.time()
           LSTM_time=end-start
           print('Training finished !!')

Epoch 1/6
53/53 [==============================] - 17s 246ms/step - loss: 0.9467 - acc: 0.4843 - val_loss: 0.8033 - val_acc: 0.7196
Epoch 2/6
53/53 [==============================] - 12s 233ms/step - loss: 0.7173 - acc: 0.7160 - val_loss: 0.3073 - val_acc: 0.8836
Epoch 3/6
53/53 [==============================] - 14s 260ms/step - loss: 0.3046 - acc: 0.8650 - val_loss: 0.2407 - val_acc: 0.9101
Epoch 4/6
53/53 [==============================] - 14s 265ms/step - loss: 0.1988 - acc: 0.9092 - val_loss: 0.2293 - val_acc: 0.9048
Epoch 5/6
53/53 [==============================] - 14s 264ms/step - loss: 0.1617 - acc: 0.9319 - val_loss: 0.2280 - val_acc: 0.8995
Epoch 6/6
53/53 [==============================] - 14s 269ms/step - loss: 0.1600 - acc: 0.9257 - val_loss: 0.2520 - val_acc: 0.8995
Training finished !!
```

Figure 3.19: The LSTM Model Training With 6 Epochs.

3.4.3 Evaluation Measures

There are several criteria for measuring the accuracy of prediction algorithms. In this thesis, the accuracy of prediction algorithms was measured using the following criteria's Precision, Recall, F-Measure, and Accuracy to consider two important things performance and effectiveness (Domingues, Filippone, Michiardi, & Zouaoui, 2018).

The Accuracy

Accuracy is the percentage of correctly predicted to the total, which is considered an important measure when using asymmetric datasets that present when the false positive and false negatives the same value. Accuracy can measured by the Equation (1) (Imran, 2016):

$$\text{Accuracy} = (TP + TN) / (TP + FP + FN + TN) \qquad (1)$$

The Precision

Precision is the function of relevant instances among the retrieved instances. It can measured by the Equation (2) (Imran, 2016):

$$\text{Precision} = TP / (FP+TP) \qquad (2)$$

The Recall

Recall is the percentage of correctly predicting positive for everyone in the actual result; it can measured by the Equation (3) (Imran, 2016):

$$\text{Recall} = TP /(TP + FN) \qquad (3)$$

The F1-Score

F1-Score means the average of Precision and recall taking into account false positives and false negatives. F1-Score is more effective than accuracy, especially if the data distribution is unbalanced. F1-Score can measured by the Equation (4) (Imran, 2016):

$$\text{F1-score}= 2 * (\text{Precision} * \text{Recall}) / (\text{Precision}+ \text{Recall}) \qquad (4)$$

Where:

- True Positives (TP): The result is the correctly predicted positive, meaning the actual results value and predicted result is "yes".
- True Negatives (TN): The result is the correctly predicted negative, meaning the actual result value and the predicted result is "No".
- False Positives (FP): This means the actual result is no and the predicted result is yes.
- False Negatives (FN): This means that the actual result is yes and predicted result in no.

Chapter Four

Experimental Results

4.1 Overview

This chapter presents the result of the experiment study, which has conducted to validate our module. The evaluation has performed with LSTM neural network and RNN.

The ROC curve, F1, Precision-Recall, confusion matrix, accuracy plot, and loss plot calculated to estimate model performance.

4.2 Results of LSTM

The LSTM trained on a training dataset with 6 epochs. First, the data was pre-processed. The model trained on extracted features. Then, to estimate model performance, the ROC curve, F1, Precision-Recall, confusion matrix, accuracy plot, and loss plot should calculated.

4.2.1 ROC Curve of LSTM

The ROC curve of the LSTM model represented in the figure below. ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR The x-axis represents FPR and the y-axis represents TPR. The curve starts from zero and moves towards one and the closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test (Skleran, 2019). The moving graph indicates the exceeding state of the graph. The false-positive rate is almost equal to one and the true positive rate is almost equal to one. In addition, the accuracy plot shows test and train accuracy.

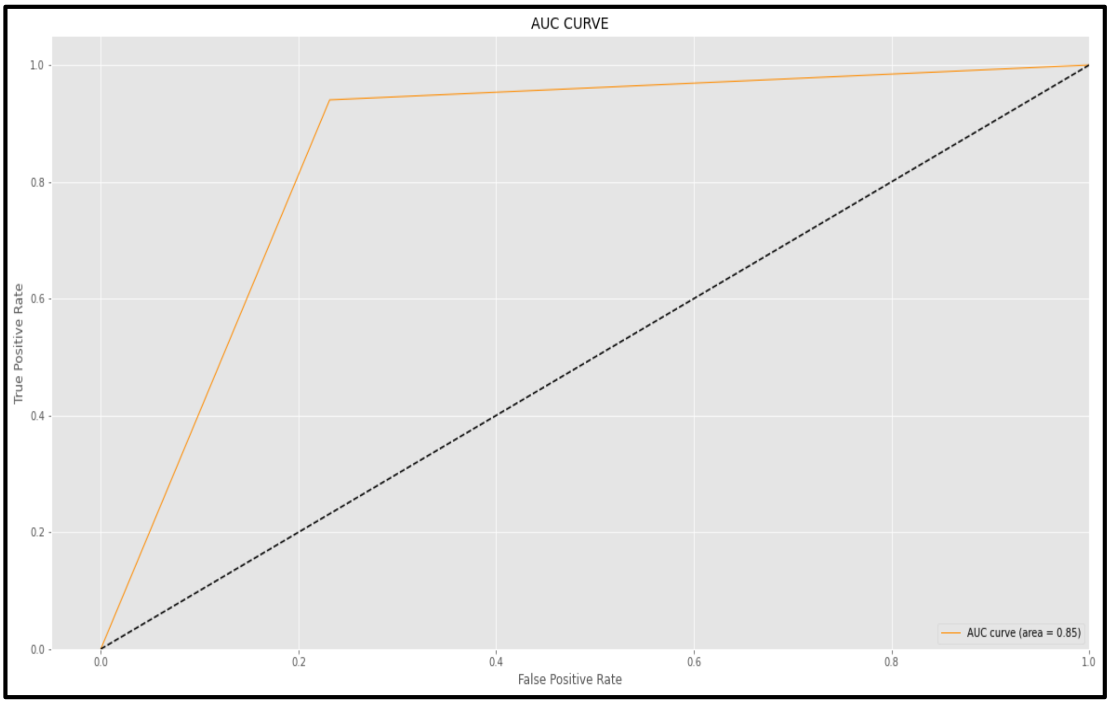


Figure 4.1: ROC Curve.

4.2.2 Confusion Matrix of LSTM

The confusion matrix of the LSTM model shown in the table below. The y-axis represents the true label and the x-axis presents the predicted labels. The confusion matrix depicts that out of 203 0-class examples 156 predicted accurately and 47 examples predicted wrongly by model. For one class, out of 268 examples, 252 correctly predicted and 16 examples predicted wrong. Overall, the accuracy of the model is very high.

Table 4.1: The Confusion Matrix Of The LSTM Model

N=467	**Predicted NO**	**Predicted YES**	
Actual: NO	TN=156	FP=47	203
Actual:YES	FN=12	TP=252	264
	168	299	

4.2.3 Measure Values Applied on LSTM

The table below shows the performance results of the LSTM model based on the level of severity. The LSTM model score accuracy rate of 0.87.

Table 4.2: Measure Values Applied on LSTM

	Precision	Recall	F1-Score
Class 0	0.91	0.77	0.83
Class 1	0.84	0.94	0.89
Macro Average	0.87	0.85	0.86
Weighted Average	0.87	0.87	0.86

4.2.4 Training and Validation Accuracy Plot of LSTM

The LSTM Neural Network experiments after the training epoch have been tried with a model in Keras frameworks that run in Python (Keras, 2019).

The validation data accuracy and loss could modified in various cases in the Keras model. The loss must be lower and higher as each epoch increases. The following cases will occur with Keras loss of validity and Keras accuracy (Brownlee, 2017):

- Validation loss starts increasing, validation accuracy starts decreasing, and the model will be cramming values and not learning.
- Validation loss and validation accuracy start increasing, the model will be over fitting probability values when softmax used in the output layer.
- Validation loss starts decreasing, validation accuracy starts increasing. The model is learning properly.

The following figure shows the training and validation accuracy plot of the LSTM model, the x-axis shows the epoch value and the y-axis depicts the accuracy of the model against each epoch. The model accuracy with the training data set indicated by blue dots and the red line indicates model accuracy with validation data set. The results of this figure show that the performance of the model is high.

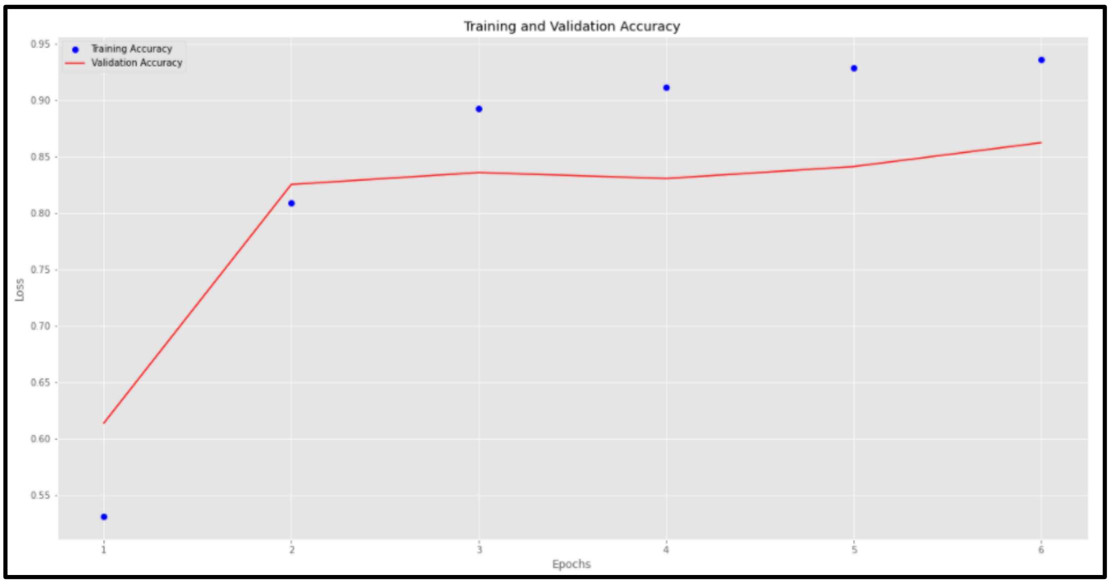


Figure 4.2: Training and Validation Accuracy of LSTM.

4.2.5 Training and Validation Loss Plot of LSTM

The loss plot of the LSTM model also generated that tells the accuracy of validation and training. The x-axis shows the Epoch value and the y-axis depicts the loss of the model against each epoch. The model loss with the training dataset indicated with blue dots and the model accuracy with the validation dataset denoted with a red line. The loss rate of the model is quite low on the training set as well as on the validation set.

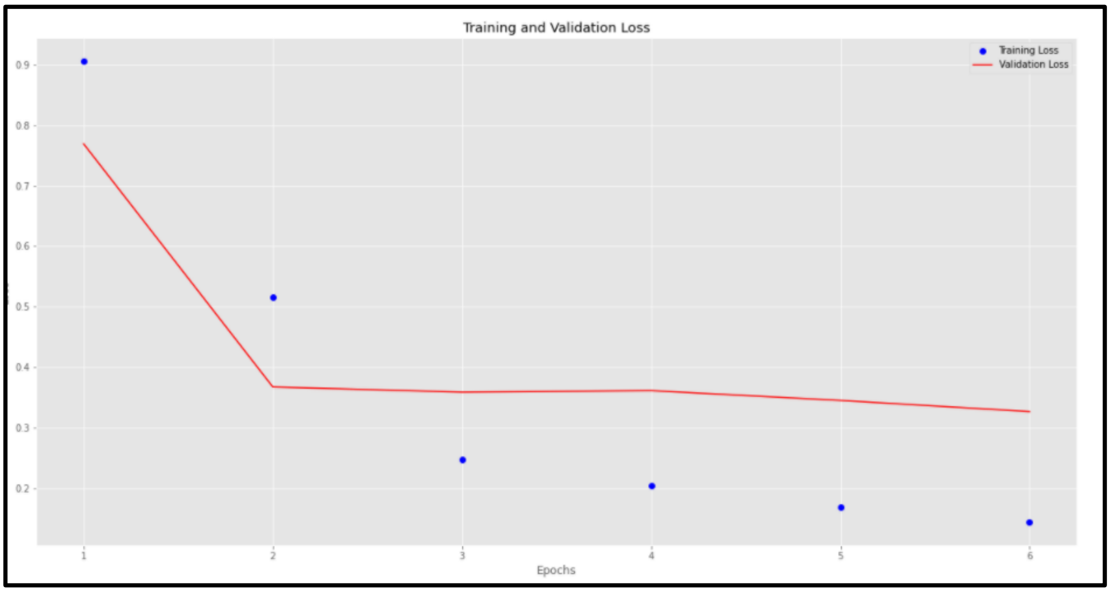


Figure 4.3: Loss Plot of LSTM.

4.2.6 Accuracy Plot of LSTM

The accuracy graph shows the validation accuracy. The x-axis of the graph shows the value of the Epoch and the y-axis shows the model accuracy of every epoch. The accuracy rate of the model is 0.89.

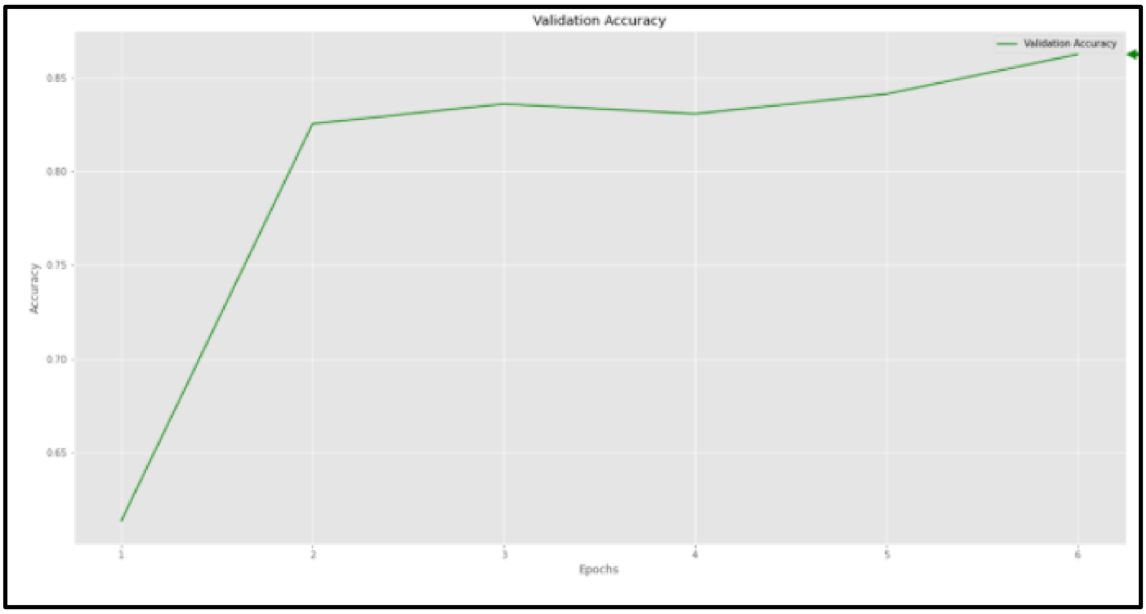


Figure 4.4: Accuracy Plot.

4.3 Results of RNN

The RNN trained on a training dataset with 6 epochs. First, the data was pre-processed. The model trained on extracted features. Then, to estimate model performance, the ROC curve, F1-score, Precision-Recall, confusion matrix, accuracy plot, and loss plot should calculated.

4.3.1 ROC Curve of RNN

The ROC curve of the RNN model represented in the figure below. ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR The x-axis represents FPR and the y-axis represents TPR. The curve starts from zero and moves towards one and the closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test (Skleran, 2019). The moving graph indicates the exceeding state of the graph.

The false-positive rate is almost equal to one and the true positive rate is quite low. The true positive rate depicts the examples that are true and predicted as true. The true negative rate depicts the examples that are true but predicted false.

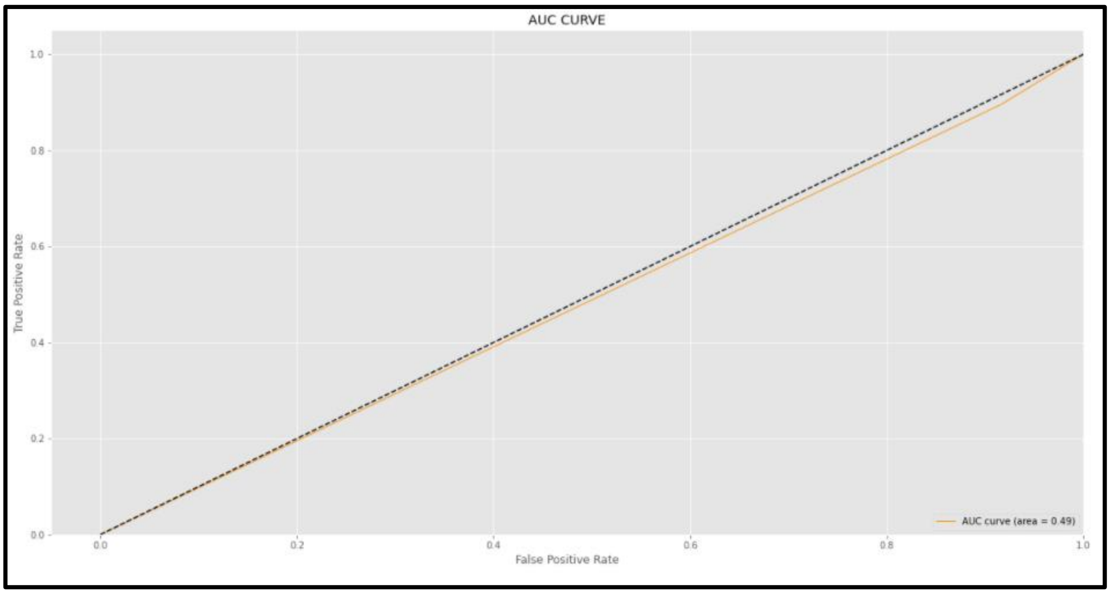


Figure 4.5: Roc Curve for RNN.

4.3.2 Confusion Matrix of RNN

The confusion matrix of the RNN model shown in the table below. The y-axis represents the true label and the x-axis presents the predicted labels. The confusion matrix depicts that out of 203 class examples 28 predicted accurately and 175 examples predicted wrongly by model. For one class, out of 268 examples, 244 correctly predicted and 24 examples predicted wrong. In general, the accuracy of the RNN model is very low.

Table 4.3. The Confusion Matrix Of The RNN Model

N=471	**Predicted NO**	**Predicted YES**	
Actual: NO	TN=28	FP=175	203
Actual:YES	FN=24	TP=244	268
	52	419	

4.3.3 Measure Values Applied on RNN

The table below shows the performance results of the RNN model based on the level of severity. The RNN model score accuracy rate of 0.58.

Table 4.4. Measure Values Applied on RNN

	Precision	Recall	F1-Score
Class 0	0.54	0.14	0.22
Class 1	0.58	0.91	0.71
Macro Average	0.56	0.52	0.46
Weighted Average	0.56	0.58	0.50

4.3.4 Training and Validation Accuracy Plot of RNN

The RNN model accuracy plot is also generated which indicates the validation and training accuracy. The x-axis of the figure shows the importance of the Epoch and the y-axis shows the model accuracy of every epoch. A dark green line used to represent model accuracy with the training data set and a light green line to indicate model accuracy with a validation data set. On both training and validation sets the performance of the model is low.

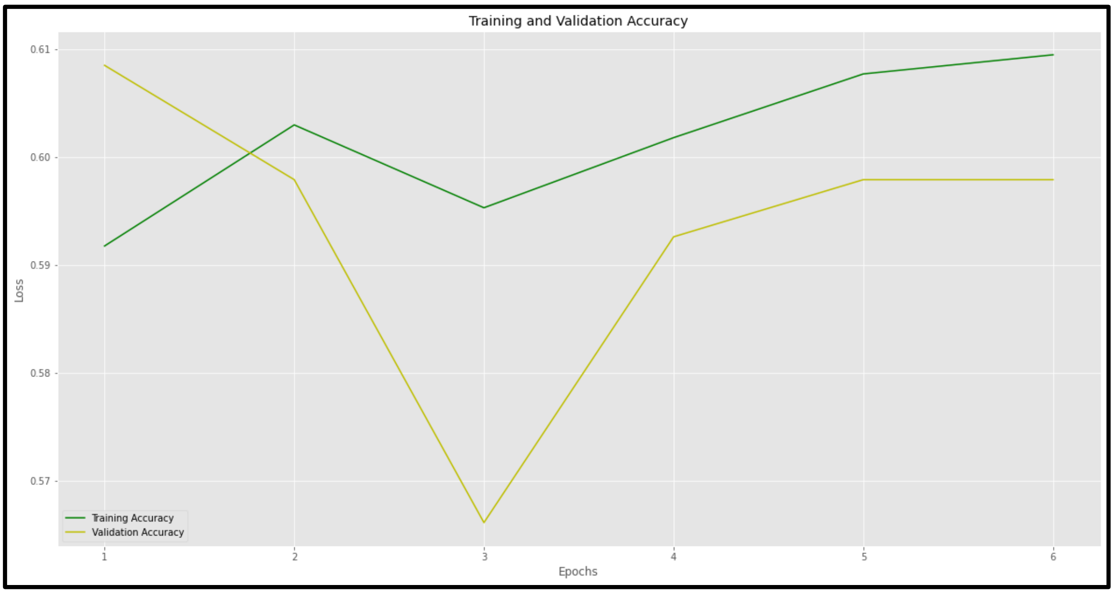


Figure 4.6: Training and Validation Accuracy Plot of RNN.

4.3.5 Training and Validation Loss Plot of RNN

The loss plot of the model also generated that tells the accuracy of validation and training. The x-axis of the graph shows the Epoch value and the y-axis depicts the loss of the model against each epoch. The model loss with the training dataset denoted with dark green and the model accuracy with the validation dataset denoted with a light green line. The loss rate of the model is high on training (in green colour) and validation (in yellow colour) sets.

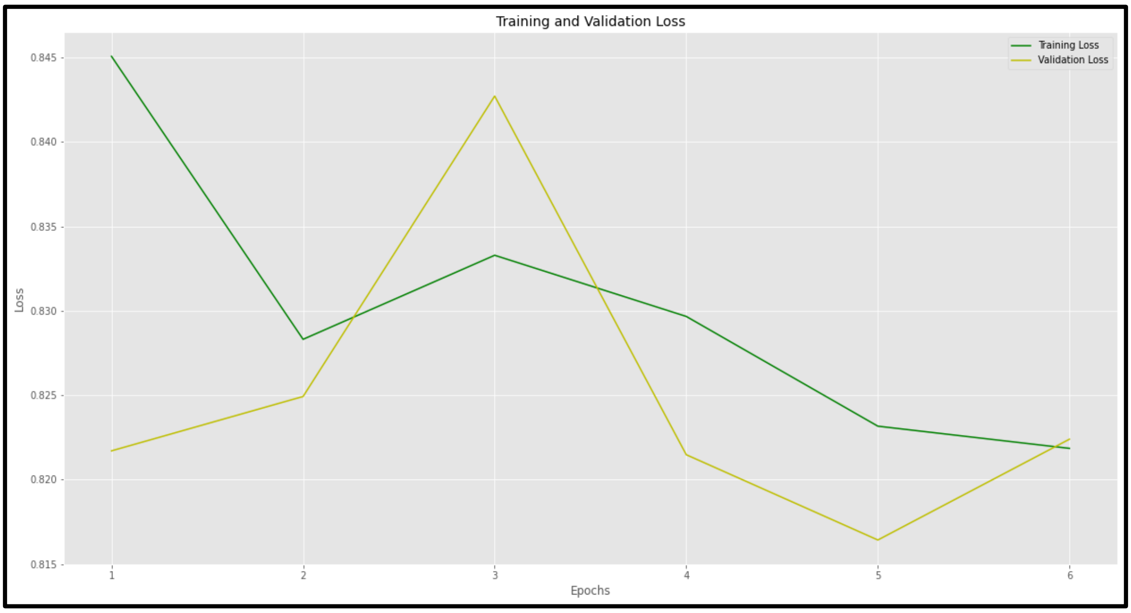


Figure 4.7: Loss Plot of LSTM.

4.3.6 Accuracy Plot of RNN

The accuracy plot tells the accuracy of validation. The x-axis of the graph shows the Epoch value and the y-axis depicts the Accuracy of the model against each epoch. The heights accuracy rate achieved by the model is 0.60

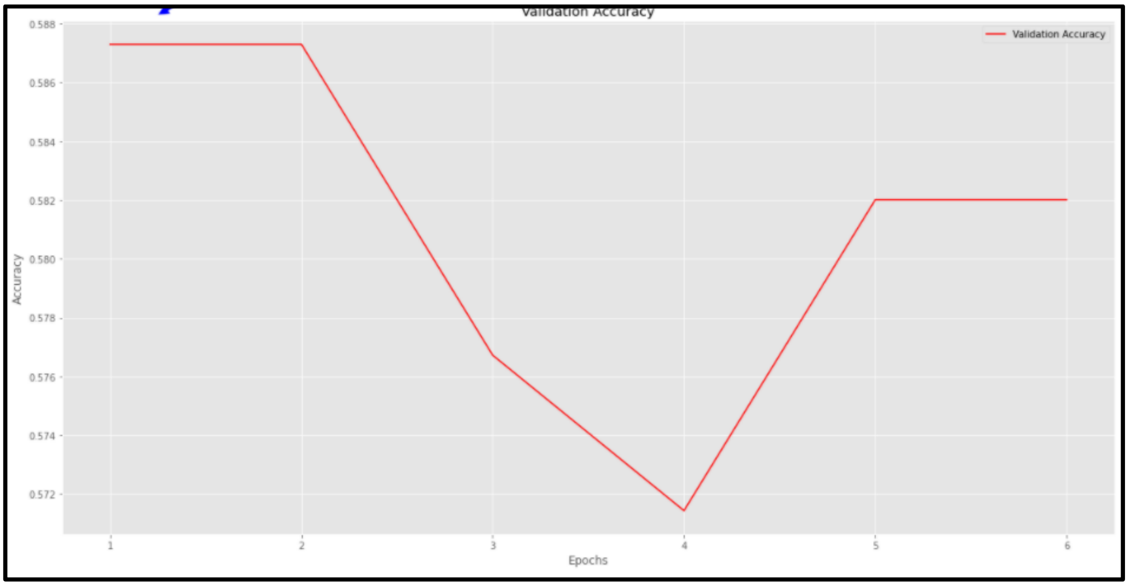


Figure 4.8: Accuracy Plot of RNN.

4.4 Comparison between LSTM and RNN

In this section, a comparison made between the algorithms that used in this thesis in order to predict the severity of the bug reports.

The following table shows the accuracy of the work of each of the algorithms, in addition to the accuracy achieved by each of these algorithms. The results show that the LSTM algorithm with score accuracy rate of 0.85 was the best among the algorithms used, followed by the RNN algorithm that achieved the lowest accuracy rate.

Table 4.5 Comparison between LSTM and RNN Results.

Algorithm	Accuracy
LSTM	0.85
RNN	0.58

A time computation-based comparison between LSTM, and RNN also performed as shown in the figure below. The x-axis shows the model name and the y-axis shows the calculation time for each model. It analyzed that the computation time of RNN is better than and LSTM. Hence, but as discussed, the LSTM performs much better.

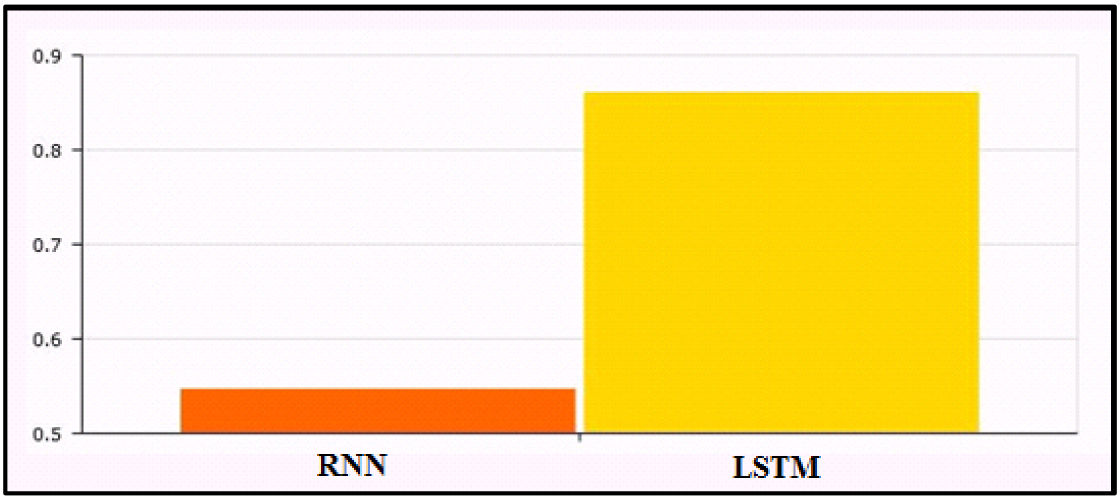


Figure 4.9:A Time Computation-Based Comparison between LSTM and RNN.

Chapter Five

Conclusions And Recommendations

5.1 Overview

This chapter summarizes the main purpose of this thesis. Section 5.2 provides conclusions that deduced based on our proposed bug severity prediction framework, and section 5.3 reviews the further of the study.

5.2 Conclusions

This thesis provides a framework for automatically assign the severity of bug for bug reports to avoid wasting limited time and resources during the software testing process.

The proposed framework involves using text pre-processing (tokenization, stop words and stemming) and then extracting an important keyword from the bug report description, this model trained on 80% of the dataset, and then tested on 20%.

The proposed framework validated on datasets extracted from JIRA using a TETCO closed-source project dashboard with over 2,300 bug reports to get better performance and higher accuracy.

The results of our experiments indicate that the proposed framework based on the LSTM algorithm achieved correctly predicts priority of bug reports and performance can significantly increase instead of RNN.

In addition, the comparison of the models shows that the LSTM performed better than the RNN, and the LSTM scored an accuracy rate of 0.858 while the RNN scored an accuracy rate of 0.58.

5.3 Future Work

In future work, Bi-directional LSTM and other deep networks-based models can applied to improve the performance of detection. Dataset can re-labelled with different annotators because the current data are not more distinguishable between the severities and non-severe. A built model can deploy to real-world applications.

References

Anvik, J., Hiew, L., & Murphy, G. C. (2006). *Who should fix this bug?* Paper presented at the Proceedings of the 28th international conference on Software engineering.

Beran, J., Schützner, M., & Ghosh, S. (2010). From short to long memory: Aggregation and estimation. *Computational statistics & data analysis, 54*(11), 2432-2442.

Bibyan, R., Anand, S., & Jaiswal, A. (2020). Assessing the Severity of Software Bug Using Neural Network *Strategic System Assurance and Business Analytics* (pp. 491-502): Springer.

Bilalli, B., Abelló, A., Aluja-Banet, T., & Wrembel, R. (2018). Intelligent assistance for data pre-processing. *Computer Standards & Interfaces, 57*, 101-109.

Brownlee, J. (2017). How to Diagnose Overfitting and Underfitting of LSTM Models. *1 September 2017*.

Bugzilla. (2021). Bugzilla Bug-Tracking System. Retrieved 1/1/2021 Available at: https://wiki.mozilla.org/BMO/UserGuide/BugFields Last visit: 1/1/2021.

Burkov, A. (2019). *The hundred-page machine learning book* (Vol. 1): Andriy Burkov Canada.

Chauhan, A., & Kumar, R. (2020). Bug Severity Classification Using Semantic Feature with Convolution Neural Network *Computing in Engineering and Technology* (pp. 327-335): Springer.

Chawla, I., & Singh, S. K. (2014). *Automatic bug labeling using semantic information from LSI.* Paper presented at the 2014 Seventh International Conference on Contemporary Computing (IC3).

Cui, J., Long, J., Min, E., Liu, Q., & Li, Q. (2018). *Comparative study of CNN and RNN for deep learning based intrusion detection system.* Paper presented at the International Conference on Cloud Computing and Security.

Dabade, T. D. (2012). *Information technology infrastructure library (ITIL).* Paper presented at the Proceedings of the 4th National Conference.

Dagao, D., & Yang, G. (2018). Preprocessing technology of consuming big data based on user interest with internet of location mining. *International Journal of Computers and Applications*, 1-6.

Domingues, R., Filippone, M., Michiardi, P., & Zouaoui, J. (2018). A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition, 74*, 406-421.

Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*.

Hamdy, A., & El-Laithy, A. (2019). Smote and feature selection for more effective bug severity prediction. *International Journal of Software Engineering and Knowledge Engineering, 29*(06), 897-919.

He, J., Xu, L., Yan, M., Xia, X., & Lei, Y. (2020). *Duplicate bug report detection using dual-channel convolutional neural networks.* Paper presented at the Proceedings of the 28th International Conference on Program Comprehension.

Iivari, J., Hirschheim, R., & Klein, H. K. (1998). A paradigmatic analysis contrasting information systems development approaches and methodologies. *Information systems research, 9*(2), 164-193.

Imran, Z. (2016). *Predicting bug severity in open-source software systems using scalable machine learning techniques.*

ISTQB. (2019). from International Software Testing Qualifications Board - Foundation Level https://www.istqb.org/ last visited at:5/1/2021.

Jindal, R., Malhotra, R., & Jain, A. (2017). Prediction of defect severity by mining software project reports. *International Journal of System Assurance Engineering and Management, 8*(2), 334-351.

JIRA. (2020). JIRA Software Developed By Atlassian Retrieved 14/1/2020, from https:/ www.atlassian.com/software/jira. Issue workflows available at: https://confluence.atlassian.com/adminjiracloud/issue-workflows-844500760.html Last visit: 14/1/2020.

Junior, J. R. B., & do Carmo Nicoletti, M. (2019). An iterative boosting-based ensemble for streaming data classification. *Information Fusion, 45*, 66-78.

Kaur, A., & Jindal, S. G. (2019). Text analytics based severity prediction of software bugs for apache projects. *International Journal of System Assurance Engineering and Management, 10*(4), 765-782.

Kaur, J., & Buttar, P. K. (2018). Stopwords removal and its algorithms based on different methods. *International Journal of Advanced Research in Computer Science, 9*(5), 81.

Keras. (2019). Deep Learning library for Theano and TensorFlow. https://keras.io/, Accessed on 20 DEC 2020. . from Deep Learning library for Theano and TensorFlow. https://keras.io/, Accessed on 01 May 2019.

Kukkar, A., Mohana, R., & Kumar, Y. (2020). Does bug report summarization help in enhancing the accuracy of bug severity classification? *Procedia Computer Science, 167*, 1345-1353.

Kumari, M., Sharma, M., & Singh, V. (2018). Severity assessment of a reported bug by considering its uncertainty and irregular state. *International Journal of Open Source Software and Processes (IJOSSP), 9*(4), 20-46.

Menzies, T., & Marcus, A. (2008). *Automated severity assessment of software defect reports.* Paper presented at the 2008 IEEE International Conference on Software Maintenance.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*: MIT press.

Nádvorníková, O. (2020). The use of English, Czech and French punctuation marks in reference, parallel and comparable web corpora: a question of methodology. *Linguistica Pragensia*, *30*(1), 30-50.

Nagwani, N. K., & Verma, S. (2011). Software bug classification using suffix tree clustering (STC) algorithm. *International Journal of Computer Science and Technology, 2*(1), 36-41.

Nagwani, N. K., & Verma, S. (2012). CLUBAS: an algorithm and Java based tool for software bug classification using bug attributes similarities.

NLTK. (2017). Natural Language Toolkit (NLTK) , preprocessing text library building Python programs . By Steven Bird, Edward Loper, Ewan Klein https://www.NLTK.org/. Last visit: 23/1/2020.

Pandey, N., Hudait, A., Sanyal, D. K., & Sen, A. (2018). Automated classification of issue reports from a software issue tracker *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications* (pp. 423-430): Springer.

Ramay, W. Y., Umer, Q., Yin, X. C., Zhu, C., & Illahi, I. (2019). Deep neural network-based severity prediction of bug reports. *IEEE Access*, *7*, 46846-46857.

Sharma, M., Kumari, M., Singh, R., & Singh, V. (2014). *Multiattribute based machine learning models for severity prediction in cross project context.* Paper presented at the International Conference on Computational Science and Its Applications.

Skleran. (2019). from Function computes subset accuracy: between true positive and false positivehttps://scikitlearn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. Last visit: 9/12/2019.

Somasundaram, K., & Murphy, G. C. (2012). *Automatic categorization of bug reports using latent dirichlet allocation.* Paper presented at the Proceedings of the 5th India software engineering conference.

Sun, Y., Song, H., & Jiao, W. (2009). *Towards Architecture-centric Collaborative Software Development.* Paper presented at the SEKE.

Tan, Y., Xu, S., Wang, Z., Zhang, T., Xu, Z., & Luo, X. (2020). Bug severity prediction using question-and-answer pairs from Stack Overflow. *Journal of Systems and Software*, 110567.

Tensorflow. (2015). from Develop and train ML models, by Google Brain Team, written in: Python, C++, CUDA, 2015 https://www.tensorflow.org/ Last visit: 15/12/2020.

TETCO. (2020). Tatweer for Educational Technologies Companey (Tetco), Riyadh, Saudi Arabia, Available At: https://tetco.sa/ar/default.aspx. Last visited at :10/1/2021.

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience, 2018*.

Wang, F., & Tax, D. M. (2016). Survey on the attention based RNN model and its applications in computer vision. *arXiv preprint arXiv:1601.06823*.

Xie, Q., Wen, Z., Zhu, J., Gao, C., & Zheng, Z. (2018). *Detecting duplicate bug reports with convolutional neural networks.* Paper presented at the 2018 25th Asia-Pacific Software Engineering Conference (APSEC).

Yang, C.-Z., Chen, K.-Y., Kao, W.-C., & Yang, C.-C. (2014). *Improving severity prediction on software bug reports using quality indicators.* Paper presented at the 2014 IEEE 5th International Conference on Software Engineering and Service Science.

Yang, C.-Z., Hou, C.-C., Kao, W.-C., & Chen, X. (2012). *An empirical study on improving severity prediction of defect reports using feature selection.* Paper presented at the 2012 19th Asia-Pacific Software Engineering Conference.

Yang, G., Min, K., Lee, J.-W., & Lee, B. (2019). Applying Topic Modeling and Similarity for Predicting Bug Severity in Cross Projects. *TIIS, 13*(3), 1583-1598.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, X.-D. (2020). Machine learning *A Matrix Algebra Approach to Artificial Intelligence* (pp. 223-440): Springer.