Application of XGBoost Algorithm and Deep Learning Techniques for Severity Assessment of Software Defect Reports

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF DEGREE

OF

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

Submitted By:

AKANKSHA CHAUHAN

2K18/SWE/02

Under the supervision of

Dr. RUCHIKA MALHOTRA

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of engineering)

Bawana Road, Delhi-110042

JUNE, 2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of engineering)

Bawana Road, Delhi-110042

DECLARATION

I, Akanksha Chauhan, Roll No. 2K18/SWE/02 student of M.Tech (Software Engineering), hereby declare that the Project Dissertation titled "**Application of XGBoost Algorithm and Deep Learning Techniques for Severity Assessment of Software Defect Reports**" which is submitted by me to the Department of Computer Science & Engineering , Delhi Technological University, Delhi in partial fulfillment for the requirement of the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: DTU, Delhi   Akanksha Chauhan

Date: 30-06-2020   (2K18/SWE/02)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled *"***Application of XGBoost Algorithm and Deep Learning Techniques for Severity Assessment of Software Defect Reports***"* which is submitted by Akanksha Chauhan, Roll No. 2K18/SWE/02, Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment for the requirement of the award of degree of Master of Technology (Software Engineering) is a record of a project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **(Dr. Ruchika Malhotra)**

Date: 30-06-2020 **SUPERVISOR**

Associate Professor

Department of C.S.E.

Delhi Technological University

ABSTRACT

Software is present in every aspect of our everyday life, and defects are bound to be found during the testing of the software, no matter how small. It is therefore imperative for software testing engineers to assess the severity of software defects to allocate proper resources for the correction of the defects and prevent software crashes. In this study, we have proposed the use of the Extreme Gradient Boosting Technique and deep learning techniques: Convolutional Neural Network and Recurrent Neural Network to predict the severity of the defects occurring in the software. AUC and sensitivity are the metrics used to evaluate the results. All three techniques: XGBoost algorithm, CNN and RNN have performed really well in predicting the severities for all the defects. It has also been noted that XGBoost algorithm is the most efficient in predicting high severity defects, while the performance of deep learning techniques is excellent for the highest as well as the lowest severity defects. For the rest of the severity values, the performance of all the three classifiers is fairly consistent.

ACKNOWLEDGEMENT

I am most thankful to my family for constantly encouraging me and giving me unconditional support while pursuing this research.

I am extremely grateful to **Dr. Ruchika Malhotra,** Associate Professor, Department of Computer Science Engineering, Delhi Technological University, Delhi for providing invaluable guidance and being a constant source of inspiration throughout my research. I will always be indebted to her for the extensive support and encouragement she provided.

I also convey my heartfelt gratitude to all the research scholars of the Web Research Group at Delhi Technological University, for their valuable suggestions and helpful discussions throughout the course of this research work.

Akanksha Chauhan
(2K18/SWE/02)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

XGBoost	Extreme Gradient Boosting
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
ML	Machine Learning
DL	Deep Learning
PITS	Project and Issue Tracking System
QA	Quality Assurance
SEVERIS	Severity Issue Assessment
ODC	Orthogonal Defect Classification
SVM	Support Vector Machine
RBF	Radial Basis Function
GloVe	Global Vectors for Word Representations
LSTM	Long Short-Term Memory
MNB	Multinomial Naïve Bayes
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
TF	Term Frequency
AUC	Area Under the ROC Curve
ROC	Receiver Operating Characteristics
IDF	Inverse Document Frequency
CC	Carbon Copy

CHAPTER 1 INTRODUCTION

1.1. OVERVIEW

Software programs are present in every walk of our daily lives. From a fitness tracker tracking our morning walks to centralized home solutions like Google Nest, Apple HomePod etc. which control the lights of our room at night among various other things, the use of software programs is so extensive that now we cannot imagine our lives without them. After the development of a software, it goes through a rigorous testing phase before being used by the end-users/customers. However, it is almost impossible to perform an exhaustive testing of any software. This leads to the software accumulating one or more defect(s). A software defect may be defined as an erroneous operation of a software when it does not meet the software requirements specified in the software requirements specifications document. A Software is bound to pick up some defects while in the development phase, and it is the job of the testing engineers to assess these defects and their impact on that software. These defects cause the software to behave abnormally and the software might end abruptly or crash, putting the reputation of the developing organization at risk. It is to be noted that not every defect has the same impact. Some defects, if triggered, can cause a major damage to the software making it extremely important as well as urgent to deal with them first. Some other defects might be so trivial that they have little or no impact on the working of the software. The extent of this impact is known as the severity of the defect. The severity of a defect may be defined as the level of impact of a failure on a software. In critical systems, mainly real-time systems like those of NASA, it is highly imperative that the testing engineers are sure that the software will not crash while operating as it may cause major damage to the project and the reputation of the organization. Similarly, in banking softwares and air traffic control, the stakes associated with software failures is so high that even the slightest defect in the software might cause huge losses in terms of money and lives. It is to be noted that the severity and the priority of a defect may not be the same. For example, it might be the case that a defect is required to be removed immediately as it is violating some copyright but the damage caused by the defect is not that high as it is not affecting the usability of the program. In such cases, the priority of the defect is high, but the severity is not. In this research, however, we are discussing the severity of the defect. A defect may not occur immediately, but if and when it occurs, it may cause some severe damage. In software testing, it is a well-known fact that the sooner the defects are recognized, the lesser it costs to correct them, hence, minimizing the overall cost of software development. Severity assessment can be used for this purpose as well, to assess the severity of the reported defects in time so that the defects can be rectified causing as less damage to the software as possible and thereby controlling the overall cost of the software. Software defect reports are an example of user generated content and are generated by the end-user of the software whenever the software does not perform in its intended manner. These defect reports are in the form of text, which is an example of highly unstructured data. Bug tracking software such as Jira, Bugzilla, Zoho etc. are integrated with the project development software in organizations and are used to keep a track of the defect. Most of these software programs allow the end-users to directly report a defect. These software programs are also known as issue tracking systems and their databases typically include the defect reports, time of occurrence of the defect, id of the person who reported the bug along with the severity of the defect. As already mentioned, these defect reports are in the form of the text and are highly unstructured. To assess the severity of defects from such an unstructured data we need to clean the data, and extract important words which will prove useful in predicting the severity of the defect.

Almost all the work that has been done previously has used standard data mining techniques or basic ML techniques, to comprehend software defect recognizers from historical records of static code features. Those data mining methods work only when the input data that is being fed to the model is highly structured, which is rarely the case, if ever. In this study, we proffer three severity classification models using ML & DL algorithms and apply it to a database that is largely unstructured. This method would provide us with a far better approach to tackle the problem at hand as compared to the previously explored data mining methods or nascent stage ML methods. Thus, we propose an automated method to assess the severity of the software defects by using a machine learning algorithm known as XGBoost and also a couple of deep learning algorithms, namely, CNN & RNN. One of the main contributions of this work is that it successfully addresses an important issue (automated defect severity assessment from unstructured or loosely structured text), which has been largely ignored by the research community. Though a lot of work has been done in this field, there's still a long way to go when it comes to the use of ensemble methods and deep learning techniques to predict the severity. In this study, text mining techniques have been employed along with XGBoost algorithm and deep learning techniques for predicting the severity of software defect reports. We have used XGBoost, which is the strongest existing ensemble machine learning method. It is trusted by a number of winning teams of machine learning competitions. A large number of teams that have won machine learning competitions organized by competition sites such as *kaggle, Topcoder etc.* have done so using the XGBoost algorithm. The performance of this method is comparable to deep learning methods. We have also employed deep learning techniques: CNN and RNN using word embeddings for the same.

1.2. RESEARCH OBJECTIVES

The research explores the various available techniques for multiclass text classification and applies it on the PITS-A dataset in order to analyze and compare those techniques. The aim is to automate the process of assignment of severity values to the software defect reports based on certain textual cues using ML & DL models. In this study, we have used two deep learning models, namely, Convolutional Neural Networks and Recurrent Neural Networks, and one machine learning model, namely, XGBoost. A primary approach to decipher the accurate severity level of a software defect report is to look for some text-based evidence. Therefore, the textual features are fed as input to the classifiers so that they can accurately predict the severity of a software defect report. All the three classifiers employed in this study, i.e., CNN, RNN and XGBoost are fed the exact same input features. The performance of each classifier is validated on the NASA's PITS-A dataset. The results of each of the classifiers are compared against one another. The main target is to achieve a high efficacy for the severity assessment tasks pertaining to software defect reports after performing tuning of different hyper-parameters of the afore-mentioned classifiers and deciding on which approach out of the three fares the best in terms of efficiency.

1.3. ORGANIZATION OF THESIS

The thesis has been divided into five chapters. Each chapter deals with one component related to this thesis. Chapter 1 being introduction to this thesis, gives us the brief introduction about the project, thereafter chapter 2 tells us about the literature survey which further includes related work section. Following up is chapter 3 which tells about the proposed work. Chapter 4 provides us with the experimental results followed by the final chapter, chapter 5, which is the conclusion of the thesis.

CHAPTER 2 LITERATURE SURVEY

The chapter explains various kinds of severity levels, the difference between severity & priority, and the work done so far in the field of automatic severity assessment of software defect reports.

2.1. TYPES OF DEFECT SEVERITY LEVELS

Severity defines the upper limit to which a particular software defect can hamper the normal functioning of an application or a system. Severity lets us know how devastating to the system a particular defect can be and what is its impact. Different organizations or bodies usually define different levels of severity based on their understanding of the topic. However, on a generic level there are four defect severity levels which are as follows:

1. Critical

A defect that brings the system to a complete halt or which disrupts the natural flow of the system resulting in the system's collapse is termed as a critical severity defect. If, for any particular reason, the application crashes or it becomes unusable, the defect is categorized as critical. For Example: If in a social media platform such as Facebook or twitter, after you have entered the correct username and password, instead of granting you access to your homepage, the application crashes or throws a fatal error message, the defect is registered as critical.

2. Major

If any particular major feature implemented in a program is not working in the way it was intended to work, such a defect is termed as a major severity defect. Basically, if a major feature is behaving in a manner that is contradictory to its defined use-cases & requirements, the defect is categorized as major. For Example: If in an email service provider such as Gmail or iCloud, the CC section is not taking more than one recipient, the defect is registered as a major defect because a major feature of the application is malfunctioning.

3. Minor

If any particular feature implemented in a program is not working in the way it was intended to work, but the impact on the program due to that difference is miniscule, such a defect is termed as a minor severity defect. In case of a minor defect, the particular feature does behave in a manner which contradicts its use-cases & requirements, but the consequences of such a defect are very narrow. For Example: In an email service provider such as Gmail or iCloud, there is a section called “Terms and Conditions” and in that section, there are several links pertaining to the terms & conditions of the website. If one of those links stop working, it is registered as a minor defect as it doesn’t have a huge implication on the workability of the application.

4. Low

Any sort of graphical or cosmetic defect including spelling errors, font issues, alignment issues etc. is categorized as a low severity defect. A low severity defect is one in which there is almost no impact on the functionality but it is still a valid defect by definition and it should be rectified. For Example: In an email service provider such as Gmail or iCloud, there is always a “License page”. If there is a spelling error or alignment issue on that page, such a defect is registered as a low severity defect.

2.2. SEVERITY VS PRIORITY

Severity of a defect report is the upper limit to which a particular software defect can hamper the normal functioning of an application or a system. Priority defines the sequence for the developers in which they should rectify the defects. A higher priority defect must be dealt with first compared to a lower priority one. Priority pertains to scheduling, whereas severity deals with functionality. Generically speaking, priority is classified into three types, namely, low, medium & high, whereas severity is classified into four types, namely, critical, major, minor & low. Priority dictates how soon a defect needs to be corrected, whereas severity depicts the seriousness of any particular defect. The priority of any defect is finalized after consulting with the client or manager, whereas the severity of any defect is finalized by the QA engineers. The value of priority is subjective in nature and it may change in the future depending upon the latest project situations. On the other hand, the value of severity is objective and is much less likely to change during the course of the project as compared to the priority value. The level of priority depends upon the customer’s requirements, whereas the value of severity depends largely on the technical aspect of the application. Basically, priority is driven by business value and severity is driven by functionality. High severity & low priority defect means that the defect is a critical one but it does not warrant an immediate rectification. On the other hand, High priority & low severity defect means that the defect is not a critical one but it needs to be fixed immediately.

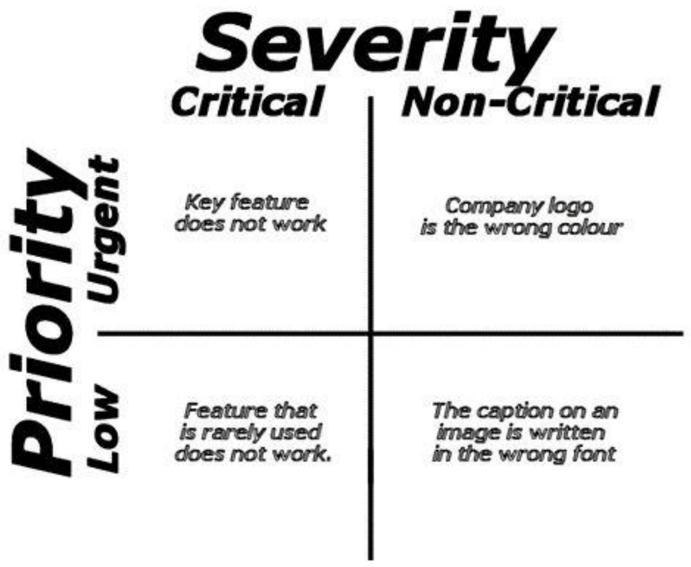
Severity
Critical
Non-Critical
Priority
Urgent
Low
Key feature does not work
Company logo is the wrong colour
Feature that is rarely used does not work.
The caption on an image is written in the wrong font

Fig. 2.1 Severity vs Priority

2.3. RELATED WORK

Menzies and Marcus [1] were the first ones who worked on the severity assessment of software defect reports and they used rule learning algorithms to do so. They designed a tool named SEVERIS. SEVERIS was applied to NASA’s PITS database. PITS contained data collected over ten years including issues of robotic satellite mission and human-rated systems. The system was applied to 5 datasets from PITS database: PITS A, PITS B, PITS C, PITS D, PITS E, which contained issues of five robotic missions of NASA. The tool was used to review reports and raise an alert if the predicted severity was anomalous. Cubranic and Murphy [2] used machine learning techniques for automatic bug triage. Bug triage refers to what needs to be done with a bug if and when it is reported. Their work was to assign the developers a particular bug based on the description of the bug as entered by the user and developers’ skills. They tested their approach on Eclipse bug reports and used Naive Bayes Classifier for classifying the bug reports. Lamkanfi *et al.* [3] compared various machine learning methods to assess the severity of reported bugs on three open-source systems: Mozilla, Eclipse and GNOME. They clubbed the six severity levels: blocker, critical, major, normal, minor, and trivial into two: severe and non-severe. Non-severe included trivial and minor whereas severe included blocker, critical and major. Normal severity bugs were not taken into account. The classifiers used by Lamkanfi *et al.* [3] were Naive Bayes, 1,2,3,4-Nearest neighbors, Naive Bayes Multinomial, and SVM using RBF kernel. Orthogonal Defect Classification is the most influential framework for defect classification and analysis. However, it does require intensive human labor and expertise of both ODC and domain knowledge to classify the defects. Huang et al. [4] worked to automate ODC by treating it as a supervised text classification problem by using SVM classifier. Patil [5] used Explicit Semantic Analysis to compute semantic similarity between defect reports and defect labels based on the concept. The defect label was assigned to the defect report based on its similarity with the defect report. Yang *et al.* [6] worked with three prevalent feature selection schemes: information gain, correlation coefficient, and chi-square to select the best features and finally used Multinomial Naive Bayes Classifier to predict the bugs. Yang *et al.* [7] worked on a textual emotion words-based dictionary, combining it with Naive Bayes Multinomial classifier to assess the severity of defect reports. Chaturvedi and Singh in [8] determined the bug severity on the software bug reports dataset obtained from NASA's PROMISE repository. They used the following machine learning techniques: Naïve Bayes, Support Vector Machine, Naïve Bayes Multinomial, k-Nearest Neighbour, J48, and RIPPER to predict the severity of bugs in the bug reports. Ramay *et al.* [9] applied deep learning methods to predict the severity of bug reports of two open-source systems: Eclipse and Mozilla. Similar to Lamkanfi *et al.* [3], they clubbed the six severities into two: severe and non-severe. They also considered the emotion score of the bug reports as users are very expressive about reporting the bugs. Senti4SD repository was used to calculate the emotion score of the bug reports. MNB, RF, CNN, and LSTM were used to predict the severity of the reported bugs.

Table 2.1: Work done in the field of Severity Assessment

Year of Publication	Author	Dataset	Techniques Used	Conclusion
2008	Menzies and Marcus [1]	PITS	Rule Learning	SEVERIS is a good predictor for issue severity levels.
2004	Cubranic and Murphy [2]	Eclipse	NB	NB classifier performed bug triaging with 30 % accuracy.

2010	Lamkanfi et al. [3]	Mozilla, Eclipse and GNOME.	NB, 1-NN, MNB, SVM using RBF kernel.	NB classifier outperforms all the others.
2015	Huang et al. [4]	403 defect records provided by the industrial company P.	ODC using SVM.	AutoODC classified defects with an accuracy of 80.2 %.
2017	Patil [5]	200 from Mahout JIRA, 200 from Lucene JIRA repository, and 100 from OpenNLP JIRA repository.	Explicit Semantic Analysis.	Concept based classification proves to be a promising approach for software defect classification.
2012	Yang et al. [6]	Eclipse and Mozilla.	Feature selection methods like information gain, correlation coefficient, and chi-square with MNB.	This study shows that feature selection can be used to improve the accuracy of severity prediction.
2017	Yang et al [7]	Eclipse, Android and JBoss.	Textual emotion words-based dictionary with MNB.	The proposed approach outperforms all the others.
2012	Chaturvedi and Singh [8]	PITS	NB, SVM, MNB, kNN, J48, and RIPPER.	Appropriate number of terms using InfoGain observed to be 125. Best F-Scores for severity level 2, 3

				& 4 by almost all ML techniques.
2019	Ramay et al. [9]	Eclipse and Mozilla.	Deep Neural classifier consisting of CNN using emotion score of defect reports.	The proposed approach outperforms the state-of-the-art by 7.90% in terms of f-score.

In this study, our goal is the same as Menzies and Marcus [1], that is to accurately predict the severity of bug reports and we have employed XGBoost algorithm and deep learning methods: CNN and RNN for the same.

CHAPTER 3 PROPOSED METHOD

In this project, we have worked on the dataset obtained from NASA's PITS database. The dataset contains defect reports in textual form along with the severity value of the defects. We divide the dataset in a ratio of 70% to train the machine learning and deep learning models and 30% to test the models. Since the text is highly unstructured, and there are no predefined features present in it as are present in structured data, we employ text mining techniques to extract the features from the data and reduce its dimension. After that, XGBoost (machine learning technique) and deep learning techniques are applied to assess the defect severity.

Preprocessing of data is performed before the textual features are fed into the classifiers. We tokenize the data, remove stop words and perform stemming. After preprocessing, in case of the XGBoost classifier, tf-idf vectoriser is used for feature extraction and then information gain is used for feature selection and then the features are finally fed to the classifier for it to generate an output. In case of CNN and RNN, GloVe word embedding is used to convert the data into a word-level matrix where each word is represented by a vector. Then, before feeding any input to the model, we create an embedding layer. The textual features are represented using the embedding layer which is then fed to the CNN and RNN classifiers. Both, CNN and RNN require input to have a static size and we know that sentence lengths can vary greatly. Therefore, we chose a maximum sentence length of 200, i.e., a sentence can have a maximum of 200 words only. If a sentence contained less than 200 tokens, a special stop word was repeatedly appended to the start of the sentence to meet the 200-word requirement. If a sentence contained over 200 words, only the first 200 were considered to be representative of that sentence.

The dataset is split into training and testing sets in the ratio of 70:30. In each run, there are different training and testing sets based on a partitioning variable (random state). The validation is performed on ten different values of partitioning variable to get more accurate and generalized results.

3.1. PREPROCESSING

Preprocessing is the task of preparing the data in a manner, which is easier for the machine learning model to comprehend. The raw data is transfigured into clean data which is then used as input to the model. The following preprocessing of the features was done to make them suitable for the severity assessment task:

Tokenization: Tokenization is the process of converting a text into tokens. These tokens can be sentences, words, or characters. For our project, we have tokenized the text up to word level. Word Tokenizer available in the NLTK library is used to tokenize the text into words. Cleaning of text is also performed wherein all the punctuation marks, special symbols, numbers and unnecessary spaces are removed from the text. Therefore, a sentence ["Hey! look, what's there?"] gets converted to a list of words as [Hey look what s there].

Stop Word Removal: Stop words are the most regularly occurring words in any language. These may be prepositions, conjunctions or interjections which are used very often in the text and do not really add to the meaning of the text. Words like a, an, the, this, that, and, in, it, etc. are omitted from the text. Stop words of the English language are available in NLTK library and can be imported from there. Stop words may also be specific to a particular application. For example, if there is an application related to the healthcare industry, the word 'doctor' might appear quite a lot of times and may not really add to the meaning of the text. In such cases, these additional words may be appended to the stop words list and all of them can be removed from the text in one go.

Stemming: Stemming and Lemmatization are performed to get the base/root form of each word. While both the techniques are used to get the base form of each word in the text, there is a major difference in how it is achieved in both the techniques. Stemming works by simply stripping off of any suffixes or prefixes that might be present with the base word. By using stemming 'run', 'running', 'runner' gets converted to 'run'. But there might occur a case of over-stemming and under-stemming. Often it might be the case that the words obtained after stemming may not make any sense because of the wrong context or wrong spelling. Lemmatization converts each word to its base form by checking the lexicon, i.e., we can say that the root words obtained after lemmatization are morphologically correct. By using lemmatization, ‘caring' gets converted to ‘care’ which would have been converted to ‘car’ using stemming. Lemmatization, therefore, takes more time than stemming. In this project, we have applied Stemming by using the *PorterStemmer*, as is the case in Menzies and Marcus [1].

3.2. MACHINE LEARNING METHODOLOGY

In this section, we are going to discuss the approach that we followed for the XGBoost classifier, from the feature extraction phase to the final output generation.

3.2.1. Feature Extraction: We have used the Tf-IDF approach to extract features from the textual data. Tf*IDF is the product of Term Frequency and Inverse Document Frequency. It assigns weight to each term in the text which signifies the importance of the term. The weight of each term is directly proportional to the number of occurrences of the term in a document while it is inversely proportional to the number of occurrences of the term in the corpus. It is used to normalize the weights of each term in cases where the occurrence of some terms is benefitted by the length of document.

Mathematically,

$$Tf * IDF = \frac{term}{totalTerms} \times log\,\frac{documents}{document[term]} \qquad (3.1)$$

where, *term* is the term in consideration,
totalTerms is the total number of terms in a document,
document[term] is the document containing term,
documents is the total number of documents.

3.2.2. Feature Selection: We have used Information gain as a measure for feature selection. Information gain is basically the calculation of entropy or surprise element in a dataset. If a dataset is split in a particular ratio *InfoGain* measures the entropy introduced in the dataset before and after the split. Information gain is low for high frequency terms and high for rare terms. We have used *MutualInfoClassifier* available in scikit-learn library of python, which gives the mutual correlation between a term and the outcome. It employs information gain in the background and tells us how much impact a term has in a particular result. Mutual information tells us how much information can be gained from a random variable.
We have selected the top 100 terms as ranked by the *MutualInfoClassifier's* score. These 100 terms act as the input features to our classifier. Rest of the features are not taken into further consideration. In this study, we have considered the top 100 features only.

3.3.3. XGBoost: XGBoost was Developed by Tianqi Chen and Carlos Guestrin at the University of Washington in 2014 [10]. Extreme Gradient Boost or XGBoost is a scalable decision tree-based ensemble machine learning algorithm that uses a gradient boosting framework. Ever since its introduction, XGBoost has proved to be the fastest of all the machine learning algorithms. In fact, it has a good competition with deep learning methods in terms of accuracy and score. Many teams that have won in various competitions organized by machine learning competition site Kaggle employed XGBoost. For unstructured data, neural networks still prove to be the most useful, but for small structured/tabular data, decision tree-based algorithm outperform all the other algorithms. XGBoost has given state-of-the-art results in many problems. It is for this reason that we have used this algorithm along with deep learning methods to predict the severity of software defect reports.

The architecture of the XGBoost algorithm is shown in Fig. 3.1. The process of boosting involves building strong classifiers from multiple weak classifiers iteratively. All samples in the dataset are assigned the same weights initially. The first weak classifier is trained by picking some of the samples from the dataset randomly. Every sample present in the dataset has an equal probability of being selected to be included in the training set. Each weak classifier tests all the samples and then updates the weights for all the misclassified samples. These samples with their updated weights are used for the training of the next weak classifier. These weak classifiers work in a consecutive manner. While predicting the results for a test sample, predictions made by all the classifiers is taken into consideration and the majority of these predictions is the final prediction.

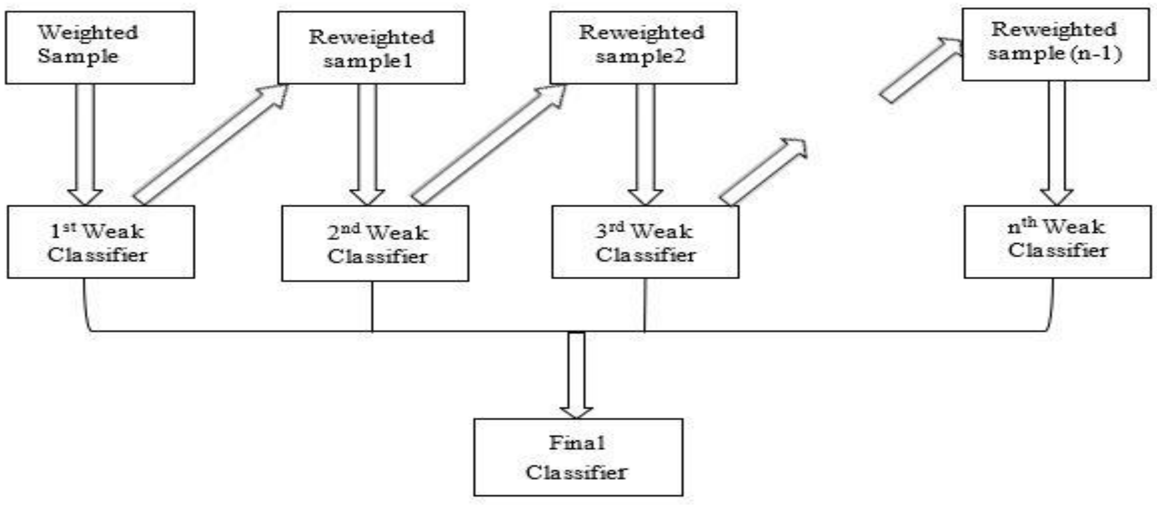

Fig. 3.1 The Architecture of XGBoost Algorithm

Regularization issues are not taken very seriously by most of the boosting algorithms. In Gradient Boosting Technique, some regularization parameters are present like maximum depth learning rate, minimum samples per leaf, etc. which can be used for controlling the tree structure. It is further improved by extreme gradient boosting. Therefore, extreme gradient boosting or XGBoost is a more regularized version of Gradient Boosted Trees.

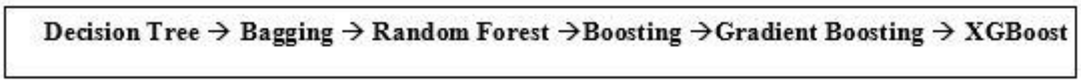
Decision Tree → Bagging → Random Forest →Boosting →Gradient Boosting → XGBoost

Fig. 3.2 Evolution of Tree-based Algorithms

The robustness of the model can be increased by changing the learning rate which minimizes weights on each step. We have taken 0.1 as the learning rate. The maximum depth of the tree is set to 6 to avoid overfitting of the model. The number of trees generated is given by the parameter n estimators and its value is set to 100. The objective function used is the multi-class version of the softmax function.

3.3. DEEP LEARNING METHODOLOGY

In this section, we are going to discuss the approach we followed for the deep learning classifiers, namely, CNN & RNN, from the creation of the embedding layer to the final output generation.

3.3.1 Embedding Layer: The embedding layer of a neural network converts an input from a sparse representation into a distributed or dense representation. Word Embedding facilitates natural language understanding by means of semantic parsing such that the meaning from text is extracted preserving the contextual similarity of words. In this research, we use the state-of-the-art pre-trained GloVe word embeddings model [11] to generate the word vectors. Basically, the vector representations for text are termed as word embeddings. Word embeddings are considered a major breakthrough when it comes to Natural Language Processing using Deep Learning. Each word is represented using a real-valued word vector often having tens or hundreds of dimensions. The vector values are learned in a way that resembles a neural network, and hence the technique is used in deep learning. The main principle behind using word embeddings is that words having similar meanings have similar representations. The GloVe word embedding used in this study is an extension to the word2vec method for learning word vectors. GloVe combines the global statistics of matrix factorization techniques like LSA with local context-based learning in word2vec. Each word is represented by the vector closest to the point obtained corresponding to that word. Similar words are closer to each other.

3.3.2 Convolutional Neural Network: Convolutional Neural Networks are several layers of convolutions. A convolution may be defined as a sliding window function applied to a matrix. These functions are non-linear activation functions like *relu* or *tanh*. The convolutions are used over the input layer to calculate the output. Each layer may use a different convolution.

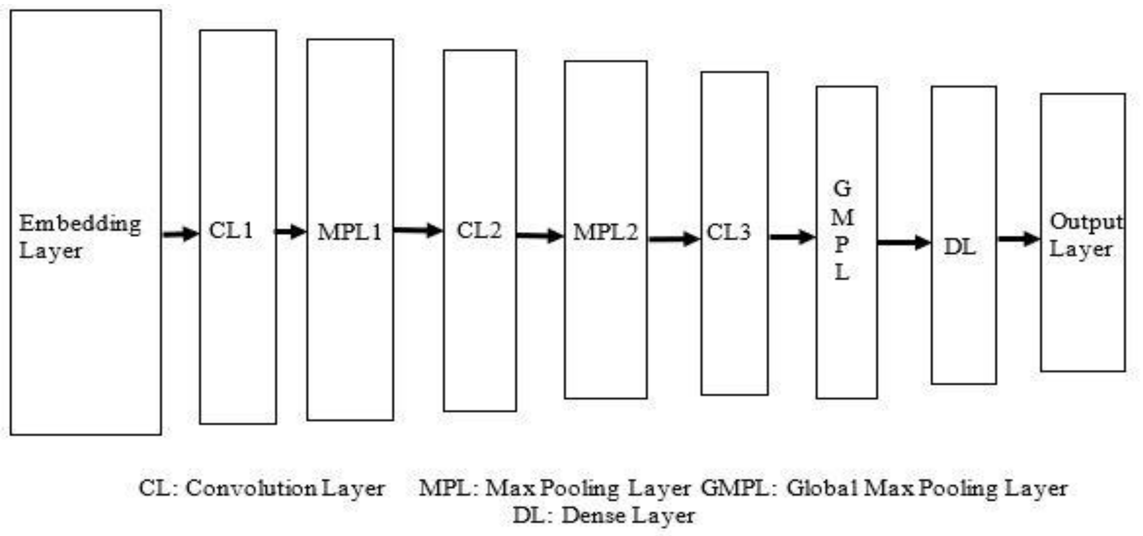

Fig. 3.3 The Architecture of CNN

A pooling layer is present after each convolution layer, which subsamples the output of the previous convolution layer and feeds it as an input to the next layer. The simplest pooling operation may be a max operation applied to the result of each filter. A global max pooling is performed at the end and its result is fed to the final fully-connected/ dense layer. Finally, the classification is performed by using a softmax activation function at the output layer. We use three 1D convolution layers with *relu* as the activation function, which decides the output. We have chosen the filter value to be 128 which indicates the number of neurons, and the value of kernel size is set to 3, which indicates the size of the filter. Global max pooling is performed, and the output of the global max-pooling layer serves as the input to a dense layer consisting of 128 neurons. Output layer maps input to a single output, which is the predicted severity of the corresponding bug report. We have used *categorical_crossentropy* as the loss function for the model, which is a cross-entropy loss that is used to measure the performance of a multi-class classification model.

3.3.3. Recurrent Neural Network: Traditional neural networks lacked persistence. They could not make an informed decision about an event based on the previous occurrences of that event. In simpler terms, they lacked memory. Recurrent Neural Networks were designed to overcome this issue. RNNs have loops in them, which allow the persistence of information. Traditional RNNs are able to connect previous relevant information to the present task, but when the gap between the relevant information and the current task becomes large, they fail, viz. they are unable to handle long term dependencies. To address this issue, a special kind of RNN was developed known as Long Short-Term Memory. LSTMs are capable of learning long term dependencies. It is in their innate nature to remember information for long periods of time. The repeating module in a LSTM has four neural network layers, which interact in a unique way as compared to the standard RNN that has just one neural network layer in its repeating module. The key components of an LSTM are a cell state and three gates. The LSTM can add or remove information from the cell state with the help of gates. The remarkable results that people have achieved using RNNs are due to the LSTM only. They are highly effective for text classification problems and are also strongly recommended for the same.

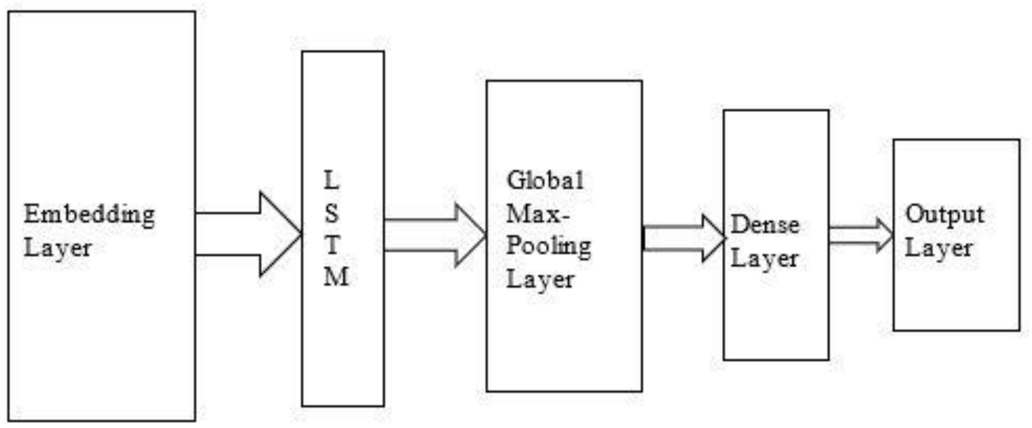

Fig 3.4 The Architecture of RNN

In our study, we feed the input to the RNN model using the embedding layer, which consists of the embedding matrix. The input goes through an LSTM network. The output of the LSTM network is passed through a global max-pooling layer. The output of the global max-pooling layer is passed onto a fully connected dense layer having 50 neurons and finally, the output layer declares its verdict on the classification of the defect report into one of the five available classes by using the softmax activation function.

CHAPTER 4 IMPLEMENTATION AND RESULTS

4.1. THE PITS-A DATASET

We evaluated our system on NASA’s original PITS A dataset. NASA’s PITS database contained data collected over ten years including issues of robotic satellite mission and human-rated systems. There are five datasets from PITS database: PITS A, PITS B, PITS C, PITS D, PITS E, which contained issues of five robotic missions of NASA. PITS A contained issues of one of the robotic missions of NASA. The dataset is available at https://zenodo.org/. There are five levels of severity, labeled 1 through 5. Label 1 is the most critical of all the bugs and may prove to be fatal to the project as well as humans involved in the project. Label 1 bugs are therefore always present in scarce amounts in such datasets as every possible measure is taken in order to avoid them. In the PITS A dataset, label 1 bugs are not present. Severity decreases with the rise in the number of labels, Label 5 being the most trivial of all the bugs. Label 5 is so trivial that more often than not the bug is corrected without even reporting. Therefore, there are very less reports having label as 5 which makes the dataset highly imbalanced.

We have used two attributes of the dataset: ‘Description’ attribute gives the textual bug report and ‘Severity’ attribute that gives the extent to which the bug can impact the system if not corrected. The total number of bug reports in the dataset is 965. Label 1 bugs are not present in this dataset, label 2 bugs are 325 in number, label 3 bugs are 375, label 4 bugs are 239 and there are 26 label 5 bugs.

Table 4.1: No. of Reports with Severity Levels in PITS A Dataset

PITS A DATASET	
Severity	**Number of Reports**
1	0
2	325
3	375
4	239
5	26
Total	**965**

4.2. PERFORMANCE MEASURES

We need to have an understanding of the below-mentioned four labels in order to understand the metrics used in this paper:

1) *True Positive (TP)*

If a sample that is positive is labeled as positive.

2) *False Positive (FP)*

If a sample that is negative is labeled as positive.

3) *False Negative (FN)*

If a sample that is positive is labeled as negative.

4) *True Negative (TN)*

If a sample that is negative is labeled as negative.

We have used the following evaluation metrics in our work:

Sensitivity:

Also called recall, sensitivity is the metric that gives the model's ability to predict true positives of each category, i.e., it provides the percentage of reports that have defects and are correctly predicted so. Mathematically,

$$Sensitivity = \frac{TP}{(TP+FN)} \times 100. \tag{4.1}$$

AUC Score:

Receiver Operating Characteristics is a probability curve that is created by plotting the True Positive Rate /Sensitivity against the False Positive Rate. AUC is the area under the ROC curve and represents the degree of separability. It indicates how capable the model is, in discriminating between the classes. The value of AUC lies between 0 and 1. Higher values of AUC represent the model's ability to distinguish between the classes more efficiently. In the multi-class model, we can plot multiple ROC curves corresponding to each class using one versus all approaches.

Accuracy:

Accuracy describes the closeness of a measurement to the true value. The accuracy is defined as the average number of correct predictions in the case of multi-class classification. Metrics calculation is done for each individual run and then overall metric calculation is done using the macro averaging. As the PITS dataset is imbalanced, we take one versus one macro average in the case of AUC as it is insensitive to class imbalance and gives a better picture of the model.

4.3. HYPERPARAMETER TUNING

Hyperparameter tuning is the process of trying out different combinations of parameters of a classifier in a bid to find the best possible combination that generates the most accurate output. In this study, experiments have been performed several times by using sets of different parameters each time in order to get the best possible result. The tables 4.2 through 4.4 below represent some of those set of parameters for XGBoost, RNN and CNN respectively.

Table 4.2: Hyperparameter tuning in XGBoost

XGBoost			
Max_Depth	**n_estimators**	**Learning rate**	**Accuracy**
4	50	0.1	73.45
5	100	0.1	76.21
6	200	0.1	76.55
6	100	0.01	77.41
6	100	0.1	78.62

Table 4.3: Hyperparameter tuning in RNN

RNN					
Embedding Dimension	**LSTM Units**	**Hidden Units**	**Epochs**	**Batch Size**	**Accuracy**
50	50	50	5	32	74
100	50	50	5	64	76
200	100	100	7	32	77
300	100	100	7	64	77
300	50	50	7	32	78

Table 4.4: Hyperparameter tuning in CNN

CNN						
Embedding Dimension	**Filters**	**Kernel Size**	**Hidden Units**	**Epochs**	**Batch Size**	**Accuracy**
50	64	3	128	6	32	75
100	64	3	128	6	64	77
200	128	4	128	7	64	77
300	64	3	256	7	64	78
300	128	4	256	7	64	78
300	128	3	128	7	32	79

4.4. FINAL PARAMETER VALUES

There are various parameters that have been used for each of the three classifiers, namely, CNN, RNN and XGBoost. These parameters were finalized upon after performing hyperparameter tuning. These are the parameters that would provide us with the best possible output in terms of accuracy from all three of the classifiers used. The tables 4.5 through 4.7 below represent the set of those final parameter values for XGBoost, RNN and CNN respectively.

Table 4.5: Parameters used in XGBoost

XGBoost	
Parameters	**Value**
Maximum depth	6
Objective	‘multi:softmax’
Num_class	4
N_estimators	100
Learning rate	0.1

Table 4.6: Parameters used in RNN

RNN	
Parameter	**Value**
LSTM Units	50
Hidden Units	50
Non-Linearity Function	ReLu
Optimizer	Adam
Dropout	0.5
MAX SEQ LENGTH	200
Embedding Dimension	300
Batch Size	32
Epochs	7
Loss	Categorical Crossentropy
Return Sequences	True
Trainable	True

Table 4.7: Parameters used in CNN

CNN	
Parameters	**Value**
Embedding	300
Filter	128
Kernel Size	3
Hidden Units	128
Epochs	7
Batch Size	32

Non-linearity function	ReLu
Optimizer	Adam
Dropout	0.5
MAX SEQ LENGTH	200
Loss	Categorical Crossentropy
Return Sequences	True
Trainable	True

4.5. EXPERIMENTAL RESULTS

We have divided the whole dataset into training and testing sets in a ratio of 70:30 based on different partitioning variables. For each partitioning variable, a unique set is generated corresponding to that variable and in each set, 70 percent of the data will be used as the training set and 30 percent will be used as the testing set. These same training sets will then be used by the XGBoost, CNN and RNN models to train, and the corresponding testing sets will be used by these models to predict the severity of defect reports. We have performed the validation on ten different values of the partitioning variable for XGBoost and five different values each for CNN and RNN to get more accurate and generalized results.

4.5.1. XGBoost: The following Table 4.8 shows the individual results for each severity for various runs of XGBoost and whereas the overall results of the model obtained by taking a macro average of the individual results are shown in Table 4.9. The results have been evaluated corresponding to the top 100 features with AUC, sensitivity, and accuracy as the evaluation metrics.

Table 4.8: Individual Results of XGBoost

XGBoost								
S. No.	Severity 2		Severity 3		Severity 4		Severity 5	
	AUC	Sensitivity	AUC	Sensitivity	AUC	Sensitivity	AUC	Sensitivity
1	0.88	81.44	0.79	81.03	0.78	64.79	0.75	50
2	0.87	77.78	**0.83**	**85.84**	0.81	70	0.81	62.50
3	0.88	81.44	0.82	85.83	0.74	55.56	0.80	60
4	0.90	85.10	0.78	78.45	0.78	65.28	0.75	50
5	0.88	82.24	0.76	76.19	0.77	61.76	0.65	30
6	**0.91**	**86.36**	0.78	78.76	0.78	65.82	0.75	50

7	0.85	77.57	0.80	79.63	**0.81**	**70.77**	0.80	60
8	0.88	80.95	0.76	81.03	0.72	52.44	0.81	62.50
9	0.90	86.11	0.80	81.73	0.77	60.87	0.78	55.56
10	0.90	84.91	0.78	78.43	0.76	61.11	**0.85**	**70**

As we can see from the above table 4.8 that the AUC measure of the model comes out to be 0.91 for severity 2 and the corresponding sensitivity comes out to be 86.36%. Whereas, as the severity is decreasing (increasing number), the AUC measure declines to 0.83 for severity 3, corresponding sensitivity being 85.84%. For severity 4, these values further decrease to AUC being 0.81 and sensitivity being 70.77%. However, for the severity value 5, the AUC measure shows a slight increase, and the value comes out to be 0.85 while the sensitivity is 70%. The trend in these values shows that the model is the most efficient in predicting high severity (severity 2) values, though, other severities do not lag behind by much. Considering AUC and sensitivity as the measure for performance evaluation, we can say that the model is consistent with respect to all the severity levels.

Table 4.9: Overall Results of XGBoost

XGBoost			
S. No.	**AUC**	**Sensitivity**	**Accuracy**
1	0.795	69	76.55
2	**0.82**	**74.02**	**78.62**
3	0.80	70.71	76.9
4	0.80	69.71	76.55
5	0.75	62.55	73.45
6	0.80	70.24	76.55
7	0.81	71.99	76.21
8	0.79	69.23	72.41
9	0.81	71.07	77.59
10	0.82	73.61	76.21

From table 4.9 we can see the overall results. The overall AUC which is obtained by the one versus one macro averaging comes out to be 0.82 while sensitivity comes out to be 74.02% with an overall accuracy of 78.62%.

4.5.2. CNN: The following table 4.10 shows the individual results for each severity for various runs of CNN, and the overall results of the model obtained by taking the macro average of the metrics used are shown in table 4.11. The results have been evaluated corresponding to the top 200 features and the evaluation metrics used are AUC, sensitivity, and accuracy.

Table 4.10: Individual Results of CNN

CNN								
S. No.	**Severity 2**		**Severity 3**		**Severity 4**		**Severity 5**	
	AUC	**Sensitivity**	**AUC**	**Sensitivity**	**AUC**	**Sensitivity**	**AUC**	**Sensitivity**
1	**0.95**	**87.23**	**0.89**	87.96	**0.91**	59.26	0.95	57.14
2	0.92	72.92	0.89	70.18	0.88	**84.51**	**0.96**	**72.78**
3	0.95	72.92	0.88	**92.97**	0.88	44.64	0.95	60
4	0.93	79.79	0.85	80.91	0.86	62.16	0.96	66.67
5	0.95	83.67	0.87	84.29	0.87	53.97	0.94	62.5

We can see from the above table 4.10 that the highest value of AUC comes out to be 0.96 for severity 5 with corresponding sensitivity being 72.78%. The second highest value is obtained for severity 2 with a sensitivity of 87.23%. For severity 3, the AUC comes out to be 0.89, and the highest sensitivity obtained is 92.97%. For severity 4, the AUC obtained is 0.91, and the highest sensitivity obtained is 84.51%. Considering AUC and sensitivity as the evaluation measure, we can say that the performance of the model is consistent and the model is efficient for all the severities. The following table 4.11 shows the overall results of CNN over five runs of the partitioning variable.

Table 4.11: Overall Results of CNN

CNN			
S. No.	**AUC**	**Sensitivity**	**Accuracy**
1	0.92	72.90	79
2	0.91	76.34	75
3	0.91	67.63	76
4	0.90	72.38	75
5	0.91	71.11	77

The overall results are calculated in a manner similar to that of XGBoost by taking the macro average of the individual results. The overall value of AUC is evaluated using one-versus-one macro averaging, which is insensitive to the class imbalance present in the data. The highest value of overall AUC comes out to be 0.92, while sensitivity and accuracy come out to be 76.34% and 79%, respectively. The high value of AUC suggests that the model is capable of differentiating between the classes.

4.5.3 RNN: Similar to XGBoost and CNN, RNN model performed various runs for different values of the partitioning variable. Individual results of each severity are being shown in Table 4.12. The results have been evaluated corresponding to the top 200 features, and the evaluation metrics used are AUC, sensitivity, and accuracy.

Table 4.12: Individual Results of RNN

RNN								
S. No.	**Severity 2**		**Severity 3**		**Severity 4**		**Severity 5**	
	AUC	**Sensitivity**	**AUC**	**Sensitivity**	**AUC**	**Sensitivity**	**AUC**	**Sensitivity**
1	0.93	77.08	0.87	**87.72**	0.88	57.75	0.95	66.66
2	**0.96**	**85.42**	**0.88**	85.16	0.90	55.36	**0.96**	50
3	0.94	79.57	0.87	79.84	**0.91**	**72.73**	0.96	**71.43**
4	0.95	82.24	0.87	74.76	0.90	71.83	0.96	55.56
5	0.96	83.33	0.87	78.07	0.86	61.97	0.94	55.56

From the individual results for each severity (Table 4.12) we can deduce that the AUC value for both severity 2 and 5 comes out to be 0.96 with their respective sensitivities being 85.42% and 71.43%. For severity 3, AUC value is 0.88, and the sensitivity is 87.72%. For severity 4, AUC value is 0.91, and the sensitivity is 72.73%. Considering AUC and sensitivity as the evaluation measure, we can see that the performance of the model is consistent. The value of AUC being close to 1 suggests the model's capability to differentiate between the classes.

Table 4.13: Overall Results of RNN

RNN			
S. No.	**AUC**	**Sensitivity**	**Accuracy**
1	0.91	72.3	76

2	0.91	68.98	78
3	**0.91**	**75.89**	**78**
4	0.91	71.09	76
5	0.89	69.73	75

Table 4.13 shows the overall results of the various runs of RNN obtained by macro averaging the individual results. The highest value of overall AUC comes out to be 0.91 while the highest sensitivity is 75.89% and the highest accuracy is 78%. The value of AUC being so close to 1 suggests the model’s capability to differentiate between the classes.

CHAPTER 5 CONCLUSION AND FUTURE WORK

In this chapter, we firstly provide a brief conclusion of this work and then summarize the whole thesis. At last, we suggest possible future work in order to better tackle the problem at hand.

5.1 CONCLUSION

Software programs are present in every aspect of our lives nowadays, and it is quite inevitable for these softwares to have defects. Users report these defects to the developers, and it is the task of the developer to assign a correct severity value to each defect in order to work on them in the right order. Our proposed approach employs XGBoost and deep learning techniques, namely, CNN & RNN to automate the severity prediction of defect reports. We have used the PITS A dataset available in NASA's PITS database to evaluate the model. The results were analyzed using the AUC measure and sensitivity as the metrics. We concluded that with XGBoost, the model performs better in predicting the high level severity (severity 2) defects as compared to the low level defects. Whereas in case of the deep learning techniques, CNN and RNN, the models were exceptionally well in predicting the highest as well as the lowest priority of the defect reports (severity 2 and severity 5), and for the rest of the severities, the models' performances were fairly consistent.

5.2 SUMMARIZATION

Our aim in this thesis is to predict the severity level for a real-time defect report submitted by a user. The PITS A dataset from NASA is used for carrying out this study.

In chapter 2, we review the non-technical and technical studies dedicated to severity assessment. The research field of automation of severity assessment of defects still has a long way to go. The development of such a model is of crucial importance for the IT industry. The chapter deals with the types of severity, the work done in this field and also the background studies that are important for performing the analysis. In this thesis, our target media object consists of textual data from the defect reports.

Chapter 3 illustrates the methodology proposed by us. In this study, we have undertaken two separate methodologies, i.e., one for ML and one for DL. We have utilized the PITS A dataset and perform preprocessing and feature extraction on the data. We further explained each of these classifiers in detail. The chapter also introduces all the features that are used as input for the ML and DL models.

Chapter 4 is where we show the implementation details, experimental setup and classification results. We have also shown the process of hyperparameter tuning of each classifier. We have explained the setting of various parameters that have been used for performing the experiments for each of the classifier. We have defined the proper distribution of the data in the PITS A dataset. Furthermore, we have analyzed our model individually for each of the four severities in the PITS A dataset as well as for all the severities combined. The results from the three classifiers are compared with one another to find out which classifier performed the best. The performance of all three classifiers was fairly consistent throughout the dataset.

5.3 FUTURE SCOPE

In our study, we have used only the PITS A dataset out of the NASA's repository which consists of four other datasets as well, i.e., PITS B, C, D and E. In the future, this study can be extended to include those other datasets as well. A severity assessment model should be generic in nature and not be dataset specific. Thus, the approach used in this study can be applied on other software defect datasets such as Eclipse, Mozilla etc as well. Another prospect for the future could be using meta data along with the textual data. Various datasets consist of some meta data related to the defect reports. This data can also be used as input to the classifiers in order to enhance their performance. Lastly, in case of our study, we have taken top 100 features in case of XGBoost and we have set maximum sequence length at 200 for CNN and RNN. These types of limitations are kept in place in order to generate the output at a faster pace but they marginally reduce the efficiency of the system. It is a trade-off that is necessary as per the latest advancements in the field of machine and deep learning but in the future, we can look for ways to reduce the computation time without having to reduce the efficiency.

APPENDICES

APPENDIX 1: LIST OF PUBLICATIONS (PUBLISHED)

Application of XGBoost Algorithm and Deep Learning Techniques for Severity Assessment of Software Defect Reports

Ruchika Malhotra

Department of Computer Science and Engineering, Delhi Technological University, Delhi, India
ruchikamalhotra2004@yahoo.com

Akanksha Chauhan

Department of Computer Science and Engineering, Delhi Technological University, Delhi, India
akankshac36@gmail.com

Abstract

Software is present in every aspect of our everyday life, and defects are bound to be found during the testing of the software, no matter how small. It is, therefore imperative for software testing engineers to assess the severity of software defects to allocate proper resources for the correction of the defects and prevent software crashes. In this paper, we have proposed the use of the Extreme Gradient Boosting Technique (XGBoost) and deep learning techniques: CNN (Convolutional Neural Network) and RNN (Recurrent Neural Network) to predict the severity of the defects occurring in the software. AUC and sensitivity are the metrics used to evaluate the results. All three techniques: XGBoost algorithm, CNN, and RNN have performed really well in predicting the severities for all the defects. It has also been noted that XGBoost algorithm is the most efficient in predicting high severity defects, while the performance of deep learning techniques is excellent for the highest as well as the lowest severity defects. Also, for the rest of the severity values, the performance of both CNN and RNN is fairly consistent.

Keywords**:** deep learning; software defects; severity; severity assessment.

1. Introduction

The severity of a defect may be defined as the impact of failure on a software. In critical systems, mainly real-time systems like those of NASA, it is highly imperative that the testing engineers are sure that the software will not crash while operating as it may cause major damage to the project as well as the reputation of the organization. Software is bound to pick up some defects while in the development phase, and it is the job of the testing engineers to assess these defects and their impact on the software. Note that the severity and priority of a defect may not be the same. For example, it may be the case that a defect is required to be removed immediately as it is not letting the users/customers to proceed further but the damage caused by the defect is not that high. In such cases, the priority of the defect is high, but the severity is not. Here, however, we are discussing the severity of the defect, i.e., the defect may not occur immediately, but if and when it occurs, it may cause some severe damage.

In software testing, it is a well-known fact that the sooner the defects are recognized, the lesser it costs to correct them, hence, minimizing the overall cost of software development. Here, we propose an automated method to assess the severity of the software defects by using machine learning and deep learning techniques. Software defect reports are generated by the user of the software whenever the software does not perform in its intended manner. Software such as Jira and Bugzilla are used to report the defect. These defect reports are in the form of text, which is an example of highly unstructured data. Though a lot of work has been done in this field, there's still a long way to go when it comes to the use of ensemble methods and deep learning techniques to predict the severity. In this paper, text mining techniques have been employed along with XGBoost and deep learning techniques for predicting the severity of software defect reports. We have used XGBoost, which is the strongest ensemble machine learning method at the time of writing of this paper. It is trusted by a number of winning teams of machine learning competitions. The performance of this method is comparable to deep learning methods. We have also employed deep learning techniques: CNN and RNN using word embeddings for the same.

REFERENCES

[1] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports”, 2008 IEEE International Conference on Software Maintenance, Beijing, 2008, pp. 346-355.

[2] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization”, Proceedings of the International Conference on Software Engineering Knowledge Engineering, Alberta, 2004, pp.92-97.

[3] A. Lamkanfi, S. Demeyer, E. Giger, B. Goethals, "Predicting the severity of a reported bug", Proc. 7th IEEE Working Conf. Mining Softw. Repositories (MSR), May 2010, pp. 1-10.

[4] L. Huang, V. Ng, I. Persing et al., “AutoODC: Automated generation of orthogonal defect classifications”, Autom Softw Eng 22, 2015, pp. 3-46.

[5] S. Patil, "Concept-Based Classification of Software Defect Reports," 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 182-186.

[6] C. Z. Yang, C. C. Hou, W. C. Kao, X Chen, “An empirical study on improving severity prediction of defect reports using feature selection”, Proc. 19th Asia-Pacific Software Engineering Conference, 2012, pp. 240-249.

[7] G. Yang, S. Baek, J.-W. Lee, B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects", Proc. Symp. Appl. Comput., Apr. 2017, pp. 1280-1287.

[8] K. K. Chaturvedi, V. B. Singh, "Determining bug severity using machine learning techniques", Proc. CSI 6th Int. Conf. Softw. Eng. (CONSEG), Sep. 2012, pp. 1-6.

[9] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu and I. Illahi, "Deep Neural Network-Based Severity Prediction of Bug Reports”, in IEEE Access, vol. 7, 2019, pp. 46846-46857.

[10] T. Chen, C. Guestrin, “XGBoost: A Scalable Tree Boosting System”, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785-794.

[11] J. Pennington, R. Socher, C. Manning, “Glove: Global Vectors for Word Representation”, EMNLP. 14, 2014, pp. 1532-1543.

[12] A. Lamkanfi, J. Péérez, S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information", Proc. 10th Work. Conf. Mining Softw. Repositories (MSR), May 2013, pp. 203-206.

[13] A. Lamkanfi, S. Demeyer, Q. D. Soetens, T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug", Proc. 15th Eur. Conf. Softw. Maintenance Reeng., Mar. 2011, pp. 249-258.

[14] G. Yang, T. Zhang, B. Lee, "An emotion similarity based severity prediction of software bugs: A case study of open source projects", IEICE Trans. Inf. Syst., vol. E101.D, 2018, pp. 2015-2026.

[15] R. Jindal, R. Malhotra, A. Jain, “Analysis of Software Project Reports for Defect Prediction Using KNN”, Lecture Notes in Engineering and Computer Science, vol. 2211, no. 1, Jul. 2014, pp. 180–185.

[16] R. Jindal, R. Malhotra, A. Jain, "Software defect prediction using neural networks”, Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization, Noida, 2014, pp. 1-6.

[17] R. Malhotra, N. Kapoor, R. Jain, S. Biyani, “Severity Assessment of Software Defect Reports using Text Classification”, International Journal of Computer Applications, 83, 2013, pp. 13-16.

[18] K. Moran, ‘‘Enhancing android application bug reporting,’’ in Proc. 10th Joint Meeting Found. Softw. Eng., New York, NY, USA, Aug. 2015, pp. 1045–1047.

[19] X. Xia, D. Lo, X. Wang, and B. Zhou, ‘‘Accurate developer recommendation for bug resolution,’’ in Proc. 20th Working Conf. Reverse Eng. (WCRE), Oct. 2013, pp. 72–81.

[20]M. Sharma, M. Kumari, R. K. Singh, and V. B. Singh, ‘‘Multi attribute-based machine learning models for severity prediction in cross project context,’’ in Computational Science and Its Applications—ICCSA. Cham, Switzerland: Springer, 2014, pp. 227–241.

[21]Y. Tian, D. Lo, and C. Sun, ‘‘Information retrieval based nearest neighbor classification for fine-grained bug severity prediction,’’inProc.19thWork. Conf. Reverse Eng., Washington, DC, USA, Oct. 2012, pp. 215–224.