EAGRA -Automated Bugs Identification Through Early Access Game Review Analytics On Game Distribution Platforms

Submitted by

ALI SHAHBAZ
(20I-2020)

Supervised by

DR. KHUBAIB AMJAD ALAM

Masters of Science (Software Engineering)

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science (Software Engineering) at National University of Computer & Emerging Sciences

Department of Computer Science

National University of Computer and Emerging Sciences

Islamabad, Pakistan

June 2022

Plagiarism Undertaking

I take full responsibility of the research work conducted during the Master's Thesis titled Automated Bugs Identification Through Early Access Game Review Analytics On Game Distribution Platforms. I solemnly declare that the research work presented in the thesis is done solely by me with no significant help from any other person; however, small help wherever taken is duly acknowledged. I have also written the complete thesis by myself. Moreover, I have not presented this thesis (or substantially similar research work) or any part of the thesis previously to any other degree awarding institution within Pakistan or abroad.

I understand that the management of National University of Computer and Emerging Sciences has a zero tolerance policy towards plagiarism. Therefore, I as an author of the above-mentioned thesis, solemnly declare that no portion of my thesis has been plagiarized and any material used in the thesis from other sources is properly referenced. Furthermore, the work presented in the thesis is my own original work and I have positively cited the related work of the other researchers by clearly differentiating my work from their relevant work.

I further understand that if I am found guilty of any form of plagiarism in my thesis work even after my graduation, the University reserves the right to revoke my Master's degree. Moreover, the University will also have the right to publish my name on its website that keeps a record of the students who plagiarized in their thesis work.

_____

ALI SHAHBAZ

Date: _____

Author's Declaration

I, Ali Shahbaz, hereby state that my Master's thesis titled Automated Bugs Identification Through Early Access Game Review Analytics On Game Distribution Platforms my own work and it has not been previously submitted by me for taking partial or full credit for the award of any degree at this University or anywhere else in the world. If my statement is found to be incorrect, at any time even after my graduation, the University has the right to revoke my Master's degree.

_____

ALI SHAHBAZ

Date: _____

Abstract

User reviews are considered one of the most important source of information about an app and game. The classification and analysis of reviews in order to extract information has proven to be a considerable difficulty. Game applications receive user input in the form of reviews, which can assist developers in selecting games with improved functionality and extracting relevant information such as user feedback, problems, and descriptions of user experiences linked with existing features. Game application review analysis has recently emerged as an important area of research in software engineering due to the sizable user base and possible advantages of automated feature and bug extraction. Numerous studies have recently been carried out to mine and classify user evaluations into useful software maintenance requests, including feature requests and problem reports for various software systems. However, existing literature have mostly focused on mining functional aspects and analysis of use for software systems. But for gaming industry existing literature is limited to manual analysis of mining functional aspects and analysis of use. In order to satisfy users and remain competitive in the gaming business, developers must fix these bug complaints. As a result, this study formulates the issue as a multi-label classification problem and suggests bug classification models utilizing RNN, LSTM, and CNN, all of which have varying degrees of accuracy. Representative features are used to train a model for bug classification. A careful selection of feature extraction methods is required since their use in training a classification model may provide results that are divergent. Recent research has put attention on the need of using simple, up-to-date taxonomies to categorize bugs. Therefore, keeping in view these taxonomies this research employs these taxonomies to provide a tools which takes a user reviews dataset and then identify bugs from them and after identification, it will classify the bug into their related bug categories using classification model. This research taking GAME STEAM ENGINE as a case study to scrap gaming reviews and train model on that reviews for bugs classification.

Acknowledgement

First and foremost, I have to thank Almighty Allah for countless blessings. ThenI have to thank my parents for their love and support throughout my life. Thankyou both for giving me strength to reach for the stars and chase my dreams. My brothers, and my entire family deserve my wholehearted thanks as well. I wouldlike to sincerely thank my supervisor, Dr. Khubaib Amjad Alam, for his guidanceand support throughout this research, and especially for his confidence in me. I would also like to thank all friends who stood by me through the ups and downs of life.

Table of Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

CNN	Convolutional Neural Network
BoW	Bag of Words
LSTM	long short-term memory
RNN	Recurrent Neural Networks
ML	Machine Learning
AI	Artificial Intelligence
RL	Reinforcement Learning
FPS	First-person Shooter
DRL	Deep Reinforcement Learning
ReLU	Rectified Linear Unit
SDLC	Software Development
GloVe	Global Vectors for word representation
NLP	Natural language processing
MLBCGR-LSTM	Multi-label bugs classification of Gaming Bugs using Long term-short term Memory

TP	True Positive
FP	False Positive
RE	Requirement Engineering

Chapter 1

1. Introduction

Over time, the popularity of computer games has expanded, creating a multi-billion-dollar business. The scale of the computer gaming market makes it challenging to build a hit game. Additionally, previous research demonstrates that gamers are notoriously difficult to satisfy, making game quality a crucial concern. Making game quality a crucial consideration. The majority of online gaming stores allow consumers to leave feedback on a game they purchased. The opinions of other prospective customers are regularly taken into account while making purchase decisions. There is a lot of research on social media reviews, product reviews, and app reviews on many dimensions in the literature that is already available [1]. They have limited work on user satisfaction, user requirements, and sentimental analysis however existing literature has non-existing automated methods to extract gaming bugs from reviews. Gaming reviews have started gaining attention recently in the gaming industry. As a result, reading game evaluations can aid game makers in better comprehending user issues and continue to increase the games' user-perceived quality [2]. It has grown rapidly and is now a significant sector of the software development industry. 2D and 3D platform games are once again available (the developed version). An immense challenge for them is that the industry lacks an automated system-level approach to gaming tests. Currently, most game development organizations. The game is tested using manual or semi-automatic techniques. Such testing methods do not meet industry requirements if you need a more systematic and repeatable approach [3].

Users rate and review programs they have downloaded on application distribution platforms like Google Play and the Apple App Store. The popularity of these platforms among both application developers and users has been on the rise in recent years. They have the potential to affect requirements engineering processes, but their effects are not yet fully understood [4]. It's critical to generate high-quality games to flourish in the gaming business given the expanding market and fierce competition. Bug reports from players are a frequent tool used by developers to find errors in their games. Play-through videos have recently gained popularity in the gaming world. Some of

these games' videos show off bugs, giving engineers more chances to gather thorough bug data. [5].

The app distribution platforms have embraced an open business approach to reduce the limitations for small-scale businesses and commercial app developers. Lowering the entry barriers has led to increased competition in the mobile app market. The app distribution platforms have embraced an open business approach to reduce the limitations for small-scale businesses and commercial app developers. Lowering the entry barriers has led to increased competition in the mobile app market [4], [5]. For a significant number of evaluations that are received every day, manual analysis is impractical. Text mining techniques have been employed in the study on app review mining to extract meaningful information from the reviews. A way to find valuable information in reviews is to train an ML-based classification model to automatically group the reviews according to their content [6].

The creators and players may both suffer if a bug continues after a game is released. Despite these Effects, it's normal for games to launch with bugs already there. In 2014, for instance, three-quarters of high-budget games "published on Xbox One, Wii U, and PS4" received an update within 24 hours after the game's initial release. This issue was later fixed through updates. These problems may have emerged due to the various ways that games and conventional software are developed. It is challenging to thoroughly evaluate every feature of a video game, according to a prior study. Because there are so many potential user interactions in games compared to other forms of software, it can be challenging for developers to write thorough tests. As a result, many games will be published with hidden defects that only become apparent once the player begins using the game [7].

This difficulty can often lead to players experiencing failure, which negatively affects the game experience for some [8]. This challenge draws attention to a more comprehensive analysis of reviews to identify issues and requirements, given the potential value that the text of the review means. As per the above discussion, there was a need for a tool that takes a user-review dataset and then identifies bugs from them, and after identification, it will classify the bug into their related bug categories. The goal of this thesis was to automate the extraction of gaming bugs from steam

engine game reviews, train classification models to categorize user reviews under different labels and make the bug categorization process more efficient.

1.1. Challenges Related to Game Reviews Analysis

Although the video game industry has changed dramatically over the previous few decades, multiple reports reveal that creators still confront numerous problems in their everyday job. One of the most difficult challenges is getting projects finished on schedule and within budget [9]. Efficient requirement gathering not only speeds up the development process but also results in more efficient and cost-effective solutions [10]. Requirements defined by IEEE are “a condition or capability needed by a user to solve a problem or achieve an objective” [11]. If these items are video games for mobile devices, the issues are more serious.

Games in particular must adhere to several significant non-functional constraints (portability, gameplay, emotional aspects, etc.). Additionally, because games are created for a broad audience, development teams must be aware of the requirements of diverse and perhaps unacquainted stakeholders [12]. The gaming business uses different requirements engineering techniques than the conventional software industry [13].

As per the above limitations, developers need to identify reported issues and suggestions by analyzing and categorizing a large corpus of reviews. When read just as a "buyer's guide," game reviews provide readers with advice on how to approach a certain game and enjoy it to the fullest. They are extensive and diversified in terms of the subjects and topics covered. More generally, game reviews also help keep a history of video games by contextualizing the connections and history connections that exist between games [14]. While these categories are valuable for improving the capabilities of your game, they need bug detection and classification when it comes to quality and user satisfaction.

1.2. Problem Formulation

Mining and analyzing user game reviews have emerged as an important challenge for game

developers. There are millions of games in the highly competitive gaming market, and these games have billions of user reviews. This excessive amount of user ratings makes it difficult to analyze and manually interpret the ratings for information extraction.
Software providers are assembling enormous sums. use information, error logs, and sensor data are all examples of implicit feedback. These patterns suggest a change toward the discovery, prioritizing, and management of software needs that is data-driven and user-centered. While dealing with a large set of data training of classification models is performed using feature extraction [15].

However, the importance of these models for multi-label text classification is not clear. This paper addresses this issue and evaluates the performance of the classification model. The approach to bug classification in game reviews refers to different datasets or evaluation strategies, so the results cannot be compared. This research assesses the performance of fundamental feature extraction techniques for labeled datasets and the performance of monitored deep learning models to address this issue. The previous studies have more complex methods to extract features from the reviews [16], [17]. However, because reviews are in text format and contain additional information that isn't important to get as a feature of your app, this strategy risks extracting a lot of irrelevant features. To address this challenge, this task involves training a convolutional neural network model to perform validation classifications without the explicit use of feature extraction methods.

1.2.1. Game reviews Classification

The classification model for analyzing Game reviews automatically assigns reviews to classes while monitoring the content of the review text [18]. To train and evaluate these models, you need data from labeled evaluations. In this data, ratings are manually categorized by humans into various categories.
In this thesis, the effectiveness of deep learning models is assessed using various feature classifications. The model implementation chapter presents many methods as well as accuracy.

1.2.2. Multi-Label Game Reviews Analysis

Previous studies have classified Game reviews as one or more classes [19], [20], [21], [22].
Most users point out multiple bugs/Issues in a single rating and identify and categorize multiple categories in a single rating, but confirming a complete rating of a review is a multi-labeled classification problem. To address this challenge, a multi-label classification approach is performed using a problem transformation approach on a deep learning-based model.
In addition to the review's multi-label classification problem, feature engineering for review classification can introduce fake feature extraction tasks when considering irrelevant information present in user reviews. To solve this problem, deep learning, a new branch of machine learning models, has recently been developed. It automatically learns characteristics from tagged data. [20].

CNN (convolutional neural network) a deep learning model performs well on text classification problems. But its performance is unknown for multi-label text classification problems that's why performing deep learning analysis of multiple models including LSTM, and RNN was required to compare their accuracy in the end. And using the one which evaluates results with high accuracy. In this thesis first, the dataset is labeled according to the bugs categories defined in the literature [8]. Categorize them into multiple categories of bugs and trained multiple deep learning models to get information from the models.

1.3. Research Objectives

This research mainly focuses on the Multi-Label Review Classification problem in the context of video games. To resolve this problem, the following objectives are intended to be achieved:

- RO1: To gain insights into the existing classification methods that need to be addressed in the context of game rating analysis.

- RO2: To assess the effectiveness of deep learning models for multi-label text classification.

- RO3: To automatically categorize gaming reviews into multiple bug labels using deep learning models (CNN, RNN & LSTM).

1.4. Research Questions

The following research inquiries have been made in light of the issue mentioned in the preceding sections:

- RQ1: What is the most widely used classification approach for game review analysis?
- RQ2: How do feature extraction impacts the performance of deep learning classification models for gaming reviews?
- RQ3: How reviews can be categorized into multiple labels using deep learning approaches CNN, RNN & LSTM?
- RQ4: What are gamers talking about in gaming reviews?

1.5. Research Approach

The whole research process is divided into four distinct phases. Below is a brief overview of the four phases of the research process according to the design science approach [23] in Figure 1.

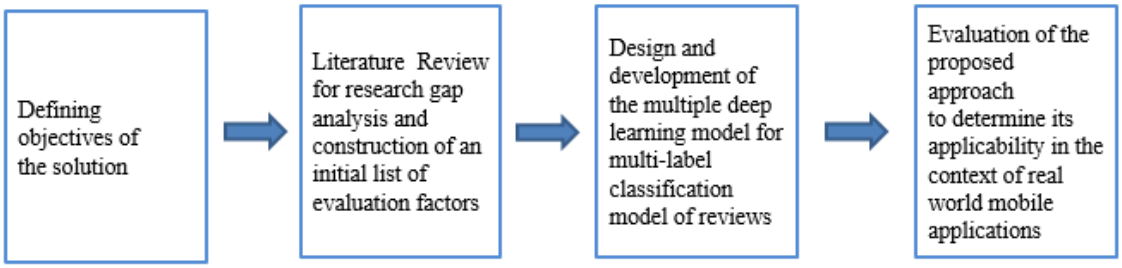


Figure 1 Research Methodology

The first phase comprises a literature review and research processes. Literature reviews have multiple perspectives. This includes a detailed discussion of the requirement engineering process, requirement elicitation processes, requirement elicitation for game development, automated text analysis, and gaming bug classification. Lastly, a case study is conducted to verify our findings. This phase concludes the identification of the research gap and a comprehensive list of factors that are needed for this research problem.

The second phase includes research methodology, research questions (RQs), and research objectives (ROs). They helped in defining the scope and focus of the current scope.

The third phase is the design and development of a multi-label classification model for gaming reviews analysis. Here, the effectiveness of many deep learning models on game reviews is evaluated. And considering the one which has high accuracy for multi-label review s classification. This phase starts with the identification of key evaluation factors among a large number of factors that can be considered in gaming reviews analytics. This phase involves gaming experts who are asked to identify the relevant gaming bugs from the reviews. The final dataset is maintained by employing a suitable screening process.

In the fourth phase proposed model is evaluated through a case study on the real-world platform name GAME STEAM. To guarantee the correctness and consistency of the outcomes, continuous evaluation checks are carried out at each stage. The designed decision model is used to take strategic decisions based on the trained data. As these models are sensitive to even little changes in the weights of the criteria, it is usually preferable to confirm their stability and robustness. The proposed solution model is validated for correction accuracy by feeding the real-world gaming reviews data. Finally, qualitative comparisons with leading evaluation and selection models highlight the shortcomings of existing models and highlight the novelty and contributions of the proposed decision-making model. After successfully evaluating the model, the trained model is tested for real-life unseen data.

1.6. Thesis Organization

The basic background of the thesis, which comprises the research aims and research questions of the thesis study, has been given in this chapter of the introduction.
Chapter 2 gives a thorough survey of the literature along with its context and flaws. The process of requirement engineering, requirement elicitation, requirement elicitation for game development, automated text analysis techniques, and the classification of gaming problems were all covered in detail. In this chapter, supervised and unsupervised neural networks' background material is described in-depth, along with its specifics.

Chapter 3 discusses in great depth the pilot tests of various deep learning models on the game reviews dataset. Several deep-learning approaches are available for text classification using multiple labels. But it wasn't known what the best multi-label text categorization performance was. Therefore, a pilot study of several deep learning models using a dataset of 1500 game reviews was conducted. This turned out to be very useful in choosing an appropriate classification model.

Chapter 4 highlights the problem which is discussed in chapter 1 and its proposed solutions. The first section describes the high-level architecture of the proposed classification model. The second section describes the concepts and plans needed for designing the classification model. And in the last section, multiple phases of the proposed solution are presented.

Chapter 4 highlights the case study Analyzing Steam Platform Games Reviews for Bugs Classification. This chapter focused on answering RQ2 (defined in chapter 1) on mining the gaming reviews in multiple bug categories [7]. This chapter is arranged into different sections introduction, explanation of the research design and data collection process, and automated methods to perform analysis on gaming reviews.

Chapter 5 shows the implementation of the proposed classification model which is implemented for game reviews. The first section describes the high-level architecture of the proposed classification model. The second section describes the concepts and plans needed for designing the classification model. And in the last section, multiple phases of the proposed solution are

presented.

Chapter 6 discuss the conclusion, research contribution, limitations & future work.

1.7. Summary

To lay the framework for the initial inquiry into the game reviews categorization challenge, this chapter gave an outline of this thesis as well as an overview and background of the topic. The established research questions, particular issue areas, and research goals attained throughout this study are all included in this chapter. The next chapter will comprehensively review the existing literature on Game reviews.

Chapter 2

2 Literature Review

This chapter initially provides background information on the issue that is crucial to address, after which game reviews and analyses of previous research are covered. Also, the requirement engineering processes which are important to discuss here as the dataset is collected and labeled by experts. Give specific details on the sorts of automated machine learning models that were employed in this thesis.

2.1 Requirement Engineering Process

Requirement Engineering is a systematic approach to managing and defining requirements. Considering the following objectives defined in [24].

- Understanding, archiving and overseeing related necessities methodically as well as achieving an understanding among stockholders.

- To decrease the risk of the stockholder's dissatisfaction with the conveying framework, determinations and administration of necessities are essential after understanding and reporting all the needs and needs of the stockholder.

The four fundamental exercises to meet these closes are as takes after:

Requirement Elicitation:

Requirement elicitation involves different activities including various techniques for the collection of requirements from multiple sources and particularly from stakeholders to define them precisely and clearly.

Requirement Documentation:

To document requirements various methods or natural language models are used to document requirements. It also follows the elicitation phase during the whole documentation process.

Requirement Validation:

To guarantee the completion of the characterized quality measure, validation and negotiation of recorded necessities are required.

Requirement Management:

This activity is different as compared to the rest of the required engineering activities. It involves the organization of requirements by any means to make them usable for multiple roles and perspectives. And also making them stable so they can accommodate any future change.
A detailed discussion of the requirements elicitation process and their importance in the games development industry is provided in the below section. Other requirement engineering process activities are not discussed because they are not within the scope of this thesis.

All the above-mentioned steps are visualized in the figure given below:

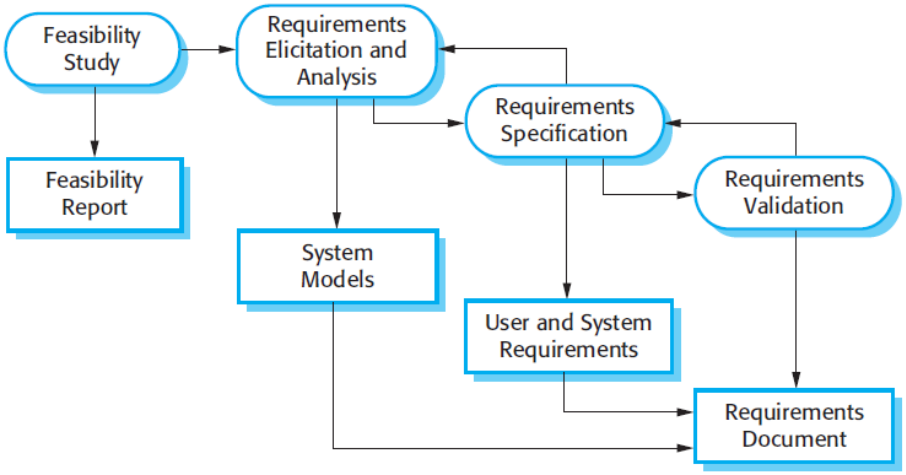


Figure 2 Requirement Engineering Process [25]

2.1.1 Requirement Elicitation Process

Among the foremost vital steps within the SE venture, the one is prerequisite elicitation. The whole process is described in the following steps:

- Related requirement sources identification
- User needs inquiry
- Examination of collected information which includes analysis of different issues
- Should have a good understanding of the completeness of user requirements
- Compile selected statements of complete requirements

Survey techniques help us to fully understand the mental, conscious, unconscious, and unconscious needs of our stakeholders. Additionally, the choice of requirements-gathering technique depends on the environment and system type, and there is no universal approach that perfectly collects requirements from all types of systems [26]. The below figure shows the required elicitation techniques:

Figure 3 Requirement Elicitation Techniques [27]

2.2 Requirements Elicitation for Game Development

Multiple factors influence the choice of elicitation technique. But the most influencing factor is the desired level of detail in the requirements [28]. Therefore, a document-centric approach can be

used to gather detailed requirements. Include the strategies for using existing documents, given that an appropriate level of detail can be extracted from these documents [29].
Commercial game developers intend to have: a wide range of users rather than meeting specific user groups, the requirements collected should be detailed and reflect the expectations of a wider range of users. As mentioned, document-centric requirements extraction techniques lead to finer-grained requirements. However, the agile and fast nature of mobile app development is the process places less emphasis on detailed documentation, despite the fact presence of informative documentation about apps, owned by developers or organizations, leads to the challenge of choosing a documentation-centric documentation approach as a requirements extraction method.

Therefore, the challenge of choosing a requirements-gathering technique for game development can be addressed using existing game feedback available on game distribution platforms such as Game Steam in the form of user reviews. Multiple studies literature involves the analysis of user reviews for requirement elicitation techniques [30]. A requirements management process Baan has been introduced by Dag et al. which searches the relationship between business requirements and user feedback comments [31].

An automated method to refine combine and analyze application reviews is introduced by Guzman et al. This method also uses topic modeling to group the requirements into meaningful groups of high-level functionality [32]. More specifically user reviews have been considered the most important source of information that can be used for games business objectives, important development decisions, etc.

2.3 Automated Text Analysis

On the market for gaming applications, several user evaluations are posted each day in text form. Therefore, the automated approach facilitates the manual task of extracting critical information from user reviews and supports a variety of game development release cycle activities. An example of a user game review, containing useful information for the improvement of a game is shown in

Figure 3: Gaming review.

Traditional automated approaches used to mine the reviews are based on supervised machine learning techniques [30][33][34].

Figure 4 Game Review [35]

In a supervised machine learning model, the algorithm picks up new information from the input-output training pair and then maps the sample using the newly acquired mapping function. The training set considered as an example is R = (Xi, Yi)^N. The review is considered full text and the target label is manually assigned to the review. Where Yi indicates the category or label that classifies the reviews into bug categories. For the classification model, the dataset is transformed into a vector format of zero and one such that number one is assigned to a review that belongs to the labeled class, and others are set to zero. An example of how one review might have many labels is shown in the image below.

Reviews	Labels
I returned it 3d interface impossible to negotiate even looking at the side view then the complete the tutorial keep an eye on the clock. Score was not updating.	Graphical error, Invalid value
still requires you to play online on top of it I suddenly have massive technical issues with the connection.	Invalid context over time

Graphical Error	Invalid Value Change	Invalid Context overtime
1	1	0
0	0	1

Figure 5 Reviews Encoding Example

The training example helps you learn the mapping function used to predict hidden data that will not be used during training. For example, the output of the results from the training model predicts a defined set of possible labels. Figure 4 above shows the review encoding example for the deep learning model.

2.3.1 Machine Learning for Reviews Classification

ML is primarily based on the concept of pattern-based decision-making with less human intervention, creating models that automate analysis, training the models, learning different patterns from data, and Artificial Intelligence (AI). Researchers are studying AI to see how machines learn from data by performing specific tasks according to learning patterns, without the explicit programming that forms the basis of ML. What is important is the repeatability of the ML. This allows the model to independently adapt new data and learn from previously trained models to produce reproducible and reliable results. Different machine-learning approaches are shown in the figure below.

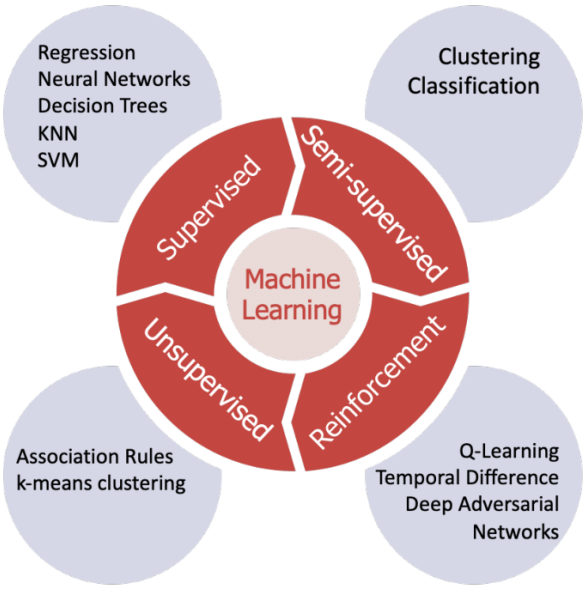


Figure 6 Machine Learning Approaches [36]

Studies also revealed that machine-learning approaches are best for text classification and analysis [30][33][34]. Another approach including deep learning is also part of the machine learning approaches. The below diagram represents the machine-learning approaches

Unsupervised Learning:

The category of ML, in which the model is well-trained for observation without prior output, can allow the autonomous construction of new models resulting from the data [37]. Unlike supervised machine learning techniques, which utilize labeled data to train the model, this form of machine learning algorithm uncovers latent patterns in unlabeled input data. The most commonly used algorithm in unsupervised machine learning is K-mean clustering [38].

Simple unsupervised learning technique for determining whether reviews are approved (with a thumbs up) or not (thumbs down) by D. Turney [39].

Reinforcement Learning:

Reinforcement learning considers learning from the environment and taking action based on the experience learned. Researchers have used this technique in many ways such as Zhang provided an RL method for learning sentence patterns by discovering the structure of related tasks [40].

Supervised Learning:

The supervised machined learning dataset is labeled and the output is known. It is the most widely used branch of machine learning. Many review analysis approaches have been implemented using supervised machine learning [41][19][42].

Supervised Machine Learning Algorithms:

The overview of supervised machine learning methods utilized in this thesis for text classification reviews may be seen below.

Deep Learning:

Deep learning is a machine learning method that instructs computers to learn by doing what comes naturally to people. Comprised several processing layers. They have minimized the effort of manual feature engineering which is used in traditional machine learning models. It has been used for text classification in different research domains. Recently, a variety of deep learning models have been utilized for text categorization and analysis, although it is uncertain how accurate these models are for this issue [43][44][45][46][47][48][49]. Therefore, this research is using RNN, CNN, and LSTM deep learning models which are evaluated for game reviews bug classification. The below section describe gives an overview of the working of these models. The details of the model's implementation are discussed in the pilot studies chapter.

Recurrent Neural Network:

An arbitrary length sequence can be processed by a recurrent neural network (RNN) by repeatedly applying a transition function to the input sequence's internal hidden state vector, ht. According to the equation below, the activation of the hidden state ht at time-step t is calculated as a function f of the current input symbol xt and the prior hidden state ht1.

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases} \qquad (1)$$

Here are some definitions of the equations required for training:

1) $h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$
2) $y_t = softmax(W^{(S)} h_t)$
3) $J^{(t)}(\theta) = \sum_{i=1}^{|V|} (y_{t_i}^{'} log y_{t_i})$

1. Contains details about the words that came before it in the list. As you can see, the word vector x t and the preceding vector h (t-1) are used to derive the value of h t. Additionally, a non-linear activation function (often a tanh or sigmoid function) was applied to the final summation. Assuming that h 0 is a vector of zeros is acceptable.

2. Predicts the word vector at time step t and calculates it. A (V,1) vector is created using the softmax function, and it has elements that add up to 1. This probability distribution provides us with the index of the vocabulary word that is most likely to come next.

3. Calculates the difference between the expected and real word at each time step t using the cross-entropy loss function. RNN working

Convolutional Neural Network:

While scanning the input I about its dimensions, the convolution layer employs filters that carry out convolution operations. Its hyperparameters include padding P and filter size F. Its activation function makes use of the Rectified Linear Unit (ReLU). The goal is to make the network less linear ($g(z) = \max(0, z)$).
Filter hyper parameters: The convolution layer contains filters; in which the following parameters are applied:

1. Dimensions of a filter: Dimension is defined for an input, which will produce the feature map.

2. Padding: Padding is applied, in which the filter sees the input end to end. It can be achieved by adding 0.

Following a convolution layer, which performs some spatial invariance, the pooling layer is a down-sampling procedure. Each input is linked to every neuron in the flattened input used by the fully connected layer.
An activation function is applied to predict the output labels of the input sentences [47].

Long Short-Term Memory:

In text categorization tasks, the Long Short-Term Memory (LSTM) recurrent neural network is frequently utilized. Recurrent neural networks are capable of capturing the structure of the sentences because predictions are generated progressively (for example, for the vector representation of each word) and the hidden layer from one prediction is passed to the hidden layer of the following prediction. A typical RNN gains extra components from LSTM that provide it with more precise control over memory.
These elements influence

1. How much does the present input affect the new memory's creation?

2. How much do the old memories influence the formation of the new memory?

3. What memory areas are crucial for producing the output?

When a late output depends on a very early input, LSTM is better at handling long-term reliance than typical RNN [51].

For instance, T1,..., and Tn are fed to the LSTM sequentially. Using the Word2Vec model, each word Ti is first transformed into its associated input vector xi and then entered one at a time into the LSTM. The output W of the hidden layer Hj at time j will be transmitted back to the hidden layer at time j+1 together with the subsequent input xj+1. The output layer will then receive the final output In [52]. Figure 6 shows the LSTM model layers that how input is modeled at the output layer.

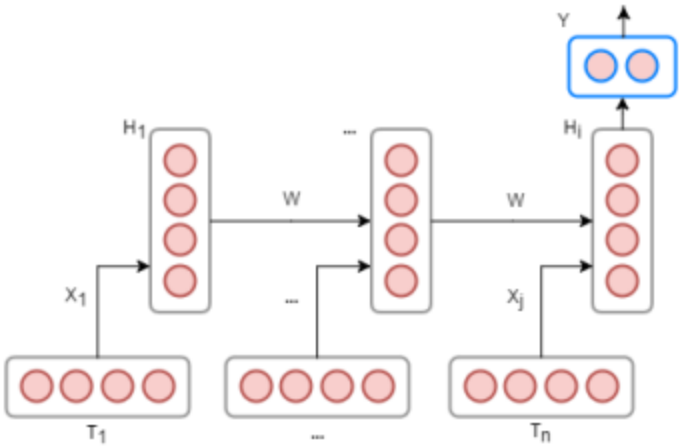


Figure 7 LSTM Layers [53]

2.4 Relevant Work On Gaming Bugs Classification

We review the existing work related to the automated classification of bugs from gaming reviews. Taxonomies and case studies are presented in the past to get information from gaming reviews

which include user responses toward the app, issues users are facing, and the new improvements users want in-game.

In research by [54], the PUMA technique—an automated, phrase-based method to extract user opinions in app reviews—was introduced. This strategy uses part-of-speech (POS) templates to extract terms from reviews. A study that looked at more than 100,000 bug reports from 380 projects discovered that defects may be categorized into four groups based on where they are fixed, which is something that many gaming businesses do manually. The bugs have been separated into four main types. Type 1 bugs are those that can be remedied by changing a single line of code, whereas Type 2 defects may be repaired in several places. Multiple bugs that are fixed simultaneously are referred to as Type 3. An extension of Type 3 is Type 4 when several issues are fixed by changing a single group of places [55].

The reviews are divided into four categories using a probabilistic method: text ratings, issue reports, feature requests, and user experiences. This method makes use of text categorization, natural language processing, sentiment analysis, and review information such as star ratings and tense [56].

Unity games are the most downloaded games in the word. To detect bad smells in the unity project a tool was developed called Unity Linter a static tool that supports unity projects to detect bad smells from the projects. It had a precision of 80% to 100% [57].
Gaming bugs can also be detected in real-time using deep enforcement learning techniques [18]. Deep enforcement learning (DRL) may be used to locate vulnerabilities, assess map difficulty, broaden test coverage, and identify recurring issues in first-person shooter (FPS) game testing. Using a random forest classifier, game flaws have also been found in gameplay videos. It classifies gameplay footage according to how likely it is that they include bugs. On a dataset of 96 films that have been manually labeled, the suggested method outperforms the naïve keyword searching method in precision by 43% [5].

It is an important factor that how much time users spend reviewing a game or app. To detect how long users, spend time using the app or playing the game an exploratory case study was presented.

They discovered that the majority of input is given soon after new releases, with frequency rapidly declining with time [4].

To classify the bugs into specific categories based on their location a taxonomy was presented by [8]. Classification of potential failures into temporal and non-temporal faults. This taxonomy can benefit designers and testers alike by directing their ideas and assisting them in identifying game issues. This taxonomy gives a pathway to automate the process of analyzing gaming reviews and divide them into related bug categories.

Gaming reviews can contain more than one bug at the same time which can be multi-type of information. Thus user reviews can be labeled into more than one category [58].
Various approaches exist to classify data having multiple labels including binary relevance, classifier chains, ensembles of binary relevance, and classifier chains. Multi-label review classification problem, feature techniques for rating classification can lead to false feature extraction attempts, due to irrelevant information in the reviews. To solve this problem, an emerging field of the machine learning model has been introduced in recent years, which automatically learns features from labeled data, known as Deep Learning. Which have proved efficient for various text classification problems [43][44][46][47][48][49].

2.4 Summary

This chapter presented brief information about the automated analysis of reviews using traditional and machine leaning methods. It describes the manual and automated methods used in the analysis of reviews. Then the classification models that are used to solve this thesis problem are explained with their working steps.

Chapter 3

3 Pilot Studies

This chapter aimed at answering the RQ1 & RQ3 (defined in chapter 1) Multiple deep learning approaches are available for multi-label text classification. But it was unclear which performed best for the multi-label text categorization. So pilot studies of multiple deep learning models on 1500 gaming reviews datasets were conducted. Which turns out much beneficial for the selection of the right classification model. RNN, CNN, and LSTM are the most popular models for classifying text. All the models are implemented using python 3 using the JUPYTER notebook created in VS code.

3.1. RNN (Recurrent Neural Networks)

A form of neural network called a recurrent neural network (RNN) uses the output from the previous step as the input for the next phase. Sequence classification, emotion classification, and video classification are the key applications for RNNs. Sequence Tagging: Identifies a named thing and marks a section of speech [50].

By repeatedly applying a transition function to the input sequence's internal hidden state vector ht, a recurrent neural network (RNN) [59] may analyze a sequence of any length. Given the current input symbol xt and the prior hidden state ht-1, the activation of the hidden state ht at time step t is calculated as a = function f. It is typical to employ affine transformations of both xt and ht-1 to construct the state-to-state transition function f as a composite of element-wise nonlinearities.

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$

A conventionally straightforward method for modeling sequences Uses Convert to RNN to convert the input sequence into a fixed-size vector, then feed the vectors into a softmax layer for

classification and other tasks.

Gradient vectors can rise or drop exponentially over a long period of Sequence during training, which is the unfortunate issue with RNNs with transfer functions of this kind. RNN models struggle to learn collections of long-range correlations as a result of the inflating or disappearing gradient issue [50].

The general working of the RNN network is shown in Figure 7 below:

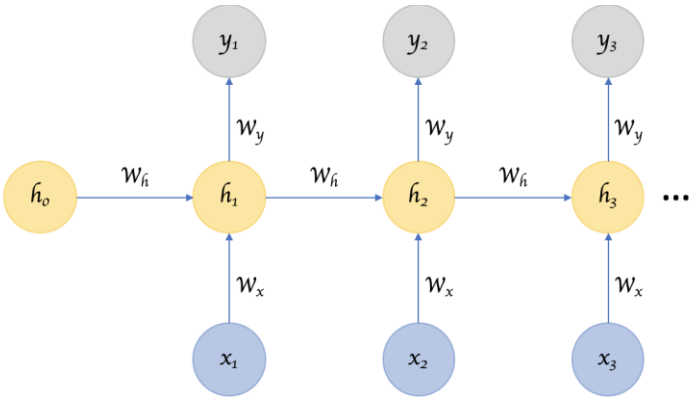
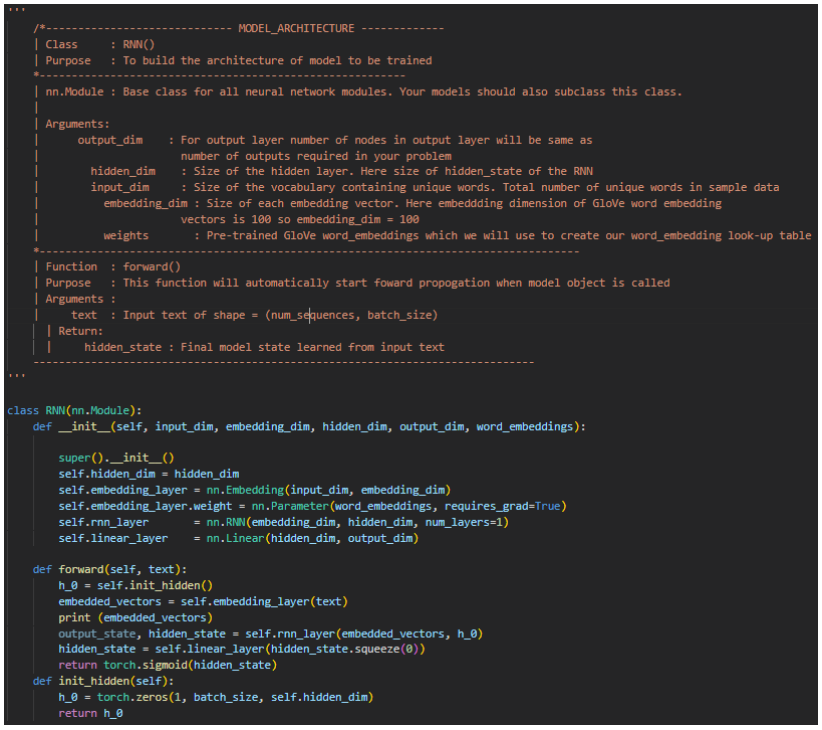


Figure 8 RNN Network [60]

The data was first separated into a **train set**, **validation set**, and **test set**. GloVe embedding is used to implement word embedding. Before word embedding data is Pre-processed. Then the parameters are initialized which include input dimension, embedding dimension, hidden dimension, output dimensions, and word embedding.



```
'''
    /*---------------------------- MODEL_ARCHITECTURE ------------
    | Class     : RNN()
    | Purpose   : To build the architecture of model to be trained
    *-------------------------------------------------------------
    | nn.Module : Base class for all neural network modules. Your models should also subclass this class.
    |
    | Arguments:
    |      output_dim    : For output layer number of nodes in output layer will be same as
    |                      number of outputs required in your problem
    |       hidden_dim    : Size of the hidden layer. Here size of hidden_state of the RNN
    |       input_dim     : Size of the vocabulary containing unique words. Total number of unique words in sample data
    |         embedding_dim : Size of each embedding vector. Here embeddding dimension of GloVe word embedding
    |                      vectors is 100 so embedding_dim = 100
    |          weights       : Pre-trained GloVe word_embeddings which we will use to create our word_embedding look-up table
    *-------------------------------------------------------------------------------------
    | Function  : forward()
    | Purpose   : This function will automatically start foward propogation when model object is called
    | Arguments :
    |     text  : Input text of shape = (num_sequences, batch_size)
    | | Return:
    | |     hidden_state : Final model state learned from input text
    -------------------------------------------------------------------------------
'''

class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim, word_embeddings):

        super().__init__()
        self.hidden_dim = hidden_dim
        self.embedding_layer = nn.Embedding(input_dim, embedding_dim)
        self.embedding_layer.weight = nn.Parameter(word_embeddings, requires_grad=True)
        self.rnn_layer       = nn.RNN(embedding_dim, hidden_dim, num_layers=1)
        self.linear_layer    = nn.Linear(hidden_dim, output_dim)

    def forward(self, text):
        h_0 = self.init_hidden()
        embedded_vectors = self.embedding_layer(text)
        print (embedded_vectors)
        output_state, hidden_state = self.rnn_layer(embedded_vectors, h_0)
        hidden_state = self.linear_layer(hidden_state.squeeze(0))
        return torch.sigmoid(hidden_state)
    def init_hidden(self):
        h_0 = torch.zeros(1, batch_size, self.hidden_dim)
        return h_0
```

Figure 9 RNN Implementation

RNN is a feed-forward model output of one node is the input of the next node. Its implementation is shown in figure 8. At the first hidden start is set to all zeros. Here all the indexes present in the input sequence of the corresponding word vector using our trained word embedding are mapped. Then Applied the RNN layer to start learning the words. Then at last linear layer is applied to the output. Model accuracy is measured using the **Jaccard-Score and f1_score & hamming loss.** The model is later saved to test it for the real-world input and calculated the accuracy which is shown in the below table.

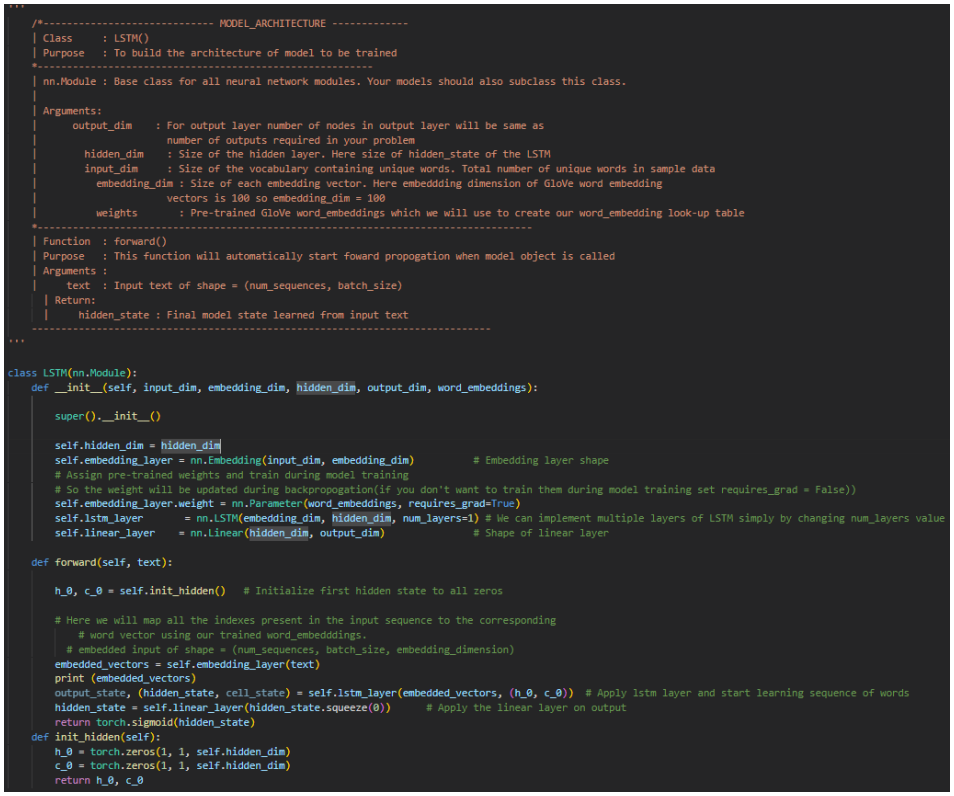
Epochs	Accuracy	Val-Accuracy
1	0.6104	0.6000
2	0.6201	0.6198
3	0.6202	0.6101
Average	0.6169	0.6099

Table 1 RNN Accuracy On 1500 Reviews

3.2. LSTM (Long Short Term Memory)

Recurrent neural networks of a certain type include LSTM, however, LSTM has a greater memory than conventional recurrent neural networks. Due to hold over remembering particular memory patterns, LSTMs function considerably better. Similar to other neural networks, LSTMs can have numerous hidden layers. As information is stored via each layer, all irrelevant information in each cell is discarded. It had been extensively utilized in recent studies [51] [52]. The chapter 3 description already covers the fundamentals.

It is implemented using the python torch (machine learning) library. Multiple layers of LSTM are implemented. As an activation function, the sigmoid function is employed. Chapter 4 of the manual explains the model's specifics. Its implementation is done using python as shown in figure 9.



```
'''
    /*---------------------------- MODEL_ARCHITECTURE -------------
    | Class     : LSTM()
    | Purpose   : To build the architecture of model to be trained
    *-------------------------------------------------------------
    | nn.Module : Base class for all neural network modules. Your models should also subclass this class.
    |
    | Arguments:
    |      output_dim    : For output layer number of nodes in output layer will be same as
    |                      number of outputs required in your problem
    |       hidden_dim    : Size of the hidden layer. Here size of hidden_state of the LSTM
    |       input_dim     : Size of the vocabulary containing unique words. Total number of unique words in sample data
    |         embedding_dim : Size of each embedding vector. Here embeddding dimension of GloVe word embedding
    |                       vectors is 100 so embedding_dim = 100
    |          weights       : Pre-trained GloVe word_embeddings which we will use to create our word_embedding look-up table
    *-------------------------------------------------------------------------------------
    | Function  : forward()
    | Purpose   : This function will automatically start foward propogation when model object is called
    | Arguments :
    |     text  : Input text of shape = (num_sequences, batch_size)
      | Return:
      |     hidden_state : Final model state learned from input text
    ----------------------------------------------------------------------------
'''

class LSTM(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim, word_embeddings):

        super().__init__()

        self.hidden_dim = hidden_dim
        self.embedding_layer = nn.Embedding(input_dim, embedding_dim)         # Embedding layer shape
        # Assign pre-trained weights and train during model training
        # So the weight will be updated during backpropogation(if you don't want to train them during model training set requires_grad = False))
        self.embedding_layer.weight = nn.Parameter(word_embeddings, requires_grad=True)
        self.lstm_layer      = nn.LSTM(embedding_dim, hidden_dim, num_layers=1) # We can implement multiple layers of LSTM simply by changing num_layers value
        self.linear_layer    = nn.Linear(hidden_dim, output_dim)              # Shape of linear layer

    def forward(self, text):

        h_0, c_0 = self.init_hidden()   # Initialize first hidden state to all zeros

        # Here we will map all the indexes present in the input sequence to the corresponding
            # word vector using our trained word_embedddings.
          # embedded input of shape = (num_sequences, batch_size, embedding_dimension)
        embedded_vectors = self.embedding_layer(text)
        print (embedded_vectors)
        output_state, (hidden_state, cell_state) = self.lstm_layer(embedded_vectors, (h_0, c_0))  # Apply lstm layer and start learning sequence of words
        hidden_state = self.linear_layer(hidden_state.squeeze(0))      # Apply the linear layer on output
        return torch.sigmoid(hidden_state)
    def init_hidden(self):
        h_0 = torch.zeros(1, 1, self.hidden_dim)
        c_0 = torch.zeros(1, 1, self.hidden_dim)
        return h_0, c_0
```

Figure 10 LSTM Implementation

The results for the model accuracy for 1500 reviews for 3 epochs are given in the table below:

Epochs	Accuracy	Val-Accuracy
1	0.6404	0.6400
2	0.6401	0.6498
3	0.6402	0.6401
Average	0.6402	0.6433

Table 2 LSTM Model Accuracy On 1500 Reviews

3.3. CNN (Convolutional Neural Network)

Convolutional neural networks (CNNs), originally developed for computer vision, have demonstrated impressive performance in text categorization tasks [47].

With growing popularity, CNN has progressed in interpretability When used for computer vision tasks [47]. Unfortunately, these techniques do not trivially hold for discrete sequences. assume a continuous input space used for the representation picture. Intuition about how CNN works for you The level of abstraction should not be carried over from the painting either. Text input - e.g. for CNN pooling. It was used to induce strain invariance [47]. this is probably different than a role in a word processor. Below Figure 10 shows the layers of CNN.

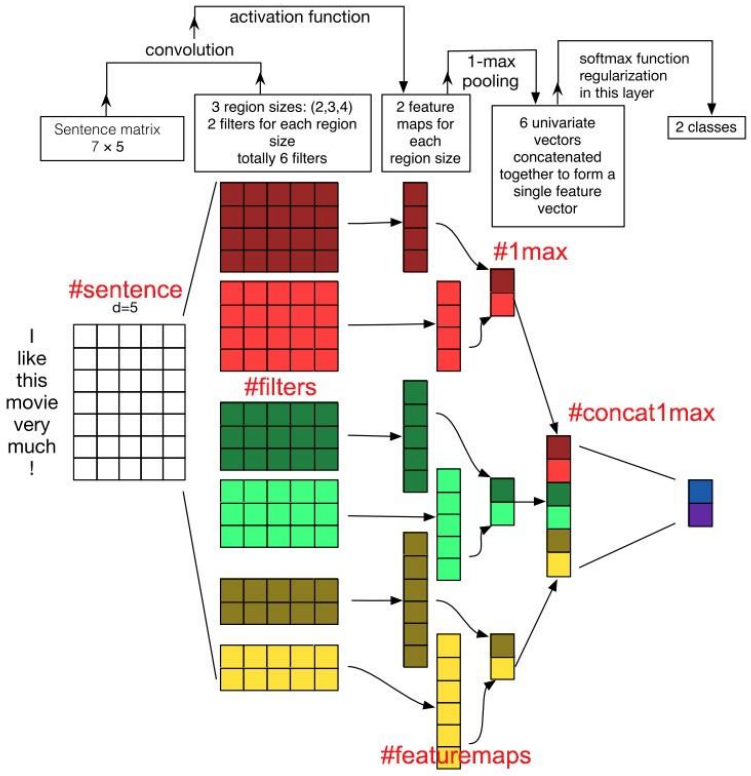


Figure 11 CNN Layers

We have implemented it using the python torch (machine learning) library. The model consists of three layers: a pooling layer, a convolutional layer, and a fully connected layer as the final layer. The first layer, a pooling layer, has a smaller feature map size. It decreased the amount of processing in the network and the number of parameters to learn. In the CNN model, the convolutional layer filter map is applied to the original dataset to create feature maps. There is where most of the user-specified parameters are in the network. Here kernel size is also defined. And the fully connected layer contained the final model output which has connected nodes with the output labels.

Since CNN is primarily used for multi-label image classification issues, it was also adjusted for multi-label text classification.
The customized multi-label bugs categorization model states that:

- The number of labels matches the number of nodes on the output layer.
- The function of activation Each node in the output layer uses a sigmoid.
- To compare each of the predicted probabilities to the actual class output, which can be either 0 or 1, a binary cross-entropy loss function is used.

Results of the 3 epochs for the testing data set of 1500 reviews are presented in the below table:

Epochs	Accuracy	Val-Accuracy
1	0.4904	0.4900
2	0.5001	0.4993
3	0.5002	0.5000
Average	0.4969	0.4963

Table 3 CNN Model Accuracy On 1500 Reviews

3.4. Summary

This chapter gives an overview of different deep learning algorithms' accuracy on the same dataset. This study helped us to continue with the best fit model for the multi-label game reviews classification which has higher accuracy. Furthermore, it shows the accuracies of other deep learning models. And the implementation details are discussed.

Chapter 4

4. Analyzing Steam Platform Games Reviews for Bugs Classification

This chapter focused on answering RQ2 (defined in chapter 1) on mining the gaming reviews in multiple bug categories [8]. The introduction is provided in section 4.1 of the chapter, which is divided into further sections. The research concept and data-collecting procedure are described in Section 4.2. The examination of game reviews using automated approaches was covered in Section 4.3. The conclusion is described in Section 4.4, and the chapter summary is provided in Section 4.5.

4.1. Introduction

User reviews are composed of sentences made up of words in a natural language format. Review sentences are unprocessed text that adds extra symbols and words that are unnecessary for further processing. To get rid of this extraneous data, text pre-processing techniques are applied, such as eliminating stop words, tokenizing sentences, and removing all punctuation and symbols [61].

To extract information from the review sentences, they are broken up into words of tokens. These words serve as review sentence characteristics, and feature extraction is the process of gathering them. Certain aspects have been extracted from evaluations using a variety of methods, and their expansions are recommended. BoW and Tf-IDF, however, are the most modern and widely used feature extraction techniques [62].

Since it is not possible to implement a classification algorithm on the raw text of the gaming reviews, common feature extraction methods are used to define a feature set for training the ML model. There is research on classifying user reviews using different machine-learning algorithms and feature extraction methods. Answering the RQ2 requires evaluating the effectiveness of the Bag-of-words and TF-IDF approaches with a machine learning algorithm on multi-label gaming

faults.

4.2. Research Design

This research follows a data-driven approach all the decisions are based on the analysis and interpretation of data rather than on observations.

To answer the RQ2 fifteen different games reviews were crawled by a crawler developed using python as shown in the figure below:

product_i	page	page_ord	recomme	date	text	hours	username	products	early_access
1144200	1	0	TRUE	1-Mar	10/10 taser shots to a civilians test	6.1	BURNtheHERETIC	74	TRUE
1144200	1	1	TRUE	1-Mar	good	2.7	Hynek	54	TRUE
1144200	1	2	TRUE	1-Mar	Love the game and would give it a	35.6	Philcrest	31	TRUE
1144200	1	3	TRUE	1-Mar	eygg	16.5	doooomkid	16	TRUE
1144200	1	4	TRUE	1-Mar	Great game, very fast pace. even t	48.6	Marshmallow_Warrior	8	TRUE
1144200	1	5	FALSE	1-Mar	If you like scare thrills this is for yo	0.5	Serevok	196	TRUE
1144200	1	6	TRUE	1-Mar	yes	11	Humanmonkey207	12	TRUE
1144200	1	7	TRUE	1-Mar	good	10.2	Alex.N	5	TRUE
1144200	1	8	TRUE	1-Mar	sex0	8.8	N0 tengo Skins NO pidas puta	30	TRUE
1144200	1	9	TRUE	1-Mar	shooting and door booting	4.8	Samaghan	21	TRUE
108600	1	0	TRUE	1-Mar	In what can only be described as a	6.5	UltraMarine716	17	TRUE
108600	1	1	TRUE	1-Mar	Grindy and realistic zombie apocal	69	Bed Wetter	74	TRUE
108600	1	2	TRUE	1-Mar	Very challenging, as a post zombie	142.5	Arkguil	42	TRUE
108600	1	3	TRUE	1-Mar	great game	18.4	josecristina8792	3	TRUE
108600	1	4	TRUE	1-Mar	fun game and very realistic	10.4	ð,fð,ƒ	12	TRUE
108600	1	5	TRUE	1-Mar	Amazing game, amazing communi	113.9	N-ine	155	TRUE
108600	1	6	FALSE	1-Mar	not good, dont reccomend buying	2.6	batzombie07	19	TRUE
108600	1	7	FALSE	1-Mar	Fun game ruined by overly punish	180.9	Kobogen	157	TRUE
108600	1	8	TRUE	1-Mar	I'm a fan of 1st person graphics	711.7	intorpere	93	TRUE
108600	1	9	TRUE	1-Mar	coollest game on thw wpor;ld	4.1	brianyu727	5	TRUE
1086940	1	0	TRUE	1-Mar	Great game in the making. Will	24.9	Elkraf	46	TRUE
1086940	1	1	FALSE	1-Mar	Like, it's definitely fun, but the sta	81.4	WickedMystic	142	TRUE

Figure 12 Crawled Reviews

Then the crawled data is preprocessed and labeled with fifteen different gaming bug categories mentioned in the taxonomy [8]. Multiple factors affect the model performance and result such as the quality of the dataset, features of the model, and language of the reviews because our interest is in the automatic extraction of gaming bugs from reviews.

4.2.1. Data Collection

Data collection steps are comprised of two steps:

- Games and category selection
- Games reviews selection

Games and category selection

During the data collection, the step focus was on game categories to get the reviews dataset. Fifteen different early-access action games on the game steam platform. To classify them into distinct bug categories [8]. The dataset with no reviews, attributes, or categories is shown in below table:

Dataset	No. of reviews	No. of target attributes	Game Categories
Ready or Not [63]	600	15	Early access action
Project Zombies [64]	600	15	Early access action
Three Finger Battle Arena [65]	600	15	Early access action
Baldur's Gate 3 [66]	600	15	Early access action
Trip In Another World [67]	600	15	Early access action
NEBULOUS: Fleet Command [68]	600	15	Early access action
ERRANTE [69]	600	15	Early access action
HITMAN 3 [70]	600	15	Early access action
Grand Theft Auto V [71]	600	15	Early access action
Tomb Raider [72]	600	15	Early access action
Red Dead Redemption 3 [73]	600	15	Early access action
Max Payne 3 [74]	600	15	Early access action
Resident Evil 6 [75]	600	15	Early access action
Just Cause 3 [76]	600	15	Early access action
Star Wars: Battlefront 2 [77]	600	15	Early access action

(classic, 2005)			

Table 4 Dataset Description & Categories

Games reviews selection

Reviews are scrapped using the custom scrapper tool SCRAPPY. Twelve thousand reviews were crawled and stored in a CSV file. Later after cleaning the reviews, the most relevant ten thousand reviews were chosen for classification. Our focused mainly was on early access games all the reviews contain a detailed description of bugs because these reviews are in detail. Which is very helpful for accurate results in deep learning models for bug classification.

4.3. Automated Bugs Classification

Here this chapter will discuss the performance of the automated method for bug classification of reviews that will automatically classify the bugs into their related categories.
Our research path is also mainly related to the automatic classification of gaming bugs from user gaming reviews.

Here each review is classified into more than one category. For its representation let's say L is a label from a set of multiple labels L1, L2, L3 …. Ln. And now, define p as the output space and q as the input space as subsets of the labels-set L. Task J can be assigned for the categorization of multi-label defects. J = (p1, q1) p, q, where p1 and q1 are examples that fall under the p and q labels.

4.3.1. Multi-label Bugs Classification

In the previous steps of gaming, reviews are classified in the form of 0 and 1 in fifteen different gaming bug categories. Where 0 represents this is not a bug and where 1 represents it is a bug. As this problem falls into multiple categories which are multi-label classifications so it cannot be solved using a simple binary text classification method.

A multi-label issue can be split up into several single-label classification problems using one of two primary kinds of multi-label procedures, referred known as "problem transformation" approaches. The second category of approaches uses a particular algorithm to directly predict multiple labels in the single-label classification issue.

To classify the bugs into multiple categories Problem Transformation Method is adopted in which one bug can contain more than one bug. So It can fall into multiple labels of bugs at the same time. Figure 12 shows a difference between multi-class and multi-label classification.

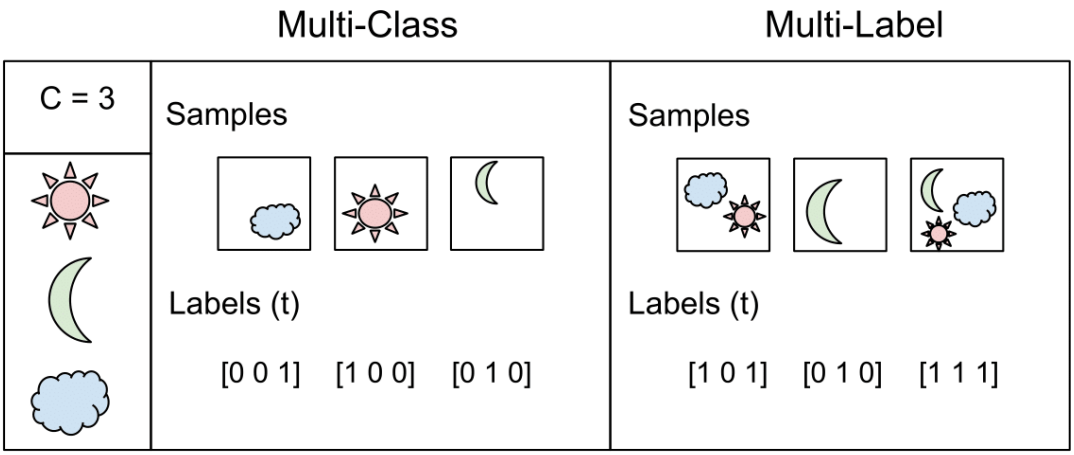


Figure 13 Multi-label and Multi-class classification [78]

4.3.2. Reviews Pre-Processing

User reviews are natural language text, contain additional information, and use text reduction techniques to remove additional information and remove words that may be present to influence the predictions of the classifier. NLP techniques were used which include stop-word-removal to remove common words such as ‘is’, ‘an’, etc. which do not change the actual meaning of the text. Other techniques of NLP which are lemmatization and stemming are applied for further preprocessing.

In linguistics, lemmatization is the act of combining a word's inflected forms into a single unit for analysis using the word's lemma, or dictionary form. Run, for instance, is the lemma of all the terms runs, running, and running since they are all variations of the word run.

Stemming is a method for eliminating affixes from words to reveal their basic structure. It is analogous to trimming a tree's branches down to the trunk. For instance, the word eat is the root of the verbs eating, eats, and eaten.

Additional pre-processing is carried out, including the elimination of spaces, special characters, and other useless data.

Features Extraction

The technique of turning raw data into numerical features that can be handled while keeping the information in the original data set is known as feature extraction. Compared to using machine learning on the raw data directly, it produces superior outcomes. While there are several feature extraction techniques, BoW and Tf-IDF are the most advanced.

Bag-of-words

A reduced version of a text document, known as a bag-of-words (BoW) model, is made up of text fragments chosen according to specific criteria, such as B. Word repetition. BoW models are used in artificial intelligence (AI) information retrieval, Bayesian spam filters, natural language processing (NLP), document categorization, and computer vision.

A phrase or document that is considered a body of text by BoW is a collection of words. In the BoW procedure, a word list is produced. Because the grammar was disregarded during collection and creation, these words are not sentences. Words frequently convey a sentence's meaning. Grammar and appearance order are disregarded, but multiplicity is recorded and can be used subsequently to evaluate the document's emphasis. Semantic connections are disregarded, but each

term's frequency is recorded.

Figure 13 shows the illustration of a Bag of words vectors with an example.

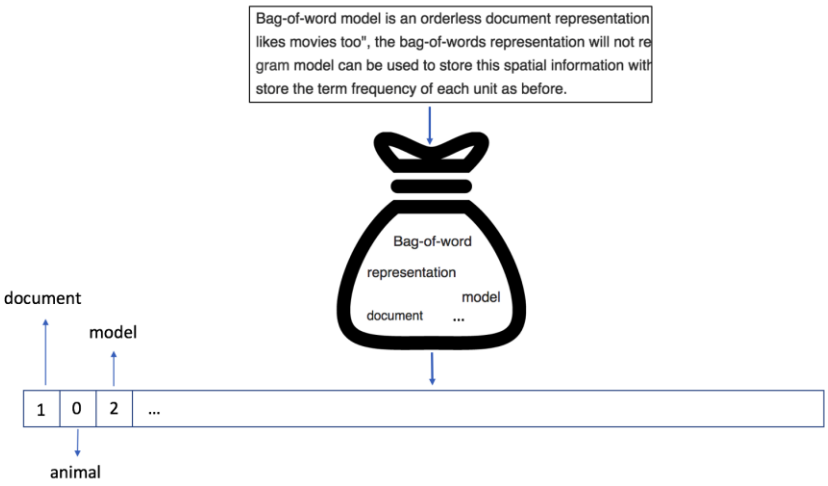


Figure 14 Bag of Words Representation [79]

The study by Maalej and Nabil [41] classified a whole review text like feature requests and problem reports using BoW features.

Term Frequency - Inverse Document Frequency

A statistical technique called TF-IDF (term frequency-inverse document frequency) assesses how relevant a word is to a document within a collection of documents.
For document and information retrieval, TF-IDF was developed. When a word appears in a document, the value is increased according to that occurrence, but offset by the number of papers that include that term. Therefore, even though they may appear frequently, terms like this, what, and if rank low since they aren't really important to that paper. It is calculated using the formula given below:

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

4.3.3. Model Classifier

Data analysis that uses classification extracts models that explain different classes of data. A classification model or classifier forecasts category labels (classes). mathematical prediction model for continuous-valued functions. The two main categories of prediction challenges are classification and numerical prediction.

Machine learning has various dimensions it can be supervised, unsupervised and semi-supervised. However, because our focus is on supervised machine learning to conduct bug categorization, the most popular techniques are covered in this section.

Random Forest:

A "forest" is created by growing and combining various decision trees using the supervised machine learning method Random Forest. By picking data randomly multiple decision trees are constructed [80].

Logistic Regression:

An illustration of supervised learning is logistic regression. It is used to determine or forecast the likelihood that a binary (yes/no) event will occur. It takes into account linear combinations of dissimilar functions and uses rating-specific parameters to determine the likelihood of a particular

type of rating.

4.3.4. Evaluation Techniques

Standard measures were used to assess the reference classification model, including accuracy and recall, and accuracy. The results of multi-label categorization might be entirely right, mostly correct, or incorrect. Therefore, new metrics that may be utilized for partial accuracy should be added to the usual accuracy and recognition measurements that are generally employed in binary classification tasks. They are used in conjunction with the aforementioned metrics to determine recall (R) and precision (P) values for the macro-mean of the model to present such statistics. To assess our findings, test hamming and the f-test were used.

4.3.5. Classification

Classification is trained over the defined labels mentioned in chapter 5 and below figure 14 dataset labels.

```
bug_categories_labels = data_set[["invalidPositionOverTime", "implementationResponseIssue", "invalidContextOverTime", "interruptedEvent",
                        "invalidEventOccurraceOverTime","actionNotPossible","actionWhenNotAllowed","informationOutOfOrder","lackOfRequiredInformation",
                        "invalidInfoAccess","objectOutOfBoundForAnyState","objectOutOfBoundForSpecificState","artificialStupidity","invalidValueChange",
                        "invalidGraphicalRespresentation","haveBugs"]]
bug_categories_labels.head()
```

Figure 15 Dataset labels

We used the python KERAS library to implement the classification model. For training and testing the model, data is split 80:20.
The utilized library is displayed in figure 15 below:

```
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, LSTM
from keras.layers import GlobalMaxPooling1D
from keras.models import Model
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.layers import Input
from keras.layers.merge import Concatenate

import pandas as pd
import numpy as np
import re

import matplotlib.pyplot as plt
```

Figure 16 Libraries

The suggested classification model's implementation is covered in full in the chapter proposed classification model.

4.4. Conclusion

This research study focused on the problem of automated classification of bugs from gaming reviews. The approach was analyzed by using automated classification models to automatically classify bugs into their related categories with feature extraction methods BoW and TF-IDF. Then the classification feature is applied to ten thousand reviews (chapter 5) to train a model and obtain an 82% accuracy.

The result of our study can help developers to better understand what kind of issues gamers are facing in games. Also, it can help to make the business decision better.

4.5. Summary

This chapter presented the implementation of an automated classification model for gaming bugs. First, the approaches used are defined for data collection and data pre-processing. Then automated classification approach to easily classify reviews using deep learning models. The next chapter focused on devising the proposed classification model LSTM.

Chapter 5

5. Proposed Classification Model

This chapter highlights the problem which is discussed in chapter 1 and its proposed solutions. It offers a thorough explanation of the architecture of the suggested model and the MLBCGR-LSTM flow. Then the working of the model with evaluation and results are presented.

5.1. The architecture of the Proposed Model

The LSTM model has demonstrated greater accuracy for multi-label text categorization, according to chapter 3. In this section, high-level architecture for the implementation of the proposed classification model LSTM is shown. This architecture mainly consists of review scrapping (collection of reviews), text preprocessing then a classification model that is depicted in figure 17.

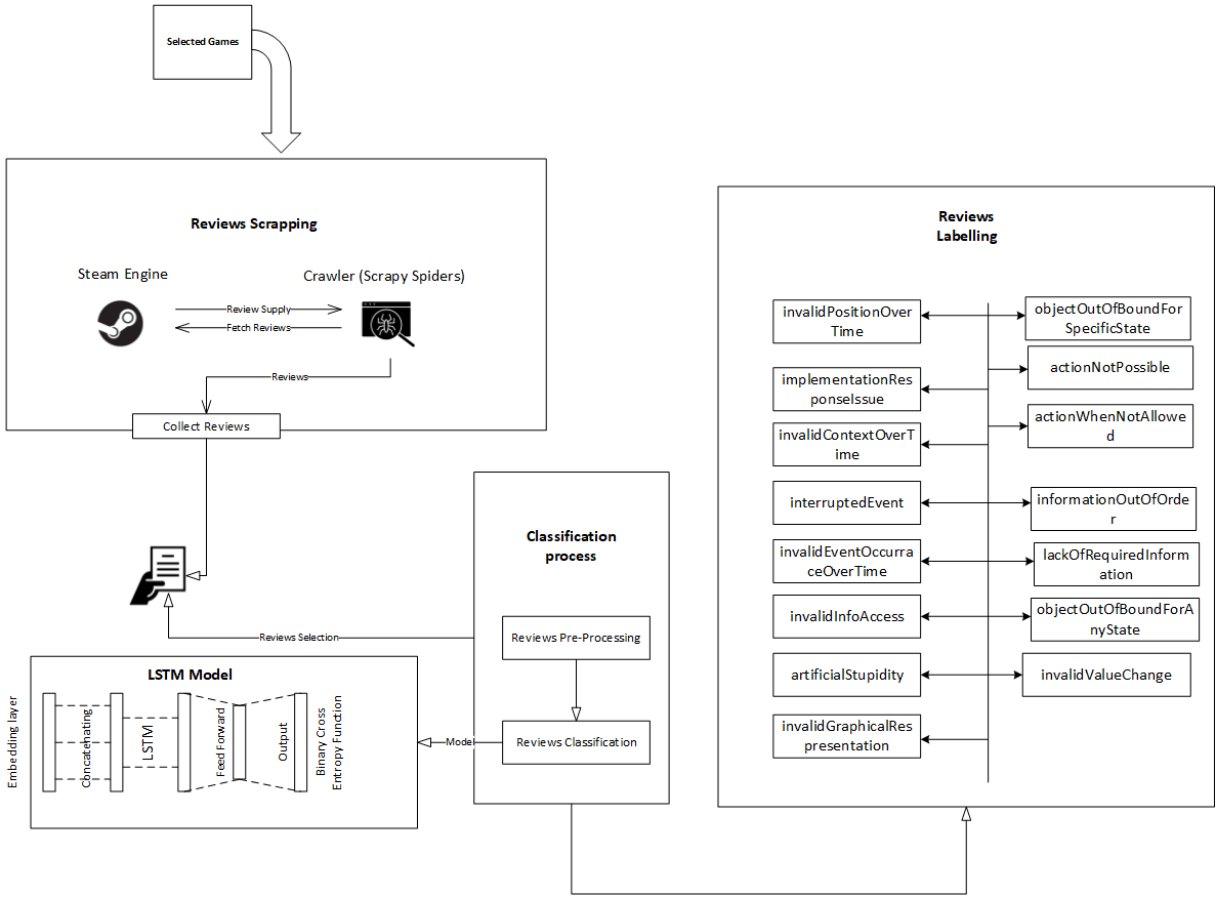


Figure 17 Proposed Solution's Architecture Diagram

The detail of each step of the architecture diagram is following.

5.1.1. Games Selection

As discussed in the chapter pilot studies dataset was needed for the classification of gaming bugs. Fifteen different games are considered on a GAME STEAM platform which is shown in below table:

Dataset	Game Categories
Ready or Not [63]	Early access action
Project Zombies [64]	Early access action
Three Finger Battle Arena [65]	Early access action
Baldur's Gate 3 [66]	Early access action
Trip In Another World [67]	Early access action
NEBULOUS: Fleet Command [68]	Early access action
ERRANTE [69]	Early access action
HITMAN 3 [70]	Early access action
Grand Theft Auto V [71]	Early access action
Tomb Raider [72]	Early access action
Red Dead Redemption 3 [73]	Early access action
Max Payne 3 [74]	Early access action
Resident Evil 6 [75]	Early access action
Just Cause 3 [76]	Early access action
Star Wars: Battlefront 2 [77] (classic, 2005)	Early access action

Table 5 Games Selection

This research problem considered the early access games on the steam platform. Because these games have detailed reviews from the users. Gaming industries post these games to get to know the user's views about the games.

5.1.2. Reviews Scrapping

Reviews are scrapped from the STEAM ENGINE GAME platform using a customized web scrapper known as **SCRAPY**. This tool was implemented in python by using the import given in

python called SCRAPY as shown in the below figure.

```
import re

import scrapy
from scrapy.http import FormRequest, Request
from w3lib.url import url_query_parameter

from ..items import ReviewItem, ReviewItemLoader, str_to_int
```

Figure 17 Python Scrapy Libraries

After carefully analyzing the steam platform thing comes to the knowledge that every game on STEAM ENGINE has a unique identification key.

To save time and effort this tool is designed in a way that it can scrap reviews of the game by inputting the game IDs (every game has a unique ID on the game steam platform). Scrappers have multiple functions to crawl the review in the correct format. To start the request a function is defined which receives a URL and makes a request and then parses the result.

```
def start_requests(self):
    if self.steam_id:
        url = (
            f'http://steamcommunity.com/app/{self.steam_id}/reviews/'
            '?browsefilter=mostrecent&p=1'
        )
        yield Request(url, callback=self.parse)
    elif self.url_file:
        yield from self.read_urls()
    else:
        for url in self.test_urls:
            yield Request(url, callback=self.parse)
```

Figure 18 URL Request STEAM Engine

After parsing the reviews, a loader function is designed which get the required information from a request-response and map that in different parameters.

```python
def load_review(review, product_id, page, order):
    """
    Load a ReviewItem from a single review.
    """
    loader = ReviewItemLoader(ReviewItem(), review)

    loader.add_value('product_id', product_id)
    loader.add_value('page', page)
    loader.add_value('page_order', order)

    # Review data.
    loader.add_css('recommended', '.title::text')
    loader.add_css('date', '.date_posted::text', re='Posted: (.+)')
    loader.add_css('text', '.apphub_CardTextContent::text')
    loader.add_css('hours', '.hours::text', re='(.+) hrs')
    loader.add_css('compensation', '.received_compensation::text')

    # User/reviewer data.
    loader.add_css('user_id', '.apphub_CardContentAuthorName a::attr(href)', re='.*/profiles/(.+)/')
    loader.add_css('username', '.apphub_CardContentAuthorName a::text')
    loader.add_css('products', '.apphub_CardContentMoreLink ::text', re='([\d,]+) product')

    # Review feedback data.
    feedback = loader.get_css('.found_helpful ::text')
    loader.add_value('found_helpful', feedback, re='([\d,]+) of')
    loader.add_value('found_unhelpful', feedback, re='of ([\d,]+)')
    loader.add_value('found_funny', feedback, re='([\d,]+).*funny')

    early_access = loader.get_css('.early_access_review')
    if early_access:
        loader.add_value('early_access', True)
    else:
        loader.add_value('early_access', False)

    return loader.load_item()
```

Figure 19 Reviews Mapper

The reviews are in JSON format and are converted into CSV format for the Pre-Processing techniques. The crawled reviews from scrapper are shown in the below figure.

product_i	page	page_ord	recomme	date	text	hours	username	products	early_acc	user_id
1144200	1	0	TRUE	1-Mar	10/10 taser shots to a civilians testicles!	6.1	BURNtheH	74	TRUE	
1144200	1	1	TRUE	1-Mar	good	2.7	Hynek	54	TRUE	
1144200	1	2	TRUE	1-Mar	Love the game and would give it a 10 out 10 but they toc	35.6	Philcrest	31	TRUE	7.66E+16
1144200	1	3	TRUE	1-Mar	eygg	16.5	doooomki	16	TRUE	
1144200	1	4	TRUE	1-Mar	Great game, very fast pace. even better with friends wh	48.6	Marshmal	8	TRUE	7.66E+16
1144200	1	5	FALSE	1-Mar	If you like scare thrills this is for you. If you like action, t	0.5	Serevok	196	TRUE	7.66E+16
1144200	1	6	TRUE	1-Mar	yes	11	Humanmc	12	TRUE	
1144200	1	7	TRUE	1-Mar	good	10.2	Alex.N	5	TRUE	7.66E+16

Figure 20 Crawled Reviews

5.1.3. Review Classification Process

The review classification process comprises of following steps:

5.1.3.1. Review Pre-processing

As data was crawled using the custom scrapper, there were many inconsistencies in the data which need to be resolved to get accurate results because most of the ML and DL models' results depend on the dataset consistency [81].

First of all, reviews were cleaned for removing empty spaces, and special characters using python data cleaning basic functions. Furthermore, missing values are those reviews that are missed when crawled from the game steam engine, filled taking the most repeated value in the dataset. User reviews are plain text, words that use additional symbols or punctuation, no order of sentences, and are used by users to post comments about the app as shown in figure 19.
Here yellow color shows the inconsistencies inside the data and the blue color means consistent data.

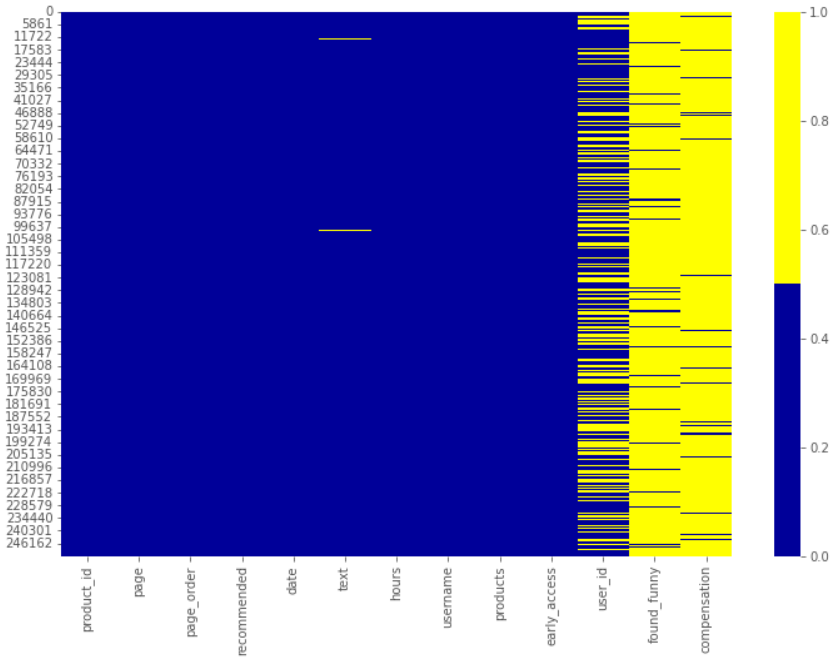


Figure 21 Missing Values Heat Map

After using fundamental cleaning techniques including eliminating spaces and special characters, deleting outliers, and filling in the missing numbers. The mean of the most frequent values was calculated to fill in any missing data. Pre-processing techniques were still required because the data is in textual format. And textual format contains so much other information which is not related to our problem and its solution. Consequently, Natural Language Processing (NLP) techniques such as tokenization, stop word removal, stemming, and lemmatization was used to eliminate these symbols and punctuation marks to improve the model's performance (explained in chapter 4). These techniques were applied using the python **KERAS pre-processing** library which supports NLP pre-processing techniques as shown in the figure given below.

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1

maxlen = 200

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

Figure 22 KERAS Tokenizer

After applying the pre-processing techniques and NLP techniques the resulting dataset was cleaned from outliers and have no inconsistences in it. As heat map in below figure show that it does not contain any missing or inconsistent value.

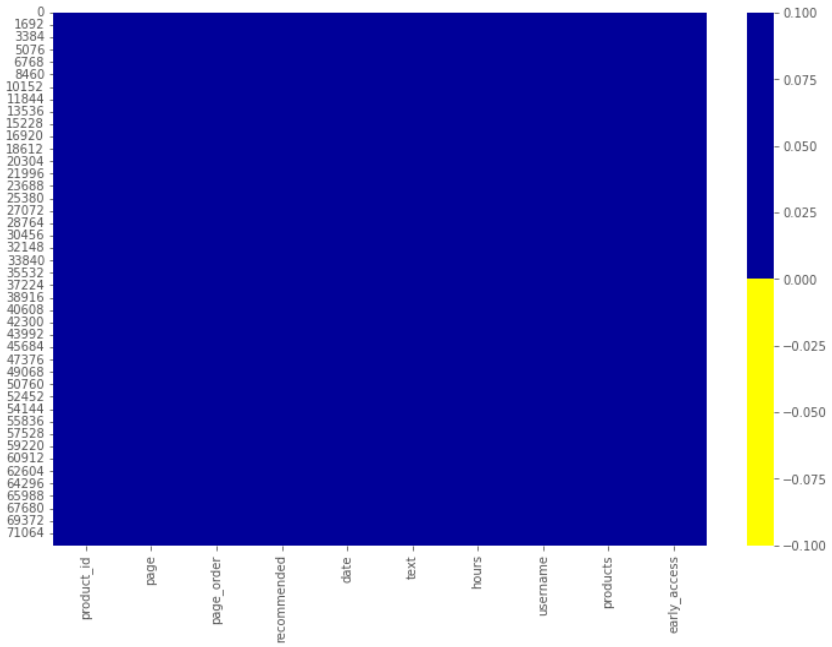


Figure 23 Dataset Heat map After Cleaning

5.1.3.2. Reviews Labelling

After the pre-processing of the dataset, the next step that was required to train the gaming bugs classification model was review labeling. Pre-processed cleaned data was distributed to gaming experts and asked them to manually read and label them in fifteen distinct gaming bug categories as described in Lewis Taxonomy [8]. These categories are discussed below with examples.

1. InvalidPositionOverTime

This mistake happened when an object moved improperly. For instance, the character moves excessively swiftly after being hurled into the air in GTA IV [8]. The figure below illustrates an incorrect wall location.

Figure 24 No Man Sky Game Invalid position on the wall [82]

2. implementationResponseIssue

Error happened when the game couldn't run quickly enough. For instance, latency might occur while playing Rainbow Six Vegas 2 when shooting a certain NPC (Non-Player Character) [8].

3. **invalidContextStateOverTime**

When an item is put into a state for an excessively short time, the error occurs. For instance, the character in Street Fighter II gets shocked for the whole eighth round.

4. **interrupted event**

An error occurred when an event that was in action has now stopped before ending. e.g. While playing Call of Duty 4: Modern Warfare while in the cut scene game characters stop moving [8].

5. **invalidEventOccurraceOverTime**

An occurrence that occurs too frequently or seldom enough in a game is considered to be an error. Using Left 4 Dead as an example Too many melee attacks occur as a result of weapon cycling [8].

6. **action not possible**

When an action is permitted but an item in the game world cannot take it, an error occurs. e.g. while playing Pokémon Gold not able to pick up items even if it is allowed in the game [8].

7. **actionWhenNotAllowed**

An error occurred when Object in-game world can take action when the action is supposedly paused. e.g. while playing Goldeneye's character can be shot when a cut scene or game is paused [8].

8. **informationOutOfOrder**

This happened when the player learned about the game's universe in an unexpected sequence (out of context). e.g. Even after a task is finished in Knights of the Old Re-Public, players will continue to ask you to do it [8].

9. lackOfRequiredInformation

Occurred when a player does not receive the required information /she required to play the game. E.g. while playing the Mass Effect camera pointing pointed in the wrong direction even if the player is looking in a different direction [8].

10. invalidInfoAccess

Occurred when a player is given extra information which is not required. e.g. Walls are transparent to players in Call of Duty: World at War [8].

11. objectOutOfBoundForAnyState

Occurred when an object is at an invalid position in a world no matter what the game state is. e.g. in Left 4 dead objects fall through the floor of the elevator no matter how you enter it [8].

12. objectOutOfBoundForSpecificState

Occurred when the object is at an invalid position in the world/map for some specific states. e.g. in left 4 dead players can skip boundaries by clipping errors in the game [8].

13. artificialStupidity

Occurred when Artificial intelligence showed poor reasoning in-game. e.g. ally wanders on the earth and thinks in "Knights of the Old Republic" [8]. The below figure shows the example of jumping form the truck while moving and hitting the ladder above it.

Figure 25 Artificial Stupidity [83]

14. invalidValueChange

Occurred when the value that needs to be changed, changes incorrectly. e.g. in GTA IV sometimes bullets do not do any damage [8]. The below figure shows the invalid value change error n the Minecraft game.

Figure 26 Invalid Value Change Error In Minecraft [84]

15. invalidGraphicalRespresentation

As its name represents graphics with different glitches. Objects in games appear incorrectly different from the normal behavior of the game [8].

Figure 27 Swimming animation while playing golf [8]

The dataset description that needs to be labeled is shown in below table:

Dataset	No. of reviews	Attributes	No. of target attributes	Missing Values	Pre-Processing Required
Ready or Not [63]	600	1	15	Yes	Yes
Project Zombies [64]	600	1	15	Yes	Yes
Three Finger Battle Arena [65]	600	1	15	Yes	Yes
Baldur's Gate 3 [66]	600	1	15	Yes	Yes
Trip In Another World [67]	600	1	15	Yes	Yes
NEBULOUS: Fleet Command [68]	600	1	15	Yes	Yes
ERRANTE [69]	600	1	15	Yes	Yes
HITMAN 3 [70]	600	1	15	Yes	Yes
Grand Theft Auto V [71]	600	1	15	Yes	Yes
Tomb Raider [72]	600	1	15	Yes	Yes
Red Dead Redemption 3 [73]	600	1	15	Yes	Yes
Max Payne 3 [74]	600	1	15	Yes	Yes
Resident Evil 6 [75]	600	1	15	Yes	Yes
Just Cause 3 [76]	600	1	15	Yes	Yes
Star Wars: Battlefront 2 [77] (classic, 2005)	600	1	15	Yes	Yes

Table 6 Dataset Features Map Details

One hot encoding technique was used to ask gaming professionals to categorize the game reviews in the aforementioned categories. One hot encoding method means if a review falls into one or more different bug categories it will be marked as 1 and the other will be marked as 0. One hot encoding is already described in detail in the chapter 2 section on automated review analysis. By that technique, 10,000 reviews of games were labeled successfully.

The gaming expert's demographics are shown in the below table.

Variable	Item	Number
Gender	Male	60
	Female	40
Age	18 ~ 20	64
	20 ~ 25	36
Education	University degree or higher	56
Major	Computer Science	55
	Software Engineering	45
Games play Experience	1 year	30
	2 years	20
	3 years	25
	4 years	25

Table 7 Gaming Experts Demographics

Analysis of the dataset after successfully getting them labeled from the gaming experts is done and also able to calculate the majority occurrence of bugs in the dataset as shown below:

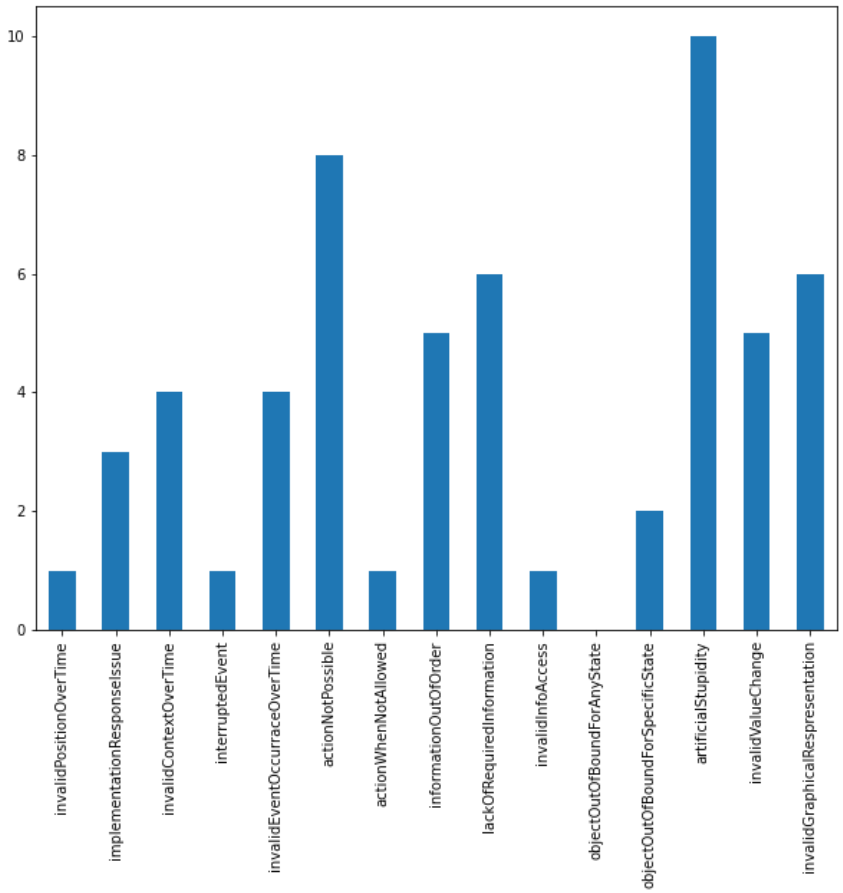


Figure 28 Bugs Occurrence Count in Reviews

5.1.3.3. Words Embedding

After labeling the data, data cannot be directly fed into the deep learning model because textual data cannot be directly inputted into the deep learning model. It needs to be converted into a vector format. Then the model is initialized. Global vectors for word representation are the abbreviation for GloVe embedding. Researchers at Stanford University created an unsupervised learning method to create word embeddings by combining global word co-occurrence matrices from a particular corpus. It is used to embed words into floating-point values and calculate the co-occurrence of values [85]. GloVe embedding is used to encode the input data into vectors.

The implementation of gloVe is shown in the below figure:

```
from numpy import array
from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()

glove_file = open('../kaggle/glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions
glove_file.close()

embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

Table 8 GloVe Word Embedding

5.1.4. Review Classifier

After embedding the reviews dataset, it was ready for the classification model. Multiple classification approaches are available which include CNN, RNN, and LSTM. To choose the classification model which performs best on the dataset pilot studies were performed (discussed in chapter 3). According to the results of the pilot studies, LSTM performed very well. The research approach proceeded with the LSTM model for the classification of gaming bugs.

Activity Flow of Classification Process:

Classification steps are comprised of four steps:

- User reviews collection
- Pre-Processing
- LSTM classification model
- Validation

The below figure shows the generic activity flow of the classification process.

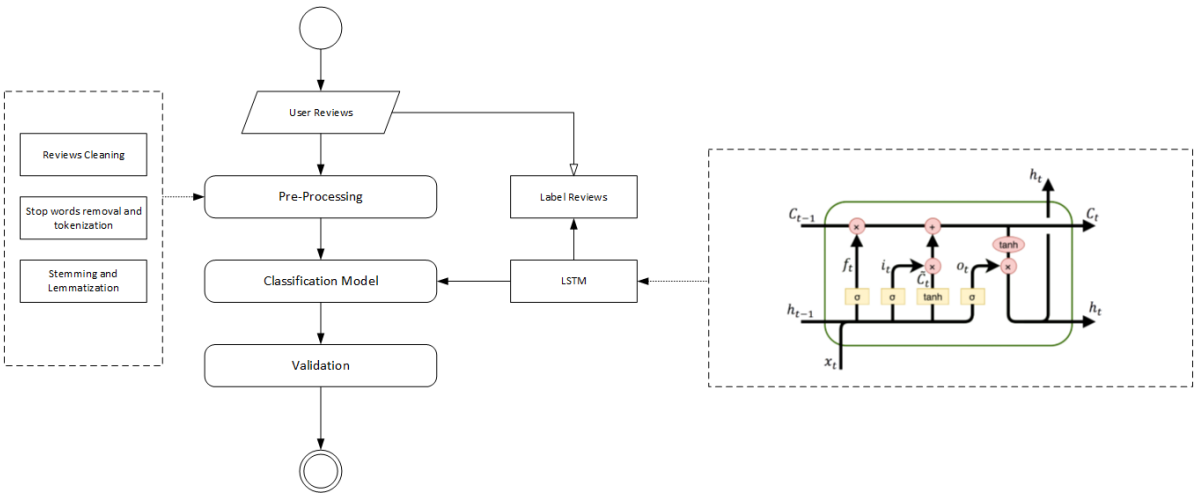


Figure 29 Flow of Classification Approach

5.2. MLBCGR-LSTM

5.2.1. MLBCGR-LSTM Design

The proposed LSTM model working is shown in figure 8. LSTM receives five arguments and has four dimensions.
Its three layers include

- Output-Dim (Output layer: number of nodes in output layer will be same as input layer)
- Hidden-Dim (Hidden layer: hidden layer size. It is the LSTM's hidden size. start's
- Input-Dim (Total number of the unique word in sample data)
- Embedding-dim (Size of each embedding vector. Here embedding dimension of GloVe word embedding vectors is 100)

Figure 18 shows the feature map implementation in the LSTM model:

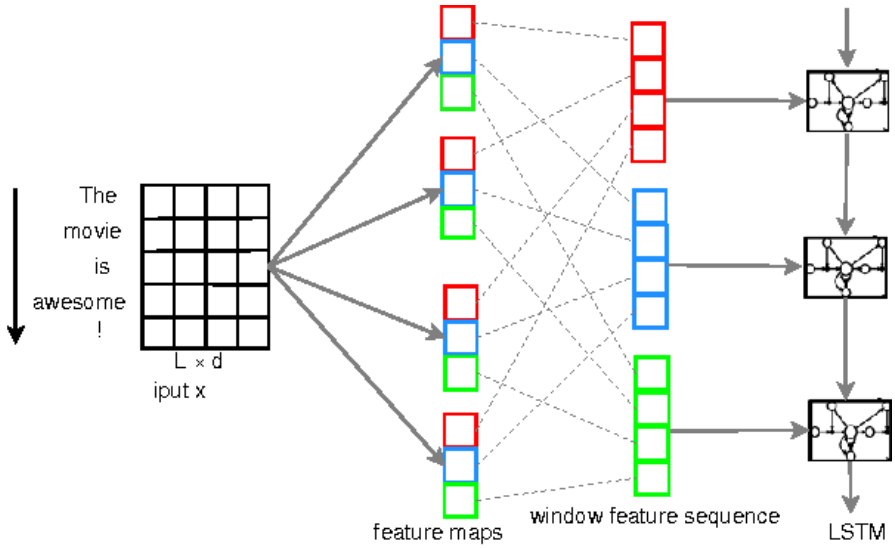


Figure 30 LSTM Model [86]

5.2.2. Dataset Distribution

Our data should be divided into three separate dataset splits for the training and testing of our model.

- **Training Set**

  The data set is what the model learns from to train and uncover any hidden characteristics or patterns.

  So that the model may be trained in all circumstances and anticipate any unobserved data sample that may arise in the future, the training set should comprise a diverse collection of inputs.

- **Validation Set**

  The validation set is a collection of data that isn't part of the training set and is used to verify.

  The major goal of dividing the dataset into a validation set is to avoid our model from overfitting, which occurs when the model is very effective at identifying samples in the training set but struggles to generalize to new data and make accurate classifications. the

performance of our model as it's being trained.

- **Test Set**

  After training is complete, the model is tested using a different collection of data called the test set.

So to get the best accurate results dataset was split into 70% training data, 20% testing data, and 10% validation data. The below figure shows the actual implementation of how a dataset is divided before MLBCGR-LSTM training:

```
def data_objects():
  # Declared a Field object
  # Field : A class that stores information about the way of preprocessing
  TEXT  = Field(sequential= True, tokenize = data_tokenization, lower = True, include_lengths = False, batch_first = False, init_token = '<sos>',
           eos_token = '<eos>')
  LABEL = data.LabelField(dtype = torch.float)
  training_data,validation_data,testing_data = TabularDataset.splits(path = '../Dataset/',
                          train       = 'Train_Data.csv',
                          validation  = 'Validation_Data.csv',
                          test        = 'Test_Data.csv',
                          format      = 'csv',
                          fields      = [('text', TEXT),('invalidPositionOverTime', LABEL),('implementationResponseIssue', LABEL),
                                         ('invalidContextOverTime', LABEL),('interruptedEvent', LABEL),('invalidEventOccurraceOverTime', LABEL),
                                         ('actionNotPossible', LABEL),('actionWhenNotAllowed', LABEL),('informationOutOfOrder', LABEL),
                          ('lackOfRequiredInformation', LABEL),('invalidInfoAccess', LABEL),('objectOutOfBoundForAnyState', LABEL),
                          ('objectOutOfBoundForSpecificState',LABEL),('artificialStupidity',LABEL), ('invalidValueChange',LABEL),
                          ('invalidGraphicalRespresentation',LABEL) ],
                          skip_header = True)
  print("\nPre-processed and Tokenized Training Data:")
  print("\n=======================================")
  for i in range(len(training_data)):
    print(training_data[i].text)
  print("\nPre-processed and Tokenized Validation Data:")
  print("\n=======================================")
  for i in range(len(validation_data)):
    print(validation_data[i].text)
  print("\nPre-processed and Tokenized Testing Data:")
  print("\n=======================================")
  for i in range(len(testing_data)):
      print(testing_data[i].text)
  return training_data, validation_data, testing_data, LABEL, TEXT
```

Figure 31 Dataset Distribution For Model Training

5.2.3. MLBCGR-LSTM Implementation

The model is implemented according to the layers which are discussed above in the design of the MLBCGR-LSTM. The classification model required an activation function. It is crucial to comprehend how a neural network picks up on complicated issues.

- **Activation Function**

SIGMOID is utilized as the foundation for identifying additional functions that result in effective and worthwhile solutions for supervised learning in deep learning architectures, as demonstrated in the accompanying image.

```
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(15, activation='sigmoid')(LSTM_Layer_1)
model = Model(inputs=deep_inputs, outputs=dense_layer_1)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

Figure 32 Activation Function Sigmoid

- **Model Architecture**

The implementation architecture is shown in the below figure.

```
class LSTM(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim, word_embeddings):

        super().__init__()

        self.hidden_dim = hidden_dim
        self.embedding_layer = nn.Embedding(input_dim, embedding_dim)          # Embedding layer shape
        # Assign pre-trained weights and train during model training
        # So the weight will be updated during backpropogation(if you don't want to train them during model training set requires_grad = False))
        self.embedding_layer.weight = nn.Parameter(word_embeddings, requires_grad=True)
        self.lstm_layer       = nn.LSTM(embedding_dim, hidden_dim, num_layers=1) # We can implement multiple layers of LSTM simply by changing num_layers value
        self.linear_layer     = nn.Linear(hidden_dim, output_dim)              # Shape of linear layer

    def forward(self, text):

        h_0, c_0 = self.init_hidden()   # Initialize first hidden state to all zeros

        # Here we will map all the indexes present in the input sequence to the corresponding
            # word vector using our trained word_embedddings.
          # embedded input of shape = (num_sequences, batch_size, embedding_dimension)
        embedded_vectors = self.embedding_layer(text)
        print (embedded_vectors)
        output_state, (hidden_state, cell_state) = self.lstm_layer(embedded_vectors, (h_0, c_0))  # Apply lstm layer and start learning sequence of words
        hidden_state = self.linear_layer(hidden_state.squeeze(0))      # Apply the linear layer on output
        return torch.sigmoid(hidden_state)
    def init_hidden(self):
        h_0 = torch.zeros(1, 1, self.hidden_dim)
        c_0 = torch.zeros(1, 1, self.hidden_dim)
        return h_0, c_0
```

Figure 33 LSTM Model Architecture

5.3. Evaluation

Accuracy is a simple estimation statistic that calculates the total proportion of correctly classified samples. TP (True Positive) is described as several correctly predicted classes of test cases that are true for both prediction and actual cases.
(FP) are predicted labels that are positive while the actual labels are negative.
Accuracy is calculated as a percentage of the total number of occurrences under all identical labels combined (TP + FP) divided by the number of instances that were properly classified under a specific label (TP). Test Hamming and Test F1 were applied to calculate the accuracy of the results.

5.4. Results of MLBCGR-LSTM

After dataset analysis, the preview text is cleaned as described above then Pre-Processing is applied using the best approaches that had been applied before in the literature. The dataset of user reviews is collected from fifteen games of different categories. Then ten thousand reviews are labeled manually by gaming experts.

A deep learning LSTM model is created and trained using the dataset of 15,000 reviews to categorize the reviews. This approach is defined in detail in chapter 4. Here results are discussed after implementing this model using python. Initially, the model is trained on 1500 reviews as described in the pilot study in chapter 3. Later model is trained over 10,000 labeled reviews in which 7200 reviews are used for training and 1800 reviews are for validation. Detailed results of each epoch are shown in below table 6.

Epochs	Accuracy	Val-Accuracy
1	0.8234	0.8219
2	0.8237	0.8222
3	0.8356	0.8333
Average	0.8263	0.8256

Table 9 LSTM Model Accuracy on 10,000 reviews

For unseen data prediction accuracy was equal to one for the trained model of LSTM.

5.5. Summary

The suggested remedy to the issue covered in Chapter 1 was summarized in this chapter. First, it defines the concepts of the proposed approach than the model design. Then the flow of the architectural approach is presented. A description of the evaluation method is also discussed in this chapter. Then the results of the proposed classification model are presented along with the accuracy table. Evaluation metrics and accuracy is used and results are reported by averaging them over epochs using the geometric mean average formula.

Chapter 6

The research study's results and a brief description are included in this chapter. The research contribution is explained in the second part. And the last section explains the limitation and future work of the research study.

6.1. Conclusion

This thesis aims at the automated classification of bugs in gaming reviews to facilitate developers about what they can improve in games. Existing literature has very serious limitations in terms of the multi-label classification of bugs in the gaming industry. Most of the existing literature is limited to the abstract level of bug classifications. In addition, an automated model for the classification of multi-label gaming reviews is rarely seen. But to meet the gamers' expectations and increase the satisfaction of gamers it is much needed to detect multiple bugs in games. Their solution can not only increase the quality of the games but also increase the number of users of the games.

State-of-the-art studies have given the taxonomies and studies for classifications of bugs which require an immediate need for a more comprehensive way to detect them from user reviews. In this research, the MLBCGR-LSTM algorithm was used to classify the bugs into specified labels.

This research suggests deep learning-based models including CNN, RNN, and LSTM as well as machine learning-based models for automatic multi-label bug categorization of game reviews. Python is used to implement these models, and the findings are detailed in the chapter on the pilot research. An automated multi-label bug classification model MLBCGR-LSTM is created using Python and fifteen different games with a 10,000 review dataset based on the accuracy of the findings of pilot trials. To classify the bugs into multi-label classification different objectives were defined and proved in the course of this research.

6.2. Research Contribution

This proposed multi-label classification model for gaming app reviews has been developed to support both game developers and researchers. The gaming app bug classification is a real-world problem that the gaming industry faces on daily basis. Extraction of bugs from the reviews helps developers and gaming companies to make improvements in games. It helps them to get information about the issues that gamers are facing while playing the games. The user expresses more than one issue in a single review which requires an automated model for multi-label classification of gaming bugs. This proposed research can be used for the classification of gaming bugs from reviews.

6.3. Limitations and Future Work

The proposed multi-label bug classification model focuses mainly on classifying the bugs into fifteen categories which are defined in existing literature [8]. Also, the number of epochs is set to the minimum which can also be increased to achieve better results. This implementation has used the GloVe word embedding technique however other embedding techniques like word2Vec can also be used. The correctness of the suggested model is assessed, but other aspects may also be taken into account to gauge its accuracy.

Bibliography

[1] G. E. Gorman, “Application of social media analytics: a case of analyzing online hotel reviews,” *Online Inf. Rev.*, vol. 38, no. 3, 2014, doi: 10.1108/oir-05-2014-002.

[2] D. Lin, C.-P. Bezemer, Y. Zou, and A. E. Hassan, “An empirical study of game reviews on the Steam platform,” *Empir. Softw. Eng.*, vol. 24, no. 1, pp. 170–207, 2019, DOI: 10.1007/s10664-018-9627-4.

[3] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, “An automated model-based testing approach for platform games,” in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 426–435, DOI: 10.1109/MODELS.2015.7338274.

[4] D. Pagano and W. Maalej, “User feedback in the AppStore: An empirical study,” in *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 125–134, DOI: 10.1109/RE.2013.6636712.

[5] D. Lin, C. P. Bezemer, and A. E. Hassan, “Identifying gameplay videos that exhibit bugs in computer games,” *Empir. Softw. Eng.*, vol. 24, no. 6, pp. 4006–4033, 2019, DOI: 10.1007/s10664-019-09733-6.

[6] C. C. Aggarwal and C. X. Zhai, *Mining text data*, 1st ed., vol. 9781461432. Springer New York, NY, 2013.

[7] A. Truelove, E. S. de Almeida, and I. Ahmed, “We’ll Fix it in Post: What Do Bug Fixes in Video Game Update Notes Tell Us?,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 258–259, DOI: 10.1109/ICSE-Companion52605.2021.00120.

[8] C. Lewis, J. Whitehead, and N. Wardrip-Fruin, “What went wrong: A taxonomy of video game bugs,” *FDG 2010 - Proc. 5th Int. Conf. Found. Digit. Games*, pp. 108–115, 2010, DOI: 10.1145/1822348.1822363.

[9] M. Westerdahl, V. Bahtijar, and D. Spikol, “Challenges in video game development - What does Agile management have to do with it?,” *Malmö Univ. och samhälle*, p. 79, 2019, doi: DiVA.org:mau-20026.

[10] J. Iqbal *et al.*, “Requirements engineering issues causing software development outsourcing failure,” *PLoS One*, vol. 15, no. 4, pp. 1–36, 2020, DOI: 10.1371/journal.pone.0229785.

[11] Institute of Electrical and Electronics Engineers, “IEEE Standard Glossary of Software Engineering Terminology,” *Office*, vol. 121990, no. 1, p. 1, 1990, DOI: 10.1109/IEEESTD.1990.101064.

[12] M. Klassen, S. Denman, and H. Driessen, “Requirements quality for a virtual world,” *Proc. - 15th IEEE Int. Require. Eng. Conf. RE 2007*, pp. 375–376, 2007, DOI: 10.1109/RE.2007.53.

[13] A. Hussain, O. Asadi, and D. J. Richardson, “A holistic look at requirements engineering practices in the gaming industry,” *arXiv Prepr. - arxiv.org*, no. Section 4, p. 12, 2018, DOI: 10.48550/arXiv.1811.03482.

[14] J. P. Zagal, A. Ladd, and T. Johnson, “Characterizing and understanding game reviews,” *FDG 2009 - 4th Int. Conf. Found. Digit. Games, Proc.*, no. McCrea 2007, pp. 215–222, 2009, DOI: 10.1145/1536513.1536553.

[15] W. Maalej, M. Nayebi, and G. Ruhe, “Data-Driven Requirements Engineering-An Update,” *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Pract. ICSE-SEIP 2019*, pp. 289–290, 2019, DOI: 10.1109/ICSE-SEIP.2019.00041.

[16] L. F. S. Britto and L. D. S. Pacífico, “Evaluating Video Game Acceptance in Game Reviews using Sentiment Analysis Techniques,” *Proc. SBGames, 2020*, pp. 399–402, 2020, doi: 10.1145/3342220.3344924.

[17] H.-N. Kang, “A Study of Analyzing Online Game Reviews Using a Data Mining Approach: STEAM Community Data,” *Int. J. Innov. Manag. Technol.*, vol. 8, no. 2, pp. 90–94, 2017, DOI: 10.18178/ijimt.2017.8.2.709.

[18] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, “Augmenting Automated Game Testing with Deep Reinforcement Learning,” in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 600–603, DOI: 10.1109/CoG47356.2020.9231552.

[19] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora, “SURF: Summarizer of user reviews feedback,” *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017*, no. I, pp. 55–58, 2017, DOI: 10.1109/ICSE-C.2017.5.

[20] A. Di Sorbo *et al.*, “What would users change in my App? Summarizing app reviews for recommending software changes,” *Proc. ACM SIGSOFT Symp. Found. Softw. Eng.*, vol. 13-18-Nove, pp. 499–510, 2016, DOI: 10.1145/2950290.2950299.

[21] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, “AR-miner: Mining informative reviews for developers from a mobile app marketplace,” *Proc. - Int. Conf. Softw. Eng.*, no. 1, pp. 767–778, 2014, DOI: 10.1145/2568225.2568263.

[22] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, “How can I improve my app? Classifying user reviews for software maintenance and evolution,” *2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc.*, no. 2, pp. 281–290, 2015, DOI: 10.1109/ICSM.2015.7332474.

[23] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.

[24] Klaus Pohl and Chris Rupp, “Requirements Engineering: Fundamentals, Principles, and Techniques,” in *IREB International Requirements Engineering Board*, ACM, 20189, p. https://news.ge/anakliis-porti-aris-qveynis-momava.

[25] “Requirement engineering process,” 2016. https://www.researchgate.net/figure/Requirement-Engineering-Process_fig1_310465092.

[26] A. M. Hickey and A. M. Davis, “Elicitation technique selection: How do experts do it?,” *Proc. IEEE Int. Conf. Requir. Eng.*, vol. 2003-Janua, pp. 169–178, 2003, doi: 10.1109/ICRE.2003.1232748.

[27] “Requirement elicitation techniques,” 2016. https://www.softwaretestinghelp.com/requirements-elicitation-techniques/.

[28] S. Lim, A. Henriksson, and J. Zdravkovic, “Data-Driven Requirements Elicitation: A Systematic Literature Review,” in *SN Computer Science*, 2021, vol. 2, no. 1, DOI: 10.1007/s42979-020-00416-4.

[29] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall, “Analyzing reviews and code of mobile apps for better release planning,” *SANER 2017 - 24th IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, no. February, pp. 91–102, 2017, DOI: 10.1109/SANER.2017.7884612.

[30] R. Deocadez, R. Harrison, and D. Rodriguez, “Preliminary study on applying Semi-Supervised Learning to app store analysis,” *ACM Int. Conf. Proceeding Ser.*, vol. Part F1286, pp. 320–323, 2017, DOI: 10.1145/3084226.3084285.

[31] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, “Release planning of mobile apps based on user reviews,” *Proc. - Int. Conf. Softw. Eng.*, vol. 14-22-May-, pp. 14–24, 2016, DOI: 10.1145/2884781.2884818.

[32] E. Guzman and W. Maalej, “How do users like this feature? Fine-grained sentiment analysis of App reviews,” *2014 IEEE 22nd Int. Required. Eng. Conf. RE 2014 - Proc.*, pp. 153–162, 2014, DOI: 10.1109/RE.2014.6912257.

[33] Z. Kurtanovic and W. Maalej, “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning,” *Proc. - 2017 IEEE 25th Int. Required. Eng. Conf. RE 2017*, pp. 490–495, 2017, DOI: 10.1109/RE.2017.82.

[34] H. X. Shi and X. J. Li, “A sentiment analysis model for hotel reviews based on supervised learning,” *Proc. - Int. Conf. Mach. Learn. Cybern.*, vol. 3, pp. 950–954, 2011, DOI: 10.1109/ICMLC.2011.6016866.

[35] “Game review example,” 2016. https://www.gamedeveloper.com/disciplines/a-call-for-responsible-steam-reviews-in-2016.

[36] "Machine learning approaches diagram," 2020. https://www.researchgate.net/figure/Machine-learning-approaches-and-selected-example-algorithms_fig2_344411516.

[37] E. Oja, "Unsupervised learning in neural computation," *Theor. Comput. Sci.*, vol. 287, no. 1, pp. 187–207, 2002, doi: 10.1016/S0304-3975(02)00160-3.

[38] M. Ahmad, S. Aftab, S. S. Muhammad, and S. Ahmad, "Machine Learning Techniques for Sentiment Analysis: A Review," *Int. J. Multidiscip. Sci. Eng.*, vol. 8, no. 3, pp. 27–32, 2017, DOI: 10.18090/samriddhi.v12i02.03.

[39] L. M. Chiappe, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews.," *Proc. 40th Annu. Meet. Assoc. Comput. Linguist. (2002), Philadelphia, Pennsylvania*, vol. 12, no. 3, pp. 417–424, 2010, DOI: 10.48550/arXiv.cs/0212032.

[40] T. Zhang, M. Huang, and L. Zhao, "Learning structured representation for text classification via reinforcement learning," *32nd AAAI Conf. Artif. Intell. AAAI 2018*, vol. 1, pp. 6053–6060, 2018, doi: 10.1609/aaai.v32i1.12047.

[41] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Required. Eng.*, vol. 21, no. 3, pp. 311–331, 2016, DOI: 10.1007/s00766-016-0251-9.

[42] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications, and Research Directions," *SN Comput. Sci.*, vol. 2, no. 3, 2021, doi: 10.1007/s42979-021-00592-x.

[43] C. N. Dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," *COLING 2014 - 25th Int. Conf. Comput. Linguist. Proc. COLING 2014 Tech. Pap.*, pp. 69–78, 2014.

[44] X. Zhang and Y. LeCun, "Text Understanding from Scratch," *arXiv Prepr. - arxiv.org*, pp. 1–9, 2015, DOI: 10.48550/arXiv.1502.01710.

[45] Y. Kim, "Convolutional Neural Network for Sentence Classification," *EMNLP 2014 -*

2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf., pp. 1746–1751, 2014, doi: 10.3115/v1/d14-1181.

[46] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *NAACL HLT 2015 - 2015 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Proc. Conf.*, pp. 103–112, 2015, DOI: 10.3115/v1/n15-1011.

[47] A. Jacobi, O. S. Shalom, and Y. Goldberg, "Understanding Convolutional Neural Networks for Text Classification," *EMNLP 2018 - 2018 EMNLP Work. BlackboxNLP Anal. Interpret. Neural Networks NLP, Proc. 1st Work.*, pp. 56–65, 2018, DOI: 10.18653/v1/w18-5408.

[48] B. Jang, M. Kim, G. Harerimana, S. U. Kang, and J. W. Kim, "Bi-LSTM model to increase accuracy in text classification: Combining word2vec CNN and attention mechanism," *Appl. Sci.*, vol. 10, no. 17, 2020, doi: 10.3390/app10175841.

[49] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep Learning Based Text Classification: A Comprehensive Review," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–43, 2020, DOI: 10.1145/3439726.

[50] P. Liu, X. Qiu, and H. Xuanjing, "Recurrent neural network for text classification with multi-task learning," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2016-Janua, pp. 2873–2879, 2016, DOI: 10.48550/arXiv.1605.05101.

[51] H. Qin and X. Sun, "Classifying bug reports into bugs and non-bugs using LSTM," *ACM Int. Conf. Proceeding Ser.*, pp. 16–19, 2018, DOI: 10.1145/3275219.3275239.

[52] J. H. Wang, T. W. Liu, X. Luo, and L. Wang, "An LSTM approach to short text sentiment classification with word embeddings," *Proc. 30th Conf. Comput. Linguist. Speech Process. ROLLING 2018*, pp. 214–223, 2018, [Online]. Available: https://aclanthology.org/O18-1021/.

[53] "LSTM Layers," 2020. https://www.iarai.ac.at/publication/mc-lstm-mass-conserving-lstm/.

[54] P. M. Vu, H. V Pham, T. T. Nguyen, and T. T. Nguyen, “Phrase-based extraction of user opinions in mobile app reviews,” in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 726–731, DOI: 10.1145/2970276.2970365.

[55] M. Nayrolles and A. Hamou-Lhadj, “Towards a classification of bugs to facilitate software maintainability tasks,” *Proc. - Int. Conf. Softw. Eng.*, pp. 25–32, 2018, DOI: 10.1145/3194095.3194101.

[56] W. Maalej and H. Nabil, “Bug report, feature request, or simply praise? On automatically classifying app reviews,” *2015 IEEE 23rd Int. Require. Eng. Conf. RE 2015 - Proc.*, pp. 116–125, 2015, DOI: 10.1109/RE.2015.7320414.

[57] A. Borrelli, V. Nardone, G. A. Di Lucca, G. Canfora, and M. Di Penta, “Detecting Video Game-Specific Bad Smells in Unity Projects,” *Proc. - 2020 IEEE/ACM 17th Int. Conf. Min. Softw. Repos. MSR 2020*, pp. 198–208, 2020, DOI: 10.1145/3379597.3387454.

[58] S. M. Liu and J. H. Chen, “A multi-label classification based approach for sentiment classification,” *Expert Syst. Appl.*, vol. 42, no. 3, pp. 1083–1093, 2015, DOI: 10.1016/j.eswa.2014.08.036.

[59] J. L. Elman, “Finding structure in time,” *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, 1990, doi: 10.1016/0364-0213(90)90002-E.

[60] “RNN Network.” https://gotensor.com/2019/02/28/recurrent-neural-networks-remembering-whats-important/.

[61] R. Collobert, J. Weston, J. Com, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural Language Processing (Almost) from Scratch,” *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011, DOI: 10.48550/arXiv.1103.0398.

[62] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *1st Int. Conf. Learn. Represent. ICLR 2013 - Work. Track Proc.*, pp. 1–12, 2013, DOI: 10.48550/arXiv.1301.3781 Focus to learn more.

[63] “Ready or Not,” 2022. http://steamcommunity.com/app/1144200/reviews/?browsefilter=mostrecent&p=1.

[64] “Project Zombied,” 2022. http://steamcommunity.com/app/108600/reviews/?browsefilter=mostrecent&p=1.

[65] “Three Finger Battle Arena,” 2022.

[66] “Baldur’s Gate 3,” 2022. http://steamcommunity.com/app/1893860/reviews/?browsefilter=mostrecent&p=1.

[67] “Trip In Another World,” 2022. http://steamcommunity.com/app/1826960/reviews/?browsefilter=mostrecent&p=1.

[68] “NEBULOUS: Fleet Command,” 2022. http://steamcommunity.com/app/887570/reviews/?browsefilter=mostrecent&p=1.

[69] “ERRANTE,” 2022. http://steamcommunity.com/app/1871830/reviews/?browsefilter=mostrecent&p=1.

[70] “HITMAN 3,” 2022. http://steamcommunity.com/app/1659040/reviews/?browsefilter=mostrecent&p=1.

[71] “Grand Theft Auto V,” 2022. http://steamcommunity.com/app/271590/reviews/?browsefilter=mostrecent&p=1%0A.

[72] “Tomb Raider,” 2022. http://steamcommunity.com/app/203160/reviews/?browsefilter=mostrecent&p=1.

[73] “Red Dead Redemption 2,” 2022. http://steamcommunity.com/app/1174180/reviews/?browsefilter=mostrecent&p=1.

[74] “Max Payne 3,” 2022. http://steamcommunity.com/app/204100/reviews/?browsefilter=mostrecent&p=1.

[75] “Resident Evil 6,” 2022.

http://steamcommunity.com/app/221040/reviews/?browsefilter=mostrecent&p=1.

[76] “Just Cause 3,” 2022. http://steamcommunity.com/app/225540/reviews/?browsefilter=mostrecent&p=1.

[77] “Star Wars: Battlefront 2 (Classic, 2005),” 2022. http://steamcommunity.com/app/6060/reviews/?browsefilter=mostrecent&p=1.

[78] “Multi-Label And Multi-Class Classification,” 2021. https://medium.com/analytics-vidhya/multi-label-classification-a9643d221954.

[79] “Bag of Words,” 2017. https://www.datacamp.com/tutorial/lda2vec-topic-model.

[80] A. Liaw and M. Wiener, “Classification and Regression by randomForest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002, DOI: 10.1021/ci034160g.

[81] F. J. Smith and L. C. Emerson, “Not All Samples Are Created Equal: Deep Learning with Importance Sampling,” pp. 12–18, 1989, DOI: 10.1109/dksme.1989.107436.

[82] “invalid position over time,” 2016. https://www.reddit.com/r/NoMansSkyTheGame/comments/cz8200/cannot_build_invalid_position_with_a_lot_of/.

[83] “Artificial Stupidity.” https://sudonull.com/post/68442-Artificial-Stupidity-The-Art-of-Intentional-Mistakes.

[84] “Invalid value change,” 2020. https://tencomputer.com/opengl-error-1281-invalid-value/.

[85] E. Hindocha, V. Yazhiny, A. Arunkumar, and P. Boobalan, “Short-text Semantic Similarity using GloVe word embedding,” *Int. Res. J. Eng. Technol.*, vol. 6, no. 4, pp. 553–558, 2019, DOI: 10.1145/2806416.2806475.

[86] “LSTM model,” 2016. https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714?gi=be2a141c1096.