SARCASM DETECTION IN POLITICAL SPEECHES USING RECURRENT NEURAL NETWORKS

BY

MULAUDZI THIKHO

DISSERTATION

Submitted in fulfilment of the requirements of the degree of

MASTERS OF SCIENCE

In

COMPUTER SCOENCE

In the

FACULTY OF SCIENCE AND AGRICULTURE

(School of Mathematical and Computer Science)

At the

UNIVERSITY OF LIMPOPO

SUPERVISOR:

PROF SN MOKWENA

2023

DEDICATION

I would like to dedicate this research to my family, my mother Miss Mulaudzi M.E she has been my main support throughout my life, she is always available when I need help at all times of the day, my father Mr. TP Mulaudzi. My sisters Phathutshedzo, Thendo, and Phindulo who are my inspiration and have set a good and great example for me to follow.

DECLARATION

I Mulaudzi Thikho declare that sarcasm detection in political speeches using recurrent neural networks is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references, and that this work has not been submitted before for any other degree at any other institution.

Signature: mulaudziT Date: 04/12/2023

ACKNOWLEDGEMENT

I start by extending my profound thanks to God, the Almighty, acknowledging His steadfast guidance, limitless love, and countless blessings that have accompanied me on my life's path. His divine presence has consistently provided strength and inspiration, empowering me to confront challenges and celebrate the victories that have moulded my journey.

- I wish to convey my heartfelt appreciation to my supervisor, Professor SN Mokwena, for providing consistent guidance and support during the entire research process. His expertise and encouragement played a pivotal role in shaping the trajectory of this project.
- My sincere thanks go to ETDP SITA for their generous financial backing of this research study.
- Lastly, I am profoundly thankful to my family and friends for their steadfast support and encouragement throughout my academic journey. Their unwavering love and belief in me have served as a continual wellspring of motivation.

Abstract

Sarcasm detection is a challenging task in natural language processing (NLP) that has received significant attention in recent years. Sarcasm is a form of indirect speech in which the speaker says the opposite of what they mean. It can be used to express a variety of emotions, such as humour, irony, or contempt. Sarcasm is often difficult to detect, especially in written text, because it often relies on context and the speaker's intent. Recurrent neural networks (RNNs) have been shown to be effective in sarcasm detection, but there is still room for improvement. In this work, we propose a stacking and weighted average ensemble model using simpleRNN, LSTM, and GRU as base models for sarcasm detection. The news headline dataset was used in the study. The dataset contains sarcastic and non-sarcastic labels for the headlines, and contains a total of 55325 headlines, the dataset is split into 80% (44260) testing and 20% (11065) validation. The aim of this study was to develop a model to detect sarcasm in political speech using Recurrent Neural Networks, incorporating sarcasm detection into sentiment analysis for political text can significantly enhance the accuracy and depth of sentiment understanding. The results suggest that the ensemble models outperform individual neural network models, with the two-level stacking model achieving the best overall performance.

Key-words: Sarcasm detection, sentiment analysis, Deep learning, SimpleRNN, LSTM, GRU, ensemble, stacking, weighted average.

Table of contents.

Contents

List of Figures

List of Tables

List of abbreviations

Abbreviations	Full word
Acc	Accuracy
ANC	African National Congress
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long-Short-Term Memory
CNN	Convolutional Neural Network
DL	Deep Learning
DT	Decision Tree
FL	Figurative Language
FN	False Negative
FP	False Positive
FS	Feature Selection
GLOVE	Global Vector
GRU	Gated Recurrent Unit
IG	Information Gain
LSTM	Long-Short-Term Memory
MI	Mutual Information
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language Toolkits
NP	NumPy
OS	Operating System

PAC	Pan African Congress
Pd	Pandas
PLRM	Pretrained Language Representation Models
re	Regular Expression
RF	Random Forest
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized Bidirectional Encoder Representations from Transformers approach
simpleRNN	Simple Recurrent Neural Network
SVM	Support Vector Machine
TF-IDF	Term Frequency-inverse Document Frequency
TN	True Negative
TP	True Positive
TRC	Truth and Reconciliation Commission

1 Chapter 1: Introduction

1.1 Introduction

Sarcasm, is a form of irony, entails the use of words to convey a meaning opposite to their intended sense. Detecting sarcasm, especially in written form, proves challenging as it hinges on the reader's comprehension of context and the speaker's intent. Recurrent Neural Networks (RNNs), a category of artificial neural networks, prove highly adept at sarcasm detection owing to their capacity to grasp long-range dependencies in data. This attribute is crucial for understanding sentence context and effectively identifying instances of sarcasm.

Detecting sarcasm in text, improves the understanding of sentiment and sentiment analysis models. [1] Sarcasm is a significant challenge for sentiment analysis, as it can lead to misinterpretations of the intended sentiment of a piece of text [2]. This is because sarcasm often involves using language that expresses the opposite of the intended meaning, and this can be difficult for machines to detect without a deep understanding of the context and nuances of human language.

Sentiment analysis, alternatively referred to as opinion mining, involves the identification, extraction, quantification, and examination of emotional states and subjective content within text. This field, a subset of natural language processing (NLP), seeks to comprehend the prevailing sentiment expressed in text, discerning whether it is positive, negative, or neutral. The applications of sentiment analysis span diverse areas such as monitoring social media, analysing customer feedback, and managing online reputation[3].

Politicians frequently employ sarcasm as a means of conveying their messages, posing a difficulty for sentiment analysis tools in accurately comprehending them. Sarcasm hinges on the deployment of irony and mockery to express the opposite of the literal meaning of words, creating a challenge for sentiment analysis tools to discern. [4] In response to this issue, researchers are actively devising novel techniques for detecting sarcasm, aiming to integrate them with sentiment analysis. These methods are designed to pinpoint and classify sarcastic statements, thereby enhancing the precision of sentiment analysis in the realm of political discourse. Although sarcasm serves as a potent tool in political communication, its complexity poses challenges for comprehension. The incorporation of sarcasm-detection capabilities into sentiment analysis tools holds the potential to deepen our understanding of political discourse and refine the public's perception of political figures.

Sarcasm detection is a challenging task, as it requires understanding the context and nuances of human language. However, recent advances in natural language processing (NLP) have made it possible to develop models that can accurately detect sarcasm with a high degree of accuracy.

Recurrent neural network structures that demonstrate efficacy in detecting sarcasm include SimpleRNN, LSTM, and GRU. These designs excel in capturing extensive textual dependencies, a crucial aspect for comprehending the nuances of sarcasm [5][6][7], However, relying solely on SimpleRNN, LSTM, and GRU may not maximise accuracy in sarcasm detection. Employing ensemble learning, a technique that amalgamates predictions from various models, proves instrumental in substantially enhancing the overall performance of these models.

Ensemble methods enhance the overall accuracy of predictions by combining the outputs of multiple machine learning models. Two widely used ensemble techniques are stacking and weighted average. Stacking achieves this by training a new model to discern relationships between the predictions of various base models. On the other hand, weighted average combines predictions by calculating a weighted mean, with weights determined from the performance of base models on a separate validation set. This research uses a sarcasm detection model that integrates RNNs with ensemble methods.

The subsequent sections are structured as follows: Section 2 delves into relevant studies on sarcasm detection. In Section 3, our proposed model is intricately detailed. The experimental results are outlined in Section 4. Finally, Section 5 provides the conclusion to this work.

1.2 PROBLEM STATEMENT.

Political discourse thrives on wit, but deciphering genuine humor from veiled criticism, often delivered through sarcasm, can be challenging. Accurately detecting sarcasm is crucial for understanding the true intent behind political statements[8], shaping public opinion, and mitigating the spread of misinformation. However, current methods utilizing sentiment analysis often misinterpret sarcastic language, leading to skewed interpretations and hampered communication.

The ability to automatically detect sarcasm in political speech holds immense significance[9]. It can enhance understanding, combat misinformation, and shape public opinion. While sentiment analysis tools exist, they often struggle with nuanced language like sarcasm, leading to misinterpretations[10][3]. The challenge lies in capturing context, domain specificity and data limitations

To address these challenges, this study aims to develop a robust model for detecting sarcasm in political speech using Recurrent Neural Networks (RNNs). The research will focus on:

- Comparing RNN architectures: Implementing and evaluating the performance of SimpleRNN, LSTM, and GRU models to identify the most effective architecture for sarcasm detection.
- Ensemble learning: Leveraging ensemble learning techniques like stacking to combine predictions from multiple deep learning frameworks, potentially improving overall accuracy.
- Weighted average ensemble: Investigating the effectiveness of combining predictions from individual RNN models through a weighted average ensemble, potentially capturing the strengths of each approach.

By overcoming the challenges of sarcasm detection in political speech, this research has the potential to significantly improve our understanding of political discourse, combat misinformation, and shape public opinion in a more informed and nuanced manner.

1.3 MOTIVATION.

Sarcasm is a type of humour or insult that uses insincere remarks [1]. Politicians often use sarcasm to mock the opposition party. Sarcasm is difficult to detect in written communication because there are no-nonverbal signals such as body language or tone of voice [11].

The issue of sarcasm has gained tremendous attention from Linguistics [12], Building computer models for the automatic detection of sarcasm, on the other hand, is still in its early stages. In earlier attempts to identify sarcasm in texts, pragmatic and lexical signs like interjections, punctuation, and emotional changes were considered because they are important indicators of sarcasm [13].

Previous attempts to detect sarcasm in text have used pragmatic and lexical features like interjections, punctuation, and emotional shifts. More recent works has used machine learning approaches as Naive Bayes, Support Vector Machines, and Decision Trees (DT), achieving accuracies of 51%, 64%, and 59%, respectively.

This study proposes a deep learning approach to sarcasm detection using Recurrent Neural Networks (RNNs). RNNs are a type of neural network that can process sequences of inputs, making them well-suited for sarcasm detection, which is often context-dependent.

RNNs have two main advantages: they can remember information over time and they can be used in conjunction with convolution layers to generate an effective pixel neighbourhood. However, RNNs can also suffer from gradient vanishing and explosion problems. The Long Short-Term Memory (LSTM) model solves these problems.

Existing attempts include, rule-based systems, lexicon base approaches, supervised machine leaning (SVM).

Rule based systems: [14] These rely on predefined rules like presence of exclamation marks or negation, often lacking flexibility and context awareness.

Lexicon-based approaches: [15],[16] Uses dictionaries of sarcastic phrases, but struggle with novel expressions and domain specificity.

Supervised machine learning: [17] Train models on labelled data, but require large, annotated datasets and struggle with generalization.

1.3.1 AIM.

The aim of this study is to develop a model to detect sarcasm in political speech using Recurrent Neural Networks.

1.3.2 OBJECTIVES.

The objectives of the study are the following

I. Implement and compare the performance of Simple Recurrent Neural Network (SRNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) for sarcasm detection in news headlines.
II. Implement ensemble learning techniques, including one-level, two-level, and three-level stacking models, to combine predictions from multiple deep learning frameworks.
III. Investigate the effectiveness of combining predictions from SimpleRNN, LSTM, and GRU through a weighted average ensemble model.
IV. Evaluate model performance using standard metrics such as accuracy, precision, recall, and F1 score.

1.3.3 RESEARCH QUESTIONS.

I. How does the performance of SimpleRNN, LSTM, and GRU compare for sarcasm detection in news headlines?
II. How effective are one-level, two-level, and three-level stacking models in combining predictions from multiple deep learning frameworks for sarcasm detection?

III. Is a weighted average ensemble model effective in combining predictions from SimpleRNN, LSTM, and GRU for improved sarcasm detection?

IV. How do the different models compare in terms of accuracy, precision, recall, and F1 score?

1.4 METHODOLOGY.

The sarcasm detection methodology begins with data collection. The news headline dataset for this study was collected from Kaggle [18]. Once the data was collected, it was pre-processed to make it suitable for machine learning[8]. The pre-processing involved removing stop words, numbers, non-English words, symbols, punctuation, and emojis, then tokenising the text, and stemming the words[11],[19],[20]. Next, feature extraction is performed to convert the text data into a numerical representation that will be used by the deep learning model. This is achieved by using Glove word embedding[1][18]. In the concluding phase, a model for sarcasm classification is developed to ascertain whether a given text exhibits sarcasm. Diverse machine learning algorithms, including simpleRNN, LSTMs, and GRUs [20],[21]. are employed for this purpose. To enhance the performance of the sarcasm classification model, an ensemble approach is implemented, amalgamating the predictions from various models.

1.5 SCIENTIFIC CONTRIBUTION.

The development of sarcasm detection models for political speech provides new insights into politicians' communication strategies, the construction of political narratives, and the tactics used to persuade and mobilise voters. These models also enable the quantitative analysis of public sentiment in response to political statements, contributing to a data-driven understanding of how political discourse influences public opinion. Sarcasm detection research in politics validates rhetorical theories regarding the persuasive power of language and has implications for political journalism and media analysis by aiding in the accurate portrayal of politicians' remarks. Furthermore, this research drives the development of advanced natural language processing (NLP) techniques, which have broader implications for sentiment analysis, opinion mining, and discourse analysis.

1.6 AVAILABILITY OF RESOURCES.

The University of Limpopo's Department of Computer Sciences provides the necessary resources and infrastructure, including software to undertake this research.

1.7 ETHICAL CONSIDERATION.

Since the analysis would be based on publicly accessible data, no ethical approval is needed. However, all the work that will be used will be properly referenced.

2 CHAPTER 2: Literature review

2.1 INTRODUCTION

Identifying sarcasm is a significant issue in natural language processing (NLP) because it constitutes a type of figurative language where words convey a meaning contrary to their literal interpretation. Despite human proficiency in discerning sarcasm, machines face difficulties in accurately detecting it within natural language text. The applications of sarcasm detection span sentiment analysis, social media analysis, and various domains, driving ongoing research in NLP to address this challenge.

Recent progress in deep learning has resulted in notable enhancements in sarcasm detection. A prevalent deep learning strategy for identifying sarcasm involves leveraging word embedding. In a notable contribution, [22] introduced a sarcasm detection model for tweets that utilises word embedding trained on an extensive corpus of unlabelled data. The model demonstrated cutting-edge performance in this domain.

Another deep learning approach that has been widely used for sarcasm detection is the use of recurrent neural networks (RNNs). [23] proposed a bidirectional LSTM with an attention mechanism for sarcasm detection in tweets. The model outperformed several baselines on the SemEval-2017 Task 4A dataset.

Convolutional neural networks (CNNs) have also been used for sarcasm detection. [24] proposed a CNN-based model that uses word embedding and character-level embedding for sarcasm detection in tweets.

In this review of the literature, we provide an overview of recent advances in sarcasm detection.

2.2 BACKGROUND

Sarcasm constitutes a form of figurative language where words or expressions are employed to communicate a meaning distinct from, and at times opposite to, their literal interpretation. While humans can readily grasp sarcasm within a given context, it poses a challenge for natural language processing (NLP) systems. The intricate and frequently ambiguous nature of sarcasm makes its detection challenging for these systems.

Sarcasm detection has become an active area of research in NLP, with various approaches proposed over the years. Traditional approaches to sarcasm detection relied on hand-crafted features and rule-based classifiers, but these methods were limited in their ability to capture the nuances of sarcasm. In recent years, deep learning approaches have shown promise in detecting sarcasm, leveraging the power of neural networks to learn representations of language that can capture the complex patterns and contextual cues that signal sarcasm.

Several studies have explored different deep learning architectures for sarcasm detection, including RNNs and CNNs [22], and, more recently, transformer-based models [25]. Other studies have focused on developing novel features and representations for sarcasm, such as sentiment embedding (e.g.,[16]) and irony-specific lexicons [15].

In general, sarcasm detection remains an important challenge in NLP, with potential applications in sentiment analysis, and social media analysis. In the following sections, we will review some of the key studies on sarcasm detection using deep learning and discuss their findings and contributions to the field.

2.3 RELATED WORK

2.3.1 UNEXPECTEDNESS AND CONTRADICTORY FACTOR APPROACH

This approach is based on the idea that sarcasm often involves saying something that is unexpected or contradictory. For example, if someone says "That's a great idea" in a sarcastic tone, they are actually saying that they think the idea is bad. This approach uses rules and features to identify these unexpected or contradictory elements in text.

Researchers [15],[26] have explored the detection of irony and sarcasm in social media by analysing the presence of unexpectedness and contradiction in expressions in figurative language. According to [27], unexpectedness is an essential aspect of irony and humour that refers to imbalances or contradictions in a given context.

To measure contextual imbalance, the researcher [26] evaluated the similarity and semantic relatedness of concepts. For example,[28] used the American National Corpus Frequency Data source to measure unexpectedness and applied random forests (RF) and decision trees (DT) classifiers to distinguish sarcastic tweets from tweets related to politics, education, and humour. Similarly [29] viewed unexpectedness as an emotional imbalance between words in the text, and [17] used support vector machines (SVM) to identify sarcasm by identifying contradictions in tweets as features.

While unexpectedness and contradictory factors are valuable elements of sarcasm, relying solely on them has limitations. Some key shortcomings are:

- Context dependency: Sarcasm often hinges on shared context and understanding between speaker and listener. This approach might struggle with references, inside jokes, or cultural nuances that aren't explicitly stated.
- Overlooking non-contradictory sarcasm: Sarcasm can exist without blatant contradictions. For example, deadpan delivery or subtle exaggerations might not be flagged by this approach.
- Literal contradictions not always sarcastic: Sentences with literal contradictions might not be intended as sarcasm, leading to false positives.

Despite the limitations mentioned previously, the unexpectedness and contradictory factor approach to sarcasm detection also holds some key strengths:

- Simplicity and interpretability: This approach is relatively straightforward to understand and implement, making it accessible to researchers and developers without extensive expertise in natural language processing.
- Focus on core elements of sarcasm: Unexpectedness and contradiction are indeed fundamental aspects of many forms of sarcasm, making this approach relevant to a significant portion of sarcastic expressions.
- Potential for efficiency: By focusing on specific linguistic features, this approach can potentially be computationally efficient, making it suitable for real-time applications or resource-constrained environments.

2.3.2 CONTENT-BASED APPROACH

This approach uses sentiment analysis and word embeddings to identify sarcasm. Sentiment analysis is a technique that can be used to determine the emotional tone of a text. Word embeddings are a way of representing words as vectors in a high-dimensional space. This approach can be effective for identifying sarcasm that is expressed through sentiment or word choice.

In the field of computer science, researchers have explored diverse approaches to detect sarcasm in both speech and text, a study by [30] delved into prosodic, spectral, and contextual clues, while another [14] investigated oral and gestural cues, including emoticons, onomatopoeic expressions, and punctuation marks, for identifying sarcasm in user-generated content. Additional efforts included [31] utilizing linguistic patterns such as sarcastic hashtags, and [32] employing support vector machines and logistic regression with sarcastic hashtags and emoticons. Another study by [33] discovered that sarcasm is often associated with positive sentiment words and negative situations. Furthermore [34] explored a transfer learning framework leveraging sentiment classification and emotion detection as intermediate tasks to detect sarcasm.

Recent research has underscored the significance of contextual signals alongside lexical and syntactic information for sarcasm detection[35]. Studies such as [13],[36],[37] utilized various features, including lexical, pragmatic, implicit, and explicit context incongruity, to identify sarcasm. In explicit scenarios, these studies integrated elements into the text to identify failed sentimental expectations.

Another approach suggested by [38] introduced content-based feature selection (FS) technique for categorizing sarcastic text. This method involves a two-stage FS strategy to identify highly distinctive characteristics. Classic FS approaches such as mutual information (MI), chi-square, and information gain (IG) are used in the initial selection of acceptable feature subsets. These subsets are then refined in the subsequent phase, employing k-means clustering methods to choose highly distinctive characteristics from a set of matching features. Support vector machine (SVM) and random forest (RF) classifiers are subsequently employed to categorize the identified characteristics.

In summary, a variety of methods have been employed for sarcasm and irony detection in text and speech, encompassing prosodic, spectral, and contextual clues, investigation of oral or gestural signals, use of linguistic patterns, and consideration of lexical, pragmatic, implicit, and explicit context incongruity. Recent emphasis has been on models that incorporate contextual signals alongside inherent lexical and syntactic information in the text. Machine learning techniques, particularly support vector machines (SVM) and random forest (RF) classifiers, have been widely utilized in sarcasm and irony detection. Feature selection (FS) techniques like mutual information (MI), chi-square, and information gain (IG) have also been proposed to identify highly distinctive characteristics[39].

Content-based approaches for sarcasm detection offer valuable insights, but they possess several weaknesses:

- Overreliance on explicit cues: They often heavily rely on explicit markers of sarcasm like exclamation marks, emojis, or specific keywords like "obviously" or "just kidding." This makes them miss implicit sarcasm that relies on context, shared knowledge, or subtle linguistic cues.
- Ignoring context and intent: They primarily focus on the text itself, neglecting the broader context of the conversation or the speaker's intent. This can lead to misinterpretations, especially in situations where sarcasm depends heavily on shared experiences or cultural references.
- Sensitivity to linguistic variations: They can be susceptible to variations in language like dialect, humor styles, and cultural nuances. This can lead to misunderstandings, especially when dealing with international audiences or diverse communication styles.

While content-based approaches for sarcasm detection have limitations, they also possess several strengths that make them valuable tools in various contexts:

- Objectivity and consistency: By relying on predefined features and rules, they can offer objective and consistent detection of sarcasm, reducing subjectivity and human bias in interpretation.

- Adaptability with domain-specific knowledge: When combined with domain-specific knowledge or dictionaries, they can be adapted to handle slang, jargon, and specific types of sarcasm relevant to a particular field or community.
- Automation Potential: Their rule-based nature allows for automation and integration into larger systems for tasks like sentiment analysis or chatbots, potentially improving their overall accuracy.

2.3.3 CONTEXT-BASED APPROACH

This approach takes into account the context of a text in order to identify sarcasm. For example, a statement that is sarcastic in one context may not be sarcastic in another context. This approach can be more effective than the other approaches at identifying sarcasm that is subtle or implied.

Studies have indicated the challenging nature of sarcasm detection when relying solely on the content of a statement, emphasizing the need for additional context. In one study, [40] explored various Long Short-Term Memory (LSTM) network configurations to model both the conversational context and the sarcastic response. Their findings revealed that the conditional LSTM network and LSTM networks incorporating sentence-level attention to both context and response surpassed the performance of the LSTM model considering only the response. Another approach, proposed by [41], involved a dual-channel convolutional neural network that considered both the semantics and emotive context of the target text, supplemented by SenticNet to provide common-sense information to the LSTM model. Additionally, [42] demonstrated that annotators often rely on context to accurately identify sarcastic comments, while machine learning models tend to misclassify such comments that necessitate additional context.

In linguistics, an investigation of the relationship between context incongruity and sarcasm [13] developed a computational method that employs context incongruity as the basis for identifying sarcasm.[43] suggested a context-based feature approach to detect sarcasm using the deep learning model, BERT, and traditional machine learning. [44] focused on sarcasm detection in tweets using both Convolutional Neural Network (CNN) feature sets obtained from the architecture and specially designed handcrafted feature sets together with deep learning-based feature extraction.

The study conducted by [45] used Support Vector Machine (SVM) for sarcasm detection, but they did not consider contextual information about the author or the tweet to improve the performance of sarcasm detection. However, since tweets are often part of a stream of posts, a wider context, such as a conversation or topic, is always available. Another study by [46] investigated different pretrained language representation models (PLRMs) such as BERT, RoBERTa, and others using Twitter data. They tested various PLRMs with and without contextual information, and found that by considering the prior three most recent utterances, the algorithm could better classify a dialogue as sarcastic or not.

In a study by [47] they compared the performance of the BERT, BiLSTM, and SVM classifiers in detecting sarcasm in conversations. The study aimed to explore the role of conversational context in identifying sarcasm by manipulating the amount of context provided with the response. The context was varied from no context to the latest one, two, or three utterances, or all utterances. The results showed that incorporating the last utterance of the conversation along with the response improved the performance of the classifiers on the Twitter dataset.

While context-based approaches have significantly improved sarcasm detection over purely content-based methods, they still face limitations. Here are some of the key weaknesses:

- Data dependence: Limited training data most models require large amounts of labelled data with clear context annotations, which can be expensive and time-consuming to collect. Limited data can lead to overfitting and poor performance on unseen contexts
- Contextual ambiguity: Incomplete context: Not all relevant context is always available. In real-world scenarios, information like shared history, cultural references, or nonverbal cues might be missing, making accurate interpretation difficult.

- Multiple interpretations: The same context can be interpreted in different ways depending on individuals' background knowledge and understanding. This ambiguity can lead to misinterpretations of sarcasm.

Strengths of using context-based approaches for sarcasm detection:

- Improved accuracy: By considering the surrounding context, these approaches can overcome the limitations of solely relying on the content of a statement, which can often be ambiguous or misleading. This leads to a more accurate understanding of the speaker's intent and helps identify sarcasm even when it's not explicitly stated.
- Handling ambiguity: Sarcasm often relies on implicit meaning and subtle cues that are difficult to capture with content-based methods. Context-based approaches can better handle these ambiguities by considering factors like the speaker's tone, previous conversation history, and shared knowledge, providing a richer understanding of the situation.
- Modelling complex relationships: By taking into account the interplay between words, context, and the speaker's intent, these approaches can model the complex relationships underlying sarcasm more effectively. This allows them to capture the nuances of humor, irony, and cultural references that are crucial for accurate detection

2.3.4 DEEP LEARNING APPROACH

This approach uses neural networks to learn complex patterns in text data. Neural networks are a type of machine learning algorithm that can be trained on large amounts of data to learn how to perform tasks. This approach can be very effective at identifying sarcasm, but it can also be computationally expensive.

Deep learning approaches for sarcasm detection have been shown to be state-of-the-art, achieving better performance than traditional machine learning approaches. Deep learning models are able to learn complex patterns and representations from raw text data, which is essential for detecting sarcasm, which is often subtle and context-dependent. Some deep learning models include SimpleRNN, LSTM, and GRU.

A SimpleRNN represents a category of recurrent neural network architecture utilised in deep learning and natural language processing, specifically tailored for handling sequential data. While SimpleRNNs boast a straightforward structure, they are constrained in capturing long-range dependencies within sequences. This limitation may render them less proficient in certain tasks compared to more sophisticated RNN variants, such as LSTMs and GRUs, which excel at addressing these challenges.

LSTM, or Long Short-Term Memory, represents a category of recurrent neural network architecture crafted to address the shortcomings of traditional RNNs when confronted with lengthy sequences and the challenge of capturing long-term dependencies in data. Originally introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [48], LSTMs have evolved into a foundational component in diverse applications of deep learning, particularly in the realm of natural language processing.

A Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture used in the field of deep learning and natural language processing. It is a variation of the traditional RNN that was designed to address some of the issues with standard RNNs, particularly the vanishing gradient problem.

Deep learning approaches have shown promising results for sarcasm detection. One common approach is to use RNNs or variations such as LSTM and GRU networks, which are specifically designed to handle sequential data.

In an RNN-based approach, the input is first encoded as a sequence of embeddings that represent the meaning of each word or token in the input text. The embeddings are then fed into the RNN, which processes them sequentially and updates its hidden state at each time step. The final hidden state is then passed to a classifier, such as a fully connected neural network, to predict whether the input text is sarcastic or not.

For example, a study by [23] used an LSTM-based approach to detect sarcasm on Twitter. They first pre-process the input text by removing stop words, replacing emoticons and slang words with their corresponding meanings, and converting the text to lowercase. They then trained an LSTM network on a dataset of sarcastic and non-sarcastic tweets, using the embedding of the pre-processed text as input.

Another study by [49] proposed a hybrid approach that combines an RNN with a Convolutional Neural Network (CNN) to detect sarcasm in Arabic text. They first used a CNN to extract features from the input text and then fed the features into an LSTM network to capture the sequential context. The final hidden state of the LSTM was then passed to a fully connected layer for classification.

This study presents an ensemble approach for detecting sarcasm in Reddit and Twitter responses, part of The Second Workshop on Figurative Language Processing during ACL 2020. The ensemble combines four-component models, utilising features like sentiment and source (Reddit or Twitter) to determine model reliability. It incorporates an LSTM, a CNN-LSTM, an MLP, and an SVM, each with unique features, and achieves F1 scores of 67% and 74% on Reddit and Twitter test data using an Adaboost classifier.

In an investigation conducted by [5] , a combination of LSTM, GRU, and CNN was employed within an ensemble model, coupled with the refinement of word-embedding models such as fastText, Word2Vec, and GloVe. The primary objective was the classification of sentiment into positive/negative or sarcastic/non-sarcastic categories. Remarkably, the ensemble model exhibited superior performance, achieving an accuracy rate of 96% for News Headlines and 73% for Reddit. Notably, the Weighted Average ensemble emerged as the most effective, attaining an impressive accuracy of approximately 99% and 82% for both datasets. This success contributed to heightened precision and stability in the proposed model.

In general, deep learning approaches for sarcasm detection have shown promising results and continue to be an active area of research. However, there are still challenges in developing robust and accurate models that can handle the complexity and nuance of sarcasm in natural language.

Deep learning offers powerful tools for sarcasm detection, but it's important to acknowledge its limitations as well. Here are some key weaknesses:

- Computational Cost: High training and inference requirements: Training deep learning models requires significant computational resources and time. This can be a barrier for smaller organizations or applications with limited resources.

- Data Dependence: Need for large, labelled datasets: Deep learning models thrive on massive datasets for training. Smaller datasets or datasets with inaccurate labels can lead to overfitting and poor generalization. Sarcasm is often nuanced and subjective, making it challenging to create truly representative datasets.
- Generalizability: Performance drop on unseen data: Models trained on specific datasets may not perform well on data from different domains or contexts. This can limit their real-world applicability.

Deep learning approaches offer several strengths for sarcasm detection, despite the weaknesses mentioned previously. Here are some key points:

- High Accuracy and Performance: Ability to capture complex patterns: Deep learning models can analyse large amounts of text data and identify subtle patterns and relationships between words, which are crucial for understanding sarcasm. This can lead to higher accuracy and performance compared to simpler models.
- Adaptability to different data formats: Deep learning models can be trained on various data formats like text, audio, and video, potentially providing a more comprehensive understanding of sarcasm across different modalities.
- Flexibility and Scalability: Ability to handle large datasets: Deep learning models can effectively handle massive datasets, enabling them to learn from diverse examples and improve their accuracy.
- Potential for continuous improvement: Deep learning models can be continuously trained and updated with new data, allowing them to adapt to evolving language styles and improve their performance over time.
- Handling Nuance and Context: Advanced architectures like LSTMs: Newer deep learning architectures like LSTMs and attention mechanisms can capture long-range dependencies and contextual information, allowing them to better understand the nuances of sarcasm.
- Integration with other features: Deep learning models can be combined with other features like sentiment analysis or speaker information, potentially

leading to a richer understanding of context and improved sarcasm detection accuracy.

Approaches	Similarities	Differences
1.1.1 UNEXPECTEDNESS AND CONTRADICTORY FACTOR APPROACH	Relies on linguistic features like negation, exaggeration, and incongruity	Requires manually defined rules and features
1.1.2 CONTENT-BASED APPROACH	Relies on sentiment analysis and word embeddings	May not capture sarcasm that is not expressed through sentiment or word choice
1.1.3 CONTEXT-BASED APPROACH	Relies on understanding the context of the text	May require a large amount of training data
1.1.4 DEEP LEARNING APPROACH	Uses neural networks to learn complex patterns in text data	Can be computationally expensive

Table 2.1 similarities and differences of the approaches.

2.3.5 WHY DO POLITICIANS USE SARCASM?

Understanding why politicians use sarcasm is essential for analysing their communication strategies, persuasive techniques, and the underlying motivations behind their statements.

2.3.6 SOUTH AFRICA’S POLITICAL HOSTORY

South African political history is a complex and multifaceted subject that spans centuries of colonisation, apartheid, and the struggle for democracy.

The early history of South Africa involves the arrival of the Dutch East India Company in the Cape of Good Hope in the 17th century. Scholars have examined the motivations, interactions, and consequences of Dutch settlement, as well as the impact of colonialism on indigenous communities [50];[51].

The nineteenth century witnessed British control over the Cape Colony and the subsequent conflicts with the Boers (Dutch settlers). Research has focused on the causes and outcomes of the Boer Wars, which ultimately led to the British victory and the establishment of British dominance in South Africa [52];[51].

The policy of apartheid, a system of legalised racial segregation and oppression, is a central theme in South African political history. Scholars have explored the origins, development, and impact of apartheid, examining its ideological underpinnings, social engineering, and resistance movements [53].

The struggle against apartheid involved various forms of resistance, both within South Africa and internationally. Extensive research has been conducted on the African National Congress (ANC), the Pan Africanist Congress (PAC) and other anti-apartheid organisations, as well as the role of leaders like Nelson Mandela, Walter Sisulu, and Steve Biko [54].

The late twentieth century saw a process of negotiations and political reforms that led to the dismantling of apartheid and the establishment of a democratic South Africa. Scholars have examined the role of political parties, international actors, and transitional justice mechanisms in this transformative period.

Since the transition to democracy, South Africa has faced numerous challenges, including addressing social inequalities, economic development, and political stability. Research has explored topics such as the Truth and Reconciliation Commission, land reform, socioeconomic disparities, and the role of political parties in shaping the post-apartheid era [55].

2.4 SUMMARY

The literature review investigates the challenging problem of sarcasm detection in natural language processing (NLP) and highlights recent advancements in deep learning methodologies. Traditional approaches, relying on hand-crafted features, were limited in capturing the nuances of sarcasm. In contrast, deep learning models, such as those using word embeddings, recurrent neural networks (RNNs), and convolutional neural networks (CNNs), have shown significant improvements in detecting sarcasm in various applications like sentiment analysis and social media analysis. The review explores approaches that focus on unexpectedness and contradictory factors, content-based features, and context-based strategies, showcasing the importance of contextual signals in addition to lexical and syntactic information for accurate sarcasm detection. The discussion also encompasses a variety of deep learning models, including ensemble approaches, highlighting their promising results while acknowledging the ongoing challenges in developing robust models capable of handling the intricate nature of sarcasm in natural language.

Beyond sarcasm detection, the literature review briefly touches on the use of sarcasm by politicians, emphasizing the importance of understanding their communication strategies and underlying motivations. Furthermore, it provides a historical overview of South African political history, spanning centuries of colonization, apartheid, and the struggle for democracy. The narrative touches upon key historical events such as the Boer Wars and the anti-apartheid movement, illustrating the complex and multifaceted nature of South Africa's political landscape.

3 CHAPTER 3: Methodology

3.1 Introduction

Sarcasm is complex. It's based on context, tone, and cultural understanding; it's difficult for even humans to reliably detect. Recurrent neural networks (RNNs) are suitable for this problem because RNNs excel at processing sequential information (like the order of words in a sentence). They can be trained to develop an internal 'memory' that helps them recognize patterns and nuances in language.

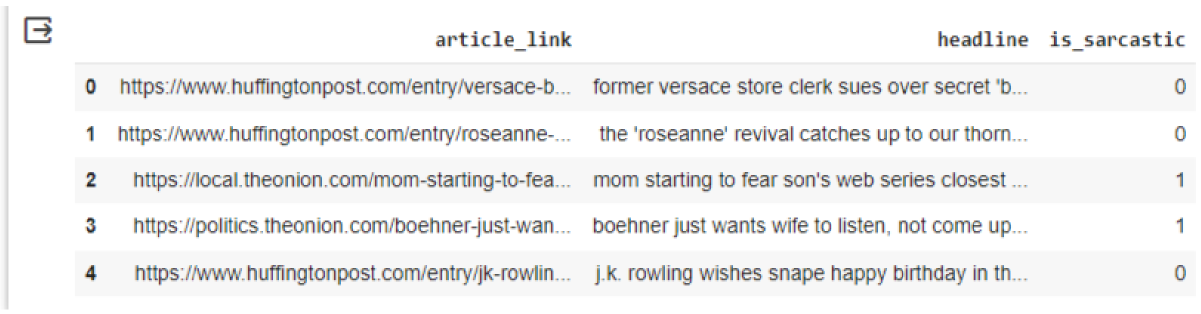
The basic structure of this chapter presents stages for our proposed method, data pre-processing, word embedding, deep learning frame work, and performance evaluation. The news headline dataset was pre-processed, glove applied, models built, 80% of the news headline dataset was used for training and 20% for validation.

3.2 Datasets

3.2.1 News headline dataset

In this research, we utilised the news headline dataset for detecting sarcasm [1] , which offers distinct advantages compared to other datasets used for sarcasm detection, such as those sourced from Twitter [2]. Notably, this dataset exhibits reduced sparsity, reducing the likelihood of spelling errors and informal language usage in headlines. This characteristic enhances the ease with which machine learning models can glean insights from the data. Furthermore, the dataset is more likely to incorporate words commonly found in pre-trained language models, enabling researchers to enhance their model's performance by leveraging pre-trained embeddings, eliminating the need to train their embedding layers from scratch. The dataset also boasts high-quality labels with minimal noise, as headlines are clearly labelled as either non-sarcastic (0) or sarcastic (1). Figure 3.1 provides a snippet of the news headline dataset, while Figure 3.2 illustrates the distribution of sarcastic and non-sarcastic headlines. The dataset is structured with three columns which are Article_link, headline, Is_sarcastic, and for experimentation purposes, it was partitioned into training and testing subsets, with 80% allocated for training and 20% for testing.

- Article_link- This attribute supplies the links to the original news articles, facilitating the retrieval of extra information.
- headline- This field furnishes the headlines of the news articles.
- Is_sarcastic- If the headline is sarcastic, the value assigned is 1; otherwise, it is 0.



	article_link	headline	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0

Figure 3.1 Snippet of the news headline dataset

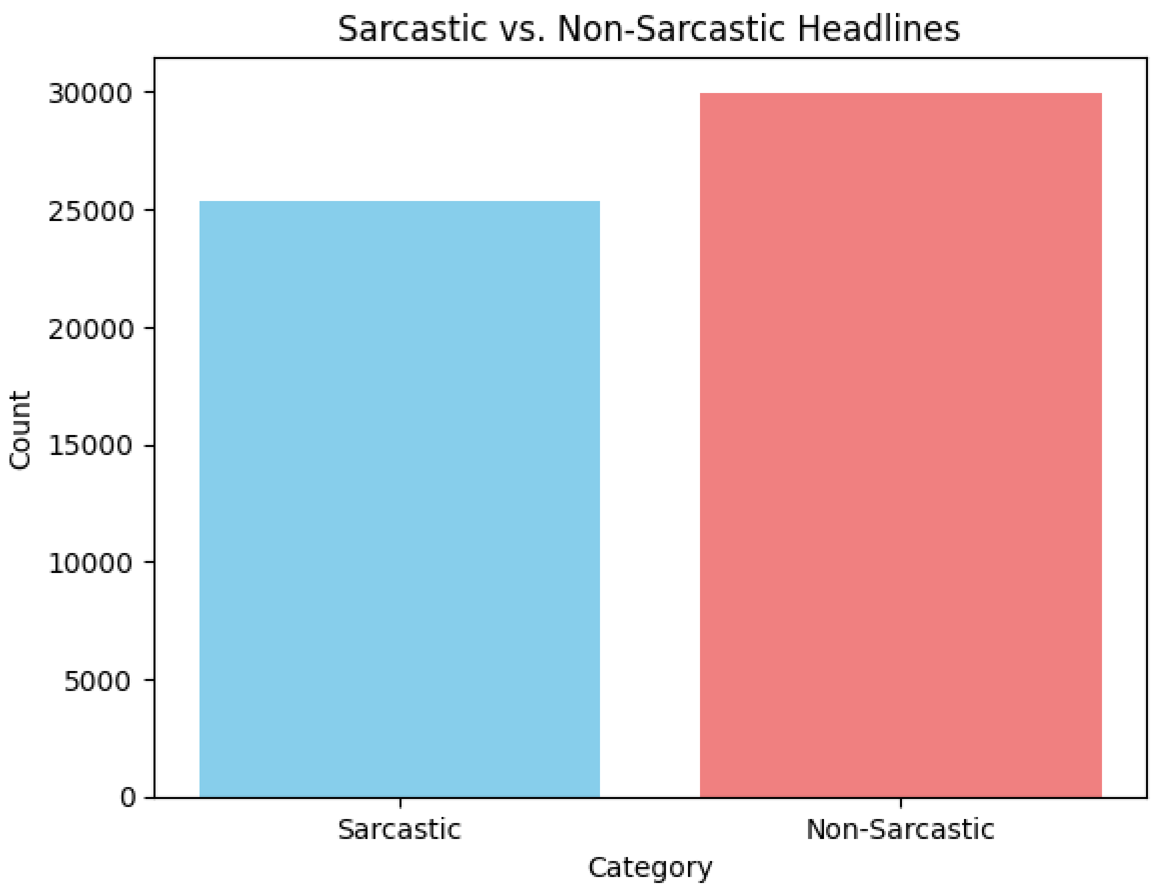


Figure 3.2 Distribution if sarcastic and non-sarcastic headlines

3.2.2 Glove dataset

The word embedding dataset used in this study is a global vector (Glove), glove combines the two major model families, global matrix factorisation and local context window [56]. The model leverages statistical information by training only on nonzero elements in a word-word co-occurrence matrix rather than the entire sparse matrix or on individual context window in a large corpus. The Glove used in this study has 1193514-word vectors.

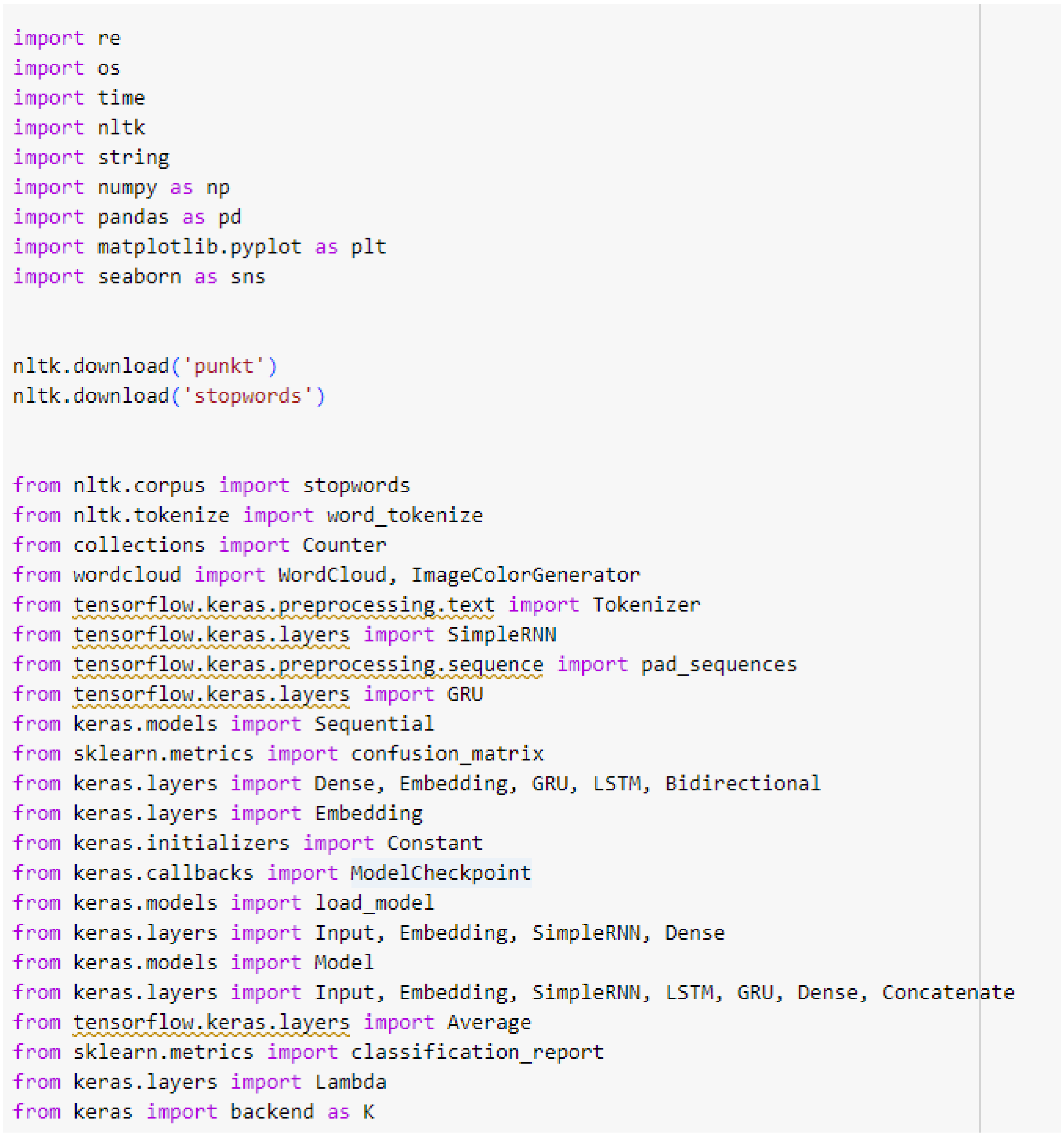
3.3 Implementation Tools

Sarcasm detection using political speeches was implemented in Google collab using python 3, using HP computer with windows 11 Enterprise 64-bit operating system, Intel(R) Core (TM) i5-8500 CPU @ 3.00GHz 3.00 GHz processor, and 16,0 GB RAM. The programming language used was Python, Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in 1991. Python is designed to be easy to learn and offers clear and concise syntax, making it an excellent choice for both beginners and experienced developers. Python is considered an excellent choice for Natural Language Processing (NLP) for several reasons:

- **Abundant NLP Libraries:** Python has a rich ecosystem of NLP libraries and frameworks, such as NLTK (Natural Language Toolkit), spaCy, Gensim, TextBlob, and more. These libraries provide pre-built tools and functionalities for various NLP tasks, saving time and effort when developing NLP applications.
- **Machine Learning and Deep Learning:** Python is the go-to language for machine learning and deep learning. Many popular deep learning frameworks like TensorFlow and PyTorch have Python APIs, enabling NLP practitioners to leverage neural networks for NLP tasks like text classification, named entity recognition, machine translation, and sentiment analysis.

- **Text Processing Tools:** Python offers built-in libraries and third-party libraries for text processing, regular expressions, and string manipulation. These tools are crucial for cleaning and pre-processing textual data in NLP.
- **Natural Language Toolkits (NLTks):** Python's NLP libraries often include corpora, lexicons, and pre-trained models that can help with various NLP tasks. For example, NLTK provides access to a vast collection of text datasets and linguistic resources.

Figure 3.3 shows the Python libraries that were used in this study.



```
import re
import os
import time
import nltk
import string
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


nltk.download('punkt')
nltk.download('stopwords')


from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
from wordcloud import WordCloud, ImageColorGenerator
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import SimpleRNN
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import GRU
from keras.models import Sequential
from sklearn.metrics import confusion_matrix
from keras.layers import Dense, Embedding, GRU, LSTM, Bidirectional
from keras.layers import Embedding
from keras.initializers import Constant
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras.layers import Input, Embedding, SimpleRNN, Dense
from keras.models import Model
from keras.layers import Input, Embedding, SimpleRNN, LSTM, GRU, Dense, Concatenate
from tensorflow.keras.layers import Average
from sklearn.metrics import classification_report
from keras.layers import Lambda
from keras import backend as K
```

Figure 3.3 Python libraries used in the study

i. re (Regular Expressions):

The ‘re’ library is used to work with regular expressions in Python. It allows you to search, match, and manipulate text based on patterns. Regular expressions are used for tasks such as pattern matching, text parsing, data extraction, and text manipulation.

ii. os

The ‘os’ library provides functions to interact with the operating system. It allows the user to perform tasks such as file and directory manipulation, working with paths, and executing system commands.

iii. time

The ‘time’ library is used to work with time-related operations. It provides functions to measure time intervals and manage time in Python code. The time library is used to profile code execution, adding delays, measure performance, and working with timestamps.

iv. nltk (Natural Language Toolkit):

‘NLTK’ is a comprehensive library for natural language processing (NLP). It provides tools and resources for working with human language data. NLTK is used for tasks like text tokenization, stemming, lemmatisation, part-of-speech tagging, and more in NLP projects.

v. string:

The ‘string’ library provides a collection of string constants and functions for string manipulation. It is used for tasks like string formatting, character manipulation, and text processing.

vi. NumPy (np):

‘NumPy’ is a fundamental library for numerical computing. It provides support for arrays, matrices, and various mathematical functions for numerical operations. NumPy is used for numerical and scientific computing, including tasks like linear algebra, statistical analysis, and data manipulation.

vii. pandas (pd):

‘Pandas’ is a data manipulation and analysis library. It offers data structures (e.g., Data Frames) and functions for cleaning, transforming, and analysing structured data, and it is commonly used for data wrangling, data exploration, and data preparation in data science and data analysis projects.

viii. matplotlib. pyplot (plt):

‘Matplotlib’ is a data visualisation library, and the pyplot module is used to create a wide range of charts and graphs. It is used to create various types of plots, including line plots, scatter plots, bar charts, histograms, and more.

ix. Seaborn (sns):

‘Seaborn’ is a data visualisation library built on top of Matplotlib. It simplifies the process of creating aesthetically pleasing statistical graphics, and it is often used for creating visually appealing statistical visualisations, including heatmaps, pair plots, and distribution plots.

x. tensorflow.keras and keras.layers:

These libraries provide tools for building and training deep learning models using the Keras API. They include various types of layers for neural networks, they can use these libraries to build neural networks for tasks like image classification, text generation, and sentiment analysis.

xi. ModelCheckpoint:

The ‘ModelCheckpoint’ callback in Keras allows you to save the best model during training based on a specified metric. It is used in deep learning to save model checkpoints to resume training from the best model or for model deployment.

xii. load_model:

The ‘load_model’ function is used to load pretrained Keras models from disk. It is used when you want to use a pre-trained model for inference or further training.

xiii. Input, Embedding, SimpleRNN, LSTM, GRU, Dense, Concatenate, Average, Lambda:

These are Keras layers and functions that are used to build complex neural network architectures for various deep learning tasks. These layers and functions are used to define the structure of neural networks for specific tasks such as text classification, sequence-to-sequence tasks, and more.

xiv. tensorflow.keras.layers.Average:

This layer of TensorFlow is used to calculate the average of the input data. It is used when you need to compute the mean of data as part of a neural network architecture.

xv. K (from keras):

The ‘K’ object is the Keras backend, allowing low-level operations and customisations within Keras models It is used when you need to define custom operations or loss functions within a Keras model.

3.4 Methodology Framework

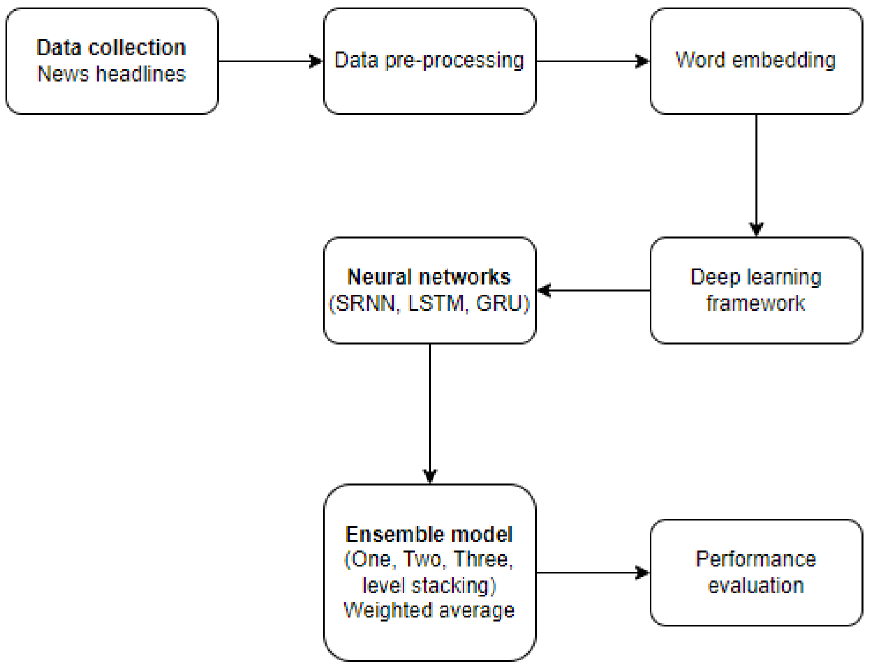


Figure 3.4 Methodology flow chat

3.4.1 Data Pre-processing.

Data preprocessing [57],[7],[5] involves converting the raw dataset into a structured format, enhancing data efficiency and thereby influencing the performance of algorithms. The news headlines underwent multiple filtering processes. Prior to the training phase, the dataset underwent preprocessing using the Natural Language Toolkit (NLTK). Various procedures were implemented to refine the datasets, encompassing the expansion of contractions into full words, removal of numbers, non-English words, stop words, special characters, and punctuations. Additionally, measures such as eliminating duplicate texts, employing tokenization, and ultimately, utilising Lemmatisation to derive the lemma of each word post morphological analysis, were undertaken to prepare the dataset for further analysis.

3.4.2 Word embedding

Word embeddings [58],[59] is a method that translates words into numerical vectors, creating distinct representations based on a given body of text. This technique, an improvement over traditional approaches like the bag-of-words model, offers the advantage of revealing previously unrecognised relationships between words. Unlike the bag-of-words model, which produces large, sparse vectors making it computationally challenging to cover the entire vocabulary, word embeddings group similar words together. Notable techniques for generating word embeddings include Word2vec, GloVe, and fastTex [60],[5]. These methods enhance the contextual relevance of words by representing them as numerical vectors through embedding techniques. Pretrained models encode specific meanings into words, representing them in binary form. In this representation, '1' indicates the presence of sarcasm, while '0' signifies its absence. The collected data is subsequently analyzed to yield additional insights [5].

The underlying principle of word embeddings [61] lies in recognising that if a user in the training data exhibits a particular personality and frequently posts sarcastic tweets, encountering new data featuring a different user with a similar style and, consequently, a comparable word embedding allows predictions about the likelihood of sarcasm in the new user's tweets. This prediction relies on the similarity between embeddings, bypassing the need to scrutinize the content of the new user's tweet.

Word embeddings present a more efficient and faster approach, facilitating improved training and learning from extensive text corpora. This method enables the sharing of word representations, leading to the creation of more robust and less common word representations. In our study, we utilized publicly available GloVe embeddings [56], trained on datasets from Common Crawl and Wikipedia.

3.4.3 Deep Learning Frameworks

Deep learning [7],[5],[62] demonstrates its proficiency in discovering valuable data representations tailored to specific tasks. It possesses the ability to extrapolate new features from a limited set of training features autonomously, without requiring human intervention. Essentially, it actively searches for and identifies additional features that have a correlation with the existing ones, eliminating the need to label everything in the dataset. Deep learning models can achieve precision levels that are at the forefront of performance, sometimes even surpassing human-level capabilities. These models are trained using neural network architectures that comprise numerous layers and extensive sets of labelled data.

To detect sarcasm, we incorporated advanced deep learning techniques like SRNN, LSTM, and GRU. The process begins by inputting the preprocessed sequence into a pre-trained word embedding layer, which assigns a distinct index to each word in the sentence, generating fixed-length vectors. Subsequently, the SRNN, LSTM, and GRU layers are employed to capture extended dependencies within the context. The pooling layer consolidates the acquired information into a feature dimension, which is then converted into a column vector through a flat layer. Ultimately, the inner layer handles the classification, concluding the entire neural network procedure.

3.4.3.1 Simple recurrent neural network (SRNN)

SRNN also known as the Elman network, was proposed by Jeffrey L. Elman in 1990[63],[62]. Jeffrey L. Elman is a cognitive psychologist and computer scientist who introduced this type of recurrent neural network to model sequential data and capture dependencies in such data. His work on SRNNs was a significant contribution to the field of artificial neural networks and played a foundational role in the development of more advanced recurrent neural network architectures like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. In this, the SimpleRNN layer in this code is used to process input sequences, capture sequential dependencies, and produce a sequence of outputs that can be used for various NLP tasks, such as sarcasm detection. Figure 3.5 shows the basic stricture of an SRNN

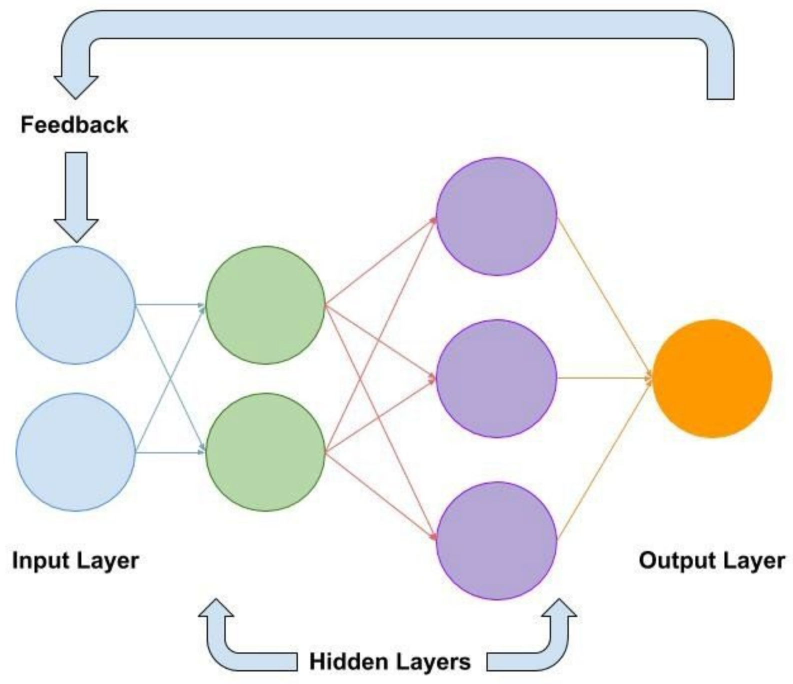


Figure 3.5 A simple RNN architecture (credit https://www.researchgate.net/figure/A-simple-RNN-architecture_fig3_337830532)

3.4.3.2 Long short-term memory neural network (LSTM)

The Long-Short-Term Memory Neural Network (LSTM), initially introduced by Hochreiter and Schmidhuber in [64], has undergone further refinement and widespread application to a wide range of problems. In this, the LSTM layer processes the input sequence step by step, maintaining an internal state that allows it to capture information from earlier time steps and use it to influence predictions at later time steps. Figure 3.6 depicts the structure on an LSTM unit

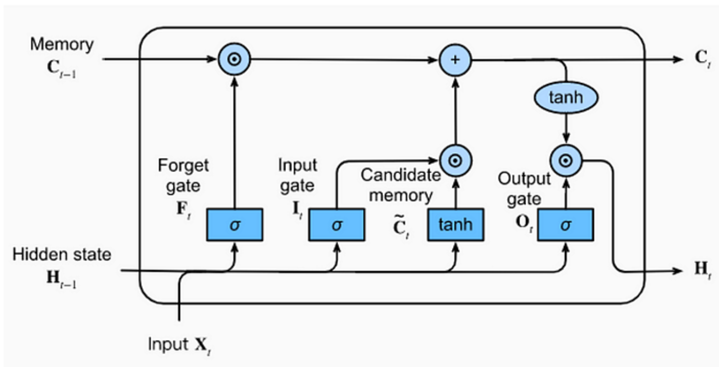


Figure 3.6 Architecture of a LSTM Unit (Credits: https://d2l.ai/chapter_recurrent-modern/lstm.html)

3.4.3.3 *Gated recurrent unit (GRU)*

The Gated Recurrent Unit (GRU), introduced by Cho [21], shares a similar architecture with LSTM. In this context, we employ the GRU [65][5] to facilitate connections across a sequence of nodes to perform machine learning tasks related to memory and clustering, such as text identification. GRU plays a crucial role in adjusting the input weights of the neural network, effectively addressing the common challenge of the vanishing gradient problem often encountered in recurrent neural networks. Figure 3.7 shows the architecture of basic Gated Recurrent Unit

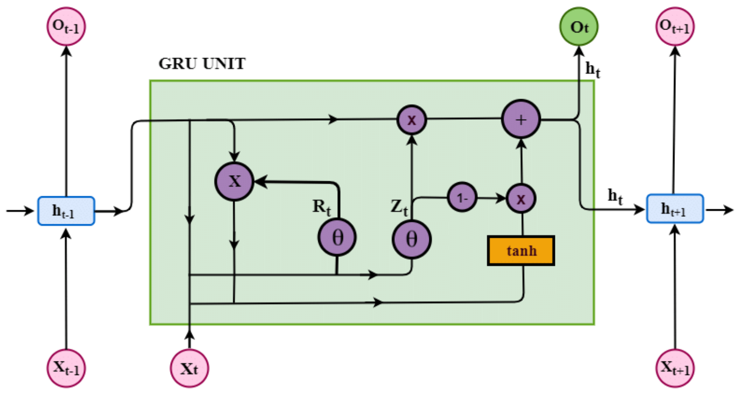


Figure 3.7 The Architecture of basic Gated Recurrent Unit (GRU).(credit https://www.researchgate.net/figure/The-Architecture-of-basic-Gated-Recurrent-Unit-GRU_fig1_343002752)

3.4.4 Proposed model - ensemble learning

Ensemble methods encompass a set of strategies involving the creation of multiple models and the integration of their outputs to achieve improved results. Generally, ensemble methods surpass individual models in terms of accuracy. Ensemble Learning, as elucidated in [66],[67], focuses on mastering the effective amalgamation of predictions from various pre-existing models known as base-learners. Each member of the ensemble contributes to the final outcome, addressing individual weaknesses with the support of other members. The combined learned model is represented by the meta-learner.

In our research, we have employed Ensemble Learning [68],[5] to enhance the performance of our models across tasks such as prediction, classification, and function approximation. The stacking technique was specifically employed in this study. Stacking is an ensemble approach that consolidates predictions from multiple machine learning models to enhance overall performance. In a stacking model, a meta-learner or aggregator model is trained using predictions generated by various base models.

3.4.4.1 Importing Libraries:

The code initiates by importing essential libraries necessary for building and training neural networks using Keras, a widely-used deep learning framework. It imports modules like Input, embedding, SimpleRNN, LSTM, GRU, Dense, Concatenate, Lambda, and backend functions from Keras. Additionally, the code imports the NumPy library for numerical operations.

3.4.4.2 One-Level Stacking

Define Input Layer:

The code starts by defining an input layer with a shape of (max_length), where max_length represents the maximum sequence length that the model will accept.

Three Separate RNN Branches:

In this one-level stacking model, the input sequence is processed through three separate branches, each utilising a different type of RNN: SimpleRNN, LSTM, and GRU. Each branch is configured with 64 units and applies dropout and recurrent dropout for regularisation. The return_sequences parameter is set to False, which means that the output of each RNN branch is a fixed-size vector, not a sequence.

Concatenation of RNN Branch Outputs:

The outputs of the three RNN branches (SimpleRNN, LSTM, and GRU) are concatenated into a single tensor using the Concatenate layer (). This step combines the information learnt by each RNN branch into a unified representation, capturing different aspects of the input data.

Dense Layer for Binary Classification:

A dense layer with a single unit and a sigmoid activation function is added to make the final binary classification decision. This layer takes the concatenated output from the RNN branches as input and computes the probability of the binary classification outcome. The sigmoid activation function outputs values between 0 and 1, indicating the likelihood that the input data belong to one of the two classes.

Stacking Model Creation:

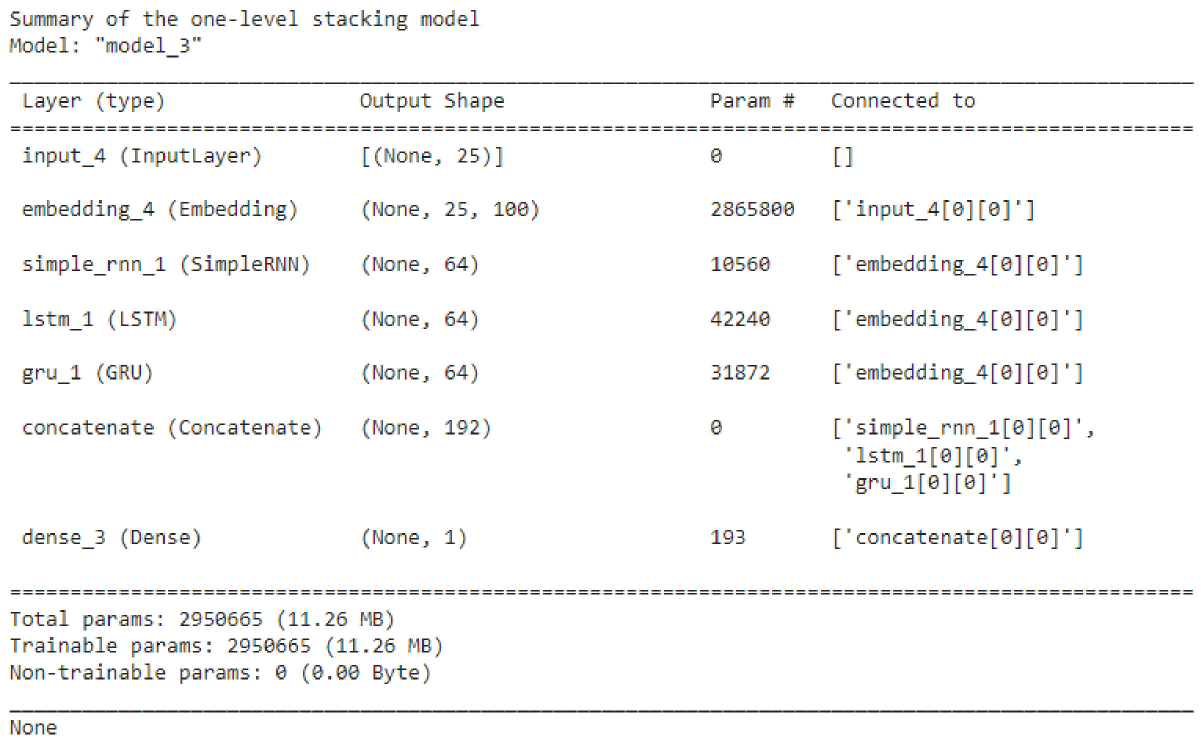
The code assembles the complete one-level stacking model, named stacking_model1, by connecting the input layer to the output layer.

Compilation:

The stacking model is then compiled with a binary cross-entropy loss function and the Adam optimiser. The model is also configured to track accuracy as a training metric, allowing performance evaluation during training.

Summary Printing:

Finally, we print a summary of the stacking_model1. Figure 3.8 provides detailed information about the model architecture, including the layers, output shapes, and the number of trainable parameters. It serves as a useful reference for understanding the model's structure and complexity.


```
Summary of the one-level stacking model
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_4 (InputLayer)        [(None, 25)]                 0         []

 embedding_4 (Embedding)     (None, 25, 100)              2865800   ['input_4[0][0]']

 simple_rnn_1 (SimpleRNN)    (None, 64)                   10560     ['embedding_4[0][0]']

 lstm_1 (LSTM)               (None, 64)                   42240     ['embedding_4[0][0]']

 gru_1 (GRU)                 (None, 64)                   31872     ['embedding_4[0][0]']

 concatenate (Concatenate)   (None, 192)                  0         ['simple_rnn_1[0][0]',
                                                                     'lstm_1[0][0]',
                                                                     'gru_1[0][0]']

 dense_3 (Dense)             (None, 1)                    193       ['concatenate[0][0]']

==================================================================================================
Total params: 2950665 (11.26 MB)
Trainable params: 2950665 (11.26 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
```

Figure 3.8 summery of the one level stacking model

In summary, this code defines a one-level stacking model that combines different RNN types at the same level to capture diverse aspects of the input data. The model is designed for binary classification tasks and aims to learn complex representations from sequential data by leveraging multiple RNN architectures.

3.4.4.3 Two-level stacking

Define Input Layer:

The code starts by defining an input layer with a shape of (max_length), where max_length represents the maximum sequence length that the model will accept.

Level 1 - Different RNN Branches:

In the first level (Level 1), the input sequence is passed through three separate branches, each utilising a different type of RNN. Specifically, it uses SimpleRNN, LSTM, and GRU with 64 units in each branch. The dropout and the recurrent dropout are applied to each RNN branch for regularisation. The return_sequences parameter is set to False, which means that the output of each RNN branch is a fixed-size vector rather than a sequence.

Concatenation in Level 1:

The outputs of the three Level 1 branches (SimpleRNN, LSTM, and GRU) are concatenated into a single tensor using the Concatenate layer (). This step combines the information learnt by each RNN branch into a unified representation, capturing different aspects of the input data.

Level 2 - Final Dense Layer:

At level 2, a dense layer with a single unit and a sigmoid activation function is added. This layer serves as the final decision-making step for binary classification. It takes the concatenated output of Level 1 as input and computes the probability of the binary classification result, where the sigmoid activation function outputs values between 0 and 1.

Stacking Model Creation:

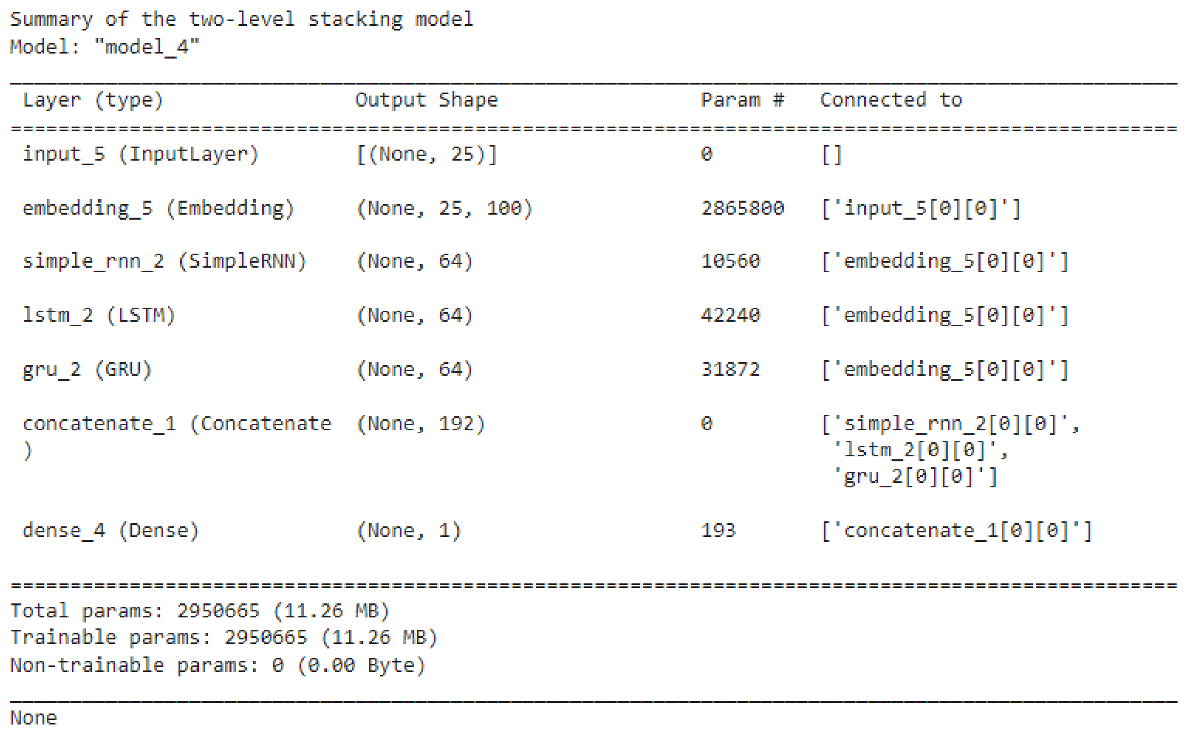
The code assembles the complete two-level stacking model, named stacking_model2, connecting the input layer to the output layer.

Compilation:

The stacking model is then compiled with a binary cross-entropy loss function and the Adam optimiser. The model is also configured to track accuracy as a training metric, allowing performance evaluation during training.

Summary Printing:

Finally, a summary of the stacking_model2. Figure 3.9 provides detailed information about the model architecture, including the layers, output shapes, and the number of trainable parameters. It serves as a useful reference for understanding the model structure and complexity.



```
Summary of the two-level stacking model
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_5 (InputLayer)        [(None, 25)]                 0         []

 embedding_5 (Embedding)     (None, 25, 100)              2865800   ['input_5[0][0]']

 simple_rnn_2 (SimpleRNN)    (None, 64)                   10560     ['embedding_5[0][0]']

 lstm_2 (LSTM)               (None, 64)                   42240     ['embedding_5[0][0]']

 gru_2 (GRU)                 (None, 64)                   31872     ['embedding_5[0][0]']

 concatenate_1 (Concatenate  (None, 192)                  0         ['simple_rnn_2[0][0]',
 )                                                                   'lstm_2[0][0]',
                                                                     'gru_2[0][0]']

 dense_4 (Dense)             (None, 1)                    193       ['concatenate_1[0][0]']

==================================================================================================
Total params: 2950665 (11.26 MB)
Trainable params: 2950665 (11.26 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
```

Figure 3.9 summery of the two- level stacking model

In summary, this code defines a two-level stacking model that combines different RNN types at the first level to capture diverse aspects of the input data. The second level, which consists of a dense layer with sigmoid activation, makes the final binary classification decision. This architecture is designed for binary classification tasks and aims to learn hierarchical representations from sequential data.

3.4.4.4 Three level stacking

Defining Input Layer:

An input layer is established using Input(shape=(max_length,)). This layer is tailored to receive sequences of fixed length, where max_length signifies the maximum sequence length the model can handle.

Level 1 - Different RNN Branches:

In the first level (Level 1), the code creates three independent branches, each employing a distinct type of recurrent neural network (RNN): SimpleRNN, LSTM, and GRU. These branches process the input sequence separately, applying the respective RNN type with 64 units. Dropout and recurrent dropout are incorporated for regularisation. The return_sequences parameter is set to False, implying that the output of each branch is a fixed-size vector, not a sequence.

Concatenation (Level 1):

After Level 1, the outputs of the three RNN branches (SimpleRNN, LSTM, and GRU) are merged into a single tensor using the Concatenate layer. This concatenation combines the knowledge acquired by each RNN branch into a unified representation.

Level 2 - Additional RNN Branches:

In the second level (Level 2), three additional branches are constructed. These branches receive the concatenated output from Level 1 as input. Each of these Level 2 branches applies a different RNN type (SimpleRNN, LSTM, GRU) and integrates dropout and recurrent dropout for regularisation.

Concatenation (Level 2):

The outputs of the Level 2 branches (SimpleRNN, LSTM, and GRU) are combined again through concatenation, generating a fresh combined representation.

Level 3 - Final Dense Layer:

Level 3 introduces a dense layer that houses a single unit with a sigmoid activation function. This layer is responsible for delivering the ultimate binary classification output. It takes the concatenated output of Level 2 as input and computes the probability of the binary classification task, distinguishing between two classes.

Stacking Model Creation:

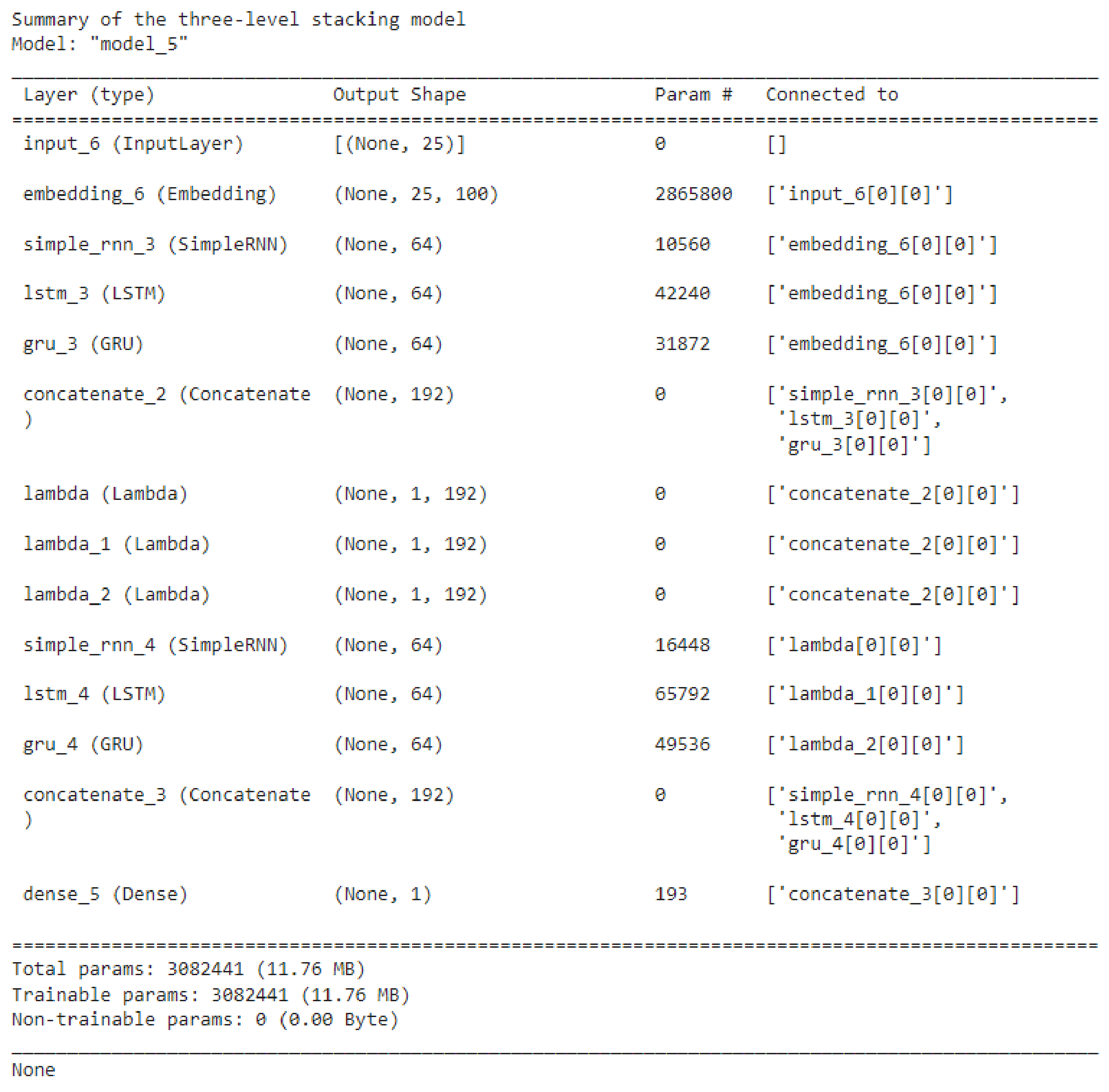
The code proceeds to form the complete stacking model, named stacking_model3. The input layer is linked to the output layer, shaping the model's architecture.

Compilation:

The stacking model is then compiled, with the binary cross-entropy loss function and the Adam optimiser being specified. The model is also configured to track accuracy as a training metric to evaluate its performance during training.

Summary Printing:

Finally, a summary of the stacking_model3. Figure 3.10 offers in-depth information about the model layers, their respective output shapes, and the number of trainable parameters within the model.



```
Summary of the three-level stacking model
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_6 (InputLayer)        [(None, 25)]                 0         []

 embedding_6 (Embedding)     (None, 25, 100)              2865800   ['input_6[0][0]']

 simple_rnn_3 (SimpleRNN)    (None, 64)                   10560     ['embedding_6[0][0]']

 lstm_3 (LSTM)               (None, 64)                   42240     ['embedding_6[0][0]']

 gru_3 (GRU)                 (None, 64)                   31872     ['embedding_6[0][0]']

 concatenate_2 (Concatenate  (None, 192)                  0         ['simple_rnn_3[0][0]',
 )                                                                   'lstm_3[0][0]',
                                                                     'gru_3[0][0]']

 lambda (Lambda)             (None, 1, 192)               0         ['concatenate_2[0][0]']

 lambda_1 (Lambda)           (None, 1, 192)               0         ['concatenate_2[0][0]']

 lambda_2 (Lambda)           (None, 1, 192)               0         ['concatenate_2[0][0]']

 simple_rnn_4 (SimpleRNN)    (None, 64)                   16448     ['lambda[0][0]']

 lstm_4 (LSTM)               (None, 64)                   65792     ['lambda_1[0][0]']

 gru_4 (GRU)                 (None, 64)                   49536     ['lambda_2[0][0]']

 concatenate_3 (Concatenate  (None, 192)                  0         ['simple_rnn_4[0][0]',
 )                                                                   'lstm_4[0][0]',
                                                                     'gru_4[0][0]']

 dense_5 (Dense)             (None, 1)                    193       ['concatenate_3[0][0]']

==================================================================================================
Total params: 3082441 (11.76 MB)
Trainable params: 3082441 (11.76 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
```

Figure 3.10 summery of the three-level stacking model

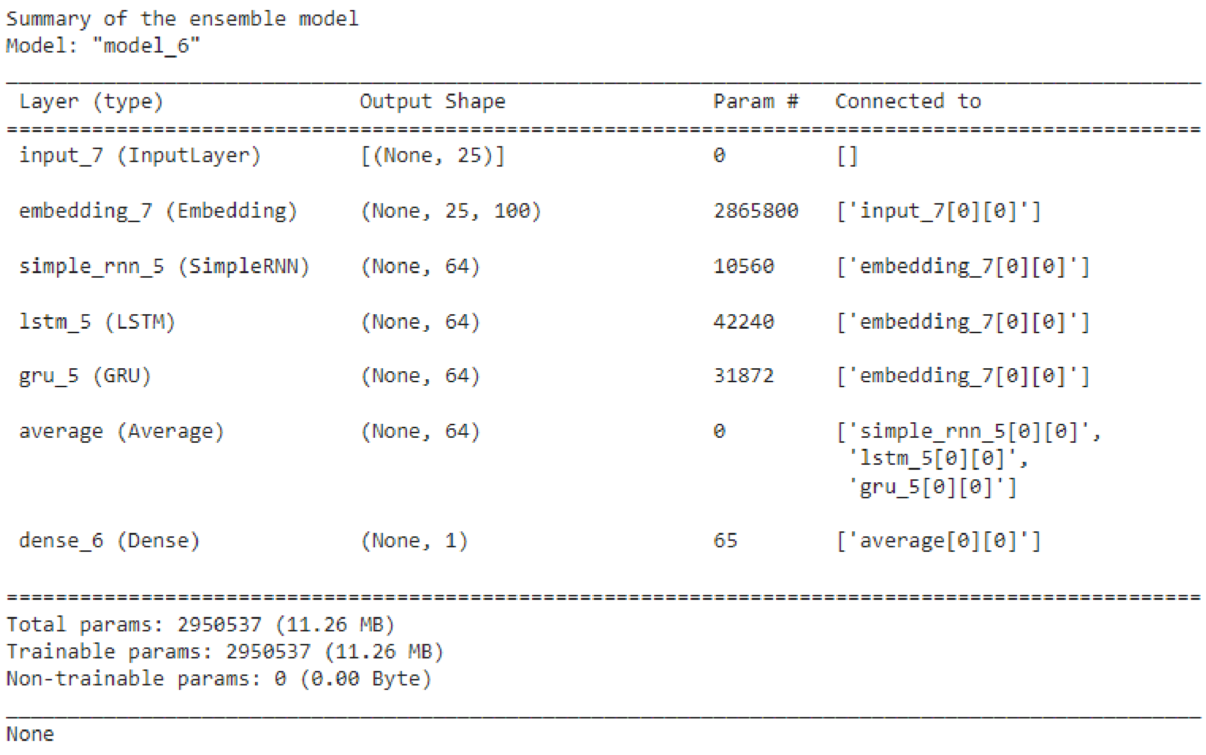
In summary, the code constructs a complex neural network architecture that combines various types of RNN at multiple levels. This architecture is designed to capture hierarchical patterns within sequential data, primarily intended for a binary classification task. The model aims to learn and generalise from sequential data, with each level contributing to a progressively more abstract representation of the input data.

3.4.4.5 Weighted Average

Three separate branches of recurrent neural networks (RNNs) are defined: SimpleRNN, LSTM, and GRU. Each branch processes the input data using a different RNN architecture. These branches are trained to capture different patterns and features in the data.

The outputs from these three branches are then combined using an averaging layer, which calculates the average prediction of each branch. This ensemble approach allows the model to benefit from the unique insights of each RNN type.

After averaging, a dense layer is added for binary classification. It makes the final decision about whether the input data belongs to one of two classes (Sarcastic and Non-sarcastic). The ensemble model is compiled, specifying how it should be trained, and prints a summary of the model's architecture. Figure 3.11 shows the model summary



```
Summary of the ensemble model
Model: "model_6"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_7 (InputLayer)        [(None, 25)]                 0         []

 embedding_7 (Embedding)     (None, 25, 100)              2865800   ['input_7[0][0]']

 simple_rnn_5 (SimpleRNN)    (None, 64)                   10560     ['embedding_7[0][0]']

 lstm_5 (LSTM)               (None, 64)                   42240     ['embedding_7[0][0]']

 gru_5 (GRU)                 (None, 64)                   31872     ['embedding_7[0][0]']

 average (Average)           (None, 64)                   0         ['simple_rnn_5[0][0]',
                                                                     'lstm_5[0][0]',
                                                                     'gru_5[0][0]']

 dense_6 (Dense)             (None, 1)                    65        ['average[0][0]']

==================================================================================================
Total params: 2950537 (11.26 MB)
Trainable params: 2950537 (11.26 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
```

Figure 3.11 summery of the weighted average ensemble model

Ensemble models combine information from three different RNN architectures (SimpleRNN, LSTM, and GRU) through element-wise averaging, and then uses a dense layer to produce the final prediction. The total number of parameters in this model is approximately 2.95 million.

In summary, an ensemble model combines the predictions of different RNN branches by averaging, leading to a collective prediction for binary classification. Ensemble models often improve performance by leveraging the diverse perspectives of multiple models.

3.5 Performance evaluation

Assessing the performance of sarcasm detection models is pivotal for the development of effective systems. [6] Various metrics come into play to gauge accuracy, precision, and recall in sarcasm detection models.

Accuracy, precision, recall, and the F1 score serve as metrics for evaluating the performance of a classification model. These metrics are derived from the following four terms:

- True Positives (TP): The number of cases that were correctly predicted as positive.
- False Positives (FP): The number of cases that were incorrectly predicted as positive.
- True Negatives (TN): The number of cases that were correctly predicted as negative.
- False Negatives (FN): The number of cases that were incorrectly predicted as negative.

Confusion matrix		
Predicted sarcasm	Sarcastic	Not-sarcastic
Sarcastic	True positives (TP):	False positives (FP):
Not-sarcastic	True negatives (TN):	False negatives (FN):

Table 3.1 confusion matrix

- Calculate accuracy. Accuracy is the proportion of all cases that were correctly predicted. It is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad 1$$

- Calculate the precision. Precision is the proportion of positive predictions that were correct. It is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \quad 2$$

- Calculate recall. Recall is the proportion of actual positives that were correctly predicted. It is calculated as follows:

$$Recall = \frac{TP}{TP + FN} \quad 3$$

- Calculate F1 score. The F1 score is a harmonic mean of precision and recall. It is calculated as follows:

$$F1 - score = 2\left(\frac{Precission * Recall}{Precission + Recall}\right) \quad 4$$

3.6 Conclusion

This chapter discussed the stages for our suggested method, data collection, data pre-processing, deep learning framework, and the performance evaluation. The models were implemented using python programming language due to its libraries like NLTK, a strong community support, and seamless integration with machine learning frameworks, offering simplicity and versatility for developing and implementing NLP tasks.

4 Chapter 4: Results

4.1 INTRODUCTION

In this section, we investigated different approaches utilised to analyse the dataset for our research. The central objective is to identify sarcasm within textual data through the application of deep learning techniques. While many studies typically employ basic deep learning algorithms like simpleRNN, LSTM, and GRU, our study introduces an ensemble model that combines the characteristics of the aforementioned models. Specifically, we incorporated ensemble models to augment our analysis, including the Weighted Average Ensemble, One-Level Stacking, Two-Level Stacking, and Three-Level Stacking techniques. We have conducted a comparative assessment and implementation of these ensemble models to evaluate their influence on accuracy.

The models underwent training using the Sigmoid activation function and the Adam optimizer, with the inclusion of dropouts set at 0.2 and recurrent dropouts at 0.25. To enhance the model's performance, we conducted additional fine-tuning by adjusting its parameters and hyperparameters, as outlined in the table below.

Parameters	Values
Kernel	3
Embedding Dimensions	100
Epochs	25
Activation Function	Sigmoid
Loss	Binary cross entropy
Batch size	32
Word Embedding	Glove
Verbose	2
Dropout	0.2

Optimizer	Adam

Table 4.1 Parameters list for Training and Validating our models.

4.2 Results for the Neural networks

The SRNN, LSTM, and GRU models were trained by utilising 80% of the news headlines dataset, and subsequently, validation was conducted on the remaining 20% of the data. The training involved 25 epochs with a batch size of 32. Figures 4.1, 4.2, and 4.3 depict the training and validation curves for the models employing GloVe word embeddings.

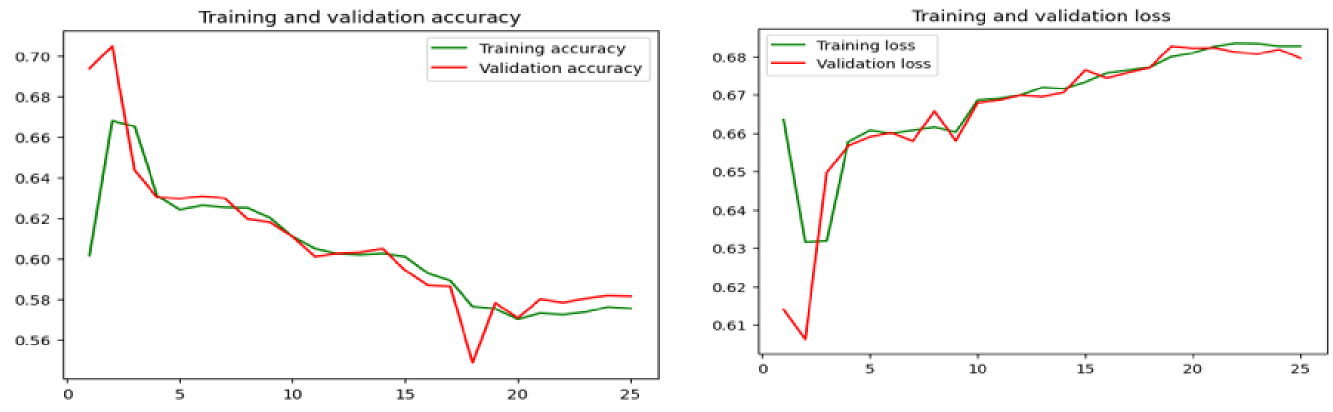


Figure 4.1 Training and validation curve of SimpleRNN

(Left): Graph illustrating the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) against the Number of Epochs for the SRNN Model.

(Right): Graph illustrating the Training Loss (shown in green) and Validation Loss (shown in red) against the Number of Epochs for the SRNN Model.

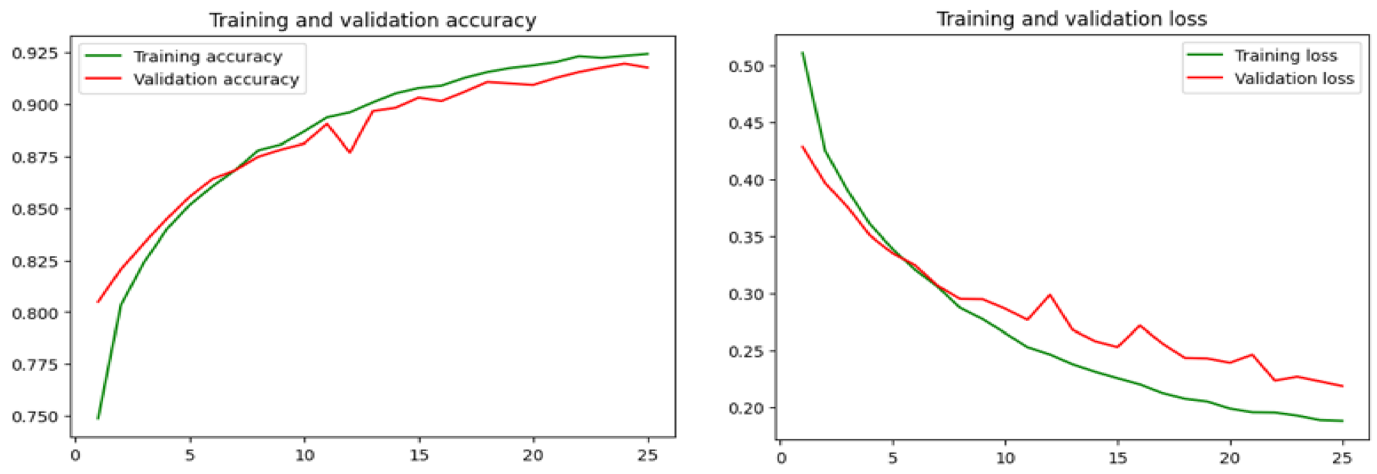


Figure 4.2 Training and validation curve of LSTM

(Left): Graph illustrating the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) relative to the Number of Epochs for the LSTM Model.

(Right): Chart showing the Training Loss (in green) and Validation Loss (in red) in relation to the Number of Epochs for the LSTM Model.

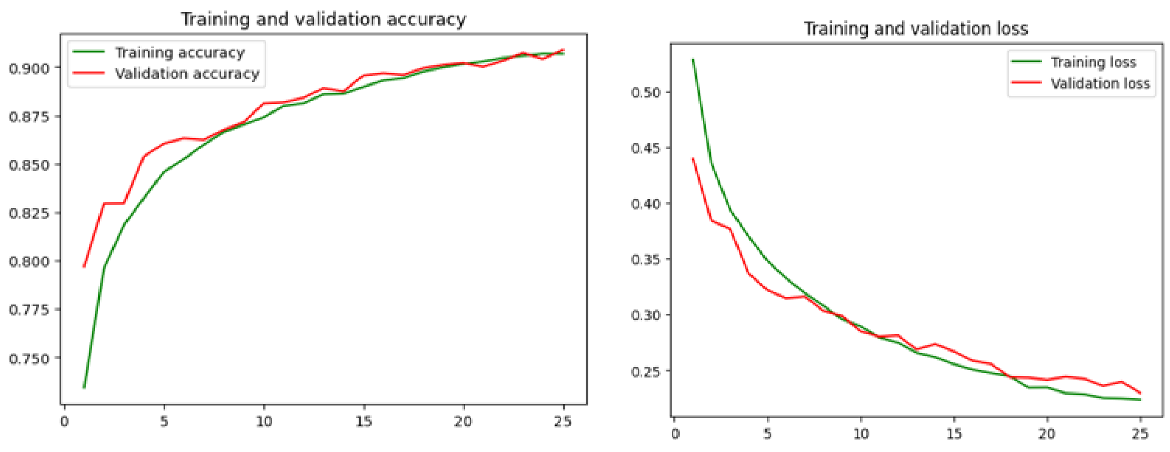


Figure 4.3 Training and validation curve of GRU.

(Left): Chart displaying the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) in relation to the Number of Epochs for the GRU Model.

(Right): Graph illustrating the Training Loss (shown in green) and Validation Loss (shown in red) against the Number of Epochs for the GRU Model.

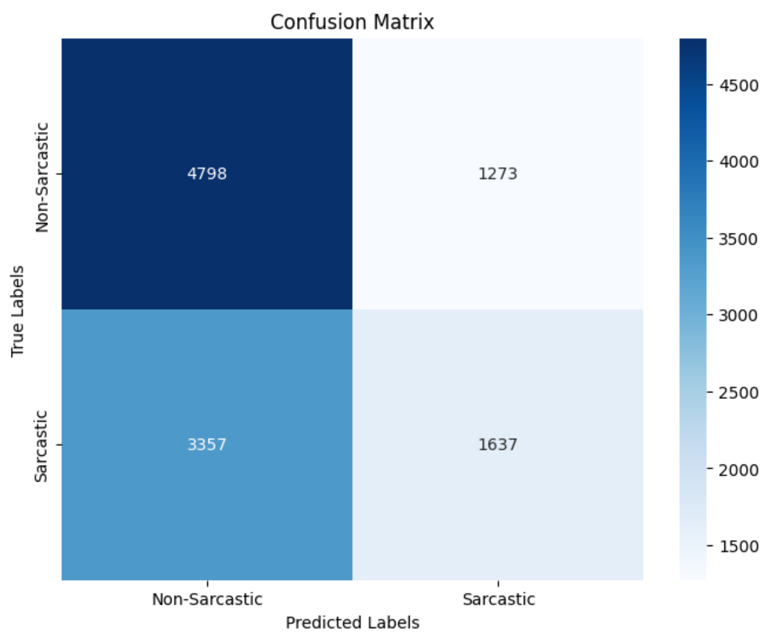
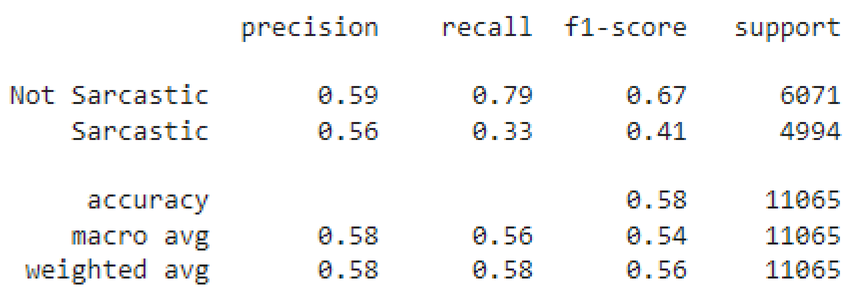


Figure 4.4 Confusion matrix of simpleRNN

The confusion matrix in Figure 4.4 was based on 11065 news headlines for the validation dataset. The results are as follows: 4798 were true positively detected, 1273 were false positively detected, 3357 were false negatively detected, and 1637 were true negatively detected by the SRNN. Figure 4.5 shows the more detailed confusion matrix of the SRNN model.



```
               precision    recall  f1-score   support

Not Sarcastic       0.59      0.79      0.67      6071
    Sarcastic       0.56      0.33      0.41      4994

     accuracy                           0.58     11065
    macro avg       0.58      0.56      0.54     11065
 weighted avg       0.58      0.58      0.56     11065
```

Figure 4.5 Detailed confusion matrix of the SRNN model.

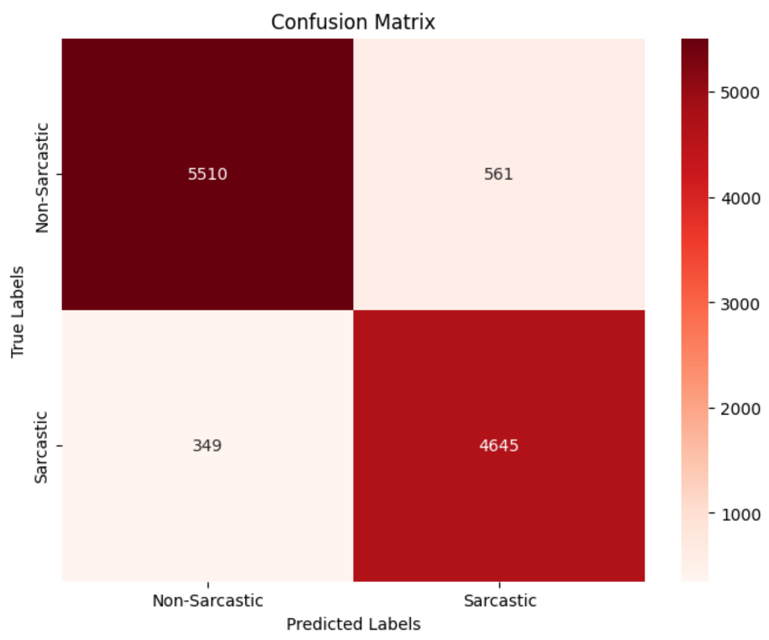
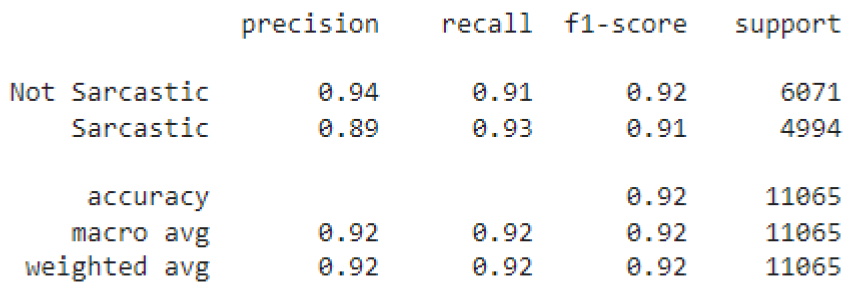


Figure 4.6 Confusion matrix of LSTM.

The confusion matrix in Figure 4.6 was based on 11065 news headlines for the validation dataset. The results are as follows: 5510 were true positively detected, 561 were false positively detected, 349 were false negatively detected, and 4645 were true negatively detected by the LSTM model. Figure 4.7 shows the more detailed confusion matrix of the LSTM model.



```
               precision    recall  f1-score   support

 Not Sarcastic       0.94      0.91      0.92      6071
     Sarcastic       0.89      0.93      0.91      4994

      accuracy                           0.92     11065
     macro avg       0.92      0.92      0.92     11065
  weighted avg       0.92      0.92      0.92     11065
```

Figure 4.7 Detailed confusion matrix of the LSTM model.

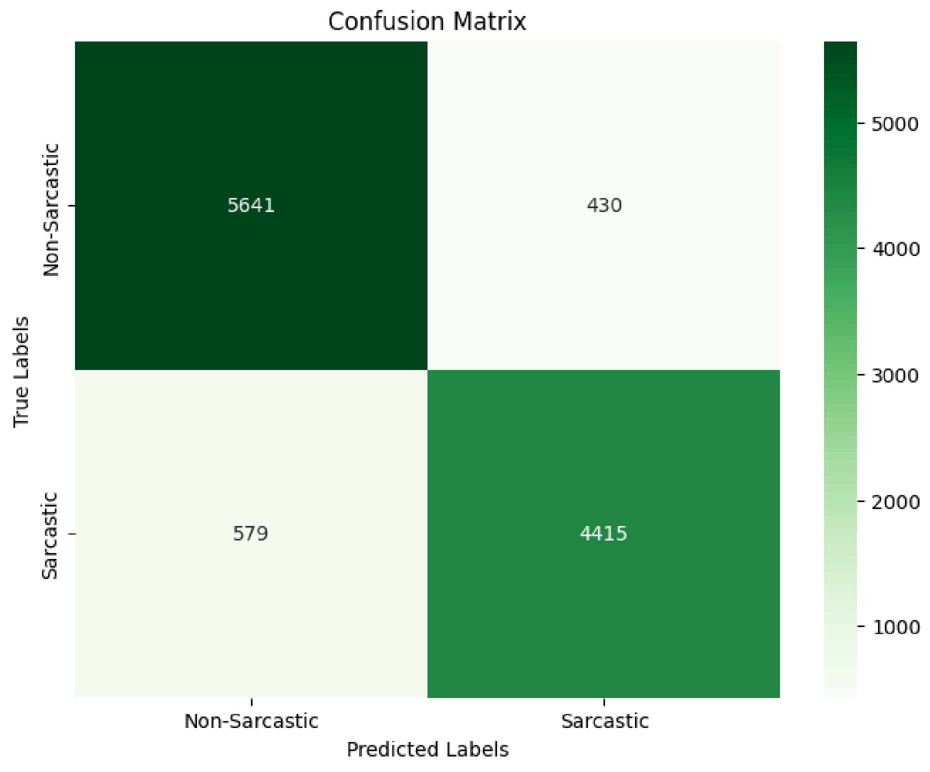
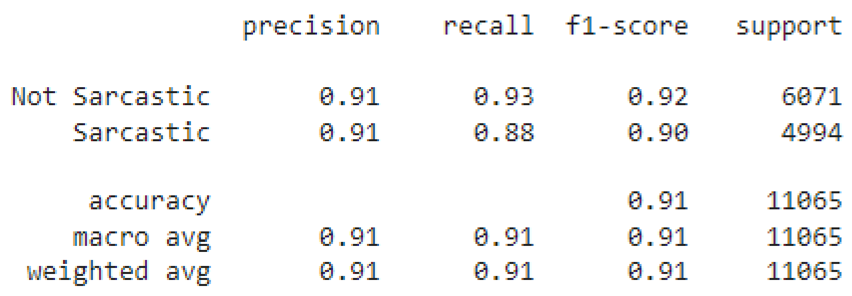


Figure 4.8 Confusion matrix of GRU.

The confusion matrix in Figure 4.8 was based on 11065 news headlines for the validation dataset. The results are as follows: 5641 were true positively detected, 430 were false positively detected, 579 were false negatively detected, and 4415 were true negatively detected by the GRU model. Figure 4.9 shows the more detailed confusion matrix of the GRU model.


```
               precision    recall  f1-score   support

Not Sarcastic       0.91      0.93      0.92      6071
    Sarcastic       0.91      0.88      0.90      4994

     accuracy                           0.91     11065
    macro avg       0.91      0.91      0.91     11065
 weighted avg       0.91      0.91      0.91     11065
```

Figure 4.9 Detailed confusion matrix of the GRU model.

4.3 Results for stacking and weighted average ensemble models

Stacking and weighted averaging are methodologies employed in ensemble learning, a technique where multiple models are integrated to enhance overall performance. Stacking entails training various models and subsequently amalgamating their predictions using another model, often referred to as a metamodel or blender. Conversely, weighted averaging involves assigning distinct weights to the predictions of individual models and then merging them to produce the ultimate prediction. Both stacking and weighted averaging aim to capitalize on the strengths of individual models while mitigating their weaknesses, culminating in a more resilient and precise ensemble model. Figures 4.10, 4.11, 4.12, and 4.13 depict the curves illustrating the training of one, two, three-level stacking, and weighted average models, each trained for 25 epochs with a batch size of 32.

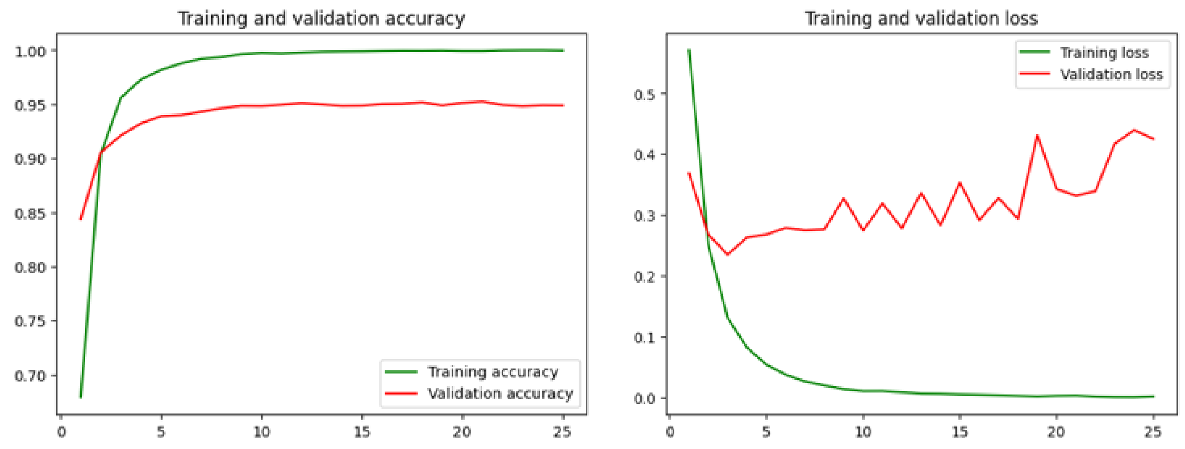


Figure 4.10 Training and validation curves of one-level stacking

(Left): Graph illustrating the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) in correlation with the Number of Epochs for the one-level stacking ensemble Model.

(Right): Chart showing the Training Loss (in green) and Validation Loss (in red) against the Number of Epochs for the one-level stacking ensemble Model.

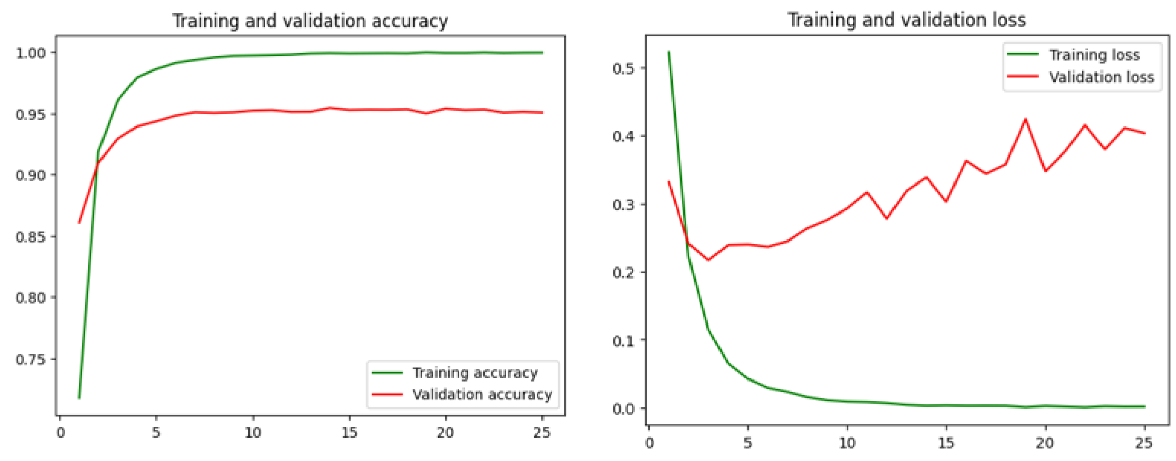


Figure 4.11 Training and validation curves of two-level stacking

(Left): Graph depicting the Training Accuracy (shown in green) and Validation Accuracy (shown in red) in relation to the Number of Epochs for the two-level stacking ensemble Model.

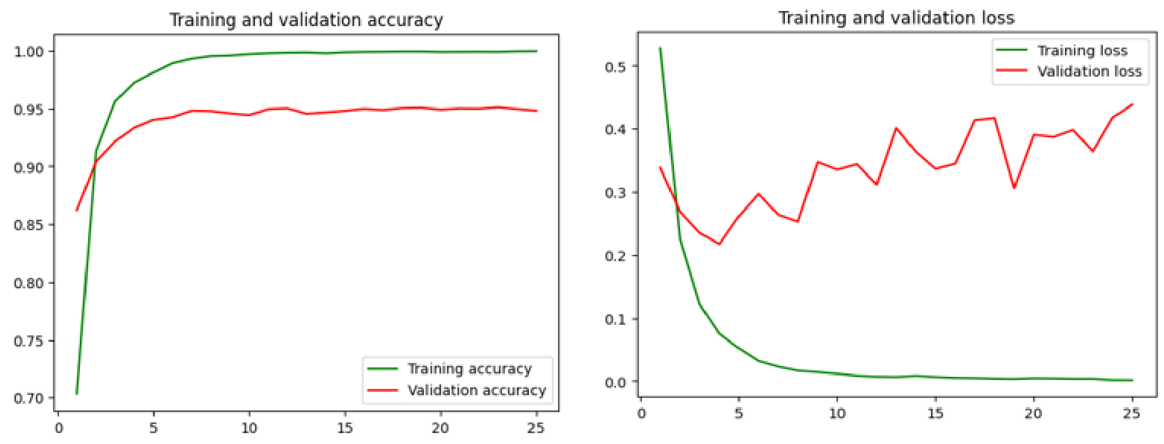


Figure 4.12 Training and validation curves of three-level stacking

(Left): Graph displaying the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) relative to the Number of Epochs for the three-level stacking ensemble Model.

(Right): Chart illustrating the Training Loss (shown in green) and Validation Loss (shown in red) against the Number of Epochs for the three-level stacking ensemble Model.

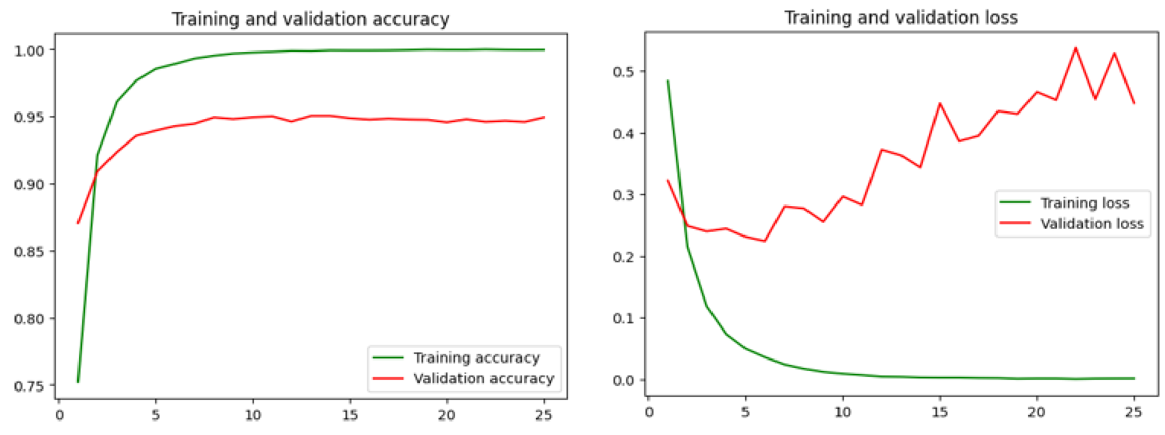


Figure 4.13 Training and validation curves of the weighted average ensemble model

(Left): Graph showing the Training Accuracy (depicted in green) and Validation Accuracy (depicted in red) in relation to the Number of Epochs for the weighted average ensemble Model.

(Right): Chart illustrating the Training Loss (shown in green) and Validation Loss (shown in red) against the Number of Epochs for the weighted average ensemble Model.

A confusion matrix is a table that is used to visualise the performance of a classification model. It shows the number of correct and incorrect predictions made by the model for each class. The rows of the matrix represent the actual classes, and the columns represent the predicted classes. The following figures show the confusion matrices of the ensemble models.

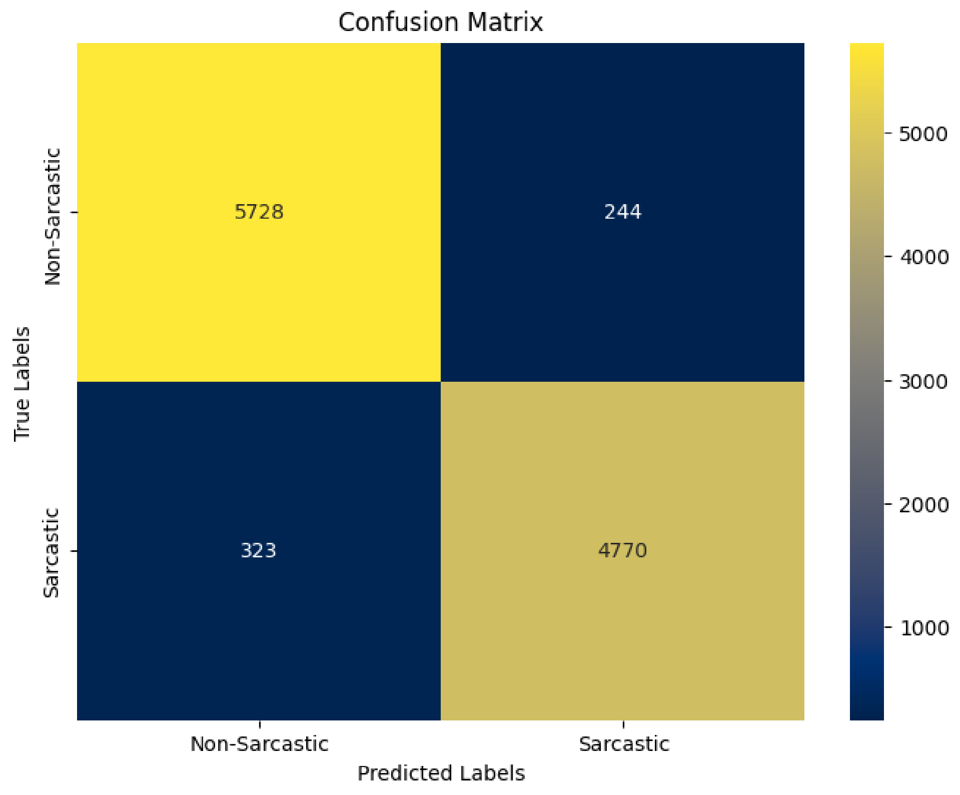
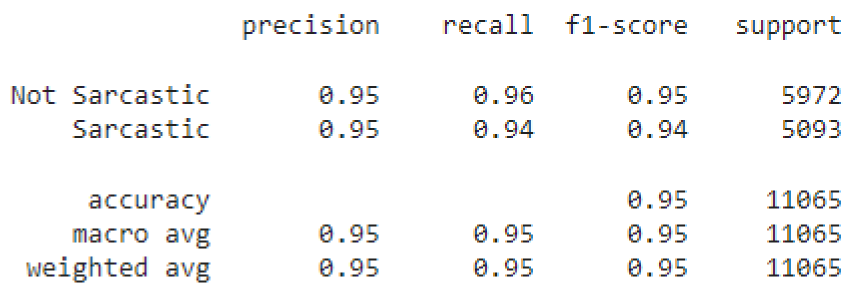


Figure 4.14 confusion matrix of one-level stacking

The confusion matrix in Figure 4.14 was based on 11065 news headlines for the validation dataset. The results are as follows: 5728 were true positively detected, 244 were false positively detected, 323 were false negatively detected, and 4770 were true negatively detected by the one-level stacking model. Figure 4.15 shows the more detailed confusion matrix of the one-level stacking model.



```
               precision    recall  f1-score   support

Not Sarcastic       0.95      0.96      0.95      5972
    Sarcastic       0.95      0.94      0.94      5093

     accuracy                           0.95     11065
    macro avg       0.95      0.95      0.95     11065
 weighted avg       0.95      0.95      0.95     11065
```

Figure 4.15 Detailed confusion matrix of the one-level stacking model.

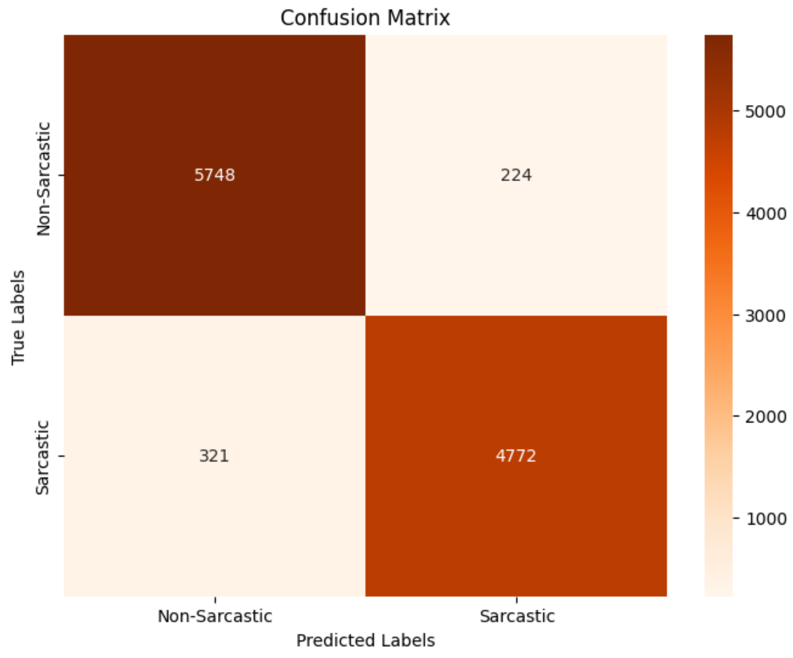


Figure 4.16 Confusion matrix if two-level stacking

The confusion matrix in Figure 4.16 was based on 11065 news headlines for the validation dataset. The results are as follows: 5748 were true positively detected, 224 were false positively detected, 321 were false negatively detected, and 4772 were true negatively detected by the two-level stacking model. Figure 4.17 shows the more detailed confusion matrix of the two-level stacking model.

```
                precision    recall  f1-score   support

 Not Sarcastic       0.95      0.96      0.95      5972
     Sarcastic       0.96      0.94      0.95      5093

      accuracy                           0.95     11065
     macro avg       0.95      0.95      0.95     11065
  weighted avg       0.95      0.95      0.95     11065
```

Figure 4.17 Detailed confusion matrix of the two-level stacking model.

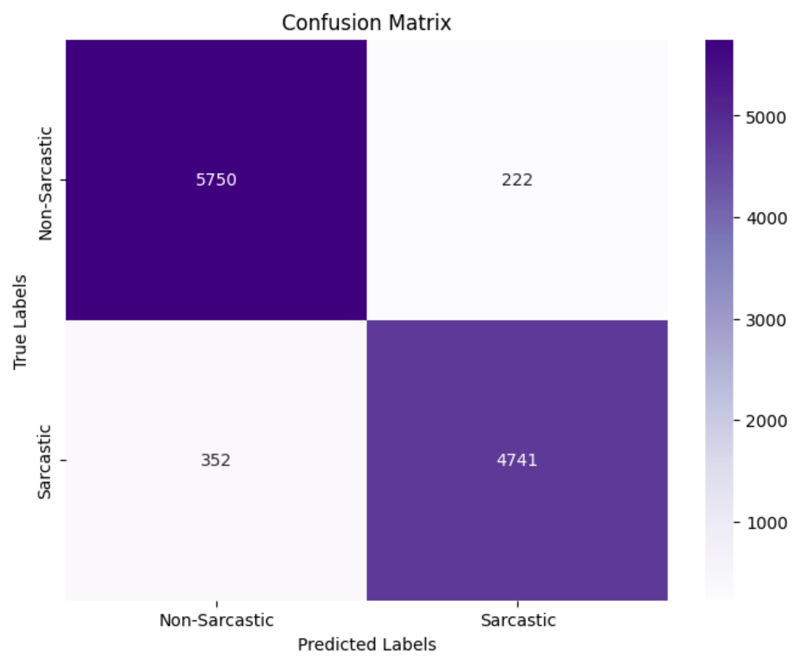
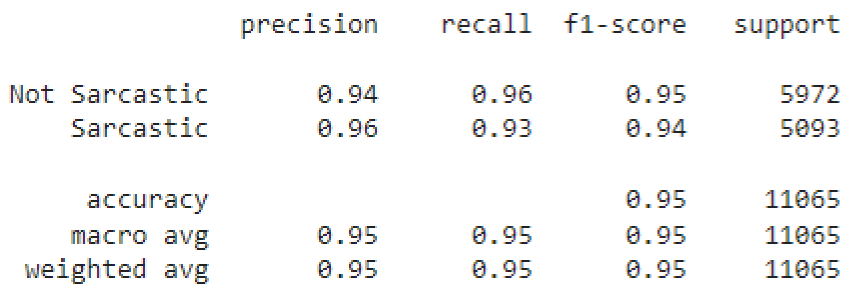


Figure 4.18 confusion matrix of three level stacking

The confusion matrix in Figure 4.18 was based on 11065 news headlines for the validation dataset. The results are as follows: 5748 were true positively detected, 224 were false positively detected, 321 were false negatively detected, and 4772 were true negatively detected by the three-level stacking model. Figure 4.19 shows the more detailed confusion matrix of the three-level stacking model.



```
                precision    recall  f1-score   support

 Not Sarcastic       0.94      0.96      0.95      5972
     Sarcastic       0.96      0.93      0.94      5093

      accuracy                           0.95     11065
     macro avg       0.95      0.95      0.95     11065
  weighted avg       0.95      0.95      0.95     11065
```

Figure 4.19 Detailed confusion matrix of the three-level stacking model.

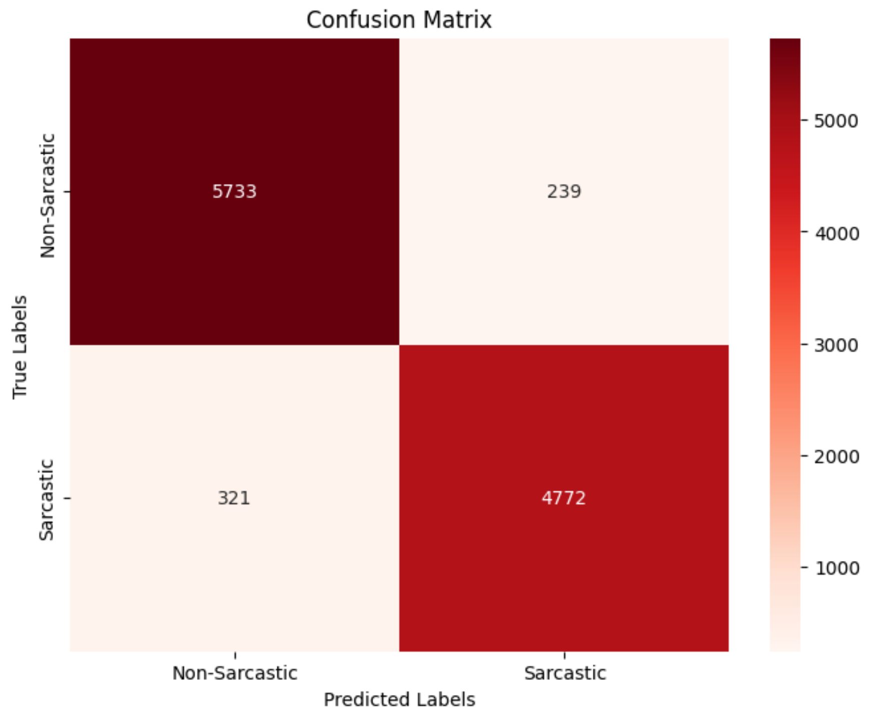
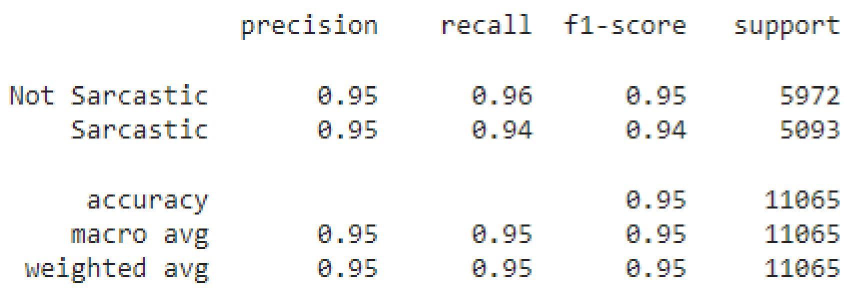


Figure 4.20 confusion matrix if weighted average.

The confusion matrix in Figure 4.20 was based on 11065 news headlines for the validation dataset. The results are as follows: 5748 were true positively detected, 224 were false positively detected, 321 were false negatively detected, and 4772 were true negatively detected by the weighted average model. Figure 4.21 shows the more detailed confusion matrix of the weighted average model.



```
               precision    recall  f1-score   support

 Not Sarcastic      0.95      0.96      0.95      5972
     Sarcastic      0.95      0.94      0.94      5093

      accuracy                          0.95     11065
     macro avg      0.95      0.95      0.95     11065
  weighted avg      0.95      0.95      0.95     11065
```

Figure 4.21 Detailed confusion matrix of the weighted average model.

4.4 Discussion

The graphs in figure 4.1,4.2 and 4.3 depict the training and validation performance of the machine learning model SRNN, LSTM, and GRU respectfully. The graphs have similar trend when it comes to training and validation.

Training and Validation Accuracy

Training Accuracy: begins at a moderate level and increases over time (epochs) as the model learns from the training data. It often increases rapidly at the beginning and eventually plateaus or increases very slowly.

Validation Accuracy: Initially lower than training accuracy, but increases as the model starts to generalize better. However, it may eventually plateau or even slightly decrease, particularly if overfitting is occurring.

Training and Validation Loss

Training Loss: Generally, decreases over time, indicating that the model is becoming better at making predictions on the training data.

Validation Loss: Also tends to decrease, but it may fluctuate more than training loss. This is because the validation set contains data the model hasn't been explicitly trained on.

There appears to be a slight case of overfitting mostly in figure 4.2 Overfitting: A common issue in machine learning, it occurs when the model becomes too specialized to the training data. This leads to excellent performance on the training set but poor performance on new data (indicated by the validation set). A significant gap between the training and validation accuracy curves, especially towards the later epochs, is a sign of overfitting.

The graphs in figure 4.10,4.11, 4.12 and 4.13 depict the training and validation performance of the machine learning model one-level, two-level, tree-level stacking and weighted average respectfully. The graphs have similar trend when it comes to training and validation.

raining and Validation Accuracy

Training Accuracy: Starts relatively high (around 0.75) and rapidly increases to nearly 1.00 within the first 10 epochs. This indicates fast learning on the training set.

Validation Accuracy: Begins lower (around 0.65) and also increases, but at a slower rate than training accuracy. It plateaus around 0.90 after about 15 epochs.

Training and Validation Loss

Training Loss: Decreases rapidly in the first few epochs, then continues to steadily decrease but at a much slower rate. This is expected – initial improvements are often substantial, followed by smaller refinements.

Validation Loss: Generally, decreases, but shows occasional fluctuations. The fluctuations suggest the model sometimes encounters unfamiliar patterns in the validation data, causing temporary increases in error.

Observations

Gap between Accuracy Curves: There's a noticeable gap between training and validation accuracy, widening slightly in later epochs. This suggests a degree of overfitting. The model is becoming highly specialized to the training data, hindering its performance on unseen data in the validation set.

The findings from the study indicate that employing ensemble models holds promise for sarcasm detection. These models merge predictions from multiple individual models, leading to a more precise overall prediction. The effectiveness of ensemble models lies in their ability to capitalize on the strengths of diverse models while offsetting their respective weaknesses.

In this study, the two-level stacking model achieved the best overall performance, with an accuracy and F1-score of both non-sarcastic and sarcastic of 95%. This model combines the predictions of three individual neural network models (SRNN, LSTM, and GRU) using a meta-model. The meta-model is trained to predict the correct class label given the predictions of the individual models. The stacking and weighted average models outperformed individual neural network models by a significant margin. This suggests that ensemble models can be more effective for sarcasm detection than individual models. Figure 4.22 shows the accuracy and F1 scores of both non-sarcastic and sarcastic of all models

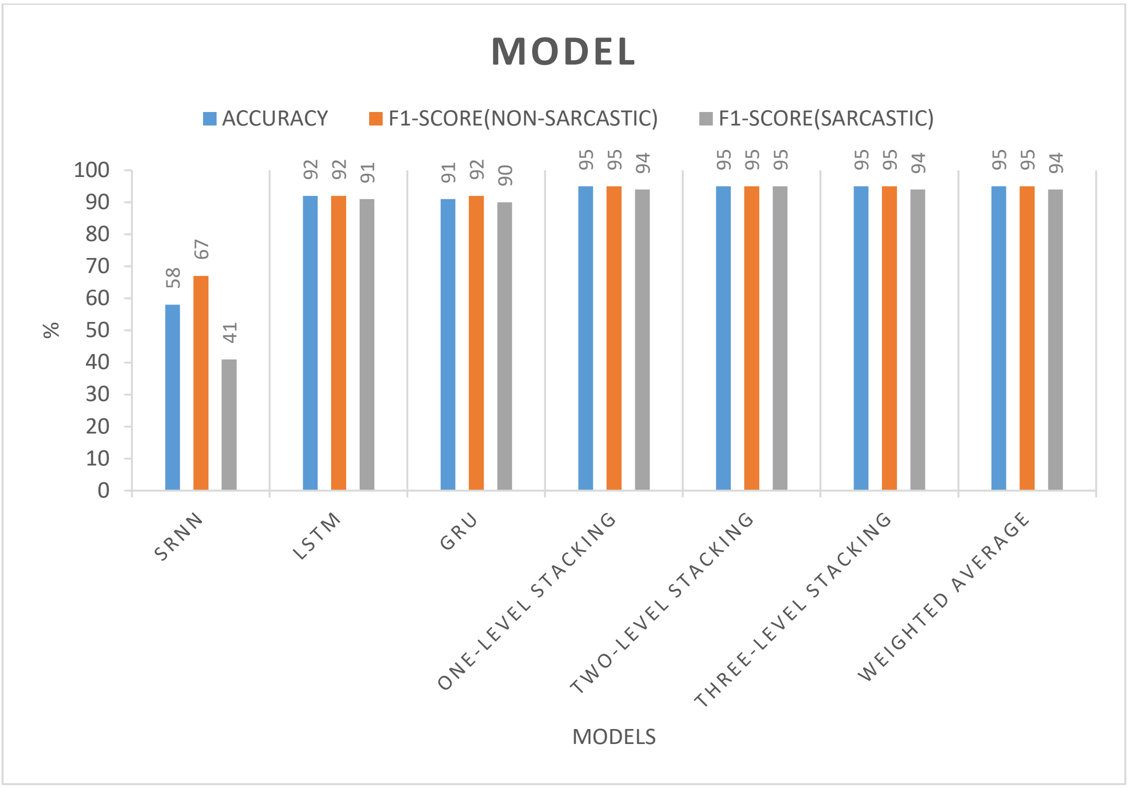


Figure 4.22 accuracy and F1 scores of both non-sarcastic and sarcastic of all models

4.5 Conclusion

The results of this study suggest that ensemble models are a promising approach for sarcasm detection. Ensemble models outperformed individual neural network models by a significant margin, suggesting that they are able to capture more complex patterns in the data and are more resistant to noise.

However, it is important to note that ensemble models can be more complex to train and deploy than simpler models, such as weighted average. Additionally, the performance of different models may vary on different datasets.

4.6 Summary

This session outlines the results of a study on sarcasm detection in textual data through deep learning methods, using traditional models and an ensemble approach. Utilising simpleRNN, LSTM, GRU, and ensemble models like weighted average and stacking, results suggest that ensemble models, notably two-level stacking, surpass individual neural network models.

5 Chapter 5: Conclusions.

5.1 Introduction.

In this section, we discuss the results of our study on using recurrent neural networks to detect sarcasm in political text and speeches. We also provide some suggestions for future research in this area.

5.2 Research summary.

Aim:

The aim of this study is to develop a robust model for detecting sarcasm in political speech using recurrent neural networks (RNNs). RNNs have demonstrated remarkable effectiveness in natural language processing (NLP) tasks, particularly in capturing sequential dependencies within text data. This study aims to harness the power of RNNs to identify the subtle nuances of sarcasm in political discourse.

Objectives:

To achieve the overarching aim, this study pursued the following specific objectives:

I. Implement and compare the performance of SRNN, LSTM, and GRU for sarcasm detection in news headlines.

This objective involved implementing three distinct RNN architectures – SRNN, LSTM, and GRU – and evaluating their performance in detecting sarcasm in political news headlines. The comparison assessed the ability of each model to capture long-range dependencies and contextual cues essential for sarcasm detection.

II. Implement ensemble learning techniques to combine predictions from multiple deep learning frameworks.

This objective explored the potential of ensemble learning to enhance the overall accuracy of sarcasm detection. Ensemble learning techniques, such as one-level, two-level, and three-level stacking models, were implemented to combine predictions from the three RNN architectures. The investigation assessed whether ensemble methods could effectively aggregate the strengths of individual models to improve overall performance.

III. Investigate the effectiveness of combining predictions through a weighted average ensemble model.

This objective focused on a specific ensemble approach – the weighted average ensemble model. This model assigned weights to the predictions from individual models based on their relative performance. By analysing the effectiveness of this approach, the study determined whether weighted averaging could improve sarcasm detection accuracy.

IV. Evaluate model performance using standard metrics such as accuracy, precision, recall, and F1 score.

To assess the effectiveness of the proposed models, standard evaluation metrics – accuracy, precision, recall, and F1 score – were employed. These metrics provided a comprehensive evaluation of the models' ability to correctly identify sarcastic and non-sarcastic utterances in political speech.

In conclusion, this study addressed the critical task of sarcasm detection in political speech through the utilization of recurrent neural networks (RNNs). By implementing and comparing three distinct RNN architectures—SRNN, LSTM, and GRU—the research investigated their efficacy in capturing the nuanced features of sarcasm within political news headlines. The findings of this comparative analysis provided insights into the strengths and weaknesses of each architecture in handling long-range dependencies and contextual cues crucial for sarcasm detection.

Furthermore, the exploration of ensemble learning techniques, including stacking models and the weighted average ensemble model, demonstrated the potential for combining predictions from multiple RNN architectures to enhance overall accuracy. Ensemble methods proved effective in aggregating the complementary strengths of individual models, showcasing their utility in improving sarcasm detection performance.

The study's commitment to evaluating model performance through standard metrics such as accuracy, precision, recall, and F1 score ensured a comprehensive understanding of the proposed models' capabilities. These metrics not only validated the effectiveness of the developed models but also provided a basis for future comparisons and advancements in sarcasm detection methodologies.

Ultimately, this research contributes to the broader field of natural language processing, offering valuable insights into the application of RNNs and ensemble learning techniques for detecting sarcasm in the challenging domain of political discourse. As technology continues to evolve, the outcomes of this study pave the way for further innovations in sentiment analysis and computational understanding of complex linguistic phenomena in political communication.

5.3 Future work.

One possible direction for future research is to investigate the use of different ensemble architectures. For example, it would be interesting to see how the performance of the two-level stacking model compares to other ensemble architectures, such as random forests and gradient-boosting machines.

Another possible direction for future research is to investigate the use of different neural network architectures for sarcasm detection. For example, it would be interesting to see how the performance of the SRNN, LSTM, and GRU models compares with more recent neural network architectures such as transformers.

Finally, it would be interesting to evaluate the performance of different models on a variety of different datasets. This would help to assess the generalisability of the results and identify the best models for different applications.

5.4 Recommendations

I. Consider using a pre-trained language model (PLM) such as BERT or RoBERTa for sarcasm detection. PLMs have been shown to be effective for a variety of natural language processing tasks, including sarcasm detection.

II. Use a variety of datasets to train the models. This will help the models to learn to generalise to different types of text and different contexts.

III. Consider not cooperating with native languages in the study.

5.5 Conclusion

The aim of the study was to detect sarcasm on texts using RNNs, the models were trained using labelled news headlines; the headlines are labelled as sarcastic and non-sarcastic, the Glove was used for word embedding, and the results shows that two-level stacking sing SRNN, LSTM, and GRU as base models achieved the highest accuracy of 94%.

References

[1] R. Akula, “Interpretable Multi-Head Self-Attention Architecture for,” 2021.

[2] Y. Y. Tan, C. O. Chow, J. Kanesan, J. H. Chuah, and Y. L. Lim, “Sentiment Analysis and Sarcasm Detection using Deep Multi-Task Learning,” *Wirel. Pers. Commun.*, vol. 129, no. 3, pp. 2213–2237, 2023, doi: 10.1007/s11277-023-10235-4.

[3] D. M. E. D. M. Hussein, “A survey on sentiment analysis challenges,” *J. King Saud Univ. - Eng. Sci.*, vol. 30, no. 4, pp. 330–338, 2018, doi: 10.1016/j.jksues.2016.04.002.

[4] R. Faria de Azevedo *et al.*, *Screening of email box in Portuguese with SVM at Banco do Brasil*, vol. 12037 LNAI. 2020. doi: 10.1007/978-3-030-41505-1_15.

[5] P. Goel, R. Jain, A. Nayyar, S. Singhal, and M. Srivastava, “Sarcasm detection using deep learning and ensemble learning,” *file///C/Users/201718044/Downloads/Identification sarcasm using word Embed. hyperparameters tuning..pdfMultimedia Tools Appl.*, vol. 81, no. 30, pp. 43229–43252, 2022, doi: 10.1007/s11042-022-12930-z.

[6] D. Šandor and M. Bagić Babac, “Sarcasm detection in online comments using machine learning,” *Inf. Discov. Deliv.*, no. June, 2023, doi: 10.1108/IDD-01-2023-0002.

[7] A. Kumar and G. Garg, “Empirical study of shallow and deep learning models for sarcasm detection using context in benchmark datasets,” *J. Ambient Intell. Humaniz. Comput.*, vol. 14, no. 5, pp. 5327–5342, 2023, doi: 10.1007/s12652-019-01419-7.

[8] E. Kušen and M. Strembeck, “Politics, sentiments, and misinformation: An analysis of the Twitter discussion on the 2016 Austrian Presidential Elections,” *Online Soc. Networks Media*, vol. 5, pp. 37–50, 2018, doi: 10.1016/j.osnem.2017.12.002.

[9] Elisabeth, “Sarcasm, Pretense, and The Semantics/ Pragmatics Distinction ∗,” *Nous*, pp. 1–7, 2011.

[10] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, “A system for real-time twitter sentiment analysis of 2012 U.S. presidential election cycle,” *Proc. Annu. Meet. Assoc. Comput. Linguist.*, no. July, pp. 115–120, 2012.

[11] D. Vinoth and P. Prabhavathy, “An intelligent machine learning-based sarcasm detection and classification model on social networks,” *J. Supercomput.*, vol. 78, no. 8, pp. 10575–10594, 2022, doi: 10.1007/s11227-022-04312-x.

[12] S. Skalicky and S. A. Crossley, “Linguistic features of sarcasm and metaphor production quality,” *Proc. Work. Fig. Lang. Process. Fig-Lang 2018 2018 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. NAACL-HTL 2018*, no. Colston 2017, pp. 7–16, 2018, doi: 10.18653/v1/w18-0902.

[13] A. Joshi, V. Sharma, and P. Bhattacharyya, “Harnessing Context Incongruity for Sarcasm Detection,” no. 2003, pp. 757–762, 2015.

[14] P. Carvalho and M. J. Silva, “Clues for Detecting Irony in User-Generated Contents : Oh ...!! It ’ s “ so easy " Clues for Detecting Irony in User-Generated Contents : Oh ...!! It ’ s “ so easy " ; -),” no. June 2014, 2009, doi: 10.1145/1651461.1651471.

[15] A. Reyes, P. Rosso, and T. Veale, “A multidimensional approach for detecting irony in Twitter,” *Lang. Resour. Eval.*, vol. 47, no. 1, pp. 239–268, 2013, doi: 10.1007/s10579-012-9196-x.

[16] C. van Hee, E. Lefever, and V. Hoste, “SemEval-2018 Task 3: Irony Detection in English Tweets,” *NAACL HLT 2018 - Int. Work. Semant. Eval. SemEval 2018 - Proc. 12th Work.*, pp. 39–50, 2018.

[17] D. Ghosh, W. Guo, and S. Muresan, “Sarcastic or Not : Word Embeddings to Predict the Literal or Sarcastic Meaning of Words,” no. September, pp. 1003–1012, 2015.

[18] J. Aboobaker and E. Ilavarasan, “A survey on Sarcasm detection approaches,” *Indian J. Comput. Sci. Eng.*, vol. 11, no. 6, pp. 751–771, 2020, doi: 10.21817/indjcse/2020/v11i6/201106048.

[19] J. P. Pinto, “Twitter Sentiment Analysis: A Political View,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 1, pp. 723–729, 2020, doi: 10.30534/ijatcse/2020/103912020.

[20] L. Kurniasari and A. Setyanto, “Sentiment Analysis using Recurrent Neural Network,” *J. Phys. Conf. Ser.*, vol. 1471, no. 1, 2020, doi: 10.1088/1742-6596/1471/1/012018.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” pp. 1–9, 2014, [Online]. Available: http://arxiv.org/abs/1412.3555

[22] A. Ghosh and T. Veale, “Fracking sarcasm using neural network,” *Proc. 7th Work. Comput. Approaches to Subj. Sentim. Soc. Media Anal. WASSA 2016 2016 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol.*,

no. April, pp. 161–169, 2016, doi: 10.18653/v1/w16-0425.

[23] A. Joshi, V. Tripathi, K. Patel, P. Bhattacharyya, and M. Carman, "Are word embedding-based features useful for sarcasm detection?," *EMNLP 2016 - Conf. Empir. Methods Nat. Lang. Process. Proc.*, no. 2013, pp. 1006–1011, 2016, doi: 10.18653/v1/d16-1104.

[24] M. Khodak, N. Saunshi, and K. Vodrahalli, "A large self-annotated corpus for sarcasm," *Lr. 2018 - 11th Int. Conf. Lang. Resour. Eval.*, pp. 641–646, 2019.

[25] R. A. Potamias, G. Siolas, and A. G. Stafylopatis, "A transformer-based approach to irony and sarcasm detection," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17309–17320, 2020, doi: 10.1007/s00521-020-05102-3.

[26] A. Reyes, P. Rosso, and D. Buscaldi, "From humor recognition to irony detection: The figurative language of social media," *Data Knowl. Eng.*, vol. 74, pp. 1–12, 2012, doi: 10.1016/j.datak.2012.02.005.

[27] J. Lucariello, "Situational Irony : A Concept of Events Gone Awry," vol. 123, no. 2, pp. 129–145, 1994.

[28] F. Barbieri and H. Saggion, "Modelling Irony in Twitter," pp. 56–64, 2014.

[29] K. Buschmeier, P. Cimiano, and R. Klinger, "An Impact Analysis of Features in a Classification Approach to Irony Detection in Product Reviews," pp. 42–49, 2014.

[30] J. Tepperman, D. Traum, and S. Narayanan, "Yeah right : Sarcasm recognition for spoken dialogue systems," no. May 2014, 2006, doi: 10.21437/Interspeech.2006-507.

[31] D. Davidov and O. Tsur, "Semi-Supervised Recognition of Sarcastic Sentences in Twitter and Amazon," no. July, pp. 107–116, 2010.

[32] R. González-ibáñez and N. Wacholder, "Identifying Sarcasm in Twitter : A Closer Look," no. 2010, pp. 581–586, 2011.

[33] F. H. Calderon and Y. Chen, "Emotion Combination in Social Media Comments as Features for Sarcasm Detection".

[34] E. Savini and C. Caragea, "Intermediate-Task Transfer Learning with BERT for

Sarcasm Detection," 2022.

[35] S. Amir, B. C. Wallace, P. Carvalho, and J. Silva, "Modelling Context with User Embeddings for Sarcasm Detection in Social Media," pp. 167–177, 2016.

[36] S. Oraby, V. Harrison, L. Reed, E. Hernandez, E. Riloff, and M. Walker, "Creating and Characterizing a Diverse Corpus of Sarcasm in Dialogue," no. September, pp. 31–41, 2016.

[37] E. Riloff, A. Qadir, P. Surve, L. De Silva, N. Gilbert, and R. Huang, "Sarcasm as Contrast between a Positive Sentiment and Negative Situation," no. October, pp. 704–714, 2013.

[38] H. M. K. Kumar and B. S. Harish, "ScienceDirect Sarcasm classification : A novel approach by using Content Based Sarcasm classification : A novel approach by using Content Based Sarcasm classification : A novel approach by using Content Based Feature Selection Method Feature Selection Met," *Procedia Comput. Sci.*, vol. 143, pp. 378–386, 2018, doi: 10.1016/j.procs.2018.10.409.

[39] B. Lala *et al.*, "The Challenge of Multiple Thermal Comfort Prediction Models: Is TSV Enough?," *Buildings*, vol. 13, no. 4, pp. 1–22, 2023, doi: 10.3390/buildings13040890.

[40] D. Ghosh, A. Richard, and F. Smaranda, "The Role of Conversation Context for Sarcasm Detection in Online Interactions," no. August, pp. 186–196, 2017.

[41] Y. Du, T. Li, M. S. Pathan, H. K. Teklehaimanot, and Z. Yang, "An Effective Sarcasm Detection Approach Based on Sentimental Context and Individual Expression Habits," *Cognit. Comput.*, vol. 14, no. 1, pp. 78–90, 2022, doi: 10.1007/s12559-021-09832-x.

[42] B. C. Wallace, D. K. Choe, L. Kertz, and E. Charniak, "Humans Require Context to Infer Ironic Intent (so Computers Probably do , too)," pp. 512–516, 2014.

[43] C. I. Eke, A. A. Norman, and L. Shuib, "Context-Based Feature Technique for Sarcasm Identification in Benchmark Datasets Using Deep Learning and BERT Model," *IEEE Access*, vol. 9, pp. 48501–48518, 2021, doi:

10.1109/ACCESS.2021.3068323.

[44] M. S. Razali, A. A. Halin, L. Ye, S. Doraisamy, and N. M. Norowi, “Sarcasm Detection Using Deep Learning with Contextual Features,” *IEEE Access*, vol. 9, pp. 68609–68618, 2021, doi: 10.1109/ACCESS.2021.3076789.

[45] Z. Wang, Z. Wu, R. Wang, and Y. Ren, “Twitter Sarcasm Detection Exploiting a,” vol. 9419, pp. 332–336, 2015, doi: 10.1007/978-3-319-26190-4.

[46] N. Jaiswal, “Neural sarcasm detection using conversation context,” *Proc. Annu. Meet. Assoc. Comput. Linguist.*, pp. 77–82, 2020, doi: 10.18653/v1/P17.

[47] A. Baruah, K. A. Das, F. A. Barbhuiya, and K. Dey, “Context-aware sarcasm detection using BERT,” *Proc. Annu. Meet. Assoc. Comput. Linguist.*, pp. 83–87, 2020, doi: 10.18653/v1/P17.

[48] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[49] S. Mohammadi, S. G. Majelan, and S. B. Shokouhi, “Ensembles of deep neural networks for action recognition in still images,” *2019 9th Int. Conf. Comput. Knowl. Eng. ICCKE 2019*, no. Iccke, pp. 315–318, 2019, doi: 10.1109/ICCKE48569.2019.8965014.

[50] U. M. D. E. C. D. E. Los, “No 主観的健康感を中心とした在宅高齢者における健康関連指標に関する共分散構造分析Title”.

[51] R. Ross, *Cape of torments: Slavery and resistance in South Africa*. 2022. doi: 10.4324/9781003307426.

[52] D. LOWRY, *THE SOUTH AFRICAN WAR RE-SURVEYED The South African War, 1899–1902 . By B ILL N ASSON . London: Arnold, 1999. Pp. xvi+304. £45 (ISBN 0-340-74154-6); paperback, £16.99 (ISBN 0-340-61427-7).* , vol. 44, no. 3. 2003. doi: 10.1017/s002185370332866x.

[53] B. New and Y. London, *Long walk to freedom: the autobiography of Nelson Mandela*, vol. 32, no. 08. 1995. doi: 10.5860/choice.32-4642.

[54] U. M. D. E. C. D. E. Los, *No 主観的健康感を中心とした在宅高齢者における*

健康関連指標に関する共分散構造分析Title.

[55] J. Seekings and N. Nattrass, “Class, race, and inequality in South Africa,” *Class, Race, Inequal. South Africa*, pp. 1–446, 2005, doi: 10.2307/20032025.

[56] C. D. M. J. Pennington, R. Socher, “GloVe: Global Vectors forWord Representation,” *Proc. 2014 Conf. Empir. Methods Nat. Lang. Process. (EMNLP),* vol. 19, no. 5, pp. 1532–1543, 2014.

[57] D. Jain, A. Kumar, and G. Garg, “Sarcasm detection in mash-up language using soft-attention based bi-directional LSTM and feature-rich CNN,” *Appl. Soft Comput. J.*, vol. 91, p. 106198, 2020, doi: 10.1016/j.asoc.2020.106198.

[58] P. Mehndiratta and D. Soni, “Identification of sarcasm using word embeddings and hyperparameters tuning,” *J. Discret. Math. Sci. Cryptogr.*, vol. 22, no. 4, pp. 465–489, 2019, doi: 10.1080/09720529.2019.1637152.

[59] A. Onan, “Topic-Enriched Word Embeddings for Sarcasm Identification,” *Adv. Intell. Syst. Comput.*, vol. 984, no. April, pp. 293–304, 2019, doi: 10.1007/978-3-030-19807-7_29.

[60] J. Patro, S. Bansal, and A. Mukherjee, “A deep-learning framework to detect sarcasm targets,” *EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf.*, pp. 6336–6342, 2019, doi: 10.18653/v1/d19-1663.

[61] N. Babanejad, H. Davoudi, A. An, and M. Papagelis, “Affective and Contextual Embedding for Sarcasm Detection,” *COLING 2020 - 28th Int. Conf. Comput. Linguist. Proc. Conf.*, pp. 225–243, 2020, doi: 10.18653/v1/2020.coling-main.20.

[62] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing [Review Article],” *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, 2018, doi: 10.1109/MCI.2018.2840738.

[63] 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.

[64] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi:

10.1162/neco.1997.9.8.1735.

[65] N. Majumder, S. Poria, H. Peng, N. Chhaya, E. Cambria, and A. Gelbukh, “Sentiment and Sarcasm Classification with Multitask Learning,” *IEEE Intell. Syst.*, vol. 34, no. 3, pp. 38–43, 2019, doi: 10.1109/MIS.2019.2904691.

[66] J. Lemmens, B. Burtenshaw, E. Lotfi, I. Markov, and W. Daelemans, “Sarcasm detection using an ensemble approach,” *Proc. Annu. Meet. Assoc. Comput. Linguist.*, pp. 264–269, 2020, doi: 10.18653/v1/P17.

[67] F. Li, G. Li, and S. W. Hwang, “Foreword,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8485 LNCS, no. June, 2014, doi: 10.1007/978-3-319-08010-9.

[68] E. Fersini, F. A. Pozzi, and E. Messina, “Detecting irony and sarcasm in microblogs: The role of expressive signals and ensemble classifiers,” *Proc. 2015 IEEE Int. Conf. Data Sci. Adv. Anal. DSAA 2015*, no. October, 2015, doi: 10.1109/DSAA.2015.7344888.