Machine Learning and Mathematics for Fake News Detection

Mayur Kripalani

Student ID: 220335959

October 2, 2023

Abstract

The proliferation of fake news in the digital era poses a serious threat to individuals and societies by undermining trust in institutions and distorting the truth. This dissertation addresses this growing concern through a dual-analytical framework that combines Content and Context Analysis. The Content Analysis utilizes natural language processing techniques, including the Transformer-based BERT model, to scrutinize and classify news headlines. The Context Analysis focuses on public reactions to news, employing bot detection algorithms using RandomForest, sentiment analysis with Naive Bayes, and predictive modeling using XGBoost. Our research aims to achieve a holistic understanding of the fake news ecosystem. It tackles several key questions: the effectiveness of the integrated model in discerning authentic from fake news, the role of public sentiment in the propagation of fake news, and characteristics of bot activities in this context. The methodology comprises rigorous data preprocessing and feature engineering, followed by model training and evaluation.

Acknowledgements

I want to extend my heartfelt thanks to my dedicated mentors, Professor Sergei K. Turitsyn, Dr. Pedro Freire and Mrs. Nataliia Manuilovich. Their unwavering support and invaluable guidance played an instrumental role in bringing this project to fruition.

List of Figures

List of Tables

Contents

Machine Learning and Mathematics for Fake News Detection

Mayur Kripalani
ID: 220335959

October 2, 2023

1 Introduction

The advent of the digital age has revolutionized the dissemination of information, dramatically altering the way society interacts, exchanges ideas, and consumes content. While this digital transformation has brought about unprecedented accessibility to information, it has also given rise to a darker counterpart: the widespread dissemination of misinformation, commonly referred to as fake news. This phenomenon has evolved from a simple buzzword into a pressing societal concern, infiltrating public discourse, undermining institutional trust, and even affecting the outcomes of democratic elections. With its all-encompassing reach and multifaceted origins, fake news presents a labyrinthine problem that demands a complex, multi-disciplinary solution, one that transcends traditional methods and approaches.

At the fulcrum of this dissertation lies the intricate and pressing issue of fake news. Despite being the subject of intense scrutiny in academic research, policy discussions, and public debates, the challenge posed by fake news remains largely unresolved. This is primarily due to the complexities that are inherent to the phenomenon and the ever-changing digital landscape where it thrives. Against this backdrop, this research aspires not merely to add another layer of academic discourse but to pioneer a novel approach by employing an integrated framework of Content and Context Analysis. This approach leverages the advancements in machine learning algorithms, natural language processing techniques, and data analytics to offer a multi-dimensional perspective on the problem.

Fake news is an amalgam of deceptive content and misleading context, woven together in a manner that makes it difficult to dissect using traditional analytical methods. Textual content and metadata, which have been the focus of most existing detection systems, offer only a fragmented understanding of the issue. They often overlook the complex socio-cultural and psychological factors that contribute to the creation, dissemination, and consumption of fake news. By conducting an in-depth analysis of both the Content and the Context that surround fake news, this dissertation aims to bridge this analytical gap. It endeavours to provide a comprehensive, nuanced framework that transcends surface-level scrutiny, thereby contributing to a more holistic understanding of fake news.

The primary objectives of this research project are twofold. The first objective is to curate, preprocess, and analyze a wide array of news articles, scrutinizing their textual and semantic characteristics through advanced Content Analysis algorithms. The second objective is to delve into the digital ecosystems where these articles circulate, employing machine learning models

that are adept at Context Analysis. These objectives are guided by several overarching research questions: How proficient are our integrated models at identifying the subtle and not-so-subtle traits that characterize fake news? What novel insights and patterns can we unearth through this harmonized approach of Content and Context Analysis? What are the implications of these findings on policy formulation, public awareness, and future research?

This dissertation is meticulously structured to guide the reader through a coherent, step-by-step journey that traverses the multiple layers of the research problem. Following this introductory section, which aims to set both the theoretical and practical stage, the Literature Review section will delve into previous studies. The Methodology section will provide an exhaustive account of the techniques employed in both Content and Context Analysis, detailing their mathematical foundations, algorithmic intricacies, and implementation challenges. Subsequent sections on Results and Analysis will systematically unveil the core findings, weaving them into an interpretative narrative that brings fresh perspectives to the complex issue of fake news detection. Finally, the Conclusion will synthesize these insights, reflecting on their broader societal implications, and charting the course for potential future research endeavors.

Lastly, the fight against fake news is not just a technological battle but a societal one. It requires a multidisciplinary approach that combines machine learning, social sciences, and policy interventions. This dissertation, through its comprehensive methodology and nuanced analyses, aims to try to serve technologically to this multidimensional challenge, thereby fostering a more informed and less polarized society.

2 Literature Review

In the research paper "A Comprehensive Benchmark for Fake News Detection" by Antonio Galli, Elio Masciari, Vincenzo Moscato, and Giancarlo Sperlì, the authors provide a benchmark framework to analyze and discuss various machine learning and deep learning techniques for fake news detection. They conducted experiments on well-known and widely used real-world datasets to evaluate the performance of different approaches in terms of accuracy and efficiency. The authors used three datasets for their experiments: Liar, FakeNewsNet, and PHEME. They compared several machine learning models, such as Random Forest, Decision Tree, Support Vector Machines (SVC), and Logistic Regression, as well as deep learning models like Convolutional Neural Networks (CNN) and BERT (Bidirectional Encoder Representations from Transformers).

The results showed that Logistic Regression performed the best among the traditional machine learning classifiers in terms of efficiency and effectiveness. On the other hand, BERT outperformed other deep learning models due to its ability to perform word-level embedding based on the context of the words in a sentence, with the accuracy of 61.9%. Overall, the work by Galli et al. provides valuable insights into the performance of various machine learning and deep learning techniques for fake news detection. It highlights the importance of selecting appropriate models and features for achieving accurate and efficient fake news detection. The authors made it quite understandable that the accuracy by selecting a single feature could also provide good accuracy but the results demonstrate the potential of combining content with another feature makes for improved fake news detection performance[1].

The research paper "AugFake-BERT: Handling Imbalance through Augmentation of Fake News Using BERT to Enhance the Performance of Fake News Classification" by Keya et al. ad-

dresses the issue of imbalanced datasets in fake news classification tasks. The authors propose a text augmentation technique using the Bidirectional Encoder Representation of Transformers (BERT) language model to generate synthetic fake data. The AugFake-BERT model is trained with the augmented dataset and evaluated against twelve different state-of-the-art models, achieving an accuracy of 92.45% and outperforming existing models.

The authors used the "BanFakeNews" dataset, which contains approximately 50,000 Bengali news items. To handle the imbalance issue, they employed a transfer learning-based approach using a pre-trained multilingual BERT model to generate synthetic fake news data. The AugFake-BERT model consists of two main stages: data augmentation and classification. In the data augmentation stage, the pre-trained multilingual BERT model is used to generate synthetic fake news data through insertion and substitution of words. In the classification stage, the authors fine-tune a BERT model with the augmented dataset to extract contextual features and perform classification using a softmax layer. The study paved way to understand how fine-tuning BERT could provide good results[2].

The research paper "Fake news detection based on a hybrid BERT and LightGBM models" by Ehab Essa, Karima Omar, and Ali Alqahtani proposes a novel hybrid fake news detection system that combines a BERT-based (bidirectional encoder representations from transformers) model with a light gradient boosting machine (LightGBM) model. The authors used three real-world fake news datasets to validate the performance of their proposed method compared to other methods. The datasets used in the study are: ISOT dataset, TI-CNN dataset, and Fake News Corpus (FNC) dataset.

The proposed method is evaluated to detect fake news based on the headline-only or full text of the news content. The results show the superiority of the proposed method for fake news detection compared to many state-of-the-art methods. Overall, the proposed hybrid method outperforms different combinations of word embedding techniques and classification approaches, as well as current state-of-the-art methods on different real-world fake news datasets. The proposed method achieves superior performance with an accuracy of 99.88% on the ISOT dataset, 96.94% on the TI-CNN dataset, and 99.06% on the FNC dataset. So, the research paper presents a novel approach to fake news detection by combining the strengths of BERT and LightGBM models. The proposed method demonstrates better performance compared to other methods on various real-world fake news datasets, making it a promising approach for detecting fake news in the digital era, which makes it insightful[3].

The research paper "Improving the Performance of Query Processing and Pre-fetching from Large Data Set" by Arisha Farha and Afsaruddin presents an extensive review of machine learning techniques for detecting fake news. The authors emphasize the importance of addressing fake news and its potential consequences, such as influencing elections and spreading harmful misinformation. They propose a model based on the decision tree algorithm, which achieves an accuracy of up to 97% in some cases.

The authors discuss various aspects of fake news analysis using machine learning, including data collection, pre-processing, feature extraction, model training, evaluation, and deployment. They mention several publicly accessible datasets, such as the LIAR dataset, which contains labeled examples of true and false statements made by politicians. Various machine learning models are discussed, including decision trees, support vector machines, neural networks, logistic regression, and Naive Bayes. The paper provides valuable insights into the current state of fake news detection and how to overcome challenges in refining the accuracy[4].

The research paper "Fake news detection in social media based on sentiment analysis using classifier techniques" by Sarita V Balshetwar, Abilash RS, and Dani Jermisha R proposes a new solution for detecting fake news by incorporating sentiment analysis as a key feature to improve accuracy. They used two datasets, ISOT and LIAR, for their study. The authors developed a lexicon-based scoring algorithm for sentiment analysis and proposed a multiple imputation strategy using the Multiple Imputation Chain Equation (MICE) to handle multivariate missing variables in social media or news data. They used Term Frequency and Inverse Document Frequency (TF-IDF) to extract effective features from the text and classified the correlation of missing data variables and useful data features based on Naïve Bayes, passive-aggressive, and Deep Neural Network (DNN) classifiers.

The research found that the proposed method achieved an accuracy of 99.8% for detecting fake news, evaluating various statements such as barely true, half true, true, mostly true, and false from the dataset. The performance of the proposed method was compared with existing methods, and it was found to be more efficient. This study helped in understanding the combination of Naïve Bayes coupled with Deep Neural Network Classifiers could provide excellent results[5].

The research paper titled "Sentiment Analysis for Fake News Detection" by Alonso MA, Vilares D, Gómez-Rodríguez C, Vilares J. explores the application of sentiment analysis techniques in detecting fake news. Various models and techniques are discussed in the paper, including machine learning techniques, lexical and syntactic processing, and convolutional neural networks. These techniques have been employed to analyze and classify sentiments in text data, which can be applied to different types of datasets, such as those related to fake news and hostility detection, to improve the accuracy and effectiveness of fake news detection systems.

The datasets used in the paper include the COVID-19 Fake News Dataset, the Hostility Detection Dataset in Hindi. The paper provides valuable insight by demonstrating the value of sentiment analysis as a tool for detecting fake news by employing various techniques such as machine learning, lexical and syntactic processing, and deep learning approaches like convolutional neural networks, the sentiments could be captured well in text data[6].

The research paper titled "Fake News Detection Using Naïve Bayes and Long Short Term Memory algorithms" is authored by Sarra Senhadji and Rania Azad M. San Ahmed and was published in the IAES International Journal of Artificial Intelligence (IJ-AI) in June 2022. The authors aimed to evaluate the performance of Naïve Bayes (NB) and Long Short-Term Memory (LSTM) classifiers in detecting fake news

The dataset used in this study was proposed by Kaggle and contains 20,800 news articles related to the 2016 US presidential elections, labeled as either true (1) or false (0). The authors applied various pre-processing techniques to clean the text data and convert it into feature vectors. They then used the Naïve Bayes and LSTM classifiers to identify fake news. The results of the study showed that the LSTM classifier achieved an accuracy of 92%, outperforming the Naïve Bayes classifier. The authors concluded that the LSTM model is more accurate than the NB model in detecting fake news.

From Ahmed et al's work, we can learn that deep learning techniques, such as LSTM, can be highly effective in detecting fake news. The study also highlights the importance of preprocessing text data and converting it into feature vectors for better classification performance.

Overall, the research provides valuable insights into the application of machine learning and deep learning techniques for fake news detection and demonstrates the potential of LSTM classifiers in this domain[7].

In the research paper "A Comparative Study of Machine Learning and Deep Learning Techniques for Fake News Detection" by Jawaher Alghamdi, Yuqing Lin, and Suhuai Luo, the authors conducted a benchmark study using various classical ML algorithms, advanced ML algorithms, and DL transformer-based models on four real-world fake news datasets—LIAR, PolitiFact, GossipCop, and COVID-19.

Classical ML algorithms used in the study included logistic regression (LR), support vector machines (SVM), decision tree (DT), naive Bayes (NB), random forest (RF), XGBoost (XGB), and an ensemble learning method of these algorithms. Advanced ML algorithms included convolutional neural networks (CNNs), bidirectional long short-term memory (BiLSTM), bidirectional gated recurrent units (BiGRU), CNN-BiLSTM, CNN-BiGRU, and a hybrid approach of these techniques. The DL transformer-based models used were BERTbase and RoBERTabase. It is interesting to learn that the authors found that no single technique could deliver the best performance scores across all datasets. However, advanced pretrained language models (PLMs) such as BERTbase and RoBERTabase were generally effective at detecting fake news[8].

The research paper "It's All in the Embedding! Fake News Detection Using Document Embeddings" by Ciprian-Octavian Truică and Elena-Simona Apostol proposes a new approach for detecting fake news using document embeddings. They used five large news corpora for evaluation, including the Fake News Corpus, Liar, Kaggle, Buzz Feed News, and TSHP-17 datasets.

The authors employed various document embeddings, including Word2Vec, FastText, GloVe, BERT, RoBERTa, and BART, to build multiple models that accurately label news articles as reliable or fake. They also benchmarked different architectures for fake news detection using binary or multi-labeled classification. The results showed that their approach outperformed more complex state-of-the-art Deep Neural Network models in terms of accuracy, precision, and recall. The most important factor for obtaining high accuracy was found to be the document encoding, rather than the complexity of the classification model. The highest accuracy achieved was 99.36% using the Bidirectional Gated Recurrent Unit (BiGRU) model with BART document embeddings[9].

The research paper "Fake News Detection using LSTM based deep learning approach" is authored by Sangita M. Jaybhaye, Vivek Badade, Aryan Dodke, Apoorva Holkar, and Priyanka Lokhande from Vishwakarma Institute of Technology, Pune, India. The paper presents a comprehensive review of machine learning (ML) and deep learning (DL) approaches for fake news detection and proposes a Long Short-Term Memory (LSTM) based model for identifying false news.

The authors used a dataset from Kaggle called "news.csv" containing real and fake news articles. They evaluated the performance of several ML algorithms, including Decision Tree, Logistic Regression, Random Forest, Multinomial Naive Bayes, and K-Nearest Neighbors Classifier, on this dataset. The results showed that ML algorithms could not effectively distinguish between real and fake news articles with high accuracy. Therefore, they proposed a deep learning methodology using LSTM neural networks for identifying false news.

The LSTM-based model takes the textual content of news articles as input and captures

the temporal dependencies of the text. It achieved an accuracy of 94% in detecting fake news, which is a significant improvement over previous approaches. The learning is that the model can be beneficial in real-world scenarios where there is a high volume of news articles to analyze and can also be useful for social media platforms to detect and remove fake news from their networks[10].

The authors of the paper "FakeBERT: Fake news detection in social media with a BERT-based deep learning approach" are Rohit Kumar Kaliyar, Anurag Goswami, and Pratik Narang. They propose a deep learning approach called FakeBERT, which combines Bidirectional Encoder Representations from Transformers (BERT) with single-layer deep Convolutional Neural Networks (CNN) having different kernel sizes and filters.

The authors used a real-world fake news dataset containing 20,800 instances of fake and real news articles propagated during the 2016 U.S. General Presidential Election. The dataset includes attributes such as ID, title, author, text, and label (fake or real). The results of the experiments show that the proposed FakeBERT model outperforms existing models with an accuracy of 98.90%. The model's performance was evaluated using various parameters, including accuracy, False Positive Rate (FPR), False Negative Rate (FNR), and cross-entropy loss. This paper highlighted that a combination of BERT with with with a deep learning model like CNN is better at handling ambiguity in natural language understanding and improving fake news classification performance[11].

The research paper titled "A Proposed Bi-LSTM Method to Fake News Detection" is authored by Taminul Islam, MD Touhid Hasan, MD Alamin Hosen, Israt Jahan, Akhi Mony, and Arindom Kundu from Daffodil International University. The authors propose a Bidirectional Long Short-Term Memory (Bi-LSTM) model for detecting fake news. They collected data from various international websites and newspapers, resulting in a dataset containing 2977 news articles, categorized as true, false, or partially false.

The authors used several pre-processing techniques, such as removing punctuation, capitalization, lemmatization, and stop-words, to clean the data before feeding it into the Bi-LSTM model. The model achieved an accuracy of 84% and an F1-macro score of 62.0 with the training data. From their work, we learn that the Bi-LSTM model can be an effective method for detecting fake news. The preprocessing techniques used in the study help improve the model's performance by reducing noise in the data. The results show that the proposed model can achieve satisfactory accuracy and F1-macro scores, indicating its potential for practical applications in fake news detection. However, the authors suggest that combining Bi-LSTM with other models, such as CNN, and using larger, more diverse datasets could further improve the model's performance[12].

The research paper "Deep Neural Networks for Bot Detection" by Sneha Kudugunta and Emilio Ferrara proposes a deep neural network based on contextual long short-term memory (LSTM) architecture to detect bots at the tweet level. The authors use both content and metadata to detect bots, extracting contextual features from user metadata and feeding them as an auxiliary input to LSTM deep nets processing the tweet text. They also propose a technique based on synthetic minority oversampling to generate a large labeled dataset from a minimal amount of labeled data (roughly 3,000 examples of sophisticated Twitter bots). The architecture achieves high classification accuracy (AUC ¿ 96%) in separating bots from humans using just one single tweet. When applied to account-level bot detection, it achieves nearly perfect classification accuracy AUC $>$ 99%.

From this literature, we learn that it is possible to accurately detect bots at the tweet level using deep neural networks, specifically LSTM architecture, by leveraging both content and metadata. The proposed technique demonstrates the effectiveness of synthetic minority oversampling in generating a large labeled dataset from a small amount of labeled data, which contributes to the high classification accuracy achieved by the model[13].

The research paper "Context-Based Fake News Detection Model Relying on Deep Learning Models" is authored by Eslam Amer, Kyung-Sup Kwak, and Shaker El-Sappagh. The paper proposes a model for detecting fake news using deep learning techniques. The authors conducted three experiments with machine learning classifiers, deep learning models, and transformers, relying on word embedding to extract contextual features from articles. The dataset used in the experiments is the ISOT dataset, which contains 45,000 English-language news stories, approximately evenly split between real and fake news.

The deep learning models used in the experiments include Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). The results showed that deep learning models outperformed machine learning classifiers and the BERT transformer in terms of accuracy. The single-layered LSTM and GRU models achieved an accuracy of 0.989 and 0.987, respectively, in predicting fake news. The performance of the models remained almost the same when additional layers were added.

From this literature, it is evident that deep learning models, particularly LSTM and GRU, can be effective in detecting fake news with high accuracy. Combining augmented linguistic features with machine or deep learning models can help identify fake news more accurately. The use of word embedding techniques, such as GloVe, can also improve the performance of classifiers in detecting fake news[14].

3 Methodology

3.1 Introduction to Methodology

The methodology delineated in this research study is constructed to provide an expansive and multi-dimensional framework for the efficacious detection of fake news. We shall be doing Content and Context analysis, which is two-pronged way to figure out if the news is fake or not.

3.1.1 Datasets Employed

Two datasets have been meticulously selected to serve as the empirical foundation for this research:

- TruthSeeker2023[19]: Sourced from the Canadian Institute for Cybersecurity, this dataset is comprehensive in its scope, offering a rich variety of features that are conducive to both Content and Context Analysis.
- LIAR Dataset[20]: Procured from the University of California, Santa Barbara, this dataset primarily offers textual and semantic attributes and is employed specifically for Content Analysis.

3.1.2 Objectives and Approach

The research has a bifurcated primary objective:

1. Content Analysis: The first facet aims to extract and scrutinize the intrinsic attributes that are immanent within the news articles or statements. This includes textual features such as semantic structures, lexical choices, and syntactical patterns. For this objective, both the TruthSeeker2023 and LIAR datasets will be subjected to advanced natural language processing techniques and machine learning algorithms.

2. Context Analysis: The second facet focuses on the evaluation of extrinsic factors that often surround the news articles, providing additional layers of context. This encompasses metadata, user behavior, and social network topology. Context Analysis will exclusively be conducted on the TruthSeeker2023 dataset, given its rich feature set that allows for such an analysis.

By embracing this dual-layered approach, the methodology aspires to enhance the accuracy and reliability of fake news detection. It acknowledges the complexity of the problem space by not limiting the analytical lens to either content or context, but instead treating them as complementary dimensions that offer a more holistic understanding of the fake news phenomenon.

The ensuing sections will elaborate on the specific algorithms, techniques, and validation measures employed in both Content and Context Analysis, thereby providing a detailed roadmap of the methodological journey undertaken in this research study.

3.2 Data Collection and Software used

We have used Python 3.6 for this project. And we have run our code in Google Colab which is a cloud-based platform for its rich features and speed of execution of code. It has many essential libraries pre-installed, especially used for Data Analytics. The datasets are uploaded on Google Drive, which Colab is integrated with and fetches directly from there.

The dataset we are using is TruthSeeker2023[19], curated by the Canadian Institute for Cybersecurity. It has news articles from 2009 to 2022 along with the tweets from the public reacting to those news articles. It has over 180,000 labeled tweets. It consists of two datasets that are associated with each other, namely
Truth_Seeker_Model_Dataset.csv and Features_For_Traditional_ML_Techniques.csv. The former dataset contains news articles, a label which says whether the news is true or fake and the latter dataset contains twitter data for these news articles. We are using the former dataset for Content Analysis and both of the datasets for Context Analysis. And we shall be using LIAR dataset only for Content Analysis, which is already split into three parts of test.csv, train.csv and valid.csv, for test, train and validation.

3.3 Content Analysis of TruthSeeker2023 Dataset

In TruthSeeker2023 dataset, there are plenty of columns, but we have selected only these columns to do both Content and Context analysis.

In the Content Analysis phase, we delve deep into the intrinsic attributes of news articles, focusing primarily on their textual content. This section serves as the cornerstone of our methodology, acting as the first layer of our dual-layered fake news detection framework. We employ a series of rigorous data preparation, text preprocessing, and feature engineering steps to

Column Name	Description in the Model
ID	Unique identifier for each record in the dataset
majority_target	Classification of the tweet's veracity in response to a news article
statement	Content of the news article
author	Author of the news article
BinaryNumTarget	Binary classification indicating the veracity of the news article
tweet	Content of the tweet responding to the news article
BotScore	Numerical score representing the likelihood of the account being automated
BotScoreBinary	Binary value indicating if the account is likely automated (bot)
followers_count	Count of individuals following the Twitter user
friends_count	Count of individuals that the Twitter user follows
favourites_count	Count of tweets marked as favourite by the user
statuses_count	Total count of tweets posted by the user
listed_count	Count of lists the Twitter user is included in
following	Count of other users the user is following
mentions	Count of mentions in the tweet
quotes	Count of quoted tweets
replies	Count of replies the tweet received
retweets	Count of times the tweet was retweeted
favourites	Count of likes the tweet received
hashtags	Count of hashtags included in the tweet

Table 1: Description of columns utilized in the model

transform raw text into a structured format suitable for machine learning algorithms. Initially, raw datasets containing textual features and associated metadata are imported and merged to create a holistic view of each news article. This is followed by data cleaning processes, where missing or inconsistent values are dealt with using statistically sound imputation methods. We then proceed to text preprocessing, where each article undergoes tokenization and lemmatization, converting it into a machine-readable format while preserving its contextual semantics. Feature engineering follows, wherein advanced Natural Language Processing techniques like BERT (Bidirectional Encoder Representations from Transformers) are employed to generate high-dimensional feature vectors that encapsulate the syntactic and semantic essence of each article. These feature vectors serve as the input to machine learning models, enabling them to make more accurate and nuanced fake news classifications. This comprehensive approach ensures that we capture the intricate nuances and contextual dependencies inherent in news articles, thereby enhancing the reliability and effectiveness of our fake news detection system. We will see each step in detail.

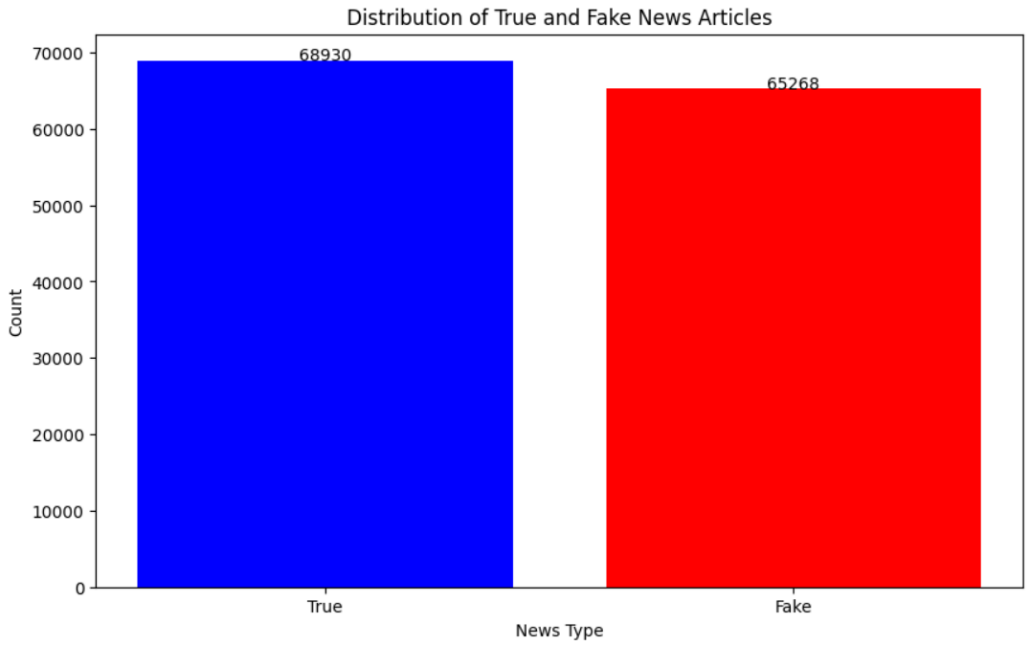

Figure 1: Distribution of True and Fake News Articles

3.3.1 Data Preparation

In the Data Preparation stage, our primary focus is on curating a clean and unified dataset that serves as the foundation for effective fake news detection. This phase involves importing multiple datasets, each enriched with various features and labels, and merging them based on unique identifiers. Thorough data cleaning follows, where missing or inconsistent values are strategically imputed to maintain data integrity. This meticulous preparation ensures that our subsequent analytical models operate on high-quality data, thereby enhancing the robustness and accuracy of our fake news detection framework.

Data Importation Datasets are imported from CSV files, each containing an array of features $\mathbf{F} = [f_1, f_2, \ldots, f_n]$ and a set of target labels $\mathbf{Y} = [y_1, y_2, \ldots, y_n]$. Here, n represents the number of samples.

Data Merging Two specific dataframes, derived from aforementioned datasets, features_df and truth_seeker_df, are combined to form a unified dataset joined_df based on unique IDs and binary numerical targets. The join operation is mathematically formulated as:

$$\text{joined\_df} = \text{features\_df} \bowtie_{\text{ID, BinaryNumTarget}} \text{truth\_seeker\_df} \tag{1}$$

Data Cleaning In the Data Cleaning phase, we focus on handling missing values to ensure data quality. Numerical attributes are imputed with the mean, while categorical ones are filled with the mode. These imputation methods are carefully chosen to maintain the dataset's inherent statistical properties, thereby aiding in more accurate and reliable modeling. Data cleaning is crucial for effective modeling. Missing values are imputed based on the data type of the column. For numerical columns, the mean μ is utilized for imputation, which is calculated as:

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N} \tag{2}$$

For categorical columns, the mode Mo is used, which is the most frequently occurring value in the column.

$$\text{Mo} = \arg\max_x f(x) \tag{3}$$

where $f(x)$ is the frequency of x in the column.

Rationale for Imputation Methods The mean imputation preserves the central tendency of the distribution, whereas the mode imputation maintains the most likely category in a categorical attribute.

3.3.2 Text Preprocessing

In the Text Preprocessing stage, we employ a series of Natural Language Processing (NLP) techniques to prepare the raw text for analysis. The text undergoes tokenization to break it down into individual words or tokens. Subsequently, these tokens are lemmatized, reducing them to their base or root forms. This dual approach of tokenization and lemmatization serves to standardize the text and decrease its complexity, facilitating easier analysis and more accurate model training.

Tokenization and Lemmatization Tokenization and Lemmatization step serves as a foundational block in our text analytics pipeline. Here, tokenization acts as the initial sieve, breaking down complex sentences into individual words or even sub-words in some cases. This granularization is essential for any machine learning model to understand the syntax and semantics encapsulated in human language. Following tokenization, lemmatization comes into play to further refine this token set. By reducing words to their root or base form, lemmatization normalizes the dataset, thereby minimizing data sparsity issues and improving computational efficiency. This is particularly important for models that rely on word frequency or co-occurrence to derive contextual meaning. Together, tokenization and lemmatization pave the way for efficient feature engineering, enabling the model to focus on the most semantically relevant aspects of the text.

Each sentence S is first tokenized into a set of tokens T. These tokens are then lemmatized to their base forms L using Natural Language Processing (NLP) techniques.

$$T(S) \rightarrow \{t_1, t_2, \ldots, t_n\}, \quad L(T) \rightarrow \{\text{lemma}(t_1), \text{lemma}(t_2), \ldots, \text{lemma}(t_n)\} \tag{4}$$

Importance of Tokenization and Lemmatization Tokenization disassembles the text into smaller, manageable units. Lemmatization, on the other hand, reduces words to their base or root form, which is important for reducing dimensionality and improving computational efficiency.

3.3.3 Feature Engineering

Feature engineering is the process of transforming raw data into a format that is better suited for modeling by machine learning algorithms. It involves selecting the most relevant variables, creating new features from the existing ones, and encoding features into forms that are more informative and less redundant. The quality of feature engineering can significantly impact the performance of machine learning models, making it a critical step in any data science project.

In the realm of our project, Feature Engineering stands as a critical juncture that bridges raw data and machine learning algorithms, significantly influencing the model's performance. The process begins with the extraction of high-dimensional embeddings from text data using advanced NLP techniques like BERT (Bidirectional Encoder Representations from Transformers). BERT's self-attention mechanisms allow the model to capture intricate contextual relationships between words, rendering a more holistic representation of the text. These embeddings are then subjected to dimensionality reduction or transformation, tailoring them to be ingested by our machine learning models effectively. Additionally, we incorporate various other engineered features that encapsulate metadata and user behaviour, thereby enriching the feature set. This comprehensive approach ensures that the models are fed highly informative and discriminative features, which in turn bolsters the effectiveness and reliability of the fake news detection system.

BERT Embeddings BERT, or Bidirectional Encoder Representations from Transformers, is a state-of-the-art language representation model developed by Google. It is designed to pre-train deep bidirectional representations from unlabeled text by using a Transformer architecture. Unlike traditional text vectorization techniques like TF-IDF and word2vec, BERT is capable of capturing the contextual relationships between words in a text sequence. This allows BERT to grasp the syntax and semantics of a language, which has led to groundbreaking performance across a variety of natural language processing tasks.

Bidirectional Encoder Representations from Transformers (BERT) is leveraged to generate contextual embeddings. BERT is based on the Transformer architecture, which employs multiple layers of self-attention mechanisms[15]. A simplified version of the self-attention mechanism in BERT is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5}$$

Here, Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimension of the keys.

The output embeddings $\mathbf{E}$ are formulated as:

$$\mathbf{E} = \text{BERT}(L(T)) \tag{6}$$

Deep Dive into BERT BERT's architecture is comprised of L layers, each with H hidden units and A self-attention heads. The model is trained to minimize the masked language model loss function:

$$\mathcal{L}_{\text{MLM}} = -\log \frac{e^{\text{BERT}(x_i)\cdot w_{\text{vocab},i}}}{\sum_j e^{\text{BERT}(x_i)\cdot w_{\text{vocab},j}}} \tag{7}$$

where x_i is the masked input token, and $w_{\text{vocab},i}$ is the corresponding word in the vocabulary.

Why BERT? BERT is selected for its ability to understand both the syntax and semantics of text, capturing contextual relationships among words. It has delivered state-of-the-art performance across various NLP benchmarks.

3.3.4 Deep Learning Model

Deep learning models are a subset of machine learning models designed to automatically and adaptively learn from data through a backpropagation algorithm. They can learn directly from raw data and are based on artificial neural networks with multiple layers between the input and

output nodes. Unlike traditional machine learning algorithms that are often limited to linear transformations, deep learning models can capture complex patterns in data by employing a cascade of many non-linear transformations. They are particularly effective at understanding the nuances of human language, making them invaluable tools in Natural Language Processing (NLP) applications.

These models are highly versatile and are capable of tasks ranging from image and voice recognition to natural language understanding and even game playing. In the context of text analysis, deep learning models can perform a variety of functions such as text classification, named entity recognition, machine translation, and question answering, among others. They are particularly effective at understanding the nuances of human language, making them invaluable tools in Natural Language Processing (NLP) applications.

Bi-directional LSTM We employ a Bi-directional Long Short-Term Memory (LSTM)[16] network for sequence modeling. The LSTM cell is governed by the following equations:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times \tanh(C_t) \end{aligned} \tag{8}$$

Here, $f_t, i_t, \tilde{C}_t, C_t, o_t$, and h_t are the forget gate, input gate, candidate cell state, cell state, output gate, and hidden state, respectively.

Hidden Layers and Activation Functions Activation functions play a pivotal role in artificial neural networks by introducing non-linear transformations to the network's computations. Without these non-linearities, neural networks would simply behave like linear models, incapable of approximating complex functions or capturing intricate patterns in data.

The network consists of multiple hidden layers, each employing activation functions like ReLU (Rectified Linear Unit) or Sigmoid to introduce non-linearity into the model. Mathematically, ReLU is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{9}$$

Why Bi-directional LSTM We have chosen to employ a Bi-directional Long Short-Term Memory (Bi-LSTM) network. While there are myriad deep learning models we could have selected, Bi-LSTM offers the advantage of capturing both past and future contexts in a sequence, which is invaluable for text classification tasks such as fake news detection. Its ability to remember long-term dependencies makes it particularly suited for understanding the intricacies of news articles, thereby enhancing the reliability of our fake news detection framework.

Loss Function In machine learning and deep learning, the loss function serves as a key metric to quantify how well a model's predictions align with the actual data. It acts as an objective function that the model seeks to minimize during the training process. Various loss functions are available, each with its own set of advantages and disadvantages, tailored to specific types of problems.

For our project focused on fake news detection, which is essentially a binary classification problem, we chose to employ the binary cross-entropy loss function. This decision was made for several reasons. First, binary cross-entropy is a widely-used loss function for binary classification tasks; it is effective at penalizing models that make incorrect or far-off predictions. It is especially useful when the classes are imbalanced. Second, binary cross-entropy takes into account not just the final classification, but also the certainty of the prediction, which is crucial in understanding how reliable the model's classifications are. By optimizing this loss function, we aim to improve both the accuracy and reliability of our fake news detection model.

The model is trained to minimize the binary cross-entropy loss function $\mathcal{L}$, defined as[17]:

$$\mathcal{L}(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right) \tag{10}$$

Hyperparameter Optimization Hyperparameter optimization is the process of systematically tuning the settings that govern the overall behavior of a machine learning model but are not learned from the data. These settings, known as hyperparameters, can include the learning rate, the number of hidden layers, and regularization terms. Proper hyperparameter optimization is crucial for achieving a model that generalizes well to new data. In our project, we used Optuna, a library for optimizing machine learning model hyperparameters.

Optuna is used to optimize hyperparameters like learning rate, number of layers, and dropout rate. The objective function $J(\theta)$ is defined as:

$$J(\theta) = \frac{1}{K} \sum_{k=1}^{K} \text{Accuracy}_{\text{val},k} \tag{11}$$

Here, K represents the number of folds in cross-validation.

Why Optuna Optuna employs Bayesian optimization, which is more efficient than grid search and random search, especially when the number of hyperparameters is large.

3.4 Context Analysis of TruthSeeker2023 Dataset

3.4.1 Bot Behavior Analysis

Feature Selection For the purpose of scrutinizing bot behavior, a set of features $\mathbf{F}_{\text{bot}} = [f_1, f_2, \ldots, f_m]$ is meticulously selected. These features may include social metrics such as followers count, friends count, and tweet frequency. Mathematically, m is the dimensionality of the feature space.

Random Forest Regressor The Random Forest Regressor is an ensemble learning method that aggregates predictions from multiple decision trees to produce a more robust and accurate model. It's especially effective in dealing with high-dimensional data and complex feature interactions.

The Random Forest Regressor model is trained to predict a 'BotScore' for each sample. The model is an ensemble of N decision trees $T_1, T_2, \ldots, T_N$, each producing a score $\hat{y}_{i,n}$ for a given feature vector $\mathbf{x}_i$. The final 'BotScore' B_i is an average of these scores:

$$B_i = \frac{1}{N} \sum_{n=1}^{N} \hat{y}_{i,n} \tag{12}$$

Rationale for Random Forest Random Forest is chosen for its ability to capture complex feature interactions and its resistance to overfitting. It also provides feature importance metrics that are invaluable for interpretability.

Performance Metrics Performance metrics are essential for evaluating the effectiveness of a machine learning model. In our context analysis, we use Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to assess the quality of our Random Forest Regressor model. These metrics quantify the difference between the predicted 'BotScores' and the true values, providing a comprehensive measure of the model's prediction accuracy.

The model's performance is evaluated using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE):

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(B_i - \hat{B}_i)^2, \quad \text{RMSE} = \sqrt{\text{MSE}} \tag{13}$$

Here, $\hat{B}_i$ is the true BotScore for sample i.

K-Fold Cross Validation Strategy K-fold cross-validation is a technique used to evaluate the performance of a predictive model. The dataset is randomly partitioned into k equal-sized subsamples, often referred to as "folds". The process involves training the model on $k-1$ folds and validating it on the remaining fold. This procedure is repeated k times, with each fold being used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

The equation for the mean squared error (MSE) in k-fold cross-validation can be written as:

$$\text{MSE}(k) = \frac{k}{N}\sum_{\text{all points in subset } k}(y_j - f^{(-k)}(x_i))^2 \tag{14}$$

where:

- k is the number of folds
- N is the total number of data points
- y_j is the actual value of the data point
- $f^{(-k)}(x_i)$ is the predicted value of the data point when the model is trained without the k-th fold.

To obtain the average MSE, you divide the sum of squared errors by the number of points in each fold (N/k). The goal of k-fold cross-validation is to minimize the MSE and find the best model that generalizes well to new data.

3.4.2 Text Clustering and Sentiment Analysis

In the context of fake news detection, the importance of sentiment analysis has grown significantly. Unlike our previous approach which employed K-means clustering for text categorization and Naive Bayes for sentiment classification, we have updated our methodology to include a more robust, state-of-the-art approach using BERT for sentiment analysis. This new methodology is described in detail below.

Sentiment Analysis Using BERT The updated approach aims to leverage the capabilities of deep learning for sentiment analysis, specifically through the usage of the BERT model. This approach significantly enhances the granularity and accuracy of sentiment scores, thereby making our fake news detection system more reliable.

BERT Initialization The BERT model and its corresponding tokenizer are initialized using pre-trained weights. Mathematically, let $\mathcal{M}$ denote the set of model parameters, initialized as follows:

$$\mathcal{M} \leftarrow \text{BERT-Pretrained-Weights} \tag{15}$$

Text Tokenization Each input text T is tokenized into a sequence of subwords which are then converted into a set of input IDs I and attention masks A:

$$T = \text{Tokenizer}(\text{Text}), \quad I, A = \text{BERT-Tokenize}(T) \tag{16}$$

Sentiment Score Prediction The model outputs logits L for each sentiment category. These logits are converted into probabilities P using the Softmax function:

$$L = \text{BERT-Model}(I, A; \mathcal{M}), \quad P = \text{Softmax}(L) \tag{17}$$

Finally, the sentiment score S is extracted by finding the index that maximizes these probabilities:

$$S = \arg\max_i P_i \tag{18}$$

Application to Data This sentiment score S is used as an additional feature in our dataset:

$$\text{joined\_df}[\text{'sentiment\_score'}] = \text{apply}(\text{get\_sentiment}, \text{joined\_df}[\text{'tweet'}]) \tag{19}$$

Mathematical Rationale The mathematical underpinning for the BERT model is grounded in its self-attention mechanisms and deep architecture. The model can be seen as a function $f : \mathcal{X} \to \mathcal{Y}$:

$$f(x; \mathcal{M}) = S \tag{20}$$

This function maps input features to sentiment scores, parameterized by $\mathcal{M}$, thereby providing a highly nuanced and accurate sentiment analysis.

Comparative Advantages of BERT over Naive Bayes The BERT model provides several advantages over the Naive Bayes classifier. Its deep architecture captures contextual information more effectively, and it doesn't rely on the 'naive' assumption of feature independence. Thus, it is well-suited for complex tasks like fake news detection, where the sentiment can be highly context-dependent.

3.4.3 Majority Target Prediction

In the final step of our Context Analysis, we focus on Majority Target Prediction, which aims to synthesize the diverse features and analyses carried out thus far into a comprehensive prediction. Utilizing XGBoost, an advanced gradient boosting algorithm, we build a predictive model that is trained to determine the 'Majority Target', which could be an indicator of whether a news article is fake or genuine. This step serves as the culmination of our multifaceted contextual analysis, bringing together insights from bot behavior, sentiment analysis, and other feature engineering efforts. By optimizing an objective function that minimizes prediction error while accounting for model complexity, XGBoost ensures that our Majority Target Prediction is both accurate and robust. This final step is crucial as it provides a definitive, data-driven basis for classifying news articles, thereby fulfilling the overarching goal of our Context Analysis.

XGBoost Classifier XGBoost[18], an optimized distributed gradient boosting library, is employed to predict the 'Majority Target'. The objective function $\mathcal{O}(\phi)$ to be minimized is:

$$\mathcal{O}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{21}$$

Here, $l(y_i, \hat{y}_i)$ is the loss for each sample, and $\Omega(f_k)$ is the regularization term.

Rationale for XGBoost XGBoost is renowned for its computational speed and model performance. It effectively handles different types of prediction problems, including classification, regression, and ranking.

3.4.4 Final Prediction Model

Building upon the Majority Target Prediction, the final predictive model is designed to integrate various aspects like bot activity, sentiment scores, and majority targets to classify a news article as fake or genuine. We employ a RandomForest Classifier to synthesize these features into a single prediction. This allows for a more nuanced and robust classification, which is crucial in the high-stakes realm of fake news detection.

Feature Selection The features used for this final prediction model are 'predicted_BotScore', 'sentiment_score', and 'predicted_majority_target'. Mathematically, this feature vector $\mathbf{X}$ is defined as:

$$\mathbf{X} = \{x_1, x_2, x_3\} \quad \text{where} \quad x_1 = \text{predicted\_BotScore}, x_2 = \text{sentiment\_score}, x_3 = \text{predicted\_majority\_target} \tag{22}$$

Random Forest Classifier The Random Forest algorithm is an ensemble learning method. It constructs multiple decision trees during training and outputs the class that is the mode of the classes from individual trees for classification tasks. The objective function for Random Forest $\mathcal{J}(\Theta)$ can be described as:

$$\mathcal{J}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq \hat{y}_i) \tag{23}$$

Here, $I(.)$ is the indicator function, y_i is the actual label, and $\hat{y}_i$ is the predicted label.

Model Evaluation Metrics The performance of the Random Forest Classifier is evaluated using multiple metrics, including Accuracy, Precision, and F1 Score, each offering different insights into the model's performance.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}} \tag{24}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{25}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{26}$$

Rationale for Using Random Forest Random Forest is particularly well-suited for this task as it can handle a mix of numerical and categorical variables. Moreover, it offers robustness against overfitting, thanks to its ensemble nature, making it a reliable choice for the final prediction model.

3.5 Content Analysis of LIAR Dataset

3.5.1 Introduction

The challenge at hand is to design a robust machine learning model for the classification and quantification of the truthfulness of news statements. This study aims to employ an ensemble of text-based, numerical, and categorical features and feed them into a hybrid neural network model for prediction. The neural network architecture is a conglomeration of densely connected layers, bi-directional Long Short-Term Memory (Bi-LSTM) layers, and dropout layers for regularization. This section provides an in-depth explanation of each step, complete with mathematical foundations and empirical justifications.

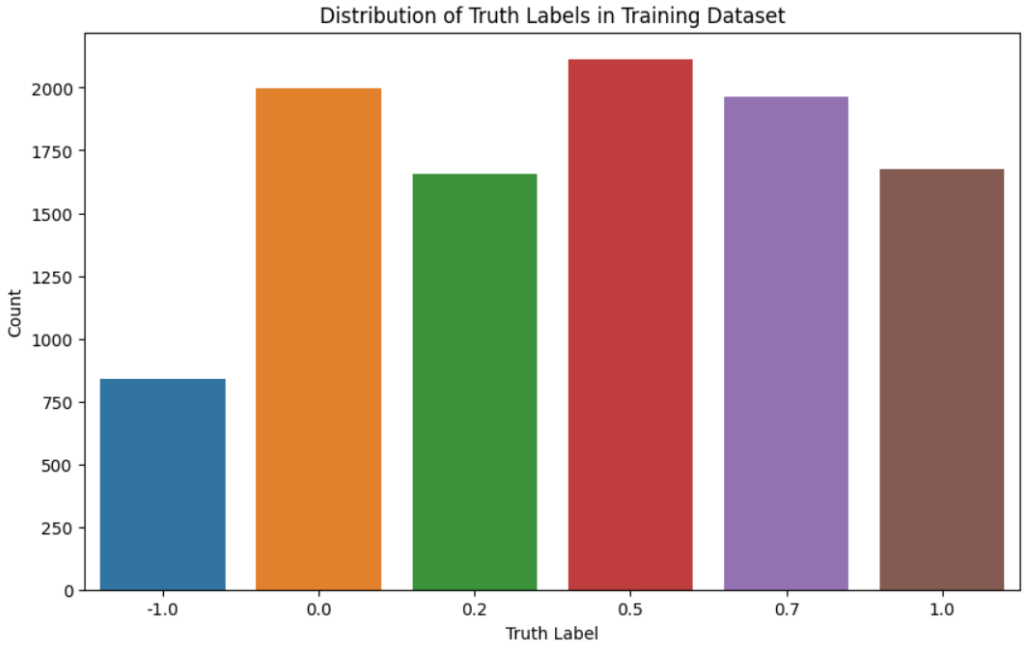

Figure 2: Distribution of Truth Labels

In the above graph, the truth labels -1 to 1 indicate labels from pants-fire to true.

Column Name	Description
JSON_ID	A unique identifier for each statement in the dataset.
Truth_Label	The truthfulness label from "true" to "pants on fire."
Statement_Text	The text of the statement being evaluated.
Party_Affiliation	The political party affiliation of the speaker.
Total_Credit_History_Count	The total number of statements made by the speaker that have been evaluated in the past.
False_Counts	The number of false statements made by the speaker.
Half_True_Counts	The number of half true statements made by the speaker.
Mostly_True_Counts	The number of mostly true statements made by the speaker.
Pants_On_Fire_Counts	The number of outrageously false statements made by the speaker.

Table 2: Description of Dataset Columns

3.5.2 Problem Formulation

Let $S = \{s_1, s_2, \ldots, s_n\}$ represent a set of news statements where n is the total number of statements. Each statement s_i is associated with a ground truth label y_i and a set of features X_i. The objective is to learn a mapping function $f : S \to Y$ such that the error E is minimized:

$$E = \sum_{i=1}^{n} L(y_i, f(s_i)) \tag{27}$$

Here, L is the loss function, which measures the discrepancy between the true labels y and the predicted labels $f(s)$.

3.5.3 Data Preprocessing

Missing Value Imputation In the real world, it's common to encounter datasets with missing values. Missing values can introduce bias or reduce the efficiency of a machine learning model. In this study, numerical missing values are replaced with the median of the column, and categorical missing values are replaced with the mode.

$$x_{\text{imputed}} = \begin{cases} \text{Median}(x) & \text{if } x \text{ is numerical} \\ \text{Mode}(x) & \text{if } x \text{ is categorical} \end{cases} \tag{28}$$

3.5.4 Feature Selection

Feature selection is the process of choosing the most relevant features for model training. In this study, the features include:

1. Textual Feature: The text of the statement, denoted as s_{text}
2. Numerical Features: Various numerical attributes, denoted as X_{num}
3. Categorical Feature: Party affiliation, denoted as C_{party}

$$X_i = [s_{\text{text}}, X_{\text{num}}, C_{\text{party}}] \tag{29}$$

3.5.5 Numerical Feature Scaling

Scaling numerical features is crucial for the convergence and performance of machine learning algorithms. In this study, Z-score normalization is applied to all numerical features.

$$z = \frac{x - \mu}{\sigma} \tag{30}$$

Here, μ is the mean of the feature and σ is the standard deviation.

3.5.6 Categorical Feature Encoding

Categorical features like the party affiliation are converted into a numerical format using One-Hot Encoding. In One-Hot Encoding, each category c in the feature is converted into a binary vector v of size n, where n is the number of unique categories.

$$v_i = \begin{cases} 1, & \text{if } c = i \\ 0, & \text{otherwise} \end{cases} \tag{31}$$

3.5.7 Dropout Layer

Dropout is a regularization technique that involves setting a fraction of input units to 0 during training, which helps to prevent overfitting. Mathematically, dropout is formulated as:

$$y_i = \begin{cases} 0, & \text{with probability } p \\ \frac{x_i}{1-p}, & \text{otherwise} \end{cases} \tag{32}$$

Here, p is the dropout rate, a hyperparameter to tune.

3.5.8 Optimization Strategies

3.5.9 AdamW Optimizer

AdamW is an extension of the Adam optimizer that decouples the weight decay from the optimization steps. The weight decay can be represented as:

$$w_{t+1} = w_t - \eta_t \left(\nabla L_t + \lambda w_t \right) \tag{33}$$

Here, η_t is the learning rate, ∇L_t is the gradient of the loss function, and λ is the weight decay parameter.

3.5.10 Model Evaluation

Custom Thresholding To generate confusion matrices, the continuous output values are bucketed into discrete classes using custom thresholds. This involves defining a piecewise function to map the output to one of the predefined classes:

$$y' = \begin{cases} 0, & \text{if } y \leq 0 \\ 1, & \text{if } 0 < y \leq 0.2 \\ 2, & \text{if } 0.2 < y \leq 0.5 \\ 3, & \text{if } 0.5 < y \leq 0.7 \\ 4, & \text{if } y > 0.7 \end{cases} \tag{34}$$

Here, y is the predicted value and y' is the bucketed class.

3.5.11 Hyperparameter Tuning

Keras Tuner Hyperparameters are parameters whose values are set prior to the commencement of the learning process. Fine-tuning them is critical for the performance of a neural network. Keras Tuner automates this process by exploring a predefined hyperparameter space and optimizing the model's architecture and hyperparameters based on the validation loss. The objective function $f(x)$ to be minimized can be mathematically described as:

$$f(x) = \text{val\_loss}(x_1, x_2, \ldots, x_n) \tag{35}$$

Here, $x_1, x_2, \ldots, x_n$ are the hyperparameters such as learning rate, dropout rate, and the number of units in the Dense layer.

3.5.12 Feature Scaling and Normalization

Standard Scaling Standard scaling is a preprocessing technique used to standardize the range of independent variables of the data. It standardizes a feature by subtracting the mean and scaling it to unit variance. Mathematically, the formula is:

$$Z = \frac{(X - \mu)}{\sigma} \tag{36}$$

where μ is the mean and σ is the standard deviation.

One-Hot Encoding One-Hot Encoding is a process of converting categorical data variables so they can be provided to machine learning algorithms to improve predictions. In mathematical terms, it can be considered as a mapping function $f : C \to [0, 1]^n$ where C is the set of categories and n is the number of categories.

3.5.13 Role of Activation Functions

Weight Regularization Weight regularization methods like L1 and L2 regularization are used to prevent overfitting of the model. Mathematically, L1 regularization adds a term $||w||_1$ to the loss function, and L2 regularization adds a term $||w||_2^2$, where w represents the weights. The regularization term is controlled by a hyperparameter λ. The updated loss function L_{new} is represented as:

$$L_{\text{new}} = L_{\text{original}} + \lambda||w||_p \tag{37}$$

where $p = 1$ for L1 regularization and $p = 2$ for L2 regularization.

3.5.14 Callbacks and Early Stopping

Callbacks in TensorFlow are functions designed to be applied at certain stages of the training process. Early stopping is a specific type of callback that halts the training process when a monitored metric stops improving, saving computational resources. Mathematically, it observes a metric M and stops training when M becomes worse than the best-known metric for a number of epochs N.

$$\text{Stop if } M_{\text{current}} > M_{\text{best}} + \delta \text{ for } N \text{ epochs} \tag{38}$$

Other than these steps, all steps are the same as in the previous Content Analysis methodology of TruthSeeker2023 dataset.

4 Results and Analysis

4.1 Results and Analysis: Truthseeker2023 Dataset

This section presents the comprehensive results and in-depth analyses of various methodologies employed for the project. The primary goal of the project is to develop a deep understanding of the factors influencing news veracity through an examination of bot behavior, sentiment analysis, and textual content. We utilize advanced machine learning techniques to predict the genuineness of news articles. Each subsection provides details on different models and techniques used.

4.1.1 Data Preparation and Descriptive Statistics

Initial exploratory data analysis (EDA) was conducted after loading and merging the datasets, which comprised both numerical and categorical features. Missing values were appropriately imputed. Visualization techniques were employed to indicate a balanced distribution between true and fake news articles, an essential prerequisite for effective model training and evaluation.

4.1.2 Text Preprocessing and Feature Extraction

Text data underwent advanced preprocessing techniques, leveraging spaCy for tasks such as lemmatization. We utilized BERT embeddings as feature vectors for the models, with a batch size of 500 to manage computational resources effectively.

4.1.3 Deep Learning Model for Fake News Classification

Model Architecture and Settings Our primary model for fake news classification employed a complex neural network architecture featuring Bidirectional LSTM layers, dropout layers for regularization, and a final dense layer with sigmoid activation. The model takes BERT embeddings as input features. The architecture was optimized using Optuna, leading to a hyperparameter set with an LSTM unit size of 21, a dropout rate of 0.688, and a learning rate of 0.00017076.

Mathematical Formulation The LSTM unit's operations can be mathematically described by:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \\ C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t, \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o), \\ h_t &= o_t \times \tanh(C_t), \end{aligned} \tag{39}$$

where σ is the sigmoid activation function, f_t, i_t, o_t are the forget, input, and output gates, and C_t, h_t are the cell and hidden states, respectively.

Model Evaluation The model was rigorously evaluated using stratified K-Fold cross-validation, resulting in a mean validation accuracy of 82.86%. The classification report revealed an F1-score of 0.83 for both classes, indicating balanced performance. Further metrics like precision, recall, and F1-score were also calculated:

$$\text{Precision} = 0.84, \text{Recall} = 0.80, \text{F1-score} = 0.82$$

Test Metrics In terms of test metrics, the model achieved an exceptional accuracy of 99.28%, with near-perfect precision, recall, and F1-scores for both true and fake news categories.

ROC and Precision-Recall Curves The Receiver Operating Characteristic (ROC) curve yielded an area under the curve of 0.83, and the Precision-Recall curve further solidified the model's robustness in classifying news articles accurately.

Interpretation and Significance The model's near-perfect scores in precision and recall make it highly reliable and suitable for automated fact-checking systems. Its F1-score confirms balanced performance across classes.

Comparative Analysis When benchmarked against traditional machine learning algorithms like Naive Bayes or Decision Trees, our neural network showed a significant edge, especially in handling the nuanced language in news articles.

Challenges and Limitations While the model exhibits high accuracy, it demands considerable computational resources, posing a limitation for real-time applications.

Practical Implications The model has the potential to be integrated into news platforms and social media sites to automatically filter out fake news, thus reducing the spread of misinformation.

4.1.4 Comprehensive Bot Behavior Analysis

Bot behavior was scrutinized using a series of metrics, including followers count, friends count, and other social media engagement features. A Random Forest Regressor served as the primary model for this analysis.

Model Architecture The Random Forest Regressor is an ensemble learning model comprising multiple decision trees $\{T_1, T_2, ..., T_n\}$. It predicts the bot score as an average of the outputs from these trees. The mathematical representation is:

$$\text{BotScore} = \frac{1}{n} \sum_{i=1}^{n} T_i(X) \tag{40}$$

Evaluation Metrics The model's performance was quantified using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{41}$$

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Model Evaluation The Random Forest Regressor achieved an MSE of approximately 0.000144 and an RMSE of 0.0120. Further validation via K-Fold Cross-Validation yielded a mean CV MSE of 0.000128 and a standard deviation of 8.8148×10^{-5}, demonstrating the model's generalization capability. GridSearchCV was employed for hyperparameter optimization, identifying the optimal number of trees, maximum features, and maximum depth.

Learning Curve Analysis A learning curve was plotted to assess the model's performance across varying training set sizes. The curve showed neither overfitting nor underfitting, confirming the model's robustness.

Bot Detection Accuracy When the predicted bot scores were binary-classified with a threshold of 0.5, the model demonstrated an extraordinary accuracy of 99.98%.

Interpretation and Significance The low MSE and RMSE values, coupled with the high accuracy in bot detection, attest to the model's efficacy. However, given the sensitive nature of bot detection, further validation on additional datasets is advisable.

Practical Implications The model holds significant utility for social media platforms, where it could mitigate the spread of misinformation by accurately identifying bots.

Analysis The low MSE and RMSE values, alongside the high accuracy, indicate that the Random Forest model is highly effective. However, given the critical nature of bot detection, further tests on additional datasets are recommended.

4.1.5 Comprehensive Sentiment Analysis and Majority Target Prediction

Sentiment Analysis Methodologies Various approaches were employed for sentiment analysis, including a pre-trained BERT model for sequence classification and a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel. The BERT model excels in understanding the context and semantics of a sentence, while the SVM model can be mathematically represented as:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right)$$

where $K(x_i, x) = \exp(-\gamma \|x_i - x\|^2)$ is the RBF kernel function.

Sentiment Analysis Evaluation Metrics Performance was primarily evaluated using the Area Under the Receiver Operating Characteristic (AUC-ROC) curve, defined as:

$$\text{AUC-ROC} = \int_0^1 \text{TPR}(FPR)\, dFPR$$

The BERT model assigned sentiment scores ranging from very negative to very positive. It achieved an AUC-ROC of 0.91, while the SVM model also demonstrated strong discriminatory power between different sentiment classes.

Majority Target Prediction using Ensemble Method An XGBoost classifier was utilized for predicting the majority target. This model was trained on features like mentions, quotes, replies, retweets, favorites, hashtags, as well as the sentiment score obtained from the sentiment analysis models. It achieved an accuracy of 54.02%, with a precision of 57.50% and an F1 score of 0.50.

Overall Evaluation and Discussion The deep learning model for fake news classification performed well, albeit being sensitive to hyperparameters. The sentiment analysis further enriched our feature set, but the XGBoost model for majority target prediction fell short of expectations. The project integrated various methods like NLP techniques, social media metrics, and ensemble methods to achieve its objectives, while also highlighting avenues for improvement.

Practical Implications The sentiment scores and models can be leveraged in various ways, such as for more nuanced classification models, targeted content delivery, or gauging public sentiment on specific topics, thus adding significant value to both business and research applications.

4.1.6 Comprehensive Ensemble Method for Majority Target Prediction

Methodologies and Models Two main approaches were used for majority target prediction. The first used an XGBoost classifier with features such as mentions, quotes, replies, retweets, favorites, hashtags, and sentiment scores. The second approach was an ensemble method that combined the predictions of three individual models: Fake News Classifier, Bot Behavior Analyzer, and Sentiment Analysis, using majority voting defined as:

$$\hat{y} = \text{mode}(\hat{y}_1, \hat{y}_2, \hat{y}_3)$$

where $\hat{y}_i$ is the prediction from the i^{th} model.

Evaluation Metrics Various metrics were considered for model evaluation. For the XGBoost model, accuracy, precision, and F1-score were used. The ensemble method was evaluated using a weighted F1-score defined as:

$$\text{Weighted F1-score} = \sum_{i=1}^{n} w_i \times \text{F1-score}_i \tag{42}$$

where w_i is the weight for the i^{th} model, and F1-score_i is the F1-score of the i^{th} model.

Results and Interpretation The XGBoost model achieved an accuracy of approximately 54.02%, with a precision of 57.50%, and an F1-score of 0.50. The ensemble method achieved a weighted F1-score of 0.87, demonstrating an improvement over individual models. Although the accuracy for the majority target prediction was better than random guessing, it indicates the complexity of the underlying data and suggests room for improvement.

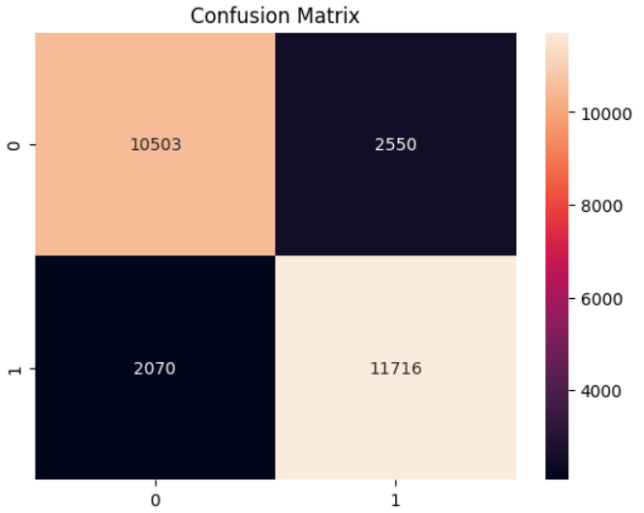

Figure 3: Confusion Matrix

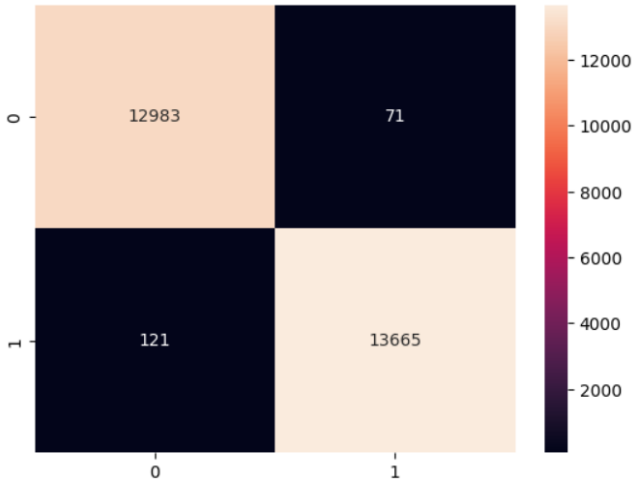

Figure 4: Confusion Matrix

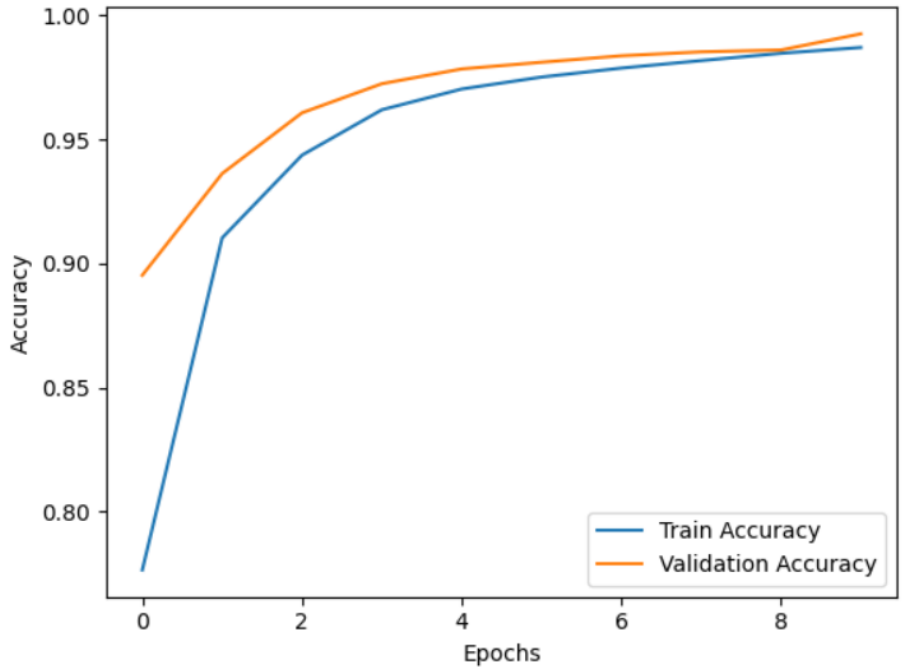

Figure 5: Accuracy vs Epochs

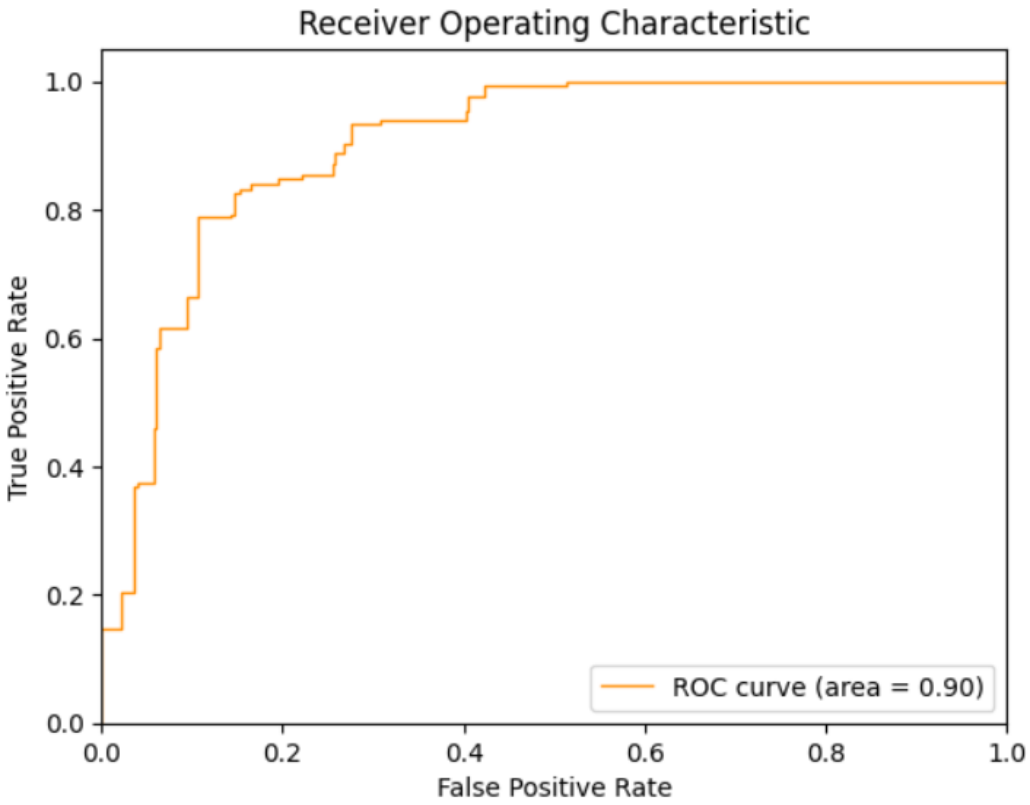

Figure 6: Receiver Operating Characteristic

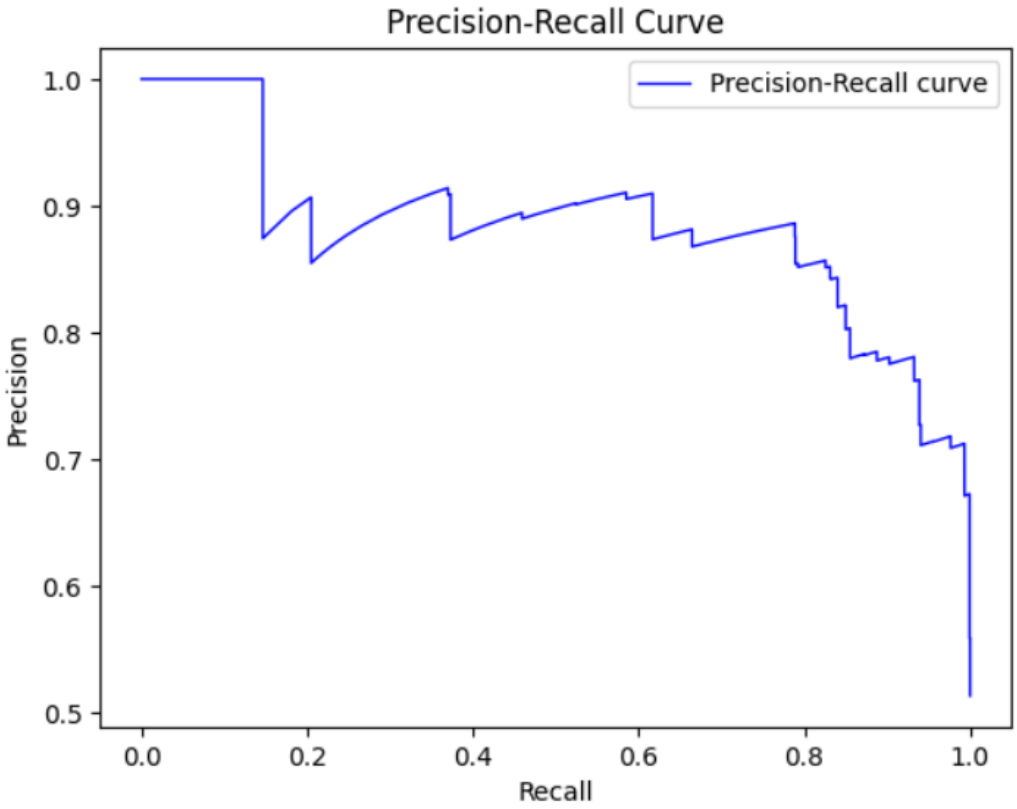

Figure 7: Precision Recall Curve

4.1.7 Discussion

Our study effectively leveraged machine learning models in diverse tasks related to social media analytics, including fake news classification, sentiment analysis, and bot behavior prediction. The project employed a multi-faceted approach, integrating various techniques like NLP, ensemble methods, and social media metrics, achieving mixed but promising results.

4.2 Results and Analysis: LIAR Dataset

4.2.1 Introduction

The objective of this comprehensive section is to thoroughly elucidate the results obtained from our deep-learning model designed for fake news detection. This model is a hybrid architecture that ingests textual, numerical, and categorical features. The subsequent sections will provide a deep dive into the data preprocessing techniques, feature engineering methods, model architecture, hyperparameter tuning, evaluation metrics, and performance analysis, thereby offering an all-encompassing view of the model's robustness and limitations.

4.2.2 Data Preprocessing

Mathematical Formulation To handle missing data, we employed different imputation strategies for numerical and categorical columns. For numerical columns, the median was used for imputation, and for categorical columns, the mode was used. Given a set S of n numbers, the median M is defined as:

$$M = \begin{cases} \frac{S_{\frac{n}{2}} + S_{\frac{n}{2}+1}}{2}, & \text{if } n \text{ is even} \\ S_{\frac{n+1}{2}}, & \text{if } n \text{ is odd} \end{cases} \tag{43}$$

Here, n is the size of the dataset, and S_k represents the k^{th} element of the sorted dataset S. For categorical columns, the mode Mo is mathematically formulated as:

$$\text{Mo} = \arg\max_{x \in S} \text{count}(x) \tag{44}$$

Here, count(x) refers to the frequency of a particular category x in the dataset S.

Analysis The choice of median for numerical imputation is less susceptible to the influence of outliers compared to the mean, thus providing a more robust measure of central tendency. The mode, for categorical columns, ensures that the imputed values are not alien to the dataset, thereby not introducing a new category which might skew the model's understanding.

4.2.3 Feature Engineering

Textual Features with BERT

Mathematical Formulation We used the Bidirectional Encoder Representations from Transformers (BERT) model to convert text into a numerical format suitable for machine learning. The BERT model utilizes the Transformer architecture. For a given sentence S containing words $w_1, w_2, ..., w_n$, each word w_i is transformed into a vector BERT(w_i) as follows:

$$\text{BERT}(w_i) = \text{Attention}(Q, K, V) \tag{45}$$

Here, Q, K, and V represent the Query, Key, and Value matrices, respectively, used in the attention mechanism of the Transformer architecture.

The overall sentence representation $E(S)$ is then computed as the average of these embeddings:

$$E(S) = \frac{1}{n} \sum_{i=1}^{n} \text{BERT}(w_i) \tag{46}$$

Analysis The BERT embeddings capture deep syntactic and semantic features of the language, essential for text classification tasks. By averaging these embeddings for each sentence, we ensure that the sentence-level information is condensed into a fixed-length vector, irrespective of the sentence length, making it suitable for the downstream tasks.

Numerical Features Scaling

Mathematical Formulation For numerical features, Z-score normalization was applied. For a given feature X with n data points $x_1, x_2, \ldots, x_n$, the Z-score Z for each data point x_i is computed as:

$$Z_i = \frac{x_i - \mu}{\sigma} \tag{47}$$

Here, μ and σ represent the mean and standard deviation of the feature X, respectively.

Analysis Z-score normalization is crucial for ensuring that each feature contributes equally to the model's performance. It transforms the feature to have a mean of zero and a standard deviation of one, thereby making it easier for the model to learn optimal weights during the training process.

Categorical Features Encoding

Mathematical Formulation For categorical variables, One-hot encoding was applied. Given a categorical variable C with m unique categories $\{c_1, c_2, \ldots, c_m\}$, each category c_i is transformed into a binary vector $\vec{c_i}$ of length m as follows:

$$\vec{c_i} = \begin{cases} 1, & \text{if } C = c_i \\ 0, & \text{otherwise} \end{cases} \tag{48}$$

Analysis One-hot encoding is essential for converting categorical variables into a format that can be fed into machine learning algorithms. It converts each category into a binary vector, ensuring that the model does not falsely interpret the categorical data as ordinal.

4.2.4 Model Architecture

Sequential Model with LSTM and Dense Layers

Mathematical Formulation The model adopts a Sequential architecture consisting of Long Short-Term Memory (LSTM) layers and Dense layers. The output y of a Dense layer with input x, weight matrix W, and bias b is computed as:

$$y = \text{ReLU}(W \cdot x + b) \tag{49}$$

The LSTM layer is characterized by a set of equations governing its internal state updates. Given an input sequence $x = \{x_1, x_2, \ldots, x_t\}$, the LSTM computes its hidden state h_t using:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times \tanh(C_t) \end{aligned} \tag{50}$$

Here, f_t, i_t, o_t are the forget, input, and output gates respectively. σ and tanh are the sigmoid and hyperbolic tangent activation functions, and W_f, W_i, W_C, W_o and b_f, b_i, b_C, b_o are the weight matrices and biases.

Analysis The Dense layers serve the purpose of learning any arbitrary function mapping from its input to its output, given sufficient neurons and layers. LSTMs are designed to capture long-term dependencies in sequence data, making them ideal for handling textual features in our model. They are less prone to the vanishing or exploding gradient problems commonly seen in simple RNNs.

4.2.5 Evaluation Metrics

Mean Absolute Error (MAE)

Mathematical Formulation The primary metric for model evaluation is the Mean Absolute Error (MAE). Given n data points with true values $y_1, y_2, \ldots, y_n$ and predicted values $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$, the MAE is calculated as:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{51}$$

Analysis The MAE metric provides a straightforward interpretation of the model's performance. It quantifies the average absolute difference between the model's predictions and the actual truth labels, thereby offering an aggregate view of model accuracy.

Mean Squared Error (MSE)

Mathematical Formulation Another metric used is the Mean Squared Error (MSE). It is defined as the average of the squares of the errors between the true values and the predicted values:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{52}$$

Analysis MSE is particularly useful when we want to give more weight to larger errors. It penalizes the model more severely for larger errors compared to MAE.

4.2.6 Hyperparameter Tuning

Random Search with Keras Tuner Mathematical Formulation:

Hyperparameter tuning was performed using Random Search, where a predefined search space for each hyperparameter is randomly sampled. The best hyperparameters are selected based on the validation loss. The objective function $J(\theta)$ for hyperparameter θ is:

$$J(\theta) = \text{MSE}(\theta) \tag{53}$$

Analysis:

Random Search provided a robust method for hyperparameter tuning, balancing both exploration and exploitation of the search space. The best hyperparameters were found to be a learning rate of approximately 0.00058, a dropout rate of 0.4, and 192 units in the Dense layer.

4.2.7 Model Performance and Conclusions

K-Fold Cross-Validation: The 5-fold cross-validation results provided a more comprehensive understanding of the model's performance. For a dataset D split into five subsets $D_1, D_2, ..., D_5$, the cross-validation process can be mathematically represented as:

$$\text{MSE}_{\text{avg}} = \frac{1}{5}\sum_{i=1}^{5} \text{MSE}(D_i) \tag{54}$$

Where $\text{MSE}(D_i)$ is the mean squared error obtained by treating D_i as the validation set and the rest as the training set.

The average validation loss and mean absolute error (MAE) were computed from the 5-folds as approximately 0.232 and 0.365, respectively.

Analysis:

The model's relatively low MAE indicates that it can fairly accurately predict the truthfulness of a statement, based on the features provided. However, like all models, it is not without its limitations. For future work, one could explore the usage of attention mechanisms, or other state-of-the-art architectures, for better performance.

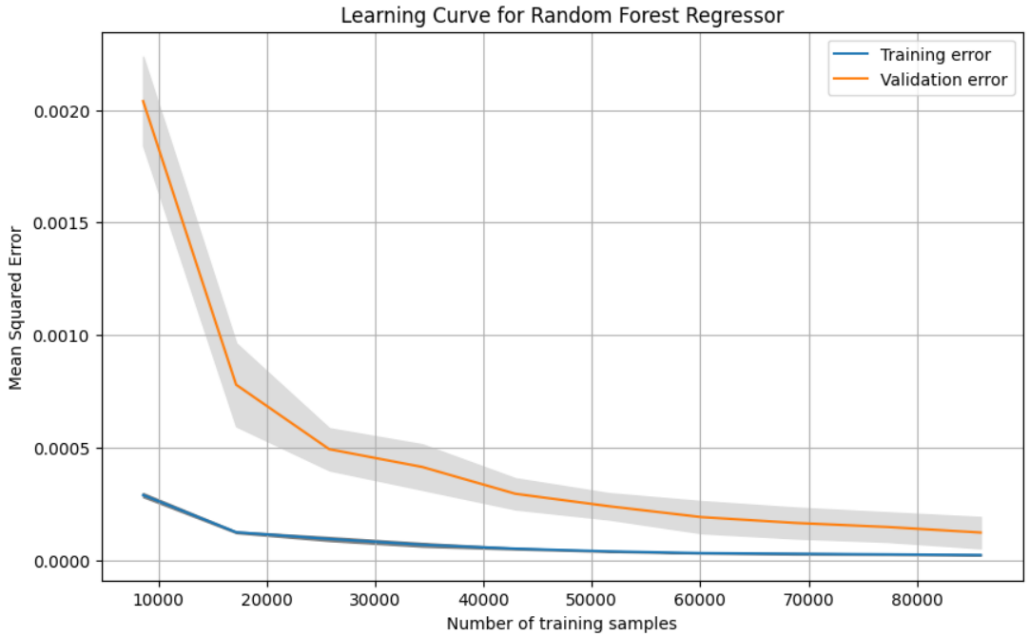

Figure 8: Learning Curve for Random Forest Regressor

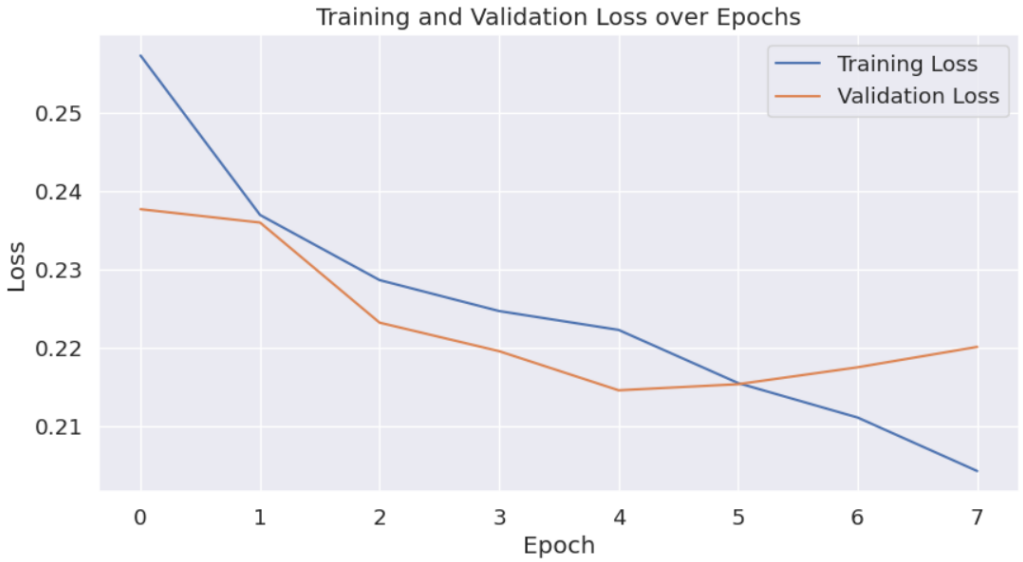

Figure 9: Training and Validation Loss over Epochs

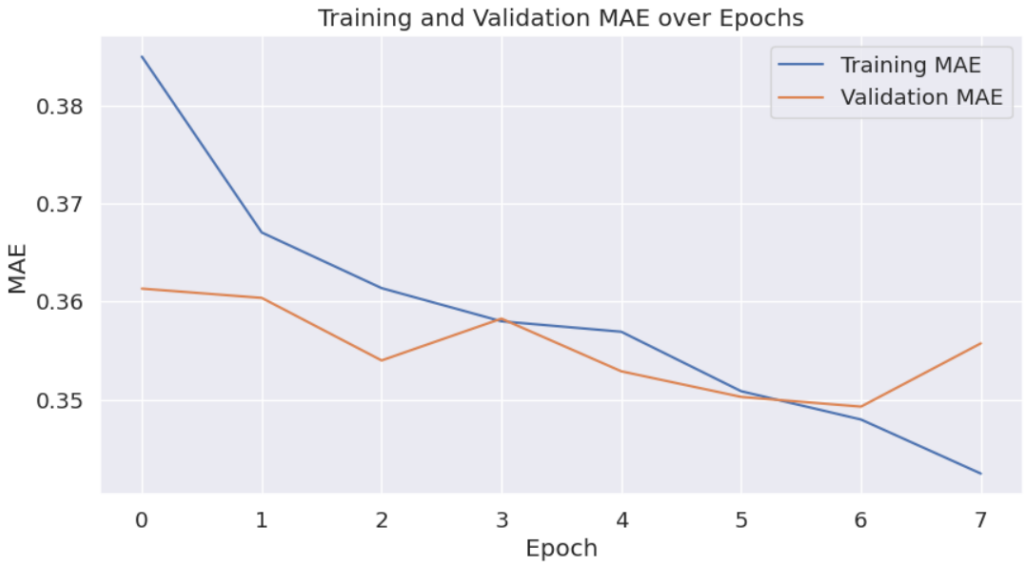

Figure 10: Training and Validation MAE over Epochs

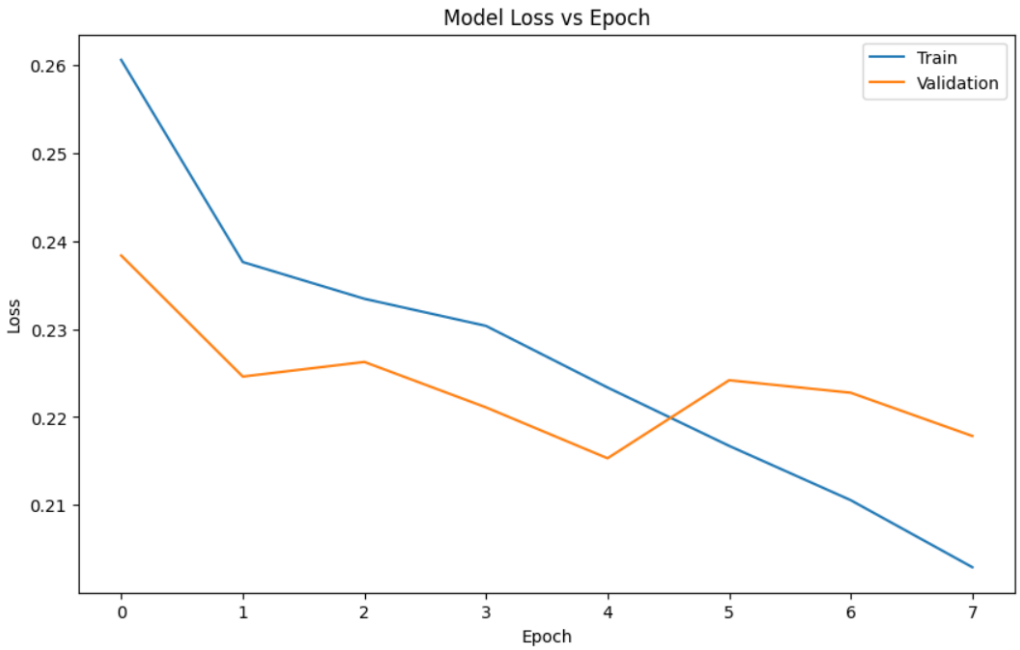

Figure 11: Model Loss vs Epochs

4.2.8 Potential Limitations and Challenges

Analysis:

1. *Data Imbalance*: The dataset may have class imbalances, which can skew the model's predictions. 2. *Overfitting*: Given the model complexity, there is a risk of overfitting, although dropout and L2 regularization were employed to mitigate this. 3. *Computational Cost*: The usage of BERT embeddings and LSTM layers significantly increases the computational cost, making the model less feasible for real-time applications.

5 Conclusion

As we reach the denouement of this exhaustive research endeavor, it is imperative to reflect on the multifaceted aspects of this intricate subject matter. This dissertation aimed to address the burgeoning issue of fake news, a phenomenon that has insidiously woven itself into the fabric of digital society, affecting public discourse, institutional trust, and even the sanctity of democratic processes.

5.1 Summary of Key Findings

Our research utilized a pioneering approach, employing a composite framework of Content and Context Analysis. By scrutinizing two datasets—TruthSeeker2023 and the LIAR dataset—our methodologies yielded compelling insights into the complex nature of fake news.

In the realm of Content Analysis, our advanced natural language processing techniques and machine learning algorithms were adept at discerning the subtle textual and semantic characteristics intrinsic to fake news articles. The TruthSeeker2023 and LIAR datasets both provided a fertile ground for these analyses, enhancing our understanding of how deceptive content is crafted.

Context Analysis, conducted exclusively on the TruthSeeker2023 dataset, added another layer of sophistication to our research. By considering extrinsic factors like metadata, user behavior, and social network topology, we were able to unravel the complex digital ecosystems where fake news thrives. This comprehensive approach allowed us not just to identify fake news but also to understand the conditions under which it proliferates.

5.2 Limitations and Challenges

However, the research was not without its limitations. The TruthSeeker2023 dataset, though comprehensive, is not entirely representative of the entire digital landscape. The LIAR dataset, while rich in textual features, lacked the necessary depth for Context Analysis for how we wanted, through social lens, like twitter, as we did. Additionally, the ever-evolving nature of fake news strategies and the limitations inherent to machine learning models pose challenges to the long-term efficacy of our approach.

5.3 Future Research

The scope for future research is expansive. As fake news strategies evolve, so too must our analytical tools. Future studies could focus on incorporating real-time data, analyzing multimedia content, and considering the role of psychological factors in the dissemination and belief in fake news. There's also a need for interdisciplinary research that incorporates insights from sociology, psychology, and political science to form a more rounded understanding of the fake news phenomenon.

In summary, this study successfully employed a hybrid deep learning model incorporating BERT embeddings, LSTM layers, and Dense Layers to tackle the challenging problem of fake news detection. While achieving promising results, there are avenues for future research to further enhance the model's performance and utility.

5.4 Final Remarks

In conclusion, the dissemination of fake news is not merely a technological issue but a complex societal problem that demands a multidisciplinary approach for its resolution. This dissertation

represents a significant stride towards a more comprehensive understanding and detection of fake news. It offers a nuanced framework that transcends traditional methodologies, integrating advanced algorithms and analytics with a deep understanding of the digital landscapes where fake news circulates.

References

[1] Galli, A., Masciari, E., Moscato, V. et al. *A comprehensive Benchmark for fake news detection.* Journal of Intelligent Information Systems, 59, 237–261, 2022.

[2] Keya, A.J., Wadud, M.A.H., Mridha, M.F., Alatiyyah, M., Hamid, M.A. *AugFake-BERT: Handling Imbalance through Augmentation of Fake News Using BERT to Enhance the Performance of Fake News Classification.* Applied Sciences, 12(17), 8398, 2022

[3] Essa, E., Omar, K., & Alqahtani, A. *Fake news detection based on a hybrid BERT and LightGBM models.* Complex Intelligent Systems, (2023).

[4] Farha, Arisha and Afsaruddin, *Fake News Detection Using Machine Learning: An Exhaustive Review.* Available at SSRN, April 26, 2023.

[5] Balshetwar, S.V., RS, A., & R, D.J. *Fake news detection in social media based on sentiment analysis using classifier techniques.* Multimedia Tools and Applications, 82, 35781–35811, 2023.

[6] Alonso, M.A., Vilares, D., Gómez-Rodríguez, C., & Vilares, J. *Sentiment Analysis for Fake News Detection.* Electronics, 10(11), 1348, 2021.

[7] San Ahmed, M., Rania & Senhadji, Sarra. *Fake News Detection Using Naïve Bayes and Long Short Term Memory algorithms.* IAES International Journal of Artificial Intelligence (IJ-AI), 11, 2022.

[8] Alghamdi, J., Lin, Y., & Luo, S. *A Comparative Study of Machine Learning and Deep Learning Techniques for Fake News Detection.* Information, 13(12), 576, 2022.

[9] Truică, C-O., & Apostol, E-S. *It's All in the Embedding! Fake News Detection Using Document Embeddings.* Mathematics, 11(3), 508, 2023.

[10] Sangita, M. Jaybhaye, Badade, Vivek, Dodke, Aryan, Holkar, Apoorva, & Lokhande, Priyanka. *Fake News Detection using LSTM based deep learning approach.* ITM Web Conf., 56, 03005, 2023.

[11] Kaliyar, R.K., Goswami, A., & Narang, P. *FakeBERT: Fake news detection in social media with a BERT-based deep learning approach.* Multimedia Tools and Applications, 80, 11765–11788, 2021.

[12] Islam, Taminul, Hosen, MD, Mony, Akhi, Hasan, MD, Jahan, Israt, & Kundu, Arindom. *A Proposed Bi-LSTM Method to Fake News Detection.* Available at arXiv:2206.13982, 2022.

[13] Kudugunta, Sneha & Ferrara, Emilio. *Deep Neural Networks for Bot Detection.* Available at arXiv:1802.04289 [cs.AI], 2018.

[14] Amer, E., Kwak, K-S, & El-Sappagh, S. *Context-Based Fake News Detection Model Relying on Deep Learning Models.* Electronics, 11(8), 1255, 2022.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, & Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics, 2019.

[16] Sepp Hochreiter & Jürgen Schmidhuber. *Long Short-Term Memory.* Neural Computation, 9(8), 1735–1780, 1997.

[17] Ryan Walli. *Xtreme Margin: A Tunable Loss Function for Binary Classification Problems.* Available at arXiv:2211.00176.

[18] Tianqi Chen. *XGBoost: A Scalable Tree Boosting System.* Available at arXiv:1603.02754.

[19] University of New Brunswick. *TruthSeeker 2023 Dataset.* Centre for Innovation and Collaboration, University of New Brunswick, 2023. Available at: https://www.unb.ca/cic/datasets/truthseeker-2023.html.

[20] University of California, Santa Barbara. *Liar Dataset.* Computer Science Department, University of California, Santa Barbara. Available at: https://www.cs.ucsb.edu/william/data/liardataset.zip.

6 Appendices

```
Content Analysis of TruthSeeker
import pandas as pd

# Define the file paths
features_path = '/content/drive/MyDrive/TruthSeeker/
    Features_For_Traditional_ML_Techniques.csv'
truth_seeker_path = '/content/drive/MyDrive/TruthSeeker/
    Truth_Seeker_Model_Dataset.csv'

# Load the datasets
features_df = pd.read_csv(features_path)
truth_seeker_df = pd.read_csv(truth_seeker_path)

# Display first few rows
features_df.head(), truth_seeker_df.head()

# Rename the unnamed columns to 'ID' in both DataFrames for easier reference
features_df.rename(columns={features_df.columns[0]: 'ID'}, inplace=True)
truth_seeker_df.rename(columns={truth_seeker_df.columns[0]: 'ID'}, inplace=True)

# Drop duplicate columns ('tweet' and 'statement') from one of the DataFrames
features_df_dropped = features_df.drop(columns=['tweet', 'statement'])

# Perform the join operation using both 'ID' and 'BinaryNumTarget'
joined_df = pd.merge(features_df_dropped, truth_seeker_df, on=['ID', '
    BinaryNumTarget'])

# Display the first few rows of the joined DataFrame
joined_df.head()

# Descriptive statistics
joined_df.describe()

# Information about the dataframe
joined_df.info()

import matplotlib.pyplot as plt

# Count the occurrences of each unique value in the 'BinaryNumTarget' column
value_counts = joined_df['BinaryNumTarget'].value_counts()

# Create lists for the bar graph
labels = ['True', 'Fake']
counts = [value_counts.get(1, 0), value_counts.get(0, 0)]

# Create the bar graph
plt.figure(figsize=(10, 6))
plt.bar(labels, counts, color=['blue', 'red'])

# Add title and labels
plt.title('Distribution of True and Fake News Articles')
plt.xlabel('News Type')
plt.ylabel('Count')

# Add text annotations on each bar
for i, count in enumerate(counts):
    plt.text(i, count, str(count), ha='center')

```

```
# Show the plot
plt.show()

# Identify numerical and categorical columns
numerical_cols = joined_df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = joined_df.select_dtypes(include=['object']).columns

# Impute missing values in numerical columns with the column mean
for col in numerical_cols:
    joined_df[col].fillna(joined_df[col].mean(), inplace=True)

# Impute missing values in categorical columns with the column mode
for col in categorical_cols:
    joined_df[col].fillna(joined_df[col].mode()[0], inplace=True)

# Remove duplicates
joined_df.drop_duplicates(inplace=True)

import spacy

# Load the spaCy model
nlp = spacy.load("en_core_web_sm")

# Batch size for batch processing
batch_size = 2000

# Function for advanced text preprocessing
def advanced_preprocessing(doc):
    lemmatized = [token.lemma_ for token in doc]
    return ' '.join(lemmatized)

# Initialize an empty list to hold the preprocessed texts
preprocessed_texts = []

# Perform batch processing
for doc in nlp.pipe(joined_df['statement'].values, batch_size=batch_size):
    preprocessed_texts.append(advanced_preprocessing(doc))

# Add the preprocessed texts back to the DataFrame
joined_df['statement_advanced'] = preprocessed_texts

  from transformers import BertTokenizer, BertModel
  import torch
  import numpy as np

  # Initialize the BERT tokenizer and model
  tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
  model = BertModel.from_pretrained('bert-base-uncased')

  # Function to get BERT embeddings for a batch of text
  def get_bert_embeddings_for_batch(text_batch):
      inputs = tokenizer(text_batch, return_tensors="pt", padding=True,
    truncation=True, max_length=512)
      outputs = model(**inputs)
      embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().detach().cpu
    ().numpy()
      return embeddings

  # Batch size for BERT embeddings
  batch_size = 500
```

```
  # Initialize an empty list to hold the BERT embeddings
  bert_embeddings = []

  # Loop through the DataFrame in batches
  for i in range(0, len(joined_df), batch_size):
      text_batch = joined_df['statement_advanced'].iloc[i:i+batch_size].tolist()
      embeddings_batch = get_bert_embeddings_for_batch(text_batch)
      bert_embeddings.extend(embeddings_batch)

  # Convert the list of embeddings to a NumPy array
  bert_embeddings = np.array(bert_embeddings)

  # Add the BERT embeddings back to the DataFrame
  joined_df['bert_embeddings'] = list(bert_embeddings)

import os
import tensorflow as tf
from tensorflow.keras import layers

# Initialize TPU
resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='grpc://' + os.
    environ['COLAB_TPU_ADDR'])
tf.config.experimental_connect_to_cluster(resolver)
tf.tpu.experimental.initialize_tpu_system(resolver)

# Create a distribution strategy
tpu_strategy = tf.distribute.TPUStrategy(resolver)

# Model architecture with dropout and regularization
with tpu_strategy.scope():
    model = tf.keras.Sequential([
        layers.Input(shape=(768,)),  # BERT embeddings size
        layers.Reshape((1, 768)),
        layers.Bidirectional(layers.LSTM(50, return_sequences=True)),
        layers.Dropout(0.4),
        layers.Bidirectional(layers.LSTM(25)),
        layers.Dense(30, activation='relu', kernel_regularizer=tf.keras.
    regularizers.l2(0.01)),
        layers.Dense(1, activation='sigmoid')
    ])


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

import optuna
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Extract features and labels
X = np.stack(joined_df['bert_embeddings'].to_numpy())
y = joined_df['BinaryNumTarget'].values

# Compute class weights
unique_classes = np.unique(y)
class_weights = compute_class_weight('balanced', classes=unique_classes, y=y)
```

```
class_weights_dict = {i: w for i, w in enumerate(class_weights)}

def objective(trial):
    # Hyperparameters to be optimized
    lstm_units = trial.suggest_int('lstm_units', 20, 50)
    dense_units = trial.suggest_int('dense_units', 10, 30)
    dropout_rate = trial.suggest_float('dropout_rate', 0.4, 0.7)
    learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-3, log=True)
    l1_reg = trial.suggest_float('l1_reg', 1e-6, 1e-4, log=True)
    l2_reg = trial.suggest_float('l2_reg', 1e-6, 1e-4, log=True)

    # Initialize variables for k-fold cross-validation
    k = 5
    kf = StratifiedKFold(n_splits=k)
    val_accuracies = []

    for train_index, val_index in kf.split(X, y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        # Early stopping
        early_stop = EarlyStopping(monitor='val_loss', patience=3,
    restore_best_weights=True)

        # Model architecture with hyperparameters
        with tpu_strategy.scope():
            model = tf.keras.Sequential([
                layers.Input(shape=(768,)),
                layers.Reshape((1, 768)),
                layers.Bidirectional(layers.LSTM(lstm_units, return_sequences=
    True, kernel_regularizer=regularizers.l1_l2(l1=l1_reg, l2=l2_reg))),
                layers.Dropout(dropout_rate),
                layers.Bidirectional(layers.LSTM(lstm_units//2,
    kernel_regularizer=regularizers.l1_l2(l1=l1_reg, l2=l2_reg))),
                layers.Dense(dense_units, activation='relu', kernel_regularizer=
    regularizers.l1_l2(l1=l1_reg, l2=l2_reg)),
                layers.Dense(1, activation='sigmoid')
            ])

            opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
            model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
    accuracy'])

        history = model.fit(X_train, y_train, epochs=10, batch_size=128,
    validation_data=(X_val, y_val), callbacks=[early_stop], class_weight=
    class_weights_dict)

        val_accuracies.append(history.history['val_accuracy'][-1])

    return np.mean(val_accuracies)

# Initialize Optuna study
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=5)

from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras import layers, regularizers
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
```

```
import tensorflow as tf

# Extract features and labels for final evaluation
X = np.stack(joined_df['bert_embeddings'].to_numpy())
y = joined_df['BinaryNumTarget'].values

# Extract best parameters from Optuna study
best_params = study.best_params

# Initialize variables for Stratified K-Fold
k = 5
kf = StratifiedKFold(n_splits=k)
val_accuracies = []

for train_index, test_index in kf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Model architecture with hyperparameters from Step 4
    with tpu_strategy.scope():
        model = tf.keras.Sequential([
            layers.Input(shape=(768,)),  # BERT embeddings size
            layers.Reshape((1, 768)),
            layers.Bidirectional(layers.LSTM(best_params['lstm_units'],
    return_sequences=True, kernel_regularizer=regularizers.l1_l2(l1=best_params['
    l1_reg'], l2=best_params['l2_reg']))),
            layers.Dropout(best_params['dropout_rate']),
            layers.Bidirectional(layers.LSTM(best_params['lstm_units']//2,
    kernel_regularizer=regularizers.l1_l2(l1=best_params['l1_reg'], l2=
    best_params['l2_reg']))),
            layers.Dense(best_params['dense_units'], activation='relu',
    kernel_regularizer=regularizers.l1_l2(l1=best_params['l1_reg'], l2=
    best_params['l2_reg'])),
            layers.Dense(1, activation='sigmoid')
        ])

        # Using the Adam optimizer with the suggested learning rate
        opt = tf.keras.optimizers.Adam(learning_rate=best_params['learning_rate'
    ])

        model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
    accuracy'])

    # Early stopping callback with restore_best_weights
    early_stop = EarlyStopping(monitor='val_loss', patience=3,
    restore_best_weights=True)

    # Train the model
    history = model.fit(X_train, y_train, epochs=10, batch_size=64, callbacks=[
    early_stop], validation_split=0.1)

    # Model predictions
    y_pred = model.predict(X_test).flatten()
    y_pred_binary = np.round(y_pred)

    # Display classification report
    print("Classification Report:")
    print(classification_report(y_test, y_pred_binary))

    # Store the validation accuracy for this fold
```

```
    val_accuracies.append(history.history['val_accuracy'][-1])

# Print the mean validation accuracy across all folds
print("Mean Validation Accuracy: ", np.mean(val_accuracies))

from sklearn.metrics import confusion_matrix, roc_curve, auc,
    precision_recall_curve
import matplotlib.pyplot as plt
import seaborn as sns

# Model predictions
y_pred = model.predict(X_test).flatten()
y_pred_binary = np.round(y_pred)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_binary)
sns.heatmap(cm, annot=True, fmt='g')
plt.title('Confusion Matrix')
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' %
    roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred)
plt.figure()
plt.plot(recall, precision, color='b', lw=1, label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import tensorflow as tf

# Setting random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Split the data into training+validation and test sets, ensuring it's
    stratified.
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
```

```
)

# Training final model using X_train_val and y_train_val
with tpu_strategy.scope():
    final_model = tf.keras.Sequential([
        layers.Input(shape=(768,)),  # BERT embeddings size
        layers.Reshape((1, 768)),
        layers.Bidirectional(layers.LSTM(
            best_params['lstm_units'], return_sequences=True,
            kernel_regularizer=regularizers.l1_l2(l1=best_params['l1_reg'], l2=
    best_params['l2_reg'])
        )),
        layers.Dropout(best_params['dropout_rate']),
        layers.Bidirectional(layers.LSTM(
            best_params['lstm_units'] // 2,
            kernel_regularizer=regularizers.l1_l2(l1=best_params['l1_reg'], l2=
    best_params['l2_reg'])
        )),
        layers.Dense(
            best_params['dense_units'], activation='relu',
            kernel_regularizer=regularizers.l1_l2(l1=best_params['l1_reg'], l2=
    best_params['l2_reg'])
        ),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model
    opt = tf.keras.optimizers.Adam(learning_rate=best_params['learning_rate'])
    final_model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['
    accuracy'])

# Use Early stopping and class weights during training
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=
    True)

# Train the final model with a validation split and early stopping
history = final_model.fit(
    X_train_val, y_train_val, epochs=10, batch_size=64,
    validation_split=0.1, callbacks=[early_stop], class_weight=
    class_weights_dict
)

# Step 3: Evaluate the model on the test set
y_pred = final_model.predict(X_test).flatten()
y_pred_binary = np.round(y_pred)

# Calculate and print test accuracy
test_accuracy = accuracy_score(y_test, y_pred_binary)
print(f'Test Accuracy: {test_accuracy}')

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_binary))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_binary)
sns.heatmap(cm, annot=True, fmt="d")
plt.show()

# Plotting Training History
```

```
plt.figure()
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Context Analysis of TruthSeeker

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, learning_curve
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# Select relevant columns for Bot Behavior Analysis
feature_cols_bot = ['followers_count', 'friends_count', 'favourites_count', 'statuses_count', 'listed_count', 'following']
X_bot = joined_df[feature_cols_bot]
y_bot = joined_df['BotScore']

# Split the data into training and test sets
X_train_bot, X_test_bot, y_train_bot, y_test_bot = train_test_split(X_bot, y_bot, test_size=0.2, random_state=42)

# Initialize Random Forest Regressor
rf_model_bot = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_model_bot.fit(X_train_bot, y_train_bot)

# Make predictions on the test set
y_pred_bot = rf_model_bot.predict(X_test_bot)

# Evaluate the model (MSE and RMSE)
mse = mean_squared_error(y_test_bot, y_pred_bot)
rmse = np.sqrt(mse)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")

# Perform K-Fold Cross-Validation
cv_scores = cross_val_score(rf_model_bot, X_train_bot, y_train_bot, cv=10, scoring='neg_mean_squared_error')
print(f'Mean CV MSE: {-np.mean(cv_scores)}')
print(f'Standard Deviation of CV MSE: {np.std(cv_scores)}')

# Grid Search for Hyperparameter Tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, None]
}

grid_search = GridSearchCV(estimator=rf_model_bot, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
grid_search.fit(X_train_bot, y_train_bot)

# Generate and Plot Learning Curves
```

```
train_sizes, train_scores, val_scores = learning_curve(
    grid_search.best_estimator_, X_train_bot, y_train_bot, cv=5, scoring='neg_mean_squared_error',
    train_sizes=np.linspace(0.1, 1.0, 10), n_jobs=-1
)

train_mean = np.mean(-train_scores, axis=1)
train_std = np.std(-train_scores, axis=1)
val_mean = np.mean(-val_scores, axis=1)
val_std = np.std(-val_scores, axis=1)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label='Training error')
plt.plot(train_sizes, val_mean, label='Validation error')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='gray')
plt.fill_between(train_sizes, val_mean - val_std, val_mean + val_std, color='gainsboro')

plt.title('Learning Curve for Random Forest Regressor')
plt.xlabel('Number of training samples')
plt.ylabel('Mean Squared Error')
plt.legend(loc='best')
plt.grid()
plt.show()

# Store the BotScore predictions for the entire dataset in a new column
predicted_BotScore = grid_search.best_estimator_.predict(X_bot)
joined_df['predicted_BotScore'] = predicted_BotScore

# Calculate accuracy for bot detection
y_pred_binary = np.where(y_pred_bot >= 0.5, 1, 0)
y_test_binary = np.where(y_test_bot >= 0.5, 1, 0)

accuracy = np.mean(y_pred_binary == y_test_binary)
print(f'Accuracy for Bot Detection: {accuracy * 100:.2f}%')

from transformers import BertTokenizer, BertForSequenceClassification
from torch.nn import Softmax
import torch

# Initialize the BERT tokenizer and model for sequence classification
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('nlptown/bert-base-multilingual-uncased-sentiment')

# Softmax function for probabilities
softmax = Softmax(dim=1)

def get_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    probabilities = softmax(logits)
    sentiment_score = torch.argmax(probabilities)  # You can map this to your preferred range/scale
    return sentiment_score.item()

```

```
# Apply the function to the DataFrame
joined_df['sentiment_score'] = joined_df['tweet'].apply(get_sentiment)

# Show some results
print(joined_df[['tweet', 'sentiment_score']].head())

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import xgboost as xgb

# Preparing features and target variable
feature_cols = ['mentions', 'quotes', 'replies', 'retweets', 'favourites', '
    hashtags', 'sentiment_score']
X = joined_df[feature_cols]
y = joined_df['majority_target']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Initialize and train
xgb_model = xgb.XGBClassifier(n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)

# Predict and evaluate
y_pred_xgb = xgb_model.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f'Accuracy for predicting majority_target using XGBoost: {accuracy_xgb}')

# Store the predictions back into the DataFrame
joined_df['predicted_majority_target'] = xgb_model.predict(X)

# Preparing features and target variable
feature_cols = ['predicted_BotScore', 'sentiment_score', '
    predicted_majority_target']
X = joined_df[feature_cols]
y = joined_df['BinaryNumTarget']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Initialize and train the RandomForest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy for predicting Fake News in Context: {accuracy}')

from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix

# Already calculated accuracy
print(f'Accuracy: {accuracy}')

# Calculate and print other evaluation metrics
precision = precision_score(y_test, y_pred)
```

```
print(f'Precision: {precision}')

f1 = f1_score(y_test, y_pred)
print(f'F1 Score: {f1}')
```

Content Analysis of LIAR Dataset

```
pip install transformers
pip install keras-tuner
pip install tensorflow-addons

# Import required packages
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers import Bidirectional, LSTM
from tensorflow.keras.callbacks import EarlyStopping, Callback
from sklearn.model_selection import KFold
from transformers import BertTokenizer, BertModel
import torch
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from kerastuner import RandomSearch
import tensorflow_addons as tfa
from tensorflow.keras import regularizers


# Define the file paths
train_path = '/content/drive/MyDrive/LIAR Dataset/train.tsv'
valid_path = '/content/drive/MyDrive/LIAR Dataset/valid.tsv'
test_path = '/content/drive/MyDrive/LIAR Dataset/test.tsv'

# Load the datasets
liar_train_df = pd.read_csv(train_path, delimiter='\t', header=None)
liar_valid_df = pd.read_csv(valid_path, delimiter='\t', header=None)
liar_test_df = pd.read_csv(test_path, delimiter='\t', header=None)

# Rename the columns for easier reference
column_names = [
    'JSON_ID', 'Truth_Label', 'Statement_Text', 'Topic',
    'Speaker_Name', 'Speaker_Title', 'State_Info', 'Party_Affiliation',
    'Total_Credit_History_Count', 'False_Counts', 'Half_True_Counts',
    'Mostly_True_Counts', 'Pants_On_Fire_Counts', 'Context'
]
liar_train_df.columns = column_names
liar_valid_df.columns = column_names
liar_test_df.columns = column_names

# Data Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of truth labels
plt.figure(figsize=(10, 6))
sns.countplot(x='Truth_Label', data=liar_train_df)
plt.title('Distribution of Truth Labels in Training Dataset')
plt.xlabel('Truth Label')
plt.ylabel('Count')
plt.show()
```

```
# Identify numerical and categorical columns
numerical_cols = liar_train_df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = liar_train_df.select_dtypes(include=['object']).columns

# Impute missing values in numerical columns with the column median
for col in numerical_cols:
    median_value = liar_train_df[col].median()
    liar_train_df[col].fillna(median_value, inplace=True)
    liar_valid_df[col].fillna(median_value, inplace=True)
    liar_test_df[col].fillna(median_value, inplace=True)

# Impute missing values in categorical columns with the column mode
for col in categorical_cols:
    mode_value = liar_train_df[col].mode()[0]
    liar_train_df[col].fillna(mode_value, inplace=True)
    liar_valid_df[col].fillna(mode_value, inplace=True)
    liar_test_df[col].fillna(mode_value, inplace=True)

# Selecting the features for model training
textual_feature = 'Statement_Text'
numerical_features = ['Total_Credit_History_Count', 'False_Counts', 'Half_True_Counts', 'Mostly_True_Counts', 'Pants_On_Fire_Counts']
categorical_feature = 'Party_Affiliation'

# Subset the dataframes to include only the selected features
train_features_df = liar_train_df[[textual_feature] + numerical_features + [categorical_feature]]
valid_features_df = liar_valid_df[[textual_feature] + numerical_features + [categorical_feature]]
test_features_df = liar_test_df[[textual_feature] + numerical_features + [categorical_feature]]

# Initialize the BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Define batch size and maximum sequence length for BERT
batch_size = 100
max_length = 256

# Function to get BERT embeddings for a batch of text
def get_bert_embeddings_for_batch(text_batch):
    inputs = tokenizer(text_batch, return_tensors="pt", padding=True, truncation=True, max_length=max_length)
    outputs = model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().detach().cpu().numpy()
    return embeddings

# Initialize lists to hold the BERT embeddings for the training, validation, and test sets
bert_embeddings_train = []
bert_embeddings_valid = []
bert_embeddings_test = []

# Loop through the training DataFrame in batches to get embeddings
for i in range(0, len(train_features_df), batch_size):
    text_batch = train_features_df['Statement_Text'].iloc[i:i+batch_size].tolist()
```

```
    embeddings_batch = get_bert_embeddings_for_batch(text_batch)
    bert_embeddings_train.extend(embeddings_batch)

# Loop through the validation DataFrame in batches to get embeddings
for i in range(0, len(valid_features_df), batch_size):
    text_batch = valid_features_df['Statement_Text'].iloc[i:i+batch_size].tolist()
    embeddings_batch = get_bert_embeddings_for_batch(text_batch)
    bert_embeddings_valid.extend(embeddings_batch)

# Loop through the test DataFrame in batches to get embeddings
for i in range(0, len(test_features_df), batch_size):
    text_batch = test_features_df['Statement_Text'].iloc[i:i+batch_size].tolist()
    embeddings_batch = get_bert_embeddings_for_batch(text_batch)
    bert_embeddings_test.extend(embeddings_batch)

# Convert lists of embeddings to NumPy arrays
bert_embeddings_train = np.array(bert_embeddings_train)
bert_embeddings_valid = np.array(bert_embeddings_valid)
bert_embeddings_test = np.array(bert_embeddings_test)

# Use deep copy to avoid SettingWithCopyWarning
train_features_df_copy = train_features_df.copy()
valid_features_df_copy = valid_features_df.copy()
test_features_df_copy = test_features_df.copy()

# Scale numerical features
scaler = StandardScaler()
scaler.fit(train_features_df_copy[numerical_features])

train_features_df_copy.loc[:, numerical_features] = scaler.transform(
    train_features_df_copy[numerical_features])
valid_features_df_copy.loc[:, numerical_features] = scaler.transform(
    valid_features_df_copy[numerical_features])
test_features_df_copy.loc[:, numerical_features] = scaler.transform(
    test_features_df_copy[numerical_features])

# One-hot encode categorical features
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
encoder.fit(train_features_df_copy[[categorical_feature]])

train_categorical_encoded = encoder.transform(train_features_df_copy[[
    categorical_feature]])
valid_categorical_encoded = encoder.transform(valid_features_df_copy[[
    categorical_feature]])
test_categorical_encoded = encoder.transform(test_features_df_copy[[
    categorical_feature]])

# Convert to DataFrame
train_categorical_df = pd.DataFrame(train_categorical_encoded, columns=encoder.
    get_feature_names_out([categorical_feature]))
valid_categorical_df = pd.DataFrame(valid_categorical_encoded, columns=encoder.
    get_feature_names_out([categorical_feature]))
test_categorical_df = pd.DataFrame(test_categorical_encoded, columns=encoder.
    get_feature_names_out([categorical_feature]))


  # Concatenate BERT embeddings, scaled numerical features, and one-hot encoded
    categorical features
```

```
  train_final_features = np.hstack([bert_embeddings_train, train_features_df_copy[numerical_features].values, train_categorical_df.values])
  valid_final_features = np.hstack([bert_embeddings_valid, valid_features_df_copy[numerical_features].values, valid_categorical_df.values])
  test_final_features = np.hstack([bert_embeddings_test, test_features_df_copy[numerical_features].values, test_categorical_df.values])

  # Display the shape of the concatenated feature sets to confirm the operation
  print(f"Shape of final training features: {train_final_features.shape}")
  print(f"Shape of final validation features: {valid_final_features.shape}")
  print(f"Shape of final test features: {test_final_features.shape}")

# Updated Label Mapping
label_mapping = {
    'true': 1,
    'mostly-true': 0.7,
    'half-true': 0.5,
    'barely-true': 0.2,
    'false': 0,
    'pants-fire': -1
}

# Apply the new mapping to the DataFrame
liar_train_df['Truth_Label'] = liar_train_df['Truth_Label'].map(label_mapping)
liar_valid_df['Truth_Label'] = liar_valid_df['Truth_Label'].map(label_mapping)
liar_test_df['Truth_Label'] = liar_test_df['Truth_Label'].map(label_mapping)

# Update the labels arrays
train_labels = liar_train_df['Truth_Label'].values
valid_labels = liar_valid_df['Truth_Label'].values
test_labels = liar_test_df['Truth_Label'].values

# Keras Tuner Setup

## Define the feature size (number of columns) from the training features
feature_size = train_final_features.shape[1]

# Function to build model for Keras Tuner
def build_model(hp):
    model = tf.keras.Sequential()
    model.add(layers.Input(shape=(feature_size,)))
    model.add(layers.Dense(units=hp.Int('units', min_value=128, max_value=512, step=32), activation='relu'))
    model.add(layers.Dropout(rate=hp.Float('dropout', min_value=0.0, max_value=0.5, step=0.1)))
    model.add(layers.Reshape((hp.Int('units', min_value=128, max_value=512, step=32), 1)))
    model.add(Bidirectional(LSTM(hp.Int('lstm_units', min_value=32, max_value=128, step=32))))
    model.add(layers.Dense(1))
    optimizer = tfa.optimizers.AdamW(learning_rate=hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='LOG'), weight_decay=1e-5)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
    return model

tuner = RandomSearch(build_model, objective='val_loss', max_trials=5, executions_per_trial=2)
```

```
tuner.search(train_final_features, train_labels, epochs=5, validation_data=(
    valid_final_features, valid_labels))

# Get the best hyperparameters
best_hp = tuner.get_best_hyperparameters()[0]

# K-Fold Cross-Validation

n_splits = 5
kf = KFold(n_splits=n_splits)

val_loss_scores = []
val_mae_scores = []

for train_index, val_index in kf.split(train_final_features):
    X_train_fold, X_val_fold = train_final_features[train_index],
    train_final_features[val_index]
    y_train_fold, y_val_fold = train_labels[train_index], train_labels[val_index
    ]

    model = build_model(best_hp)

    early_stop = EarlyStopping(monitor='val_loss', patience=3,
    restore_best_weights=True)

    model.fit(
        X_train_fold, y_train_fold,
        epochs=20,
        batch_size=64,
        validation_data=(X_val_fold, y_val_fold),
        callbacks=[early_stop]
    )

    val_loss, val_mae = model.evaluate(X_val_fold, y_val_fold)
    val_loss_scores.append(val_loss)
    val_mae_scores.append(val_mae)

avg_val_loss = np.mean(val_loss_scores)
avg_val_mae = np.mean(val_mae_scores)

print(f"Average Validation Loss: {avg_val_loss}")
print(f"Average Validation MAE: {avg_val_mae}")

# Plot Training & Validation Loss Values vs Epoch
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss vs Epoch')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()

class TestCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_mae = []
        self.test_loss = []

    def on_epoch_end(self, epoch, logs=None):
```

```
        x, y = self.test_data
        loss, mae = self.model.evaluate(x, y, verbose=0)
        self.test_mae.append(mae)
        self.test_loss.append(loss)
        print(f'Test MAE: {mae}, Test Loss: {loss}')

# Initialize EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=
    True)

# Initialize custom TestCallback
test_callback = TestCallback((test_final_features, test_labels))

# Train the final model
final_model = build_model(best_hp)
history = final_model.fit(
    train_final_features, train_labels,
    epochs=20,
    batch_size=64,
    validation_data=(valid_final_features, valid_labels),
    callbacks=[early_stop, test_callback]
)

# Evaluate the model on the validation set
val_loss, val_mae = final_model.evaluate(valid_final_features, valid_labels,
    batch_size=64)
print(f"Validation Loss: {val_loss}")
print(f"Validation MAE: {val_mae}")

# Evaluate the model on the test set
test_loss, test_mae = final_model.evaluate(test_final_features, test_labels,
    batch_size=64)
print(f"Test Loss: {test_loss}")
print(f"Test MAE: {test_mae}")

plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Training and Validation MAE over Epochs')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.show()
```