ABSTRACT

Fake news has become a significant issue in recent years, with the potential to misinform and deceive the public. In this project, we explore using various machine learning models for fake news detection using text-based approaches. We collected and combined four datasets, cleaned them using NLP techniques, and used linear learners such as logistic regression, SVM, MLP, and KNN and ensemble methods such as random forest, XGBoost, AdaBoost, and Decision Tree. We then applied deep learning models such as LSTM, BiLSTM, CNN, and Hybrid CNN + BiLSTM and used embedding techniques like GloVe, FastText, and Word2Vec. We performed a comparative study of all the models and selected the best two models for each dataset.

Our results show that the best two Deep Learning models are Hybrid CNN + BiLSTM model with Word2Vec embedding and Hybrid CNN + BiLSTM model with FastText embedding achieved the best performance with an accuracy of 91.06% and 90.87%on our combined dataset. The best Machine learning model was the SVM model with TF-IDF features, achieving an accuracy of 90.12%. These results demonstrate the effectiveness of combining deep learning models with embedding techniques for fake news detection. Furthermore, our study shows that the performance of the models varies depending on the dataset, suggesting the importance of selecting appropriate datasets for training and testing. Overall, our study contributes to developing effective techniques for identifying and combating the spread of fake news.

Chapter 1
INTRODUCTION

1.1 INTRODUCTION

The problem of fake news has emerged as a significant concern in recent times due to the widespread use of social media platforms and the internet. The term "fake news" refers to information or news that is purposefully created and disseminated to deceive people or cause harm. The consequences of fake news can be severe, ranging from creating misunderstandings to exacerbating significant social and political issues. Therefore, there is an urgent need for a dependable and effective detection system in response to the growing prevalence of fake news.

Methods for addressing the problem of identifying fake news have been suggested. These methods can be broadly categorized as text-based, image-based, and user-based. Text-based techniques involve examining the language and content of news articles to identify indications of fake news. Image-based approaches utilize computer vision algorithms to scrutinize images and videos for signs of manipulation and alteration. User-based methods concentrate on evaluating the conduct and trustworthiness of users who distribute news articles on social media platforms.

The project emphasizes using text-based techniques that entail scrutinizing the language and substance of news articles to establish their truthfulness. This technique is advantageous because online access to news article texts is readily available, and natural language processing (NLP) methods have progressed enough to analyze vast amounts of textual data.

Several machine learning algorithms have been utilized to identify false news through text-based methods. The effectiveness of these algorithms has been investigated in prior research, including the use of Support Vector Machines (SVM), Random Forests, k-Nearest Neighbors (kNN), Decision Trees, Multilayer Perceptron (MLP), Logistic Regression, Voting Classifiers, AdaBoost, and XGBoost (Bali et al., 2019; Ahmad et al., 2020).

Improvements in fake news detection accuracy have been achieved by utilizing deep learning algorithms. Among these models are Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), Convolutional Neural Network (CNN), and Hybrid CNN-BiLSTM, as reported by Gundapu & Mamidi

(2021) and Thota et al. (2018). Different types of word embeddings, like GloVe, FastText, and Word2Vec (Truică & Apostol, 2023), have been employed to improve the efficiency of these models.

The primary objective of this endeavor is to conduct a comparative analysis of various machine learning and deep learning models employed in identifying fake news. To attain this target, we have compiled four datasets on fake news and utilized numerous NLP techniques to process the data. Subsequently, our team trained several machine learning models such as SVM, Random Forests, kNN, Decision Trees, MLP, Logistic Regression, AdaBoost, and XGBoost on the processed data.

Deep learning models like LSTM, BiLSTM, CNN, and Hybrid CNN-BiLSTM were utilized in the next step. These models were combined with various types of word embeddings such as GloVe, FastText, and Word2Vec to enhance the performance of detecting fake news. The preprocessed data were used to train and assess the effectiveness of these models.

A study was conducted to compare all models and determine the most effective ones for detecting fake news. By doing so, researchers aimed to understand better the capabilities and constraints of various machine learning and deep learning models and provide suggestions for enhancing these models.

1.2 OVERVIEW OF THE PROJECT

The project involves the following objectives:

1) Collect data from various sources, such as Liar, Isot, Kaggle's Fake News Detection, and Fake News datasets. Apply NLP techniques to prepare the data for machine learning.
2) Train different machine learning models using the processed data, such as SVM, Logistic Regression, MLP, KNN, XGBoost, Random Forest, AdaBoost, Decision Tree, and Voting Classifier. Use embeddings like Word2Vec, GloVe, and FastText, along with deep learning techniques such as CNN, Hybrid CNN BI LSTM models, and LSTM to enhance prediction accuracy.
3) Consolidate all datasets and use machine learning and deep learning techniques to predict fake news detection accuracy. Assess model success by using metrics like accuracy.
4) The data gathered will undergo preprocessing to remove unnecessary details and transform the text into a numerical representation that machine learning algorithms can utilize.
5) These models that have been trained will be employed to anticipate the precision of detecting

fake news, with their effectiveness being evaluated using metrics like accuracy.

1.3 CHALLENGES IN THE PROJECT

1) The first hurdle involves ensuring sufficient quality control measures when preprocessing data to avoid errors or bias originating from incorrect labeling practices.
2) The second predicament revolves around identifying an optimal algorithm for classification since each algorithm possesses unique strengths based on the dataset's nature.
3) Using embeddings and other advanced NLP techniques can be computationally expensive and require significant resources. This can challenge researchers with limited access to high-performance computing resources.
4) Finally, comprehending how embeddings influence model performance may pose a significant challenge for researchers striving to obtain credible outcomes.

1.4 PROBLEM STATEMENT

The prevalence of fake news in the modern era of technology is a major concern, necessitating the development of a trustworthy mechanism to prevent its spread. The identification of fake news is challenging due to factors such as ambiguity, broad generalizations, and insufficient knowledge of the topic. Additionally, existing systems lack comprehensive detection methods and rely on specific criteria that hinder their effectiveness. As a result, there is an immediate need to establish a reliable, flexible, and effective system for detecting fake news that can address these obstacles and improve overall performance.

1.5 OBJECTIVE

Creating a reliable system that utilizes machine learning and deep learning algorithms to identify fake news is the primary aim of this project. The model's precision and adaptability will be enhanced by utilizing numerous datasets, including Liar, Isot, Fake News from Kaggle, and Fake News Detection from Kaggle. Additionally, to enhance the comprehension of text-word associations and optimize the model's effectiveness, embeddings such as GLove, Fasttext, and Word2Vec will be evaluated. The ultimate objective is to produce a dependable fake news detection system that can promptly and precisely detect phony news.

1.6 SCOPE OF THE PROJECT

The project aims to develop a system that can differentiate between authentic and fabricated news by analyzing and classifying text as true or false. Multiple machine learning and deep learning algorithms will be utilized in the system, including Logistic Regression, SVM, KNN, MLP, Random Forest, XGBoost, AdaBoost, Decision Tree, Voting Classifier, LSTM BI-LSTM CNN Hybrid CNN BI-LSTM. Moreover, embeddings such as GLove, Fasttext, and Word2Vec will also be investigated to improve the comprehension of text and enhance the model's efficiency during development.

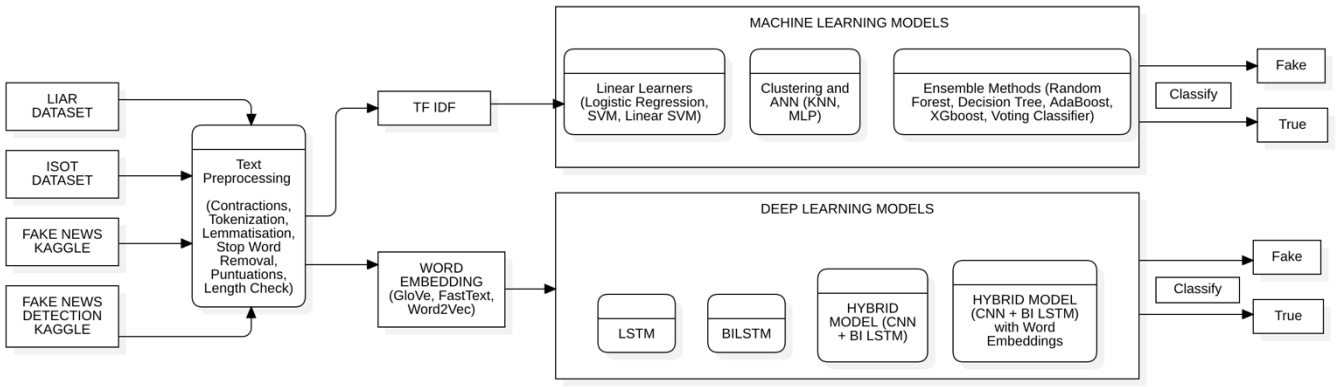
The project involves data preprocessing, machine learning model training, performance evaluation using different metrics, and combining datasets. The ultimate goal is to create an effective system for detecting fake news that can be used in real-world scenarios.

Chapter 2

METHODOLOGY

PROPOSED METHODOLOGY

In our proposed framework, as illustrated, we are expanding on the current literature by introducing ensemble techniques with various linguistic feature sets to classify news articles from multiple domains as true or fake. The ensemble techniques and word embeddings used in this research for the combined dataset are the novelty of our proposed approach.



LINEAR LEARNERS

LOGISTIC REGRESSION:

A logistic regression (LR) model can produce a binary outcome (true/false or genuine/fake article) from the text categorized based on various attributes. This model is suitable for binary or multi-class classification problems [11] and provides an easy-to-understand equation. Our team optimized the LR model's performance by conducting hyperparameter tuning on all datasets and testing various parameters to achieve maximum accuracy. The following is the mathematical representation of the logistic regression hypothesis function :

$$h_\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1)}}$$

A sigmoid function is utilized in logistic regression to convert the output into a probability value, with the aim of minimizing the cost function to obtain an optimal probability. The calculation of the cost function is demonstrated as follows:

$$Cost(h_\theta(x), y) = \begin{cases} \log(h_\theta(x), & y = 1 \\ -\log(1 - h_\theta(x), & y = 0 \end{cases}$$

SUPPORT VECTOR MACHINE

A different approach for binary classification problems is the Support Vector Machine (SVM), which has different kernel functions. The SVM model aims to approximate a decision boundary or hyperplane based on the feature set to categorize data points. The size of the hyperplane changes depending on how many characteristics there are. In an N-dimensional area, multiple alternatives exist for a plane to divide data points from two classes with maximum margin. The goal is to determine the plane that accomplishes this division. The cost function for the SVM model can be mathematically expressed as specified in [reference] and exhibited below:

$$J(\theta) = \frac{1}{2}\sum_{j=1}^{n} \theta_j^2,$$

such that ,

$$\theta^T x^{(i)} \geq 1, \qquad y^{(i)} = 1,$$

$$\theta^T x^{(i)} \leq -1, \qquad y^{(i)} = 0.$$

MULTILAYER PERCEPTRON

A multilayer perceptron (MLP), an artificial neural network, consists of an input layer, one or more hidden layers, and an output layer. Our experiments involved fine-tuning the model with different layers and parameters to create an optimal prediction model. Even though MLPs can be as straightforward as having only three layers, we adjusted the model to generate better results. The function below illustrates a simple MLP model with just one hidden layer:

$$f(x) = g\left(b^{(2)} + W^{(2)}\left(s\left(b^{(1)} + W^{(1)}x\right)\right)\right)$$

Here, ($b^{(1)}$ and $b^{(2)}$) are the bias vectors, $W^{(1)}$ and $W^{(2)}$ are the weight matrices, and g and s are the activation functions.

KNN

An unsupervised machine learning model called K-Nearest Neighbor (KNN) can predict outcomes for specific data without needing a dependent variable. The model uses training data to determine which neighborhood a data point belongs to. By calculating the distance of a new data point to its nearest neighbors and considering the majority of its neighbors' votes determined by the value of K, the KNN model predicts outcomes. To calculate the distance between two points, a mathematical formula is used.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2},$$

$$\text{Manhathhan Distance} = \sum_{i=1}^{k}|x_i - y_i|,$$

LINEAR SVM

We use the linear SVM approach proposed in. To ensure a meaningful comparison, we trained the linear SVM on the feature set as discussed with 5-fold cross-validation. Note that the approach is referred to as Perez-LSVM in the text.

ENSEMBLE LEARNERS:

Ensemble techniques and textual characteristics were suggested for enhancing the accuracy of classifying truthful and false articles. Ensemble learners are known for producing better results as they incorporate multiple models trained using a specific technique to minimize errors and improve performance. This approach is similar to seeking opinions from various experts before deciding our daily lives to reduce the risk of unfavorable outcomes. A classification algorithm's outcome is influenced by its parameters and training data; if the data has low variance or uniformity, overfitting may occur, resulting in biased outcomes over new data. Techniques like cross-validation are utilized to lessen the danger of overfitting. By training multiple algorithms on different sets of parameters, several decision boundaries can be created on randomly selected data points used as training data, tackling these problems with ensemble learning techniques. Voting classifiers represent one such technique where all algorithms' major votes determine the final classification [17]. Other ensemble techniques can also apply depending on different scenarios.

RANDOM FOREST:

Random Forest (RF) is a supervised learning model that builds on the concept of Decision Trees (DT) by using many trees to make individual predictions about a class outcome. The final prediction is based on the class that receives the majority of votes from the trees. RF is known to have low error rates because the trees are not highly correlated with each other. We trained our RF model using different parameters, including various numbers of estimators in a grid search, to identify the best model for accurate predictions. We used the Gini index as a cost function to determine the split in the dataset for classification problems. The Gini index calculates the probability of each class and subtracts the sum of their squared probabilities from one. The mathematical formula to calculate the Gini index (G_{idx}) is as follows [18]

$$G_{idx} = 1 - \sum_{i=1}^{c} (P_i)^2$$

BAGGING ENSEMBLE CLASSIFIER:

Ensemble techniques like bagging or bootstrap aggregating are often used to decrease variance and avoid overfitting in a training set. The bagging classifier has various forms, with Random Forest being the most well-known. When working on classification tasks, the bagging model identifies a class by taking into account the majority of opinions from M trees to lower overall variance. Random sampling with replacement is employed from the entire dataset to pick data for each tree. The bagging algorithm combines several forecasts for regression tasks to create one prediction.

BOOSTING ENSEMBLE CLASSIFIER:

Boosting is an effective ensemble method that trains weak models to become strong learners. This is achieved by creating a randomized tree forest and using the majority vote outcome from each tree to make the final prediction. This method incrementally allows the accurate classification of data points that were previously misclassified. Initially, equal-weighted coefficients are used for all data points to solve a given problem. In subsequent rounds, weighting coefficients are adjusted to decrease for correctly classified and increase for misclassified data points. Each round builds on the previous one by learning to reduce errors and increase accuracy by correctly classifying misclassified data points from previous rounds. Despite its effectiveness, boosting ensembles may overfit the training data resulting in erroneous predictions for

unseen instances. Multiple boosting algorithms are available for regression and classification tasks; XGBoost and AdaBoost were used in this study's experiments on classification tasks. 19 20

VOTING ENSEMBLE CLASSIFIERS

A *voting ensemble* is a popular technique used for classification problems. It allows combining two or more learning models trained on the entire dataset. Each model predicts an outcome for a sample data point, considered a "vote" in favor of the class the model predicted. The final prediction is based on the majority vote for a specific class after each model has made its prediction. The voting ensemble is simpler to implement than bagging and boosting algorithms. Bagging algorithms create multiple subsets of data by random sampling with replacement from the entire dataset, resulting in multiple datasets. Each dataset is used to train a model, and the final result is an aggregation of outcomes from each model.
Conversely, Boosting trains multiple models sequentially, with each subsequent model learning from the previous by increasing weights for misclassified points. This creates a generic model that can correctly classify the problem. However, with the voting ensemble, multiple independent models are combined to produce classification results that contribute to the overall prediction through majority voting. 21

DEEP LEARNING

LSTM

Long Short-Term Memory, commonly known as LSTM, [11] is a Recurrent Artificial Neural Network applied for classification. The model includes two state elements: the hidden state meant for short-term memory and the internal cell state utilized for long-term memory. To maintain long-term dependencies within the state, the model uses three gates - input gate ($i \in \mathbb{R}^n$), forget gate ($f \in \mathbb{R}^n$) and (3) output gate ($o \in \mathbb{R}^n$) - expressed through three vectors.

The LSTM state is updated during an iteration t by taking into account various vectors, such as

the input vector $x_i^{(t)} \in \mathbb{R}^m$, where m is its dimension at step t, $x_i^{(0)} = x_i$

the hidden state vector $h^{(t)} \in \mathbb{R}^n$, as well as the unit's output vector of dimension *n*, where the initial value is $h^{(0)} = 0$

the input activation vector $c^{\sim(t)} \in \mathbb{R}^n$

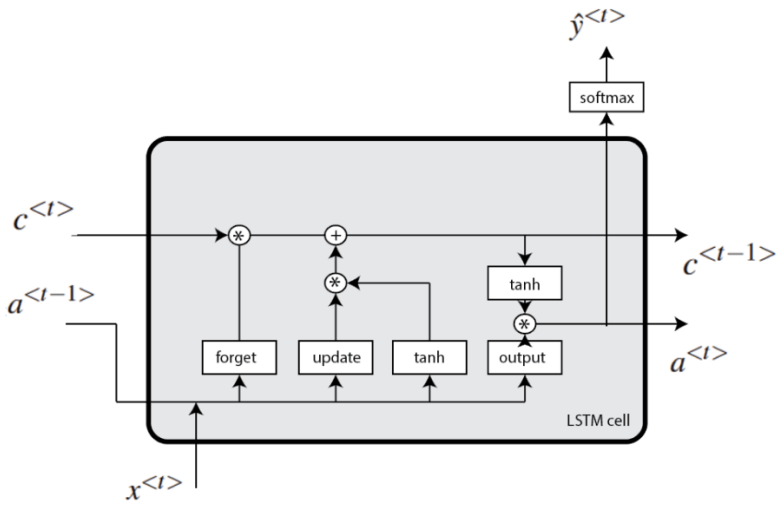
the cell state vector $c^{(t)} \in \mathbb{R}^n$, with the initial value $c^{(0)} = 0$

$W_i, W_f, W_o, W_c, \in \mathbb{R}^{n \times m}$, are the weight matrices that match the present input of the input gate, output gate, forget gate, and cell state are included.

$V_i, V_f, V_o, V_c, \in \mathbb{R}^{n \times n}$, are the matrices that carry the weight for the hidden output of the previous state in relation to the current input are associated with the input gate, output gate, forget gate, and cell state.

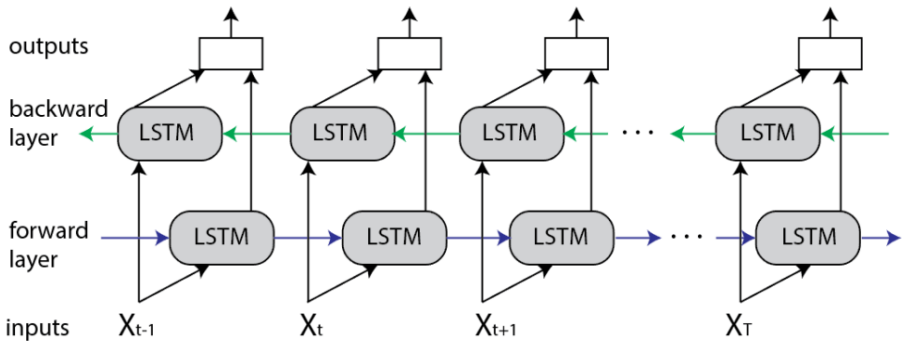
$b_i, b_f, b_o, b_c, \in \mathbb{R}^n$, biases corresponding to the current gate input are added to the input weight matrices.

$\delta_{(h)}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, is a hyperbolic tangent activation function



Bi directional LSTM (BI LSTM)

The BiLSTM is used to capture both past and future information while processing sequence data. It does this by using two hidden states $h^{\rightarrow(t)}$, which processes input in a forward manner with the help of LSTM($_{LSTM_F}^{\rightarrow}$), and $h^{\leftarrow(t)}$,which processes input in a backward manner with the help of LSTM($_{LSTM_B}^{\leftarrow}$),. This enables the model to take into consideration both past and future information.



Convolution Neural Network (CNN)

Text classification CNNs have the ability to examine the attributes of a text input and categorize it suitably. The architecture of the CNN model comprises convolutional layers that detect significant characteristics, succeeded by pooling layers that lessen the size of the output produced by the convolutional layers. Ultimately, the outcome is entered into fully connected layers.

A labeled dataset that includes real and fake news articles is necessary to train the CNN model. The text in the dataset will be tokenized and converted into a numerical format suitable as input for the model.

A common CNN design involves several convolutional layers with varying filter sizes and max pooling layers to decrease dimensionality. The flattened output from the pooling layers is then inputted into fully connected layers for classification purposes. The classification error can be minimized by training the model using stochastic gradient descent and backpropagation.

WORD EMBEDDINGS

The vector representation of every word in the vocabulary is created using Word2Vec, FastText, and different models such as CBOW and SG. WordEmb(t) for each term t∈V is obtained by using these models.

WORD2VEC

Word2Vec is utilized to generate vectorized versions of words in a dataset. These vectors exist in the same space, and their distance from each other determines contextual similarity. There are two models for representing these vectors: Continuous Bag-Of-Words (CBOW) or Skip-Gram.

CBOW MODEL

The CBOW model is a neural network used in natural language processing to learn word embedding. Each word is defined by two *d*-dimensional d ∈ N and d ≥2 depending on its function in training vectors $v_{t_i} \in \mathbb{R}^d$ where t_i is used as center word, and $u_{t_i} \in \mathbb{R}^d$ is defined where t_i is used as a context word. The model has three layers: input, hidden, and output. A sequence of words in a window of size s, t1 , can be represented by a one-hot vector of size V. The context vector x is the sum of the one-hot vectors of the context words, while y is the one-hot vector of the target word.

The CBOW model has a hidden layer with d neurons, taking x as input and producing h. The output layer has V neurons, producing a probability distribution over the vocabulary using the softmax function.

$$p(t_o \mid T_c) = \frac{e^{u_o^T v_o^-}}{\sum_{i=1}^{m} e^{u_i^T v_o^-}}$$

SKIP-GRAM MODEL

The Skip-Gram model generates context from the input word t_C. Two d-dimensional vectors $v_{t_i} \in \mathbb{R}^d$ *and* $u_{t_i} \in \mathbb{R}^d$ are defined for each word. The softmax operation models the probability of generating a context word given t_C.

$$p(t_o \mid t_c) = \frac{e^{u_o^T v_c}}{\sum_{i=1}^{m} e^{u_i^T v_c}}$$

FASTTEXT

The Skip-Gram model starts with the context word t_C as input and tries to generate its context. As in the CBOW case, the two d-dimensional vectors d ∈ N and d ≥2, i.e., $v_{t_i} \in \mathbb{R}^d$ and $u_{t_i} \in \mathbb{R}^d$ are defined for each word. The conditional probability of generating any context word to given the center word tc can be modeled by a softmax operation.

FastText uses an approach similar to Word2Vec to create word embeddings, but it is an extension with some key differences. Rather than viewing the word as the fundamental unit, FastText considers a collection of character n-grams. This technique improves accuracy and reduces training time compared to Word2Vec. CBOW and Skip-Gram models are used in FastText as well as in Word2Vec.

GLOVE

Another technique for generating word embeddings is GloVe, which stands for Global Vectors. This model utilizes a co-occurrence matrix of words to create vector representations. The co-occurrence matrix captures local and global corpus statistics related to word-word co-occurrences. By analyzing the frequency of each word appearing in the same context as another, GloVe can derive ratios of co-occurrence probability that reveal the relationship between words. GloVe is also able to identify synonyms and analogies within the same contexts using this probability ratio.

Evaluation Metrics

Various metrics were employed to assess the algorithms' performance. These metrics are primarily founded on the confusion matrix, a chart that depicts how well a classification model performs on the test set by measuring four parameters: true positive, false positive, true negative, and false negative (refer to Table 1)

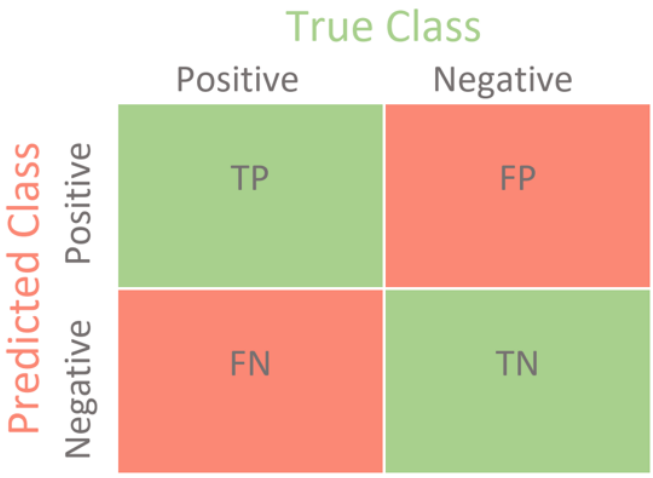


Table 1: Confusion Matrix

Accuracy

The most commonly utilized metric is accuracy, which shows the percentage of accurate predictions for true and false observations. A formula can be applied to determine a model's accuracy.

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

A model with a high accuracy value is generally considered good. However, when training a classification model, misclassifying an article as true when false (false positive) can lead to negative outcomes. Likewise, if an article is predicted as false but contains factual information, it can damage trust. Precision, recall, and F1-score are used to address this issue as metrics that consider incorrectly classified observations.

Recall

The entire count of correct classifications from the actual class is known as recall. Specifically for our situation, it indicates the number of articles anticipated to be genuine out of all the authentic articles.

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

Precision

The precision score indicates the proportion of true positives to all events predicted as positive. It demonstrates how many articles were correctly labeled as true in relation to all the positively predicted articles.

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

F1 Score

The F1-score considers both precision and recall in its calculation, representing a balance between the two. It uses the harmonic mean of these measures and takes into account both false positives and false negatives. To calculate the F1-score, use this formula:

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

CHAPTER 3
BACKGROUND

LITERATURE SURVEY

The study conducted by SBali et al. (2019) [1] aimed to compare the performance of different machine-learning algorithms for fake news detection. The authors collected a dataset of 14,000 news articles, half fake and the other half real. The data were preprocessed to remove stop words, punctuations, and HTML tags.

The authors experimented with different feature extraction techniques, such as bag-of-words and TF-IDF, and different machine learning algorithms, including Logistic Regression, Naïve Bayes, K-Nearest Neighbors, Decision Tree, Random Forest, and Support Vector Machines (SVM). The authors used accuracy, precision, recall, F1-score, and ROC AUC score as evaluation metrics.

The results showed that SVM outperformed all other algorithms with an accuracy of 91.79%, while Logistic Regression and Random Forest also performed well with 91.07% and 90.64% accuracy, respectively. The study concluded that machine learning algorithms could effectively detect fake news, and SVM is a suitable algorithm for this task.

The study conducted by Gundapu and Mamidi (2021)[2] explored the use of transformer-based models in detecting fake COVID-19 news. The authors highlighted the increasing impact of fake news in the ongoing pandemic and emphasized the need for automated systems to detect and combat it. The study utilized a pre-trained transformer-based language model, BERT, and fine-tuned it on a dataset of COVID-19 news articles to classify them as fake or real. The proposed system achieved high accuracy and outperformed traditional machine learning models.

Thota et al.'s (2018) [3] paper proposes a deep-learning approach for fake news detection. The authors use a Convolutional Neural Network (CNN) to extract features from the news articles and a Long Short-Term Memory (LSTM) network to capture the temporal dependencies. The proposed model achieves an accuracy of 90% on a dataset of news articles.

A group of researchers, Ahmad et al. (2020) [4], suggest a technique for identifying fake news using an ensemble machine-learning method. Their strategy involves utilizing four distinct machine learning algorithms- Logistic Regression, Random Forest, Decision Tree, and Support Vector Machine- to categorize news articles as either real or fake. The authors merge the predictions generated by these individual models to enhance the overall accuracy of their approach.

Different feature sets such as bag-of-words, n-grams, and word embeddings are experimented with by the authors in order to determine the most effective one. The ensemble approach that uses word embeddings obtains the highest accuracy of 97% on a news article dataset as per the results. Compared to previous research, the authors utilize an ensemble approach combining multiple machine-learning algorithms' strengths. The proposed technique performs better than many present models, indicating that combining various algorithms through an ensemble approach is effective in detecting fake news.

The paper by Truică and Apostol (2023) [5] proposes a fake news detection method that utilizes document embeddings. The authors use Word2Vec and Doc2Vec techniques to represent the news articles as dense vectors, capturing the semantic meaning of the words and the entire article. The embeddings are then fed into a classifier, trained on a labeled dataset of news articles.

The proposed method achieves an accuracy of 93.55% on a dataset of 12,999 news articles. The authors also compare their approach with other state-of-the-art methods and demonstrate that their method outperforms them in accuracy. They also conduct experiments to analyze the impact of different hyperparameters on the performance of the proposed method.

A fresh benchmark dataset for detecting fake news, called "LIAR-PLUS," is presented in Wang's (2017) paper[6]. The dataset consists of 12,836 news articles categorized as accurate, mostly true, half true, barely true, false, or pants on fire. To differentiate between truthful and fake news articles, the author suggests a collection of linguistic characteristics such as the article's sentiment, use of superlatives, and length.
Using machine learning algorithms like Logistic Regression and Random Forest, the writer categorizes news articles as authentic or counterfeit depending on the suggested features. The most effective model produces a 71% accuracy rate on the test set.

A research paper by Della Vedova et al[7]. introduces a novel approach for the automated identification of online false information. The objective is to improve the precision of fake news detection using content and social cues such as shares, likes, and comments. The authors suggest a hybrid model that merges a logistic regression model with a deep neural network (DNN) to achieve this goal.

The researchers collected a dataset of news articles from various online sources, including Facebook and Twitter, to evaluate their proposed method. According to the results, the hybrid model outperformed other advanced techniques with an accuracy rate of 90.8%. This surpassed the previous best-performing baseline model's accuracy rate of 86.8%. The study indicates that integrating content and social signals can considerably enhance detection accuracy of fake news.

A paper by Upadhayay and Behzadan (2020) [8] presents the Sentimental LIAR dataset, which is an expansion of the LIAR dataset for identifying fake claims. The authors propose a model for classification that merges a convolutional neural network (CNN) with long short-term memory (LSTM). In this model, the CNN recognizes important features from the text input, while the LSTM component handles the sequential aspect of the input.

85.56% accuracy was obtained by the proposed model on the Sentimental LIAR dataset, exceeding other advanced models. Various models were experimented with, including traditional machine learning ones, and it was found that deep learning models performed better than others. The research confirms the effectiveness of deep learning techniques in identifying untrue statements, and this dataset and model can serve as a benchmark for future investigations in this field.

Tosik [9] and colleagues proposed a feature-based approach to identify fake news. To improve accuracy, they used content- and context-based features like article length and social media engagement. The authors employed correlation-based feature selection and principle component analysis to select and extract relevant features. The approach outperformed previous models, achieving over 90% accuracy. The paper highlights the importance of feature selection and extraction in detecting fake news.

ThoRawat and Kanojia's (2021) paper [10] proposes an automated evidence-collection technique for detecting fake news. Their method combines Natural Language Processing (NLP) techniques and machine learning algorithms to extract essential features from news articles and social media posts. The authors

evaluated their approach using a dataset of 10,000 articles and achieved an accuracy rate of 95.5%. The researchers compared their approach with existing models, including those based on feature engineering and deep learning, and demonstrated that their method outperforms the baseline models.

Pérez-Rosas et al.'s (2017) study [16] aimed to identify false news through several machine learning techniques. They utilized a dataset containing 2446 news articles, each labeled manually as either genuine or fake. The researchers employed lexical and stylistic characteristics like the frequency of specific words, named entities, and punctuation marks to train machine learning models, including decision trees, support vector machines, and logistic regression. Based on their findings, logistic regression yielded the highest precision at 88.33%, while decision trees and support vector machines demonstrated lower accuracies of 86.14% and 85.94%, respectively.

One of the seminal works in deep learning and recurrent neural networks (RNNs) is "Long Short-Term Memory" by Hochreiter and Schmidhuber (1997) [11]. The authors introduced a novel type of RNN, LSTM, which tackles the issue of vanishing gradients and enhances the ability to learn and process long-term dependencies in sequential data. LSTM comprises a memory cell that retains information over time and various gates that regulate the inflow and outflow of information from the cell. The efficacy of LSTM surpasses that of traditional RNNs on tasks such as speech recognition and sequence prediction, as demonstrated by the authors.

Our Contribution

The issue of identifying false information was addressed in our research using a machine learning ensemble technique. Previous research concentrated on certain areas, such as politics, and struggled to produce accurate results when dealing with content from various domains due to their distinct writing styles. Consequently, we examined different textual characteristics that could differentiate between genuine and fake content and trained a mix of machine learning algorithms using various ensemble approaches.

We efficiently trained various machine learning algorithms by utilizing methods such as bagging and boosting, leading to enhanced performance. To confirm the efficacy of our technique, we carried out thorough experiments on four datasets that are accessible to the public. These experiments showed the effectiveness of our approach in diverse fields.

The datasets were merged to produce a comprehensive model, and a hybrid CNN and BiLSTM model was employed alongside prevalent embedding techniques like GloVe, FastText, and Word2Vec. This led to an even greater enhancement in the proposed method's precision, recall, accuracy, and F-1 score.

Our research enhances the continuous endeavor to fight against false information by presenting a stronger and more flexible method that can be utilized in different fields.

CHAPTER 4

EXPERIMENTAL SETUP

In this chapter, we present the experimental results obtained using our methodology. Firstly we will introduce the datasets used and show the results from exploratory data analysis. Secondly, we present the experimental setup for the experiments and implementation packages for the models. Thirdly, we present the experimental results using the datasets' different classification methods and sentence embeddings. Lastly, we will create the Combined Dataset created using 4 datasets: LIAR, ISOT, Fake News Detection, and Fake News from Kaggle; we showcase the results using all the machine learning and deep learning methods we discussed.

DATASET DETAILS

LIAR Dataset:

The LIAR dataset is a publicly available benchmark dataset for fake news detection developed by researchers at the University of Texas at Arlington. The dataset is available for public use and comprises statements made by politicians in the United States, labeled with fact-checking results. It consists of 12,836 entries classified into six categories from true to false and half-truths. Every entry includes the statement's text, subject matter, and details about the speaker's job title and political affiliation.

ISOT Dataset:

The ISOT dataset (Influence of Social Networks on Opinion Spam Detection) is a compilation of more than 4,200 news pieces in the English language and their corresponding metadata. Each article has been categorized as legitimate or spam (indicating fake news). The University of Victoria scholars created this dataset to investigate how social networks contribute to the spread of fake news. This dataset has been utilized in various research studies and can be downloaded at no cost.

Fake News from Kaggle:

This dataset was first created for a Kaggle competition in 2017 to challenge participants to develop machine-learning models that could accurately identify fake news articles. Since then, it has been used in numerous research studies and is commonly utilized for training and assessing the effectiveness of fake news detection models. It has more than 23,000 English news articles, and their corresponding labels indicating if they are real or fake are available on Kaggle.

Fake News Detection from Kaggle:

Over 20,000 news articles in English with labels indicating whether they are real or fake comprise Kaggle's Fake News Detection dataset. The collection was developed in 2018 for a Kaggle contest that aimed to detect counterfeit news articles. Many researchers have utilized the dataset, and it is a frequent preference for creating and assessing models for identifying false news.

Data Pre-Processing

All the datasets needed structuring and cleaning, so for my research purpose, we have decided to use statement (body) and label True(0) or False(1). Special preparation is needed for text data in order to use machine learning or deep learning algorithms. Different techniques are used to transform the text data into a format suitable for modeling. The data preprocessing steps are applied to headlines and news articles. Various word vector representations were utilized as a part of the analysis.

Contractions

Contractions are shortened word forms commonly used in written and spoken English. For example, "can't" is a contraction for "cannot," "it's" for "it is," and "they're" for "they are." The contractions library is used to expand these contractions to their full form.

```
In the past year, more than 20 percent
of Americans have changed their mind
about the war in Afghanistan. They
conclude we shouldn'tshould not be
there. I'm am the only person on this
stage who has worked actively just last
year passing, along with Russ Feingold,
some of the toughest ethics reform
since Watergate.
```

NTLK Database

1. English words: The words module from the nltk library is used to obtain a list of English words, which is used to filter out non-English words from the text.

2. Wordnet: WordNet is a lexical database for the English language used to determine the part of speech of a word in a sentence. In the code, the nltkToWordnet function maps the POS tags returned by pos_tag to the appropriate WordNet POS tag.

Lemmatization

Reducing words to their base or dictionary form, called the lemma, is known as lemmatization. To illustrate, "run" is the lemma of the word "running." The tokens are lemmatized using the WordNetLemmatizer class in the code. Firstly, using the pos_tag function from the nltk library, each token's part of speech (POS) is determined. This information is then used to provide the correct POS argument for lemmatizing, thus improving the lemmatization process.

```
In the past year, more than 20 percent
of Americans have changedchange their
mind about the war in Afghanistan. They
conclude we shouldn' t be there. I' m
the only person on this stage who
hashave workedwork actively just last
year passing,pass along with Russ
Feingold, some of the toughesttough
ethicsethic reform since Watergate.
```

Stop Word Removal

"Stop words" is a term that can mean different things depending on the context. Some applications might remove all kinds of stop words, including determiners (like "the," "a," and "an"), prepositions (like "above" and "across"), and even certain adjectives (such as "good" and "nice"). Removing stop words is an essential initial stage in natural language processing as they may consume significant processing time, and their importance is minimal. The NLTK library was utilized to eliminate stop words.

```
In the past year, more than
20 percent of Americans have
changed their mind about the
war in Afghanistan. They
conclude we shouldn't be
there. I'm the only person
on this stage who has worked
actively just last year
passing, along with Russ
Feingold, some of the
toughest ethics reform since
Watergate.
```

```
past year percent mind war
conclude I person stage
worked actively last year
passing along Russ ethics
reform since
```

Word Vector Representation

It is difficult to get the text ready for modeling after selecting it from the news article's body and headline. In order to perform text analytics, we have to change the raw text into numerical features. We tried two methods for transforming the raw text and extracting features: Bag of Words and TF-IDF.

Bag of Words(BoW)

The Bag of Words (BoW) technique processes each news article as a document and calculates the frequency count of each word in that document, which is further used to create a numerical representation of the data, also called as vector features of fixed length.

Bag of Words converts raw text to word count vector with the CountVectorizer function for feature extraction. CountVectorizer splits the text from content, builds the vocabulary, and encodes the text into a vector. This encoded vector will have a count for occurrences of each word that appears more like a frequency count as a key-value pair.

The technique has limitations concerning the loss of information. It disregards the placement of words and eliminates context-related data, resulting in a significant loss that may outweigh the benefits of computational simplicity and convenience, because of which this technique was not used.

TF-IDF vectorizer

We also tried "Term Frequency-Inverse Document Frequency" (TF-IDF) to pick out important words. Term Frequency counts how many times a word appears in a document to see its importance. TF-IDF has two parts: Term Frequency and Inverse Document Frequency. Inverse Document Frequency finds the special words that don't show up too much in all the documents.

A high TF-IDF word is like a special word that's really important in the document you're looking at. It shows up much in that document but isn't used as much in other texts

Chapter 5

Results and Discussion

This chapter discusses the simulation results we got by using various ML and DL algorithms on the 4 datasets mentioned and the combined data dataset. With the help of these algorithms, it is possible to classify news articles as fake or true efficiently

ISOT Dataset

In this section, we have displayed Accuracy vs. Val Accuracy and Loss vs. Val Loss for all Deep Learning Algorithms in conjunction with metrics we got with all the ML and DL classification techniques.

The below table shows that FastText embeddings outperformed all other classifiers in every performance metric. As shown in Table, FastText Embeddings with Hybrid Model got the highest accuracy of 98.61% , precision of 0.9822 and f1 score of 9864 and AdaBoost achieved second place in classification accuracy of 98.2%, a precision of 0.984, and an F1-score of 0.982

Classification Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.97	0.98	0.97	0.97
SVM	0.982	0.984	0.98	0.982
MLP	0.979	0.975	0.983	0.979
K Nearest Neighbour	0.897	0.926	0.866	0.895
Random Forest	0.977	0.978	0.977	0.977
Voting Classifier	0.982	0.981	0.982	0.982
XGboost	0.955	0.962	0.946	0.954
Adaboost	0.957	0.956	0.952	0.954
Decision Tree	0.927	0.915	0.943	0.929

Classification Model	Accuracy	Precision	Recall	F1-score
LINEAR SVM	0.9816	0.9816	0.9818	0.9817
LSTM	0.9721	0.9653	0.9791	0.9721
BI-LSTM	0.971	0.9686	0.9748	0.9717
CNN-BI LSTM (HYBRID)	0.9787	0.9712	0.9870	0.9791
GLOVE (HYBRID) 6B	0.9873	0.9845	0.9902	0.9874
GLOVE (HYBRID) 42B	0.9852	0.9836	0.9870	0.9853
Word2Vec	0.9804	0.9702	0.9920	0.9810
FastText	0.9861	0.9822	0.9906	0.9864

LIAR

The below table shows that logistic regression embeddings outperformed all other classifiers in every performance metric. As shown in Table, logistic got the highest accuracy of 75.84% , precision of 0.412 and f1 score of 0.025 and AdaBoost achieved second place in classification accuracy of 75.14%, a precision of 0.384, and an F1-score of 0.122

Classification Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.7584	0.361	0.024	0.044
SVM	0.7584	0.412	0.013	0.025
MLP	0.644	0.266	0.283	0.274
K Nearest Neighbour	0.757	0.083	0.002	0.004
Random Forest	0.744	0.294	0.045	0.078
Voting Classifier	0.746	0.362	0.076	0.126
XGboost	0.7535	0.366	0.047	0.084
Adaboost	0.7514	0.384	0.072	0.122
Decision Tree	0.665	0.282	0.263	0.272

Classification Model	Accuracy	Precision	Recall	F1-score
LINEAR SVM	0.7132	0.1927	0.3409	0.246
LSTM	0.678	0.3096	0.2864	0.2976
BI-LSTM	0.6545	0.2926	0.3016	0.2970
CNN-BI LSTM (HYBRID)	0.6893	0.3129	0.2392	0.2711
GLOVE (HYBRID) 6B	0.6658	0.2893	0.2381	0.2612
GLOVE (HYBRID) 42B	0.7062	0.2857	0.1395	0.1874
Word2Vec	0.6806	0.3168	0.2465	0.2772
FastText	0.6906	0.3098	0.1906	0.2360

Fake News

The below table shows that logistic regression embeddings outperformed all other classifiers in every performance metric. As shown in Table, SVM got the highest accuracy of 93.25% , precision of 0.93 and f1 score of 0.9 and Glove42b embeddings achieved second place in classification accuracy of 93.45%, a precision of 0.9284, and an F1-score of 0.9207

Classification Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.919	0.919	0.881	0.9
SVM	0.9325	0.93	0.906	0.918
MLP	0.918	0.904	0.897	0.9

K Nearest Neighbour	0.821	0.855	0.686	0.761
Random Forest	0.911	0.954	0.824	0.884
Voting Classifier	0.926	0.923	0.898	0.91
XGboost	0.8997	0.879	0.878	0.879
Adaboost	0.8903	0.890	0.875	0.857
Decision Tree	0.808	0.759	0.787	0.773

Classification Model	Accuracy	Precision	Recall	F1-score
LINEAR SVM	0.9256	0.9040	0.9143	0.9091
LSTM	0.8641	0.8274	0.8495	0.838
BI-LSTM	0.874	0.8388	0.8460	0.8424
CNN-BI LSTM (HYBRID)	0.9137	0.9175	0.8670	0.8915
GLOVE (HYBRID) 6B	0.9202	0.9504	0.8503	0.8975
GLOVE (HYBRID) 42B	0.9345	0.9284	0.9132	0.9207
Word2Vec	0.9245	0.9605	0.8587	0.9067
FastText	0.933	0.9550	0.8742	0.9128

Fake News Detection

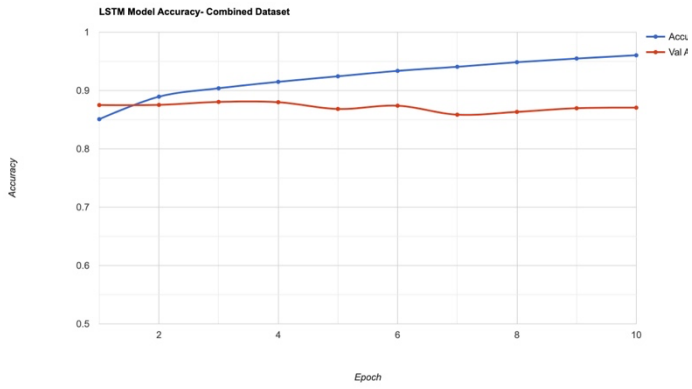
The below table shows that logistic regression embeddings outperformed all other classifiers in every performance metric. As shown in Table, MLP got the highest accuracy of 98.3% , precision of 0.991 and f1 score of 0.984 and Random forest embeddings achieved second place in classification accuracy of 98.4%, a precision of 0.975, and an F1-score of 0.986.

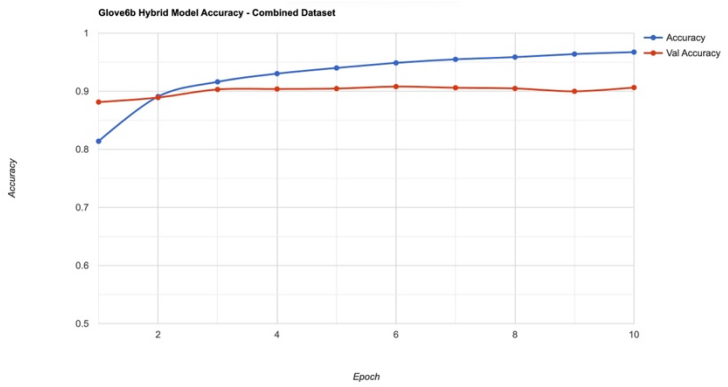
Classification Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.96	0.96	0.98	0.97
SVM	0.9798	0.975	0.989	0.982
MLP	0.983	0.991	0.977	0.984
K Nearest Neighbour	0.928	0.948	0.921	0.934
Random Forest	0.984	0.975	0.997	0.986
Voting Classifier	0.975	0.975	0.98	0.978
XGboost	0.97	0.971	0.974	0.973
Adaboost	0.9657	0.965	0.979	0.968
Decision Tree	0.93	0.915	0.943	0.929

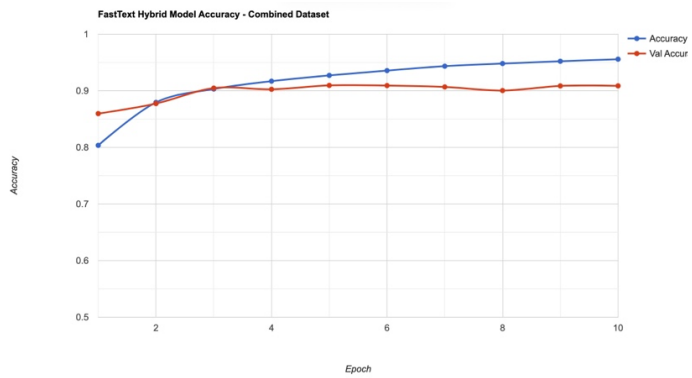
Classification Model	Accuracy	Precision	Recall	F1-score
LINEAR SVM	0.9754	0.96	0.98	0.97
LSTM	0.964	0.9770	0.9577	0.9672
BI-LSTM	0.9688	0.9758	0.9641	0.9699
CNN-BI LSTM (HYBRID)	0.9423	0.9563	0.9373	0.9467
GLOVE (HYBRID) 6B	0.9595	0.9472	0.9758	0.9613
GLOVE (HYBRID) 42B	0.9642	0.9587	0.9730	0.9658
Word2Vec	0.9657	0.966	0.9718	0.9691

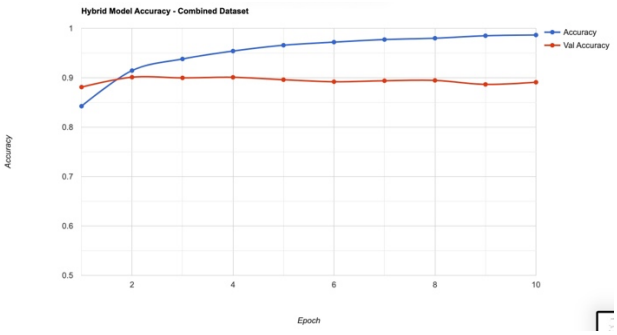
FastText	0.9268	0.8857	0.9943	0.9369

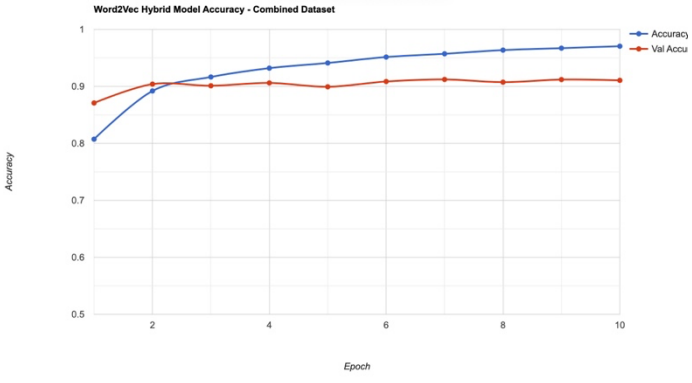
Combined Datase

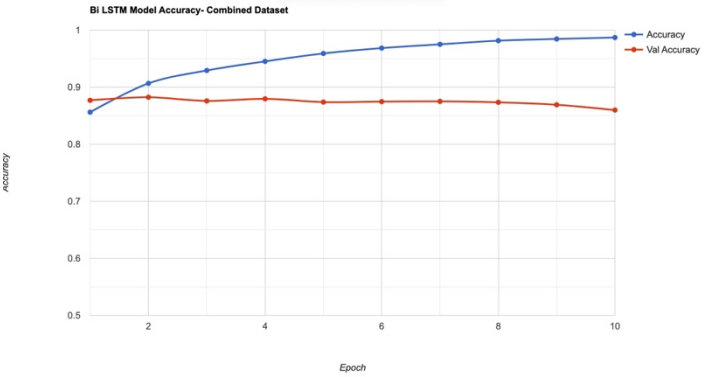


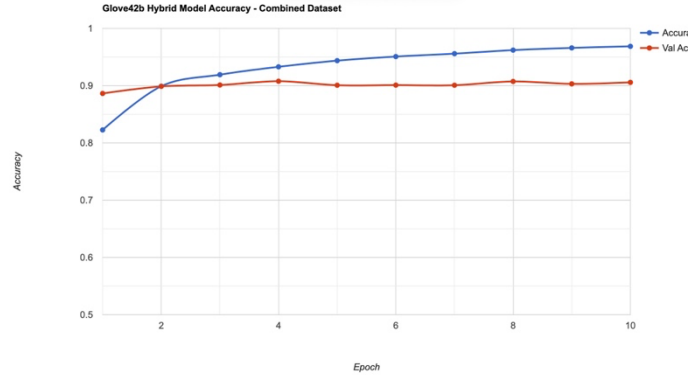




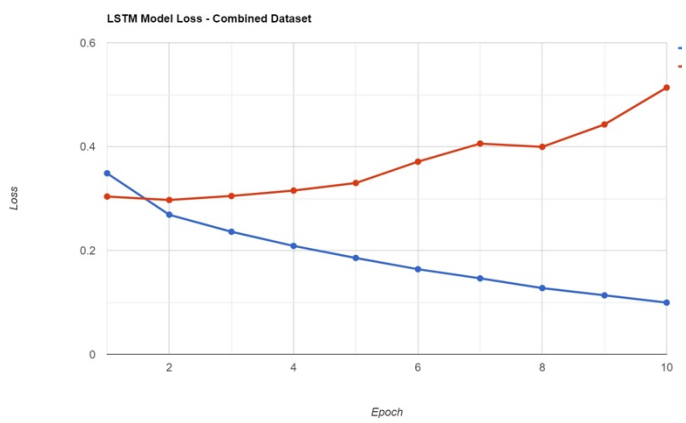


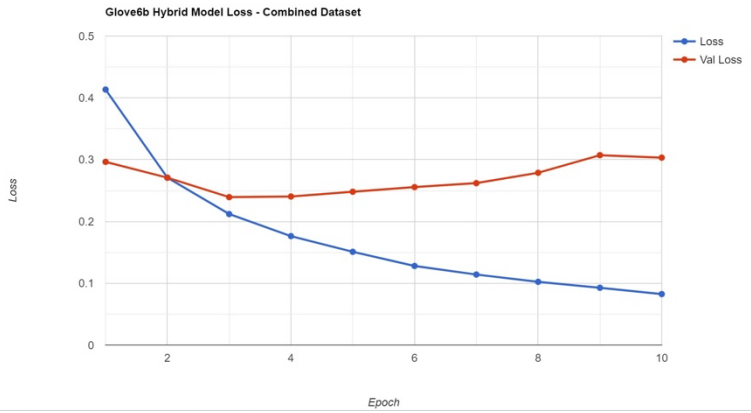


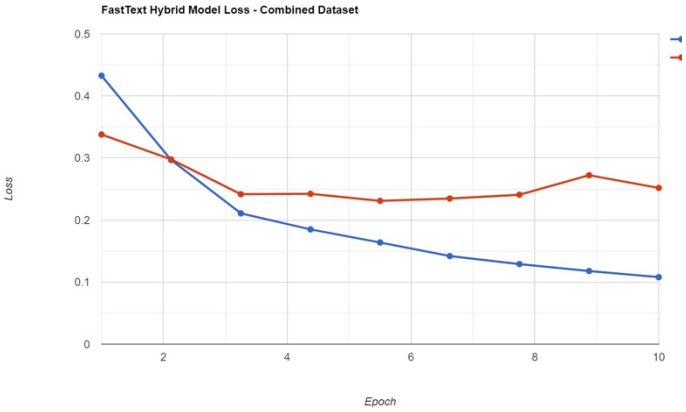


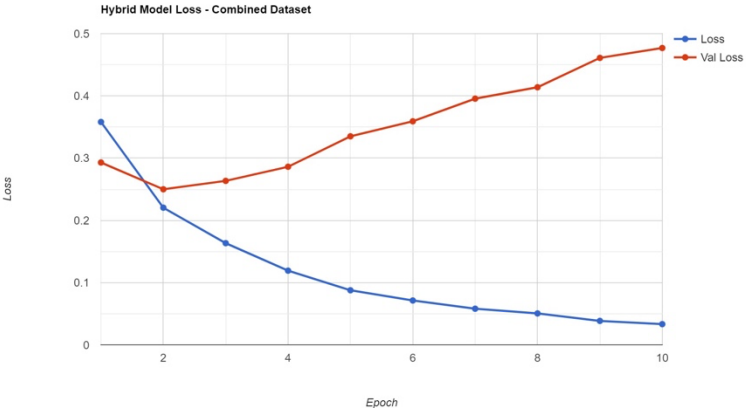


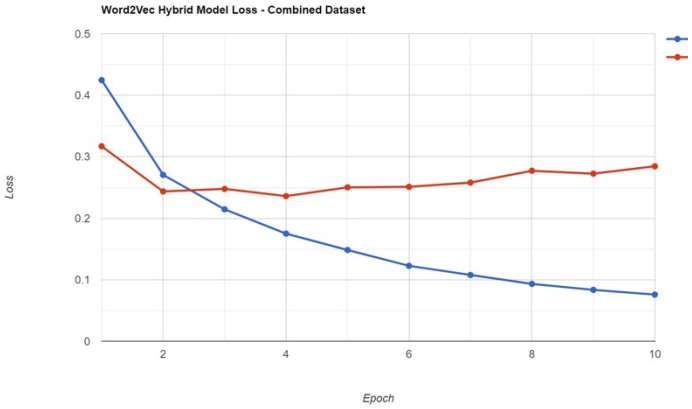
Accuracy Vs Val Accuracy (Combined Dataset)

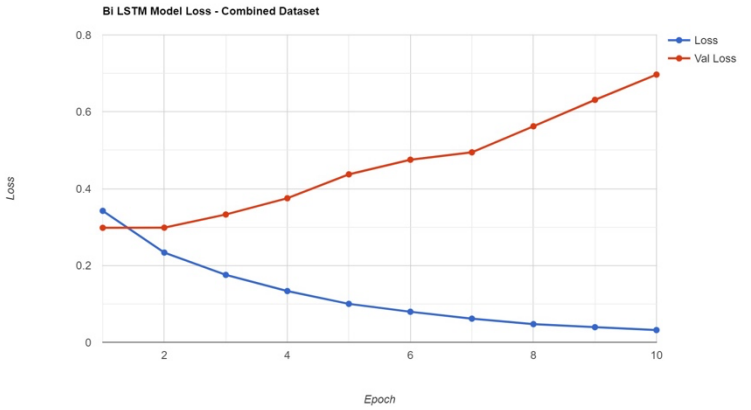


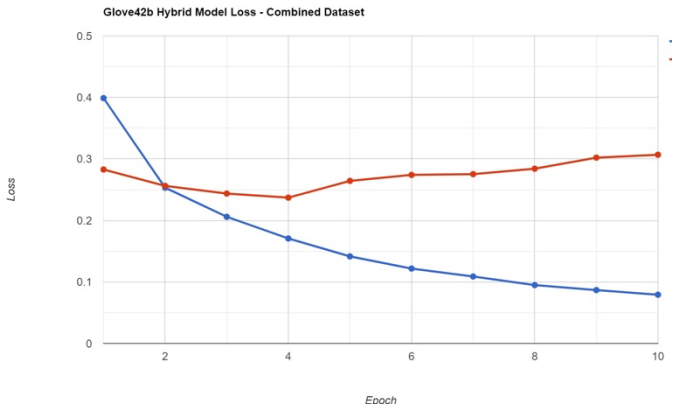












Loss Vs Val Loss (Combined Dataset)

The hybrid CNN-BiLSTM model with Word2Vec embeddings on a combined dataset achieved an accuracy of 91.06%, outperforming all other models.

Classification Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.8726	0.882	0.823	0.851
SVM	0.9012	0.925	0.846	0.884
MLP	0.873	0.862	0.849	0.855
K Nearest Neighbour	0.624	0.802	0.202	0.323
Random Forest	0.887	0.909	0.827	0.866
Voting Classifier	0.88	0.888	0.835	0.861
XGboost	0.8592	0.863	0.859	0.857
Adaboost	0.8397	0.842	0.839	0.838
Decision Tree	0.813	0.784	0.797	0.791

Classification Model	Accuracy	Precision	Recall	F1-score
LINEAR SVM	0.8706	0.8284	0.7889	0.8082
LSTM	0.8706	0.8819	0.8225	0.8512
BI-LSTM	0.8599	0.8254	0.8713	0.8477
CNN-BI LSTM (HYBRID)	0.891	0.8911	0.8588	0.8746
GLOVE (HYBRID) 6B	0.9064	0.9347	0.8481	0.8893
GLOVE (HYBRID) 42B	0.9058	0.9234	0.8619	0.8916
Word2Vec	0.9106	0.9205	0.8737	0.8965
FastText	0.9087	0.9306	0.8570	0.8923

Chapter 6

Conclusion and Future Work

After combining and cleaning four datasets, this research paper utilized various machine learning and deep learning algorithms to classify fake news. The results of the experiments showed that deep learning models outperformed machine learning models, especially the hybrid CNN-BiLSTM model with Word2Vec embeddings on a combined dataset achieved an accuracy of 91.06%, outperforming all other models.

In conclusion, this study demonstrated the effectiveness of combining multiple datasets and using deep learning techniques for fake news classification. The embedding employed for classification is the most relevant factor, as it can make a difference. Furthermore, the results suggest that using hybrid models and embeddings can significantly improve the accuracy of fake news detection.

For future work, we can use the embedded dataset to train deep learning models and select the best model. We can then use transfer learning to retrain the ML algorithms on the embedded dataset. We can also use different embeddings, such as BERT or ELMo, and compare their performance with the embeddings used in this project. Furthermore, we can explore other techniques for combining datasets and use ensemble methods to improve the performance of the models. Finally, we can analyze the models' behavior and interpret their decisions to gain insights into their internal mechanisms.

APPENDICES

Data Cleaning

```
import re
import contractions
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('words') #download list of english words
nltk.download('stopwords') #download list of stopwords
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
stopWords = stopwords.words('english')
englishWords = set(nltk.corpus.words.words())

def remove_contractions(text):
    return ' '.join([contractions.fix(word) for word in text.split()])

tokenizer = RegexpTokenizer(r'\w+')
lemmatizer = WordNetLemmatizer()

def nltkToWordnet(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize(tokens):
    pos_tags = nltk.pos_tag(tokens)
    res_words = []
    for word, tag in pos_tags:
        tag = nltkToWordnet(tag)
        if tag is None:
            res_words.append(word)
        else:
            res_words.append(lemmatizer.lemmatize(word, tag))
    return res_words
```

```
def remove_stopWords(tokens):
    return [w for w in tokens if (w in englishWords and w not in stopWords)]

def split_and_filter_words(string):
  words = string.split()
  if len(words) > 20:
    return " ".join(words)
  return None

def clean_text(text):
    text = ' '.join([contractions.fix(word) for word in text.split()])
    # Tokenize text
    tokens = tokenizer.tokenize(text)
    # Lemmatize tokens
    tokens = lemmatize(tokens)
    # Remove stopwords
    tokens = remove_stopWords(tokens)
    # Print results
    text = " ".join(tokens)
    return text
```

Machine Learning Algorithms

TF – IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_v = TfidfVectorizer(max_features=5000, ngram_range=(1,3))
X = tfidf_v.fit_transform(df['statement'].values.astype('U'))
y = df['label']
import joblib
# Save the TfidfVectorizer
joblib.dump(tfidf_v, '/content/drive/MyDrive/Colab
Notebooks/weights/LIAR/learners/tfidf_dataset_name.pkl')
```

Dataset Split to Train and Test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

Imports

```
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import accuracy_score as acs
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

Metrics Code for all ML algorithms

```
precision, recall, fscore, train_support = score(y_test, y_pred, pos_label=1,
average='binary')
print('Precision: {} / Recall: {} / F1-Score: {} / Accuracy: {}'.format(
    round(precision, 3), round(recall, 3), round(fscore,3),
round(acs(y_test,y_pred), 3)))
# Calculate the accuracy of the model
print(classification_report(y_test, y_pred))
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
class_label = [0, 1]
df_cm = pd.DataFrame(cm, index=class_label,columns=class_label)
sns.heatmap(df_cm, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Logistic Regression

```
from sklearn import linear_model
logr = linear_model.LogisticRegression()
logr.fit(X_train,y_train)
y_pred = logr.predict(X_test)
# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'.
format(acs(y_test, y_pred)))
```

SVM

```
# import SVC classifier
from sklearn.svm import SVC
# import metrics to compute accuracy
# instantiate classifier with default hyperparameters
svc=SVC()
# fit classifier to training set
svc.fit(X_train,y_train)
# make predictions on test set
y_pred=svc.predict(X_test)
```

MLP

```
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
# Create model object
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=True,
                    learning_rate_init=0.01)
# Fit data onto the model
clf.fit(X_train,y_train)
# Make prediction on test dataset
y_pred=clf.predict(X_test)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
# Predict on dataset which model has not seen before
y_pred = knn.predict(X_test)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=150, max_depth=None, n_jobs=-1)
rf_model = rf.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

Voting Classifier

```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
# Initialize the classifier
rf_clf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
lr_clf = LogisticRegression(max_iter=1000, random_state=42)
knn_clf = KNeighborsClassifier(n_neighbors=5)
# Combine the classifiers using a voting ensemble
voting_clf = VotingClassifier(estimators=[('rf', rf_clf), ('lr', lr_clf), ('knn',
knn_clf)], voting='hard')
voting_clf.fit(X_train, y_train)
# Make predictions using the voting classifier
voting_preds = voting_clf.predict(X_test)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

XGboost

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
regressor = xgb.XGBRegressor(
    n_estimators=100,
    reg_lambda=1,
    gamma=0,
    max_depth=3
)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE : % f" %(rmse))
```

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=100,
                         learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE : % f" %(rmse))
```

Linear SVM

```
import joblib
from sklearn.svm import LinearSVC
# Creating a Linear SVM classifier
```

```
svm = LinearSVC()
# Performing five-fold cross-validation
scores = cross_val_score(svm, X, y, cv=5)
```

Deep Learning Models

Imports

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, Bidirectional
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM,
Dense, Dropout
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
```

```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=128)
# Evaluate the model
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)
# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

Tokenization, Padding and Train Test Split

```
# Tokenize the news statements
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(df['statement'])
```

```
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
# Convert the news statements to sequences of integers
sequences = tokenizer.texts_to_sequences(df['statement'])
# Pad the sequences
Max_Len = max([len(x) for x in df['statement']])
padded = pad_sequences(sequences, maxlen=Max_Len)
print('Shape of data tensor:', padded.shape)
# Define the maximum number of words to keep in the vocabulary
MAX_NUM_WORDS = 20000
# Convert the labels to one-hot encoding
y = df['label']
print('Shape of label tensor:', y.shape)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded, df['label'].values,
test_size=0.2)
```

LSTM

```
# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=5000, output_dim=32, input_length=100),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 32)           160000

 lstm (LSTM)                 (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 168,353
Trainable params: 168,353
Non-trainable params: 0
_________________________________________________________________
```

Bi LSTM

```
# Define model architecture
model = Sequential()
model.add(Embedding(len(word_index) + 1, 128, input_length=max_len))
model.add(Bidirectional(LSTM(64, return_sequences=True)))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(1, activation='sigmoid'))
```

```
Found 34902 unique tokens.
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 100, 128)          4467584

 bidirectional (Bidirectiona  (None, 100, 128)         98816
 l)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dense_1 (Dense)              (None, 1)                 65

=================================================================
Total params: 4,607,681
Trainable params: 4,607,681
Non-trainable params: 0
_________________________________________________________________
```

Hybrid

```
# Build the model
model = Sequential()
model.add(Embedding(input_dim=MAX_NB_WORDS, output_dim=EMBEDDING_DIM,
input_length=Max_Len))
model.add(Conv1D(128, 5, activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling1D(2))
model.add(Conv1D(128, 5, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(1, activation='sigmoid'))
```

```
Found 34902 unique tokens.
Shape of data tensor: (75023, 29569)
Shape of label tensor: (75023,)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 29569, 100)        5000000

 conv1d (Conv1D)             (None, 29565, 128)        64128

 dropout (Dropout)           (None, 29565, 128)        0

 max_pooling1d (MaxPooling1D  (None, 14782, 128)       0
 )

 conv1d_1 (Conv1D)           (None, 14778, 128)        82048

 max_pooling1d_1 (MaxPooling  (None, 7389, 128)        0
 1D)

 bidirectional (Bidirectiona  (None, 64)               41216
 l)

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 5,187,457
Trainable params: 5,187,457
Non-trainable params: 0
_________________________________________________________________
```

Word Embeddings

GloVe

```
# Load the GloVe embeddings
embeddings_index = {}
f = open('/content/drive/MyDrive/Colab Notebooks/weights/glove.6B.300d.txt',
encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

```
Found 34902 unique tokens.
Shape of data tensor: (75023, 29569)
Shape of label tensor: (75023,)
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 29569, 300)        6000000

 conv1d_6 (Conv1D)           (None, 29565, 128)        192128

 dropout_3 (Dropout)         (None, 29565, 128)        0

 max_pooling1d_6 (MaxPooling  (None, 14782, 128)       0
 1D)

 conv1d_7 (Conv1D)           (None, 14778, 128)        82048

 max_pooling1d_7 (MaxPooling  (None, 7389, 128)        0
 1D)

 bidirectional_3 (Bidirectio  (None, 64)               41216
 nal)

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 6,315,457
Trainable params: 315,457
Non-trainable params: 6,000,000
_________________________________________________________________
```

Word2Vec

```
from gensim.models.word2vec import Word2Vec
import gensim.downloader as api
# Load the pre-trained Word2Vec model
w2v_model = api.load('word2vec-google-news-300')
```

```
Found 34902 unique tokens.
Shape of data tensor: (75023, 29569)
Shape of label tensor: (75023,)
[==================================================] 100.0% 1662.8/1662.8MB downloaded
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 29569, 300)        6000000

 conv1d_8 (Conv1D)           (None, 29565, 128)        192128

 dropout_4 (Dropout)         (None, 29565, 128)        0

 max_pooling1d_8 (MaxPooling  (None, 14782, 128)       0
 1D)

 conv1d_9 (Conv1D)           (None, 14778, 128)        82048

 max_pooling1d_9 (MaxPooling  (None, 7389, 128)        0
 1D)

 bidirectional_4 (Bidirectio  (None, 64)               41216
 nal)

 dense_4 (Dense)             (None, 1)                 65

=================================================================
Total params: 6,315,457
Trainable params: 315,457
Non-trainable params: 6,000,000
```

FastText

```
# Load the Word2Vec embeddings
from gensim.models.word2vec import Word2Vec
import gensim.downloader as api
# Load the pre-trained Word2Vec model
ft_model = api.load('fasttext-wiki-news-subwords-300')
```

```
Shape of data tensor: (75023, 29569)
Shape of label tensor: (75023,)
[==================================================] 100.0% 958.5/958.4MB downloaded
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 29569, 300)        6000000

 conv1d_4 (Conv1D)           (None, 29565, 128)        192128

 dropout_2 (Dropout)         (None, 29565, 128)        0

 max_pooling1d_4 (MaxPooling  (None, 14782, 128)       0
 1D)

 conv1d_5 (Conv1D)           (None, 14778, 128)        82048

 max_pooling1d_5 (MaxPooling  (None, 7389, 128)        0
 1D)

 bidirectional_2 (Bidirectio  (None, 64)               41216
 nal)

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 6,315,457
Trainable params: 315,457
Non-trainable params: 6,000,000
_________________________________________________________________
```

Embedding Matrix

```
# Create an embedding matrix
embedding_dim = 300
word_index = tokenizer.word_index
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, embedding_dim))
for word, i in word_index.items():
    if i >= MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Hybrid with Word Embeddings

```
# Build the model
model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=embedding_dim,
input_length=Max_Len, weights=[embedding_matrix], trainable=False))
model.add(Conv1D(128, 5, activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling1D(2))
model.add(Conv1D(128, 5, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(1, activation='sigmoid'))
```

REFERENCES

1) SBali, A. P. S., Fernandes, M., Choubey, S., & Goel, M. (2019, April).Comparative-performance of machine learning algorithms for fake news detection. In International conference on advances in computing and data sciences (pp. 420-430). Springer, Singapore.
2) Gundapu, S., & Mamidi, R. (2021). Transformer-based automatic COVID-19 fake news detection system. arXivpreprint arXiv:2101.00180.
3) Thota, A., Tilak, P., Ahluwalia, S., & Lohia, N. (2018). Fake news detection: a deep learning approach. SMU Data Science Review, 1(3), 10.
4) Ahmad, I., Yousaf, M., Yousaf, S., & Ahmad, M. O. (2020). Fake news detection using machine learning ensemble methods. Complexity, 2020, 1-11.
5) Truică, C. O., & Apostol, E. S. (2023). It's All in the Embedding! Fake News Detection Using Document Embeddings. Mathematics, 11(3), 508.
6) Wang, W. Y. (2017). " liar, liar pants on fire": A new benchmark dataset for fake news detection. arXiv preprint arXiv:1705.00648.
7) Della Vedova, M. L., Tacchini, E., Moret, S., Ballarin, G., DiPierro, M., & de Alfaro, L. (2018, May). Automatic online fake news detection combining content and social signals. In *2018 22nd conference of open innovations association (FRUCT)* (pp. 272-279). IEEE
8) Upadhayay, B., & Behzadan, V. (2020, November). Sentimental LIAR: Extended Corpus and Deep Learning Models for Fake Claim Classification. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)* (pp. 1-6). IEEE.
9) Tosik, M., Mallia, A., & Gangopadhyay, K. (2018). Debunking fake news one feature at a time. *arXiv preprint arXiv:1808.02831*.
10) ThoRawat, M., & Kanojia, D. (2021). Automated Evidence Collection for Fake News Detection. *arXiv preprint arXiv:2112.06507*.
11) Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
12) Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511801389
13) Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning.
14) Kecman, V. (2005) Support Vector Machines: An Introduction. In: Wang, L.P., Ed., Support Vector Machines: Theory and Applications, Springer, Berlin, 1-47.

15) Akhtar, S., Hussain, F., Raja, F. R., Ehatisham-ul-haq, M., Baloch, N. K., Ishmanov, F., & Zikria, Y. B. (2020). Improving mispronunciation detection of arabic words for non-native learners using deep convolutional neural network features. *Electronics*, *9*(6), 963.

16) Pérez-Rosas, V., Kleinberg, B., Lefevre, A., & Mihalcea, R. (2017). Automatic detection of fake news. *arXiv preprint arXiv:1708.07104*.

17) Ruta, D., & Gabrys, B. (2005). Classifier selection for majority voting. *Information fusion*, *6*(1), 63-81.

18) L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Springer, Berlin, Germany, 1984.

19) Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

20) Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, *2*(3), 349-360.

21) Lam, L., & Suen, S. Y. (1997). Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *27*(5), 553-568.

22) Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

23) Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, *26*

24) Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, *5*, 135-146.

25) Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).