Bug Report Prioritization

Using Hierarchical Attention Networks

By

Anurag Yadav : 2020BCS-013

Under the Supervision of

Dr. Santosh Singh Rathore

Department of Computer Science

विश्वजीवनामृतं ज्ञानम्

ABV-INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

AND MANAGEMENT GWALIOR

GWALIOR, INDIA

DECLARATION

I hereby certify that the work, which is being presented in the report, entitled Bug Report Prioritization, in fulfillment of the requirement for the Colloquium Based on Summer Internship (BCCS-4105) and submitted to the institution is an authentic record of my/our own work carried out during the period May-2023 to July-2023 under the supervision of Dr. Santosh Singh Rathore. I also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

Dated: **Signature of the candidate**

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dated: **Signature of supervisor**

Acknowledgements

I want to extend my sincere gratitude to everyone who helped make this seminar report possible. I want to start by expressing my sincere gratitude and respect to my boss, Dr. Santosh Singh Rathore, who has been the driving force behind this work. I owe him a great debt for his invaluable advice, ongoing support, and insightful criticism of my work. I had the good fortune to have such a mentor who allowed me the freedom to think for myself and consider novel concepts. More importantly, I want to express my gratitude for the patience he displayed while carefully reading, commenting on, and revising the manuscripts for this dissertation. His commitment and dedication to his work have always inspired me, and they will continue to do so. I am positive that without his assistance, I would not have been able to complete my degree in a proper and timely manner.

I am thankful to ABV IIITM for providing the research scholarship to undertake my research work. Finally, I want to express my gratitude to my parents and friends for their unwavering support.

Anurag Yadav

Abstract

Software consumption has increased a lot in recent years. These software systems receive lots of bug reports. A Traigger is a person or program who reads he report and classifies the bugs and assigns priorities to these bugs so that bugs which are necessary to be fixed urgently could be fixed on time.Manually assigning priority to bugs is a time consuming task. Previously several other Machine Learning and Deep Learning techniques have been employed for the task of Bug Report Prioritization. Some models gave poor results whereas some models gave better results but their result were found to be inconsistent on newer Bug Reports. Thus, this project focuses on the prioritization of software bug reports using a Hierarchical Attention Network (HAN) model. The objective is to effectively handle class imbalance in bug report data and improve classification accuracy. The bug reports are preprocessed through text normalization, tokenization using the DistilBERT tokenizer, and attention mechanisms to capture hierarchical relationships within the text. The model's architecture incorporates bidirectional GRU layers and attention mechanisms at both word and sentence levels. The implementation includes data preprocessing, resampling using ADASYN for class imbalance, model training, and performance evaluation. The achieved results highlight improved classification performance through the HAN model, demonstrating its potential for enhancing software bug report prioritization.. The proposed model is evaluated on the following metrics : Precision, Recall and F1-score and it is compared with the state-of-the-art methods.

Keywords : Software Bug Reports, Bug report Prioritization, DistillBERT, Heirarichal Attention Networks (HAN), ADASYN, Deep Learning techniques, Software Reliability.

Contents

List of Figures

1 Introduction

1.1 Introduction

As software utilization continues to expand across various domains, software systems inevitably receive a substantial influx of bug reports. These reports are typically reviewed and categorized by individuals or automated triaging systems, which assign priorities to ensure that critical issues are promptly addressed. However, the manual assignment of priorities is a labor-intensive endeavor, prompting the need for an efficient solution. This paper proposes an automated bug report prioritization approach using deep learning models to alleviate the challenges associated with manual prioritization.

Businesses and enterprises frequently release software with imperfections due to the limitations of testing procedures. Developers rely on users to report defects using issue tracking platforms such as JIRA, GitHub Issue Tracker, Bugzilla, Mantis, and Google Code Issue Tracker. These platforms often employ unstructured natural language descriptions to capture bug details.

The act of reporting defects serves as a valuable contribution to the software maintenance process. A pivotal aspect of software maintenance involves effectively managing and addressing user-reported issues. As software systems increase in scale and complexity, the influx of problem reports grows exponentially, amplifying the intricacy of resolution efforts. Statistics reveal that during the months of January to July 2000, both Mozilla and Eclipse encountered an average of approximately 170 and 120 new bug reports, respectively.

This paper aims to streamline the prioritization process by leveraging advanced deep learning models, contributing to more efficient bug resolution and enhanced software quality.

1.2 Bug or Defect

As software applications are developed, systems regularly display unexpected behaviour and side effects that might be linked to bugs or flaws. These issues typically arise when creating software systems. Even though some technologies can analyse and record these errors, repairing them can be a difficult and time-consuming task. Simply described, a software bug is a flaw, error, or defect in a program or system that causes expected behaviour. A “defect” is described as a flaw or malfunction in a system component that prevents it from meeting its intended requirements and calls for repair or replacement in accordance with the ISO/IEC/IEEE 24765 standard. “Bug” and“defect” are synonymous concepts.The opposite is a ”bug” in a circumstance or weakness in a system that, if used or activated, could possibly cause an error in a component. In essence, multiple file problems could cause a single issue.

Repairs are actions taken to address one or more flaws, whereas debugging is a technique used to identify and fix issues. A fix frequently entails the elimination of a bug, although it can also consist of several smaller fixes to address a significant bug. The end result is the creation of“co-fixes,” which may solve a number of issues in various files. Since defects may have significant effects despite being the result of minor code flaws, they can be difficult to locate and correct. There are very few times when a user cannot detect a bug in the code. As an illustration, a bug could be present in a part of code that is either never run or is only ever performed under unusual circumstances, resulting in dead code. It’s possible that a number of input values or settings came together to generate this issue. A bug appears when all the requirements for a flaw or the sequence of circumstances required to bring about a flaw or imperfection in the system come together.

1.3 Software Bug Report

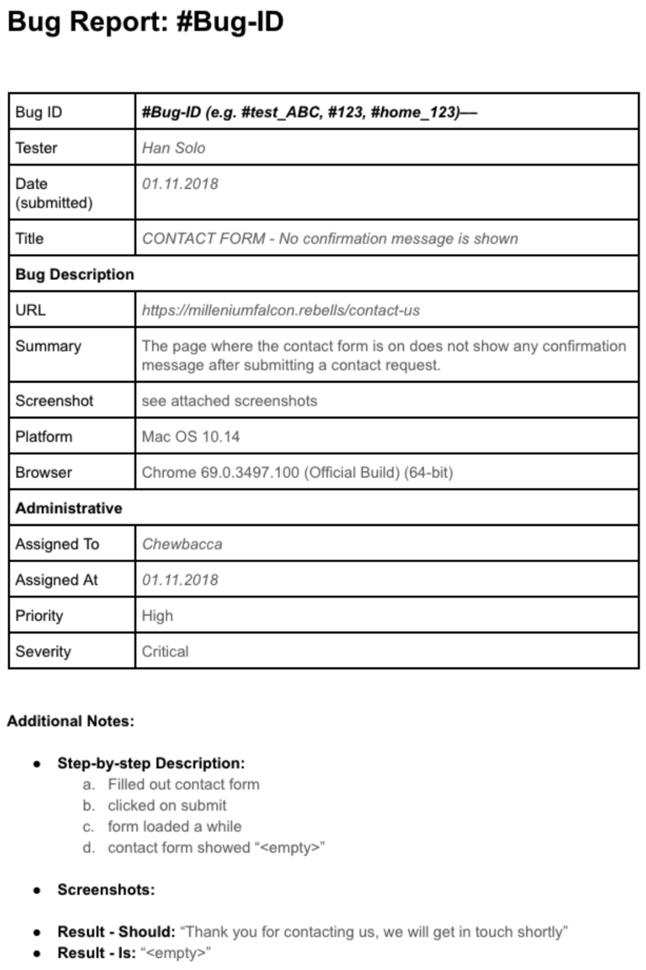
Developers and maintainers frequently deal with a large volume of bug reports when creating and maintaining software systems. To effectively resolve these problems, they must assess and rank the bug reports in accordance with their levels of seriousness, giving more weight to critical and severe flaws. A thorough report detailing the problems with the software and potential fixes is known as a bug report. The term “defect report” as well as “fault report,” “failure report,” “error report,” “issue report,” and “trouble report” can all be used to refer to this document. The bug report often includes crucial information which contains such the identification number of the bug , title, date of reporting and modification, severity, the developer in charge of resolving the flaw, and the bug's status, including whether it has been fixed or not. Figure 1.1 contains an illustration of a bug report.

Bug reports are necessary for keeping the software operational and play a critical part in maintaining the product's performance. Data mining techniques including clustering, classification, and retrieval of information have been used in several studies over the past ten years to prioritise problem reports. These methods aid in preventing improper bug prioritisation and guarantee that critical issues are fixed quickly and effectively. Software developers and maintainers can more effectively direct their resources and efforts towards fixing serious and important flaws by analysing and prioritising bug reports, which enhances the functionality of the software as a whole.

1.4 Bug Report Priority

Bug prioritising determines the order in which issues should be repaired based on their severity, which reflects the level of risk they pose and priority level. It's crucial to start with the most serious flaws and assign the proper priorities in order to handle big software maintenance quickly. Bug reports should be given the appropriate priority throughout the prioritising process because a delay in correcting major problems may have severe repercussions. The priority level given to a problem determines how quickly it will be solved. QA analysts and software testers must therefore understand the ideas of priority in order to manage their work. Bugs can have a priority level ranging from Pl, which denotes the highest priority, to P5, which denotes the lowest importance. There are five categories for bug priority levels: P1, P2, P3 P4. and P5 [2]. They priority levels are explained below:

- P1-A bug categorized as P1 or "Blocker" denotes a critical issue that requires immediate attention as it can cause the application to malfunction or become une



Bug Report: #Bug-ID

Bug ID	***#Bug-ID (e.g. #test_ABC, #123, #home_123)—***
Tester	*Han Solo*
Date (submitted)	*01.11.2018*
Title	*CONTACT FORM - No confirmation message is shown*
Bug Description	
URL	*https://milleniumfalcon.rebells/contact-us*
Summary	The page where the contact form is on does not show any confirmation message after submitting a contact request.
Screenshot	see attached screenshots
Platform	Mac OS 10.14
Browser	Chrome 69.0.3497.100 (Official Build) (64-bit)
Administrative	
Assigned To	*Chewbacca*
Assigned At	*01.11.2018*
Priority	High
Severity	Critical

Additional Notes:

- **Step-by-step Description:**
  a. Filled out contact form
  b. clicked on submit
  c. form loaded a while
  d. contact form showed "<empty>"
- **Screenshots:**
- **Result - Should:** "Thank you for contacting us, we will get in touch shortly"
- **Result - Is:** "<empty>"

Figure 1.1: Example Bug Report

usable.

- P2-Bugs can significantly impact the fundamental functionality of one or multiple application, thus resolving them expeditiously within a few days is imperative.
- P3- The matter at hand is not urgent and can be dealt with during the regular course of activities, for instance, in the second sprint.
- P4-The defect is not of high severity and can be dealt with once the more important bugs are resolved.
- P5-Bugs that have no impact on the core functionality of the software and may or

may not be addressed in the future.

1.5 Organization of the Report

The report is separated into form of 5 chapters. Chapter 1 consisted of Introduction. A brief description of what the remaining chapters consist of is discussed below:

- Chapter 2 mentions about the existing methods for Bug Report Prioritization.
- Chapter 3 describes the Research Gaps, the problem statement and the proposed methods objectives
- Chapter 4 discusses the proposed methodology and the work done.
- Chapter 5 disucsses the Results
- Chapter 6 discusses the Conclusions and the Future Scope.

2
A Review on Bug Report Priortization

In the initial phase of the research, a major focus has been given to reviewing existing state-of-the-art classification methods. Overall task can be divided into following major subtask, i.e, identifying the existing model implementations for the problem, looking for new ideas to implement on the classification task. Several methods have already been used for classification of bug reports priority, but there is a scope for improvement with the newer concepts like Hierarichal Attention Networks(HAN), and transformer model like BERT.

2.1 Review on Bug Priortization Techniques

Tan et al. [6] proposed a novel method of determining severity. The “Mozilla”, “Eclipse”, and “GCC” bug reports were linked to the bug repository entry on stack overflows. To determine the severity of defects, three classification algorithms-namely ,K-Nearest Neighbour (KNN) , “Naive Bayes”, and “Long Short-Term Memory” (LSTM) were used. The trials findings revealed a increase in the mean F1-score using the suggested method of 23.03 percent, 21.56%, and 20.59%.
Xiao et al. [7] for automatic bug localization, DeepLoc, an improved Convolutional Neural Networks (CNN)-based model, was proposed. CNN-inspired word embedding techniques were used by DeepLoc to replace the functionality of bug reports and source files The experiments were conducted on 18,500 bug complaints that were taken from 5 different places : Aspect J. Eclipse UI, JDT, SWT, and Tomcat between 2001 and 2014 They compared the effectiveness of DeepLoc to the 4 localization of bug strategies BugLo cator, LR+WE, HyLoc, and DeepLoc. According to the trial findings, using DeepLoc has increased “Mean Average Precision” (MAP) for bug localization from 10:87% to 13-1% when compared to regular CNN.
Shakripur et al. [8] employing the Time “TF-IDF weighting” method, the ABA-TF-IDE time-based methodology was proposed. The information was gathered from the Version Control System (VCS) software repository, where other project details and source code updates are kept. The model was trained using four machine learning algorithms:

"Support Vector Machine (SVM), Naive Bayes, Vector Space Model (VSM), and Smooth Unigram Model (SUM)". The findings demonstrate that the suggested method functioned satisfactorily, with "Mean Reciprocal Ranks (MRR)" up to approximately 12

Lamkanfi et al. [9] investigated whether it is possible to forecast the severity of a reported defect by studying its description which is in the form of text on which text mining techniques can be applied. They used a crude Bayes algorithm on historical information gathered from the open-source commu- nity ("Mozilla, Eclipse, and GNOME"). They found that "GNOME" 's overall perfomance was superior than "Mozilla and Eclipse", with precision and recall ranging from 0.70-0.85 for GNOME. Abdelmoez et al. [10] proposed a method to forecast the priority of bug reports using naive Bayes classifier. They made use of data from four different systems from the Mozilla, Eclipse, and GNOME open-source projects, which are all very large. The bug reports were ranked in order of mean time.

For the purpose of predicting the priority of bug reports, Qasim et al. [11] developed an automated categorization system ("DRONE"). They classified priority using linear regression (LR) and obtained an average F1-score nearly 29.

According to Tian et al. [13], it was able to predict the importance of bug reports by utilising the nearest neighbour method to identify precise bug report labels. On a larger collection of problem reports, which included over 65,000 Bugzilla complaints, they applied the suggested methodology. A convolutional neural network-based automatic method to forecast the multiclass priority for bug reports was put out by Hui et al. [1]. The suggested method greatly outperforms cutting-edge methods and raises the average F1-score by more than 24%.

An SVM-based model for priority prediction was recently developed, according to Choudhary et al. [14], and it prioritises Firefox crash reports in the "Mozilla Socorro server" based on the frequency and entropy of the crashes. Table 2.1 lists a few methods for prioritising bugs along with their drawbacks.

Table 2.1: Review on Bug Priortization

Technique Names	**Description**	**Limitations**
CLASSIFICATION DRONE with GREY engine [15]	Labels the priority of bug reports. Contains new GREY classification engine that improves linear regression with our thresholding method to handle unbalanced bug report data.	The learning thresholds may not be incorrect since the validation data differs from the test data.
Information-gain and Chi-square are contrasted with Naive Bayes Multinomial (NBM) and K closest Neighbour (KNN). [16]	The TDM matrix of the dictionary of words is trained using algorithms after the most informative phrases are selected from a corpus of the matrix.	Information gained: The severe categories are less precise than the non- severe ones. Chi square: SWT compo- nent's classification of bug severity has the lowest accuracy
Context-Based Pri- oritization using impact analysis te- chnique VSM [17]	Utilises an issue tracking system to assist projects that use issue-driven software development in maintaining a list of issues,	The study was limited to one project and one metric, which has an influence on the reliability of the findings.
"Support Vector Machine (SVM)" [11]	A supervised ML method that classifies and carries out regression ou data Utilises a kernel approach to alter data and identify the best border between two potential outputs.	Performance across all priority levels is in- consistent, and trivial reports were given P1 priority.
Logistic Regression (LOGR) [18]	Using a technique called logistic regression, which takes a number of independent variables as input and forecasts the output as a categorical dependent variable. It is possible to classify bugs automatically using the taxonomy.	The target label is more likely to forecast less accurate outcomes if it does not correlate with the problem report features.

Table 2.2: Review on Bug Priortization

Technique Names	**Description**	**Limitations**
K Nearest Neighbor, TF-IDF [19]	The method needs stack traces from the bug data set as well as its categorical attributes in order to assess a problem's severity	It employs instance learning and is a lazy learner. Every time, the temporal complexity is O(n).
Random Forest (RF) (20)	Methods for analysing and cate gorising code according to Clean, Smelly, Buggy, and Harmful categories, boosting programmes like Ada Boost	The algorithm can become ineffective when dealing with a lot of trees since it can be quite sluggish.
Convolutional Neural Network (CNN) [21]	An automated method to forecast the multi-class priority of bug reports,	Due to the "Maxpool" dataset's size and the difficulty of processing and training it on a subpar GPU, the classifier operates substantially more slowly.

3

Problem Statement based on Detected Research Gaps

3.1 Research gaps

The process of prioritising problems or defects heavily relies on automated techniques that can assist software engineer, developers and various business shareholder in processing a huge number of bug reports. These systems allow for the anticipation of a lang's importance and the discovery of the bugs in the code base using specialised algorithms. In a business setting, it is useful because these techniques can cut down on time. Because ML algorithms may automatically assign the proper large severity and developers with the necessary skills to fix an imur, the repair and maintenance costs may be lowered. The outcome from these preliminary steps is used as training data for ML classifiers. In order to prioritise bugs properly, a number of actions must be carried out in the proper sequence. It cannot provide effective outcomes if these first stages are not effectively carried out or if some procedures, are skipped. Few challenges faced are :-

- Managing new bugs: When bug reports are generated, it is discovered that the priorities given are inconsistent.
- Time Requirement - Some models use instance learning, which is time-consuming to run and assign priorities to.
- Bug report evaluation - Analysing the reports to decide what level of importance should be given to the bug
- Computation difficulty - The complexity of computation increases as the dataset gets bigger.
- Inconsistent findings: Testing on new datasets produces erratic results.

3.2 Problem Statement

Given a bug report the aim is to develop a model which can assign priorities to bug reports which gives accurate results with good consistency and can handle new bug reports with consistency.

3.3 Objectives

The proposed method aim to fulfill the following Objectives

- To create a model that can automatically prioritise bug reports using deep learning approaches after preprocessing the data using NLP techniques.
- To train the suggested title on open-source bug reports from “Mozilla”, “Eclipse”, etc., and analyse the differences between the results to the state-of-the-art approach.

4 Proposed Methodology

4.1 Proposed methodology

By downloading bug reports from Bugzilla, a bug report dataset is constructed in the first phase. Long reports from the Eclipse JDK, Eclipse Platform, Mozilla.com, and Mozilla Firefox are created in this document. The columns in this data include Issue.id and Component. Title of the Bug, The bug's description and the priority given to the bug report. The dataset's features are utilised to assess the severity of the problem. Title, Description, and other features will be provided to the model as inputs, and the model will then predict the label, which in this case is priority. The block diagram for dataset preprocessing is shown in Fig. 4.1.

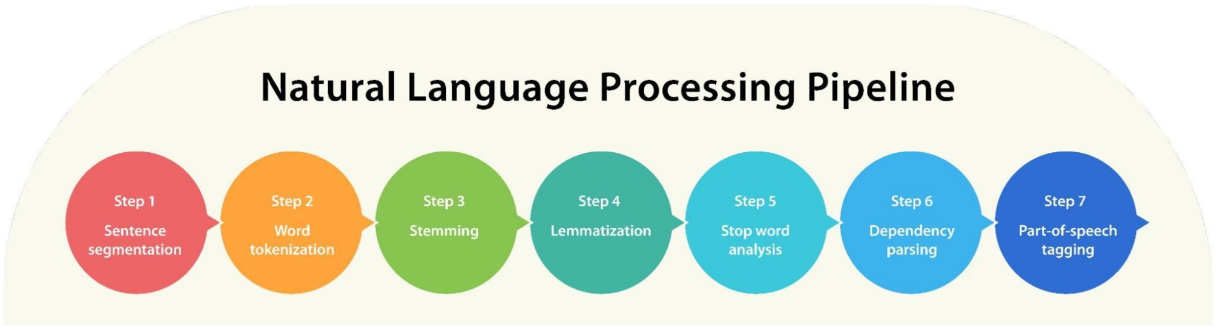


Figure 4.1: Text preprocessing

To complete this task of automatic assigning of priority to bug reports in this paper we propose a novel prototype in which data is pre-preprocessed using techniques, such as Tokenization, Spell correct, Stop-word removal, Lemmatization and then is data is passed to DistilBert Tokenizer. The DistillBert tokenizer use WordPiece algorithm to break down input text into individual tokens, which are then converted into numerical representations that can be processed by machine learning model. These numerical representations are passed to DistilBert model for training Fig 4.2 shows the block diagram of the proposed model.

DistilBert is a smaller, distilled version of the popular BERT (Bidirectional Encoder Representations from Transformers) model. DistillBert tokenizer will be used in the text preprocesseing. The key innovation behind DistilBert is that it uses a "distillation" technique to compress the larger BERT model into a smaller version. During this process, the model's architecture is simplified, and some of its parameters are removed. DistillBert uses a WordPiece tokenizer to break input text into tokens, which are then represented as vectors, Fig 4.1 shows the diagram of example of Bert tokenizer.

Also for Word embedding GloVe is used. GloVe (Global Vectors for Word Representation) is a word embedding technique. It utilizes word co-occurrence statistics from text data to create word vectors that encode semantic relationships. Key aspects of GloVe include:

- **Word Co-Occurrence Matrix:** GloVe constructs a matrix capturing word co-occurrence frequencies in a text corpus context window.
- **Word Vectorization:** GloVe transforms the co-occurrence matrix into word vectors through optimization, aiming to predict co-occurrence probabilities.
- **Loss Function and Optimization:** A loss function measures the difference between predicted and actual co-occurrence probabilities. An iterative process optimizes word vectors.
- **Final Word Vectors:** The learned vectors represent words, preserving semantic relationships and placing similar words closer in vector space.

Thus, this paper proposes the use of DistilBert as tokenizer for text preprocessing and Hierarchical Attention Network, which is attention based mechanism which when trained on dataset possibly provide better data than existing methods. Fig 4.1 shows the DistilBert architecture.

Utilising stacked recurrent neural networks on the word level and an attention model, "Hierarchical Attention Networks (HAN)" learns about the words that are crucial to the meaning of the sentence and combines their representation into a sentence vector. Once again the algorithm is applies to the sentence vector, which creates a vector that understands the content of the provided document and can then be used for text categorization.

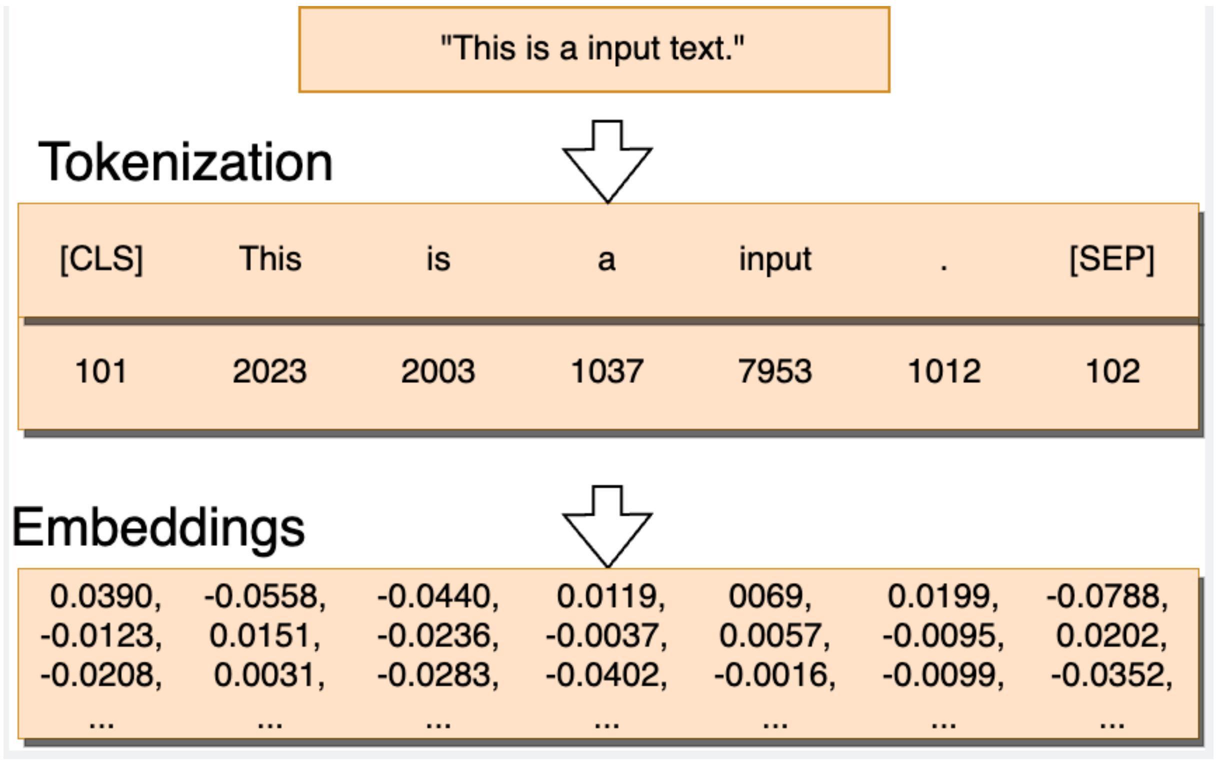


Figure 4.1: DistillBert-Tokenizer

The Hierarchical attention is based on the concept that : "Words form sentences, and sentences form documents." To capture sentence and document meanings, an attention model is used, giving importance to key words. This model involves Bidirectional RNN and Attention networks. Bidirectional RNN interprets word sequences, yielding individual word vectors. Attention network assigns weights to these vectors via a neural network, aggregating them into a sentence vector through weighted summation. This process extends to sentence and document levels, forming a hierarchical attention network. The vectors undergo a shallow neural network to determine weights, and their weighted sum captures combined meaning.

4.1.1 Working Methodology of HAN

By first developing phrase representations and then combining those into a document representation, the network first takes into account the hierarchical structure of docu-

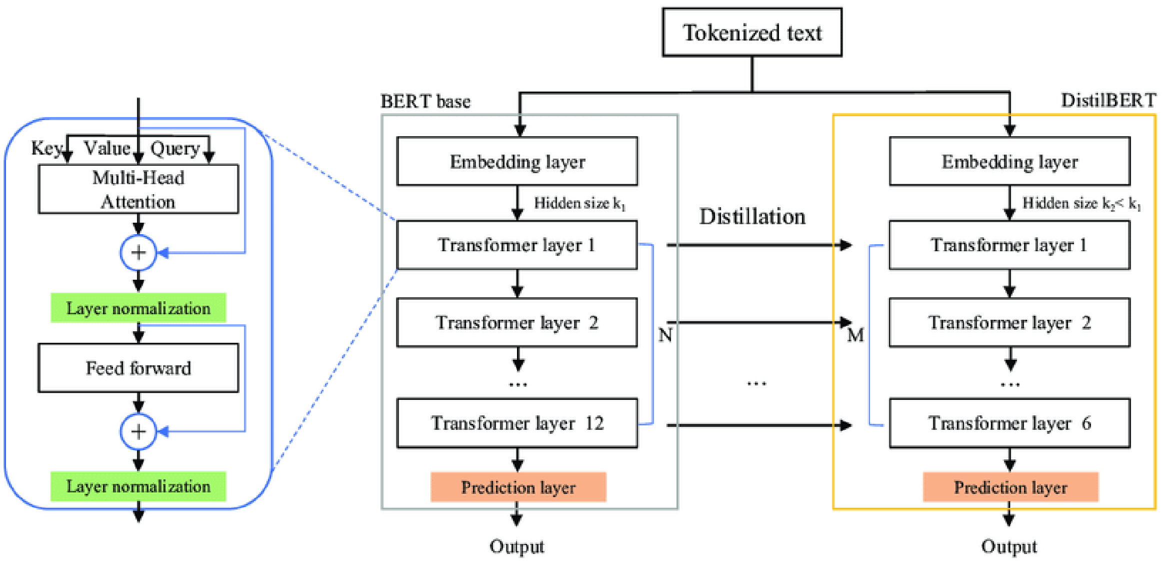


Figure 4.1: DistillBert Architecture

ments. ’Sentence representations’ are created by first encoding a sentence’s words, then using the attention process to create a *sentence vector* from those words. The ”document representation” is constructed in a similar manner, except it only takes as input the sentence vectors for each sentence in the document. The algorithms are used twice: firstly one the level of words and then sentence level later. The model consists of

- relevant contexts returned by the encoder, and
- the “attention mechanism”, which combines various contexts’ significance weights into a single vector.

Word Level

Referring Figure 4.4

$$x_{it} = W_e w_{it}, \quad t \in [1, T]$$

- As input the structured tokens **wit ** which represent the word i per sentence t are provided. All the words that are in the sentence are not taken for processing.

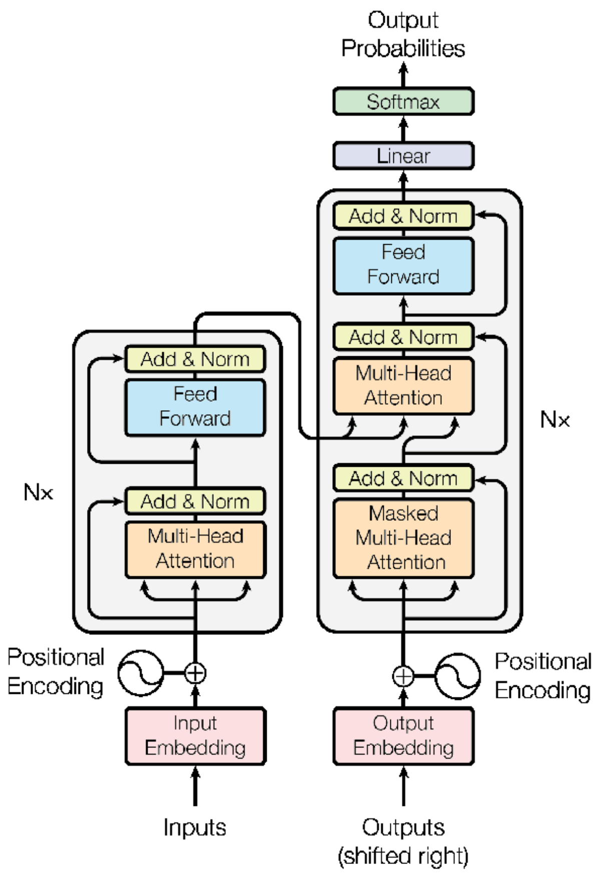


Figure 4.2: Transformer Architecture

- The tokens move through an embedding layer that 'assigns' multidimensional vectors. **We** **wit ** to each token because the model cannot interpret raw text of the data type string. In this method, words are projected into a continuous vector space and represented mathematically as **xit **.

- There are numerous embedding algorithms, with word2vec and GloVe being the most well-liked. Pretrained Glove embeddings have been used for this methods.

Word Encoder

First, convert the words from a sentence with words $w_i t$ to vectors using the embedding matrix W_e, where $x_i j = W_e w_i j$. By summarising data from both directions for each word and using contextual information in the annotation, we employ a "bidirectional

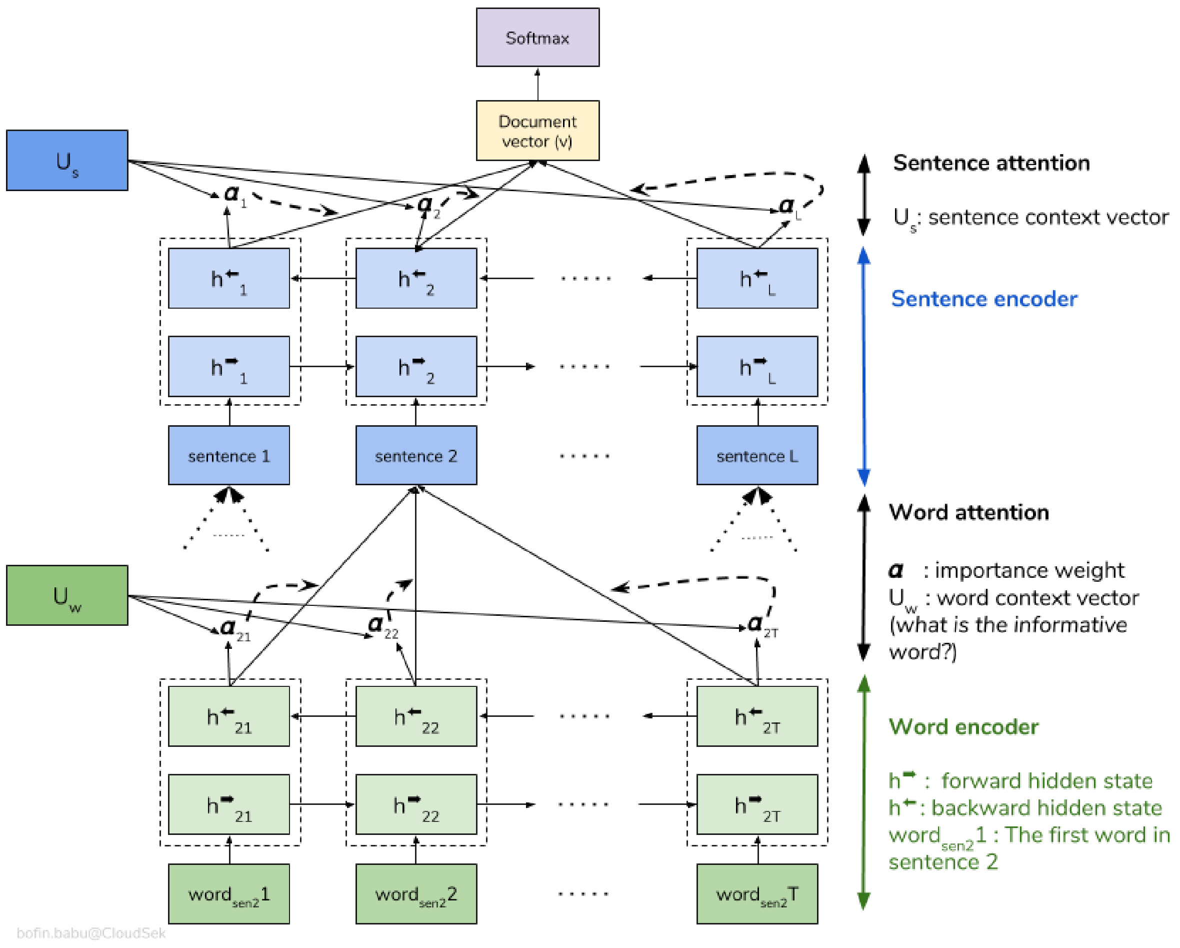


Figure 4.3: HAN Architecture

GRU" ("Bahdanau et al., 2014") to produce word annotations. The "bidirectional GRU" involves the forward gated reccurent unit, vec_f, which reads the sentence S_i from w to w_{ir}, and a backward gated reccurent unit, vec_b, which reads from w_{iT} to w_{i1}.

$$x_{it} = W_e w_{it}, \qquad t \in [1, T]$$

$$\text{vec}_{h_{it}} = \text{vec GRU}(x_{it}), \qquad t \in [1, T]$$

$$\overline{h}_{it} = \overline{\text{GRU}}(x_{it}), \qquad t \in [T, 1]$$

given word's annotation as w_{it} by concatenating the forward hidden state $\text{vec}_{h_{it}}$ and the backward hidden state $\overline{h}_{it}$, i.e., $h_{it} = [\text{vec}_{h_{it}}, \overline{h}_{it}]$, this enumerates the key details of

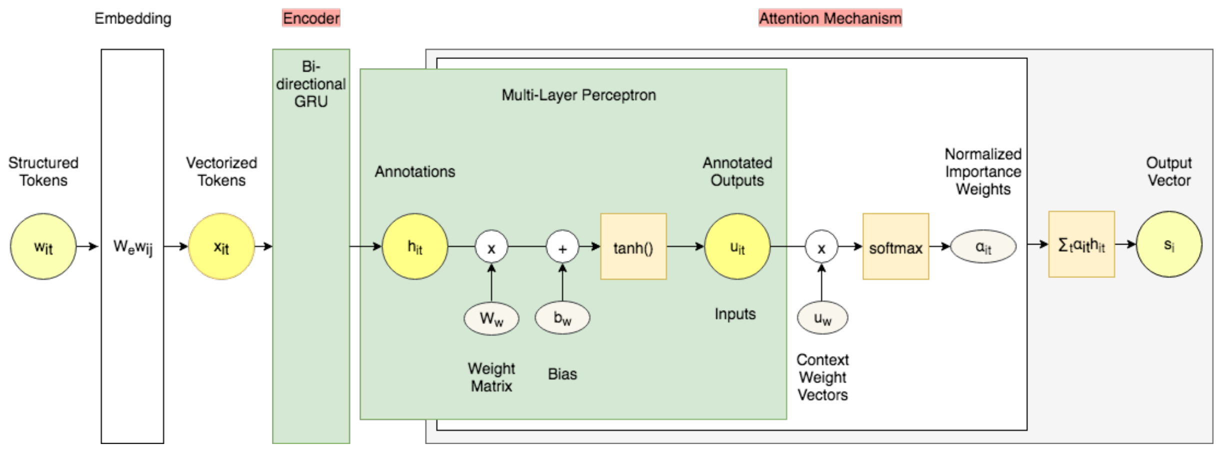


Figure 4.4: Word Level Encoding

the entire phrase with an emphasis on w_{it}.

Word Attention

- They are annotated The foundation for the attention mechanism is built by $\mathbf{h}_{i1}$, which begins with a one-layer "Multilayer Perceptron" hidden layer. The intention is to train the model with weights and biases that were randomly initialised. Then, $\mathbf{u}_{it}$ is used to represent the 'enhanced' annotations. This layer also makes sure that the network won't crash due to a *tanh* function. This function converts zeros to values that are close to zero and corrects input values to be between -1 and 1.

$$\mathbf{u}_{it} = \tanh(\mathbf{W}_w \mathbf{h}_{it} + \mathbf{b}_w)$$

- Once more, we normalise our updated annotations to a word weight $\mathbf{u}_w$ and multiplied by a trainable context vector $\mathbf{C}_L$ using a softmax function. During the training phase, the word context vector $mathbfu$ is jointly learned and initialised at random.

$$\alpha_{it} = \frac{\exp(\mathbf{u}_{it}^\top \mathbf{u}_w)}{\sum_t \exp(\mathbf{u}_{it}^\top \mathbf{u}_w)}$$

- The sentence vector is the result of adding the total of these important weights to the previously determined context annotations.

$$\mathbf{s}_i = \sum_t \alpha_{it} \mathbf{h}_{it}$$

Sentence Level

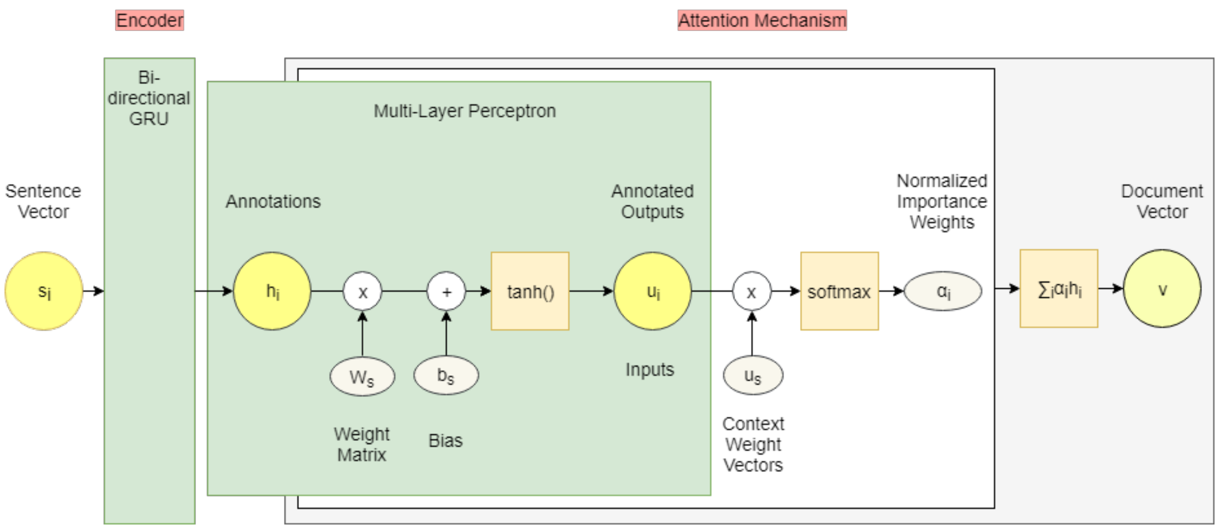


Figure 4.5: Sentence Level Encoding

The network is subsequently implemented at the sentence level using much the same method as at the word level, with the exception that we now give special emphasis to sentence i. An embedding layer is not required because we already obtain sentence vectors **si** from word level as input.

Sentence Encoder

With a bidirectional GRU, sentence contexts are condensed by reading the document both forward and backward.

$$\overrightarrow{h} = \mathrm{GRU}(\mathbf{s}), \quad t \in [1, L]$$

$$\overleftarrow{h} = \mathrm{GRU}(\mathbf{s}), \quad t \in [L, 1]$$

$$h_1 = [\overrightarrow{h}_i, \overleftarrow{h}_i]$$

Sentence Attention

- Once more randomly initialised, trainable weights and biases are jointly learned throughout the training phase. A document vector **v** that can be utilised as features for document classification is the outcome of the process.

$$\begin{aligned} u_i &= \tanh(W_{sh}\mathbf{i} + b_s) \\ \alpha_i &= \frac{\exp(u_i^\top u_s)}{\sum_t \exp(u_i^\top u_s)} \\ v &= \sum a_i h_i \end{aligned}$$

4.2 Dataset

A bug report dataset is created by downloading bug reports from Bugzilla. In this paper we have downloaded the dataset from a GitHub repository which has a collection of all these bug reports and is freely available for research purposes. In this paper bog reports from "eclipse jdt", "eclipse-platform", "Mozilla.core", "Mozilla Firefox" is created. This dataset has several columns such as Issue.id, Component, Title of the bog, Description of the bug and Priority assigned to the bug report. These features of the dataset are used to determine the priority of the bug. Here, priority will be the label and Title Description, etc. will be the features which will be given as input to the model and the model will predict the label Le, the priority. This dataset consists of bug reports from the open-source software mentioned above. Four csv files are generated for each of the above mentioned open-source software, "eclipse.jdt, eclipse platform, Mozilla.core. Mozilla Firefox".

4.3 Evaluation Measures

The proposed model is evaluated on the following metrics: Precision, Recall and F1-score and the model is compare with the state-of-the-art methods:

- Precision: ‘When referring to positive observations, precision is defined as the proportion of properly predicted observations to all expected positive observations’.
- Recall : ‘Recall is the ratio of accurately predicted positive observations to all actual class observations’.
- FI-score: ’The FI-Score is a statistic that combines recall and accuracy. Precision and Recall are weighted averaged to determine the FI Score. As a result, both FP (“false positives”) and FN (“false negatives”) are included while calculating this score’.

4.4 Work Done

We tested the proposed model on eclipse platform dataset on one feature which is the Description of the bug feature. Dataset was created by downloading several bug report of eclipse eclipse from Bugzilla and combining. The dataset has the following com Issue.id, Title, Description, Status, Resolution and Created-time the and the label Priority. The data was preprocessed and then vector representations was obtained using DistilBert Tokenizer. The data was then oversampled using ADASYN. Glove Word embedding was used for the pretrained numerical values. The glove numerical representations was passed on the HAN model and the model got trained.

Text Preprocessing

The textual data underwent a series of preprocessing steps to enhance its quality and relevance for model training. The following preprocessing steps were applied to each text sample:

- Lowercasing:To guarantee uniform word representation, all text was changed to ‘lowercase’.
- Punctuation Removal: Punctuation marks and “special characters” were removed from the text.

- Stopword Removal: Common stopwords from the English language were eliminated from the text to focus on meaningful content.
- Stemming: Words were stemmed using the “Porter stemming algorithm” to reduce them to their root forms.
- Lemmatization: Lemmatization was applied to further normalize words by converting them to their base or dictionary form.

These preprocessing steps aimed to create a cleaner and more standardized textual dataset for model training.

GloVe Word Embeddings

GloVe (Global Vectors for Word Representation) embeddings were employed to enhance the representation of words in the textual data. These pre-trained word vectors capture semantic relationships between words and help improve the model’s understanding of the textual content. The GloVe embeddings were loaded from a pre-trained file (`glove.6B.100d.txt`) and utilized for initializing the embedding layer of the HAN model.

ADASYN Oversampling

To address class imbalance in the dataset, the ADASYN (Adaptive Synthetic Sampling) technique was employed. This method oversamples the minority class by generating synthetic samples that are similar to existing samples. ADASYN helps improve the model’s ability to learn from underrepresented classes and avoid bias towards majority classes, thereby enhancing the overall performance of the model on all classes.

Hierarchical Attention Network (HAN) Model

The “Hierarchical Attention Network (HAN)” model was constructed to effectively capture the different level structure of textual data. The model consists of two levels of attention mechanisms: “word-level attention and sentence-level attention”. The HAN model processes text in a hierarchical manner, allowing it to hone attention on crucial phrases and clauses while comprehending the context and links in the text.

The model architecture consists of the following layers:

- Embedding Layer: Initialized with pre-trained GloVe embeddings to represent words in a dense vector space.
- Bidirectional GRU Layers: These layers encode both forward and backward contextual information for words and sentences.
- Attention Mechanisms: Word-level and sentence-level attention mechanisms highlight relevant words and sentences within the text.
- Dense Layers: Fully connected dense layers for feature extraction and classification.

Training and Evaluation

The model was trained using a batch size of 64 and a maximum of 10 epochs on the preprocessed and oversampled dataset. Several performance indicators, 'including as accuracy', 'loss', 'precision', 'recall', and 'F1-score', were used to track the training process. In order to avoid overfitting and maximise training effectiveness, early stopping was used.

Plots and Visualizations

Several plots and visualizations were generated to analyze and interpret the model's performance:

- Accuracy and Loss Plots: Plots depicting the 'training' and 'validation' accuracy, as well as the loss, across the training epochs.
- Confusion Matrix: A heatmap representation of the confusion matrix, displaying the distribution of predicted labels against true labels for each class.

- Precision-Recall-F1 Curves: Precision, recall, and F1-score were calculated and presented for each class, providing insights into the model's class-specific performance.

Both undersampling and oversampling were used in the model's training process. In the situation of undersampling, the performamodel did not perform very well. The model performed well after oversampling, though. As a result, the dataset was subjected to the oversampling approach known as ADASYN. The model was tested on a test dataset after the training phase. The loss metric utilised was sparse categorical cross-entropy, and the optimizer employed was the Adam optimizer. categorical cross-entropy that is sparse. For multi-class classification, sparse categorical cross-entropy is utilised. A development of the "stochastic gradient descent" (SGD) method for updating network weights during training is the optimisation algorithm known as Adam. Adam changes the learning rate for each network weight independently, as contrast to (SGD) Stochastic Gradient Descent, which utilises a single learning rate for the entirety of the training process. 128-piece batches were used to train the model. Ten epochs were used to train the model. The model's outcomes were contrasted with those of previously employed methods.

5 Results

5.1 Results

The evaluation metric used for this purpose is F1-score and it is evaluated on the priority levels. F1-score for every priority level is calculated and compared. F1-core is defined as the harmonic mean of precision and recall.

“Precision (Positive Predictive Value)”

“Out of all positive cases (true positives plus false positives), precision is the proportion of positive instances (true positives) that are correctly anticipated. ”

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

“Recall (Sensitivity, True Positive Rate)”

“Recall measures the percentage of true positives (positive cases) that were correctly predicted out of all positive cases (positive cases + false negatives). ”

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

“F1-Score”

“The harmonic mean of recall and accuracy is known as the F1-score.” It provides a balance between precision and recall, making it useful for evaluating models when there is an imbalance between the classes.

$$\text{F1-Score} = \frac{2 \cdot (\text{Precision} \cdot \text{Recall})}{\text{Precision} + \text{Recall}}$$

Here for every priority level i.e. “P1, P2, P3, P4 and P5” Precision, Recall and El-score is calculated. The proposed model is evaluated on priority level and it is compared with existing approaches. For the priority levels P1, P2, P3, P4 and P5 the results obtained are as follows:

- Precision values obtained for the proposed model: 54.84%, 28.57%, 38.71%, 37.14%, and 38.33% respectively.
- Recall values obtained for the proposed model: 32.69%, 33.33%, 23.08%, 53.06%, and 47.92% respectively.
- F1-score obtained for the proposed model: 40.96%, 30.77%, 28.92%, 43.70%, and 42.59% respectively.

In the result proposed model is compared with 4 base models: Logistic Regression, SVM. Naive Bayes and CNN. It is homogenous comparison as the dataset and preprocessing steps etc., used are same. The results obtained by the proposed method and its comparison with other base models implemented (homogenous comparison) is displayed in Tables 5.1, 5.2 and 5.3. Table 5.1 shows Precision, Table 5.2 shoes Recall and Table 5.3 shoes F1-Score.

Table 5.1: Results of the present HAN model in terms of Precision

Techniques	RESULTS (Precision)				
	P1	P2	P3	P4	P5
Proposed Method	54.84%	28.57%	38.71%	37.14%	38.33%
CNN	37.16%	33.51%	43.80%	39.71%	27.39%
SVM	35.41%	28.31%	40.38%	37.26%	25.93%
Random Forest	34.23%	30.29%	38.56%	35.09%	24.64%
Logistic Regression	32.17%	25.50%	35.68%	33.11%	22.93%

Table 5.2: Results of the presented HAN model in terms of Recall

Techniques	RESULTS (Recall)				
	P1	P2	P3	P4	P5
Proposed Method	32.69%	43.33%	23.08%	53.06%	47.92%
CNN	40.67%	21.86%	44.15%	39.77%	30.43%
SVM	36.92%	21.13%	40.54%	38.66%	27.72%
Random Forest	36.52%	20.14%	43.50%	39.59%	27.13%
Logistic Regression	33.25%	20.19%	38.40%	27.72%	25.36%

Table 5.3: Results of the presented HAN model in terms of F1-score

Techniques	RESULTS (F1-score)				
	P1	P2	P3	P4	P5
Proposed Method	40.96%	30.77%	28.92%	43.70%	42.59%
CNN	33.83%	26.46%	23.54%	39.74%	28.83%
SVM	37.02%	23.54%	31.22%	38.39%	26.52%
Random Forest	35.52%	24.89%	29.53%	36.79%	26.09%
Logistic Regression	32.70%	22.54%	26.99%	33.85%	24.08%

5.2 Performance Analysis

From the obtained results, we can observe that the F1-score obtained for priority P1. P2. P4 and P5 is better than that of the existing methods. It can be observed that the performance of the proposed model is a little bit better than that of the existing models when compared with base models (homogenous comparison) that were trained on the same dataset as that of the proposed model and also performed the same pre-processing procedures.Additionally, it can be deduced that the performance of the proposed model would greatly increase as it is trained on larger datasets.

6
Conclusions and Future Scope

6.1 Conclusions

A review of the research work had been finished, as can be seen in chapter 2. Different research and studies have various characteristics, advantages, disadvantages, and most significantly, promise in the future. It is possible to infer from reading the review that bug reports can be automatically prioritized with the aid of Ml ('Machine Learning') and DL ('Deep learning') techniques, as well as their combination with NLP approaches to produce superior outcomes. While it has been found via the study that precise prioritization of problems may be very beneficial for the overall functionality of the software as well as for the programmers or maintainers, which will boost the overall effectiveness of the task

6.2 Future work

The results above show that the suggested technique performs admirably when trained on the dataset. By using better preprocessing methods, this outcome can be enhanced: To enhance the outcomes, the suggested model can also be trained on bigger datasets and for more epoelis. Improved accuracy can be achieved by fine-tuning the model. The findings can be improved by enhancing the overall model, which will also improve the hyperparameter of the HAN model. Other Ml ('Machine Learning') and DL ('Deep learning') approaches can be investigated for this problem in additional study and contrasted with the current methods and the suggested ways.

[?]