ISyE 6740 - Summer 2023
Final Report

Team Member Names:	Hira Tanweer (htanweer3), Hunny Kanwar (hkanwar3), Felipe Munoz (fmunoz30)

Project Title: Fake News Detection Using Long Short-Term Memory (LSTM) Model

Problem Statement

During the past decade, influence of social media on our daily lives has increased exponentially. Though Social Media has helped bring people and communities closer, it has also exposed millions of users to consuming Fake News in an attempt to influence their opinions. Big social media companies such as Meta, and Twitter are using Artificial Intelligence and Machine Learning tools to combat this growing challenge but the problem is gigantic in scope, and more nuanced at the same time.

The term, "Fake News", has been defined in Cambridge dictionary [1] as:

False stories that appear to be news, spread on the internet or using other media, usually created to influence political views or as a joke

Fake news dissemination is either intentional or accidental, and detection can vary among different demographic groups [2]. False information tends to spread much faster and wider than true information, with political and urban legend topics being particularly susceptible to fake news [3]. In the realm of politics, individuals tend to believe news that aligns with their political views, regardless of its credibility. This preference for alignment occurs as frequently with true headlines as it does with false headlines, suggesting that people's capabilities of distinguishing between truth and falsehood may be compromised [4].

The consequences of fake news are far-reaching and can have devastating effects on individuals, communities, and even democracies. Therefore, it is imperative to develop effective tools and methodologies to combat the dissemination of fake news.

In this project, we attempted to build different machine learning models including a deep learning LSTM based RNN classification model that would be able to flag fake news stories by comparing the word vectors generated from raw data with legitimate news sources.

Data Source

We used Fake News dataset from one of the *Kaggle Machine Learning Competitions* [5]. Training dataset has 20,799 rows of unique text stories, along with attributes such as **id, title, author, and text**. It also provides output labels in a binary format to train the classifier if a given text story is reliable or not. Once the model was trained and validated, we tested our model accuracy on a separate test dataset provided by Kaggle, which has 5200 articles, and the same set of attributes.

Exploratory Data Analysis

To ensure that our models are not biased due to disproportionate number of data points in either Reliable or Not Reliable category, we started off exploratory analysis by looking at the per cent distribution of articles grouped by their classification labels (1 = Not Reliable, 0 = Reliable).

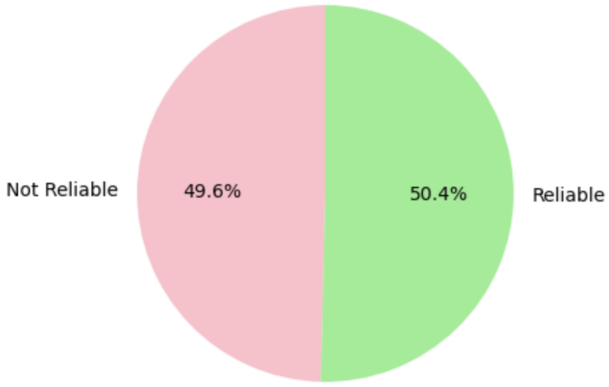

Figure 1: Percentage of Reliable, and Not Reliable articles

It can be noticed that the data set we are working with is quite balanced for model training purposes.

Next, we wanted to look at the frequency distribution of article labels grouped by author to learn if adding author as an input feature would be useful for prediction (without inducing bias in our models). When we merged the counts of Reliable, and Not Reliable articles by author, we found almost no overlap. Out of 4175 authors, there were only 5 with some degree of overlap. Rest of the authors had either written Reliable or Not Reliable articles as shown below:

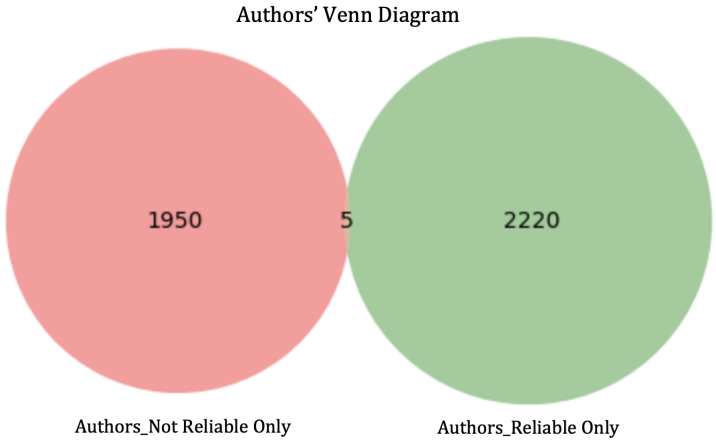

Figure 2: Venn Diagram

Upon looking at the spread of article counts per author for each label type, we noticed a couple of outliers with exceptionally high counts of articles per author. More interestingly, rest of the distribution for both label types seems quite uniform (less than 5 articles per author). As there was a lot of text data associated with these outliers, we discussed to whether flag those authors as "bots" and exclude them from further consideration or not. Due to limited context around data, we decided to keep the text data associated with these authors with significantly higher counts for model training but excluded "author" as a feature in our LSTM model.

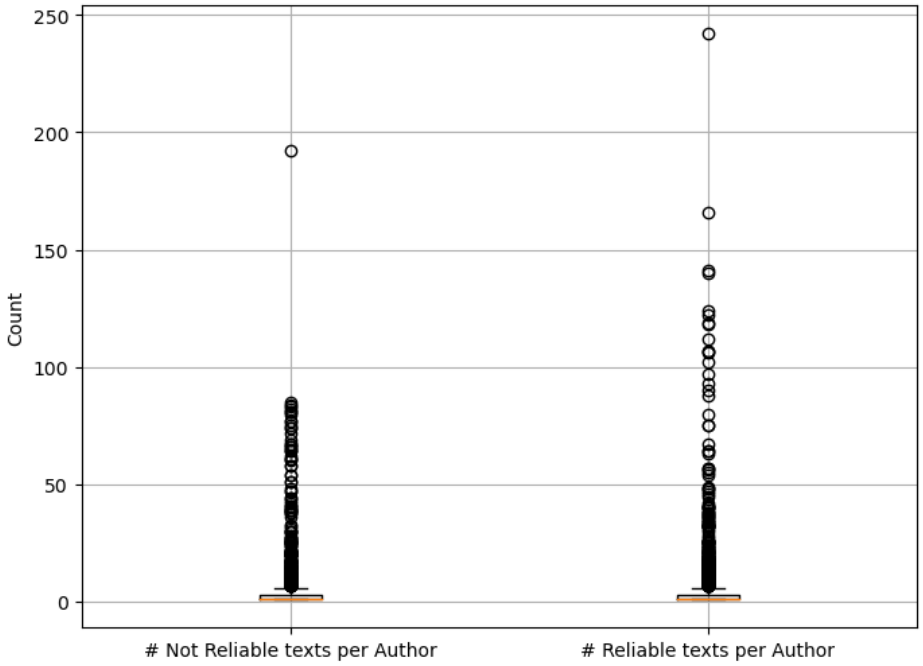

Figure 3: Box plots of article counts per Author

Below are bar charts providing further details about top 10 authors by article count for each category

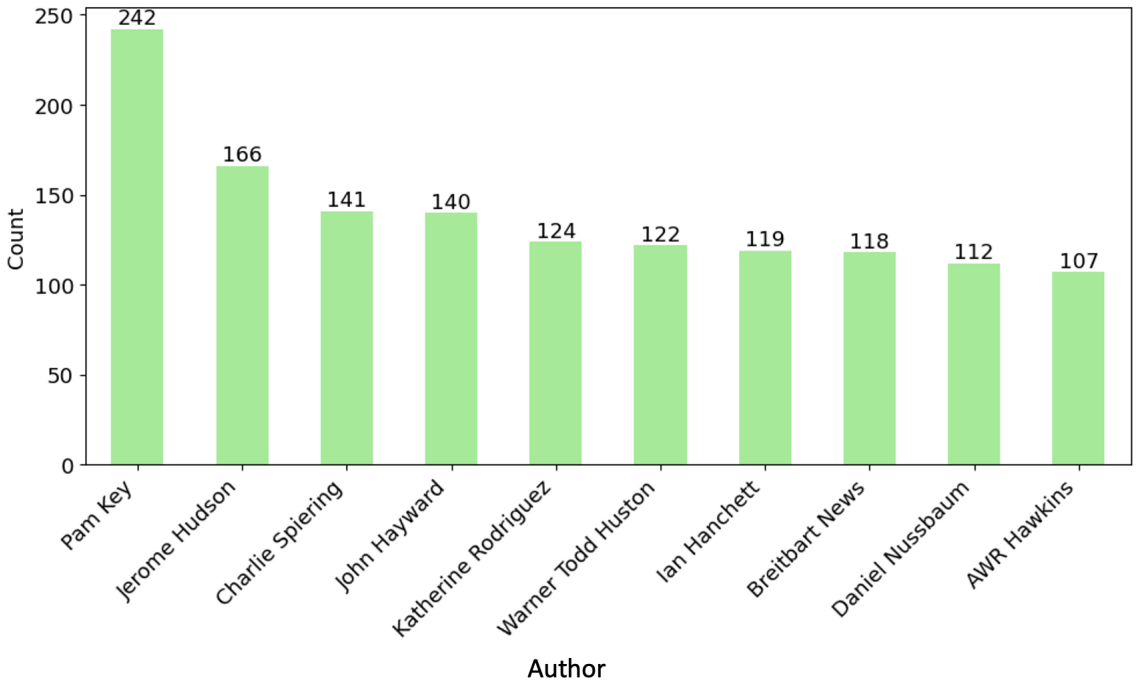

Figure 4: Top 10 Authors by Reliable article counts

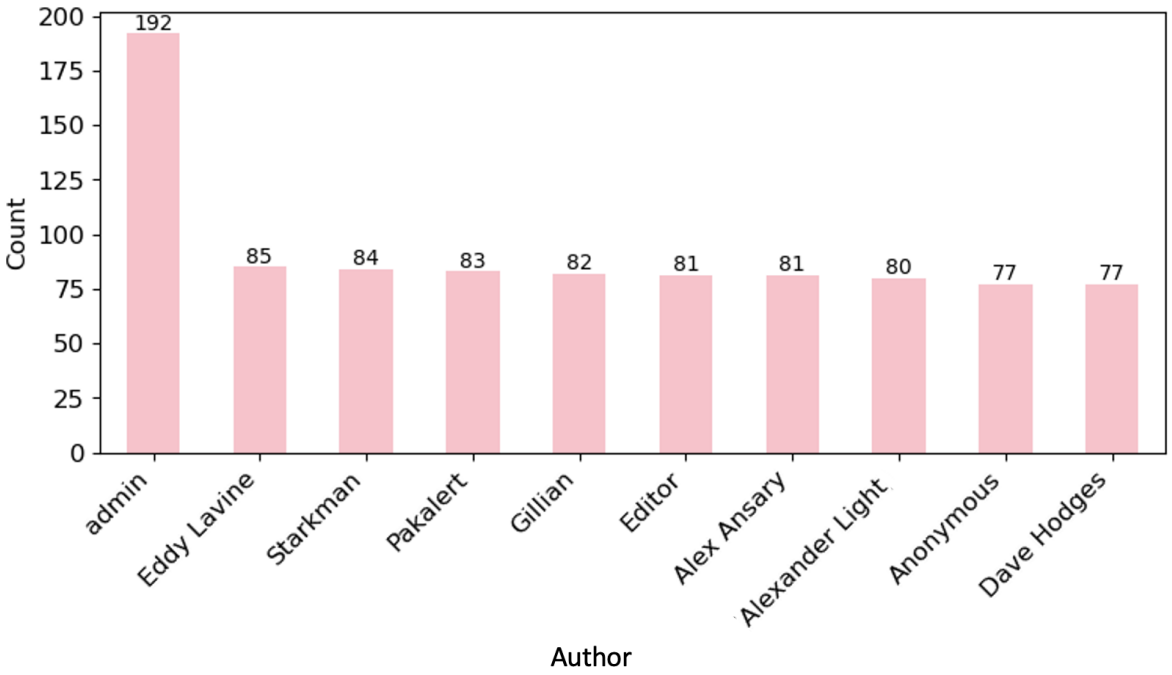

Figure 5: Top 10 Authors by Not Reliable article counts

We used Term Frequency Inverse Document Frequency (TF-IDF) scores for top 100 words in fake news articles and visualized them as a word cloud, where words with higher scores have proportionally larger font size (shown below):

Figure 6: Word Cloud for Top 100 Key Words in Fake News Articles

Methodology

Traditional neural networks process the input data independently without taking the order or temporal dependencies of the input elements into account. However, in many natural language processing tasks the sequential dependencies among the input elements are critical to building robust models.
To address this limitation, we explored Long Short-Term Memory (LSTM) model, which is a type of Recurrent Neural Network (RNN) and is capable of learning long term dependencies in sequential input data [6]. It incorporates specialized memory cells that can retain and drop information selectively.
The key components of an LSTM are the Input gate, Forget gate, Output gate and Memory cell. These gates enable the neural network to regulate the flow of information, learn relevant information, and forget unnecessary information. The memory cell serves as a long-term storage unit that holds information over multiple time steps [7].

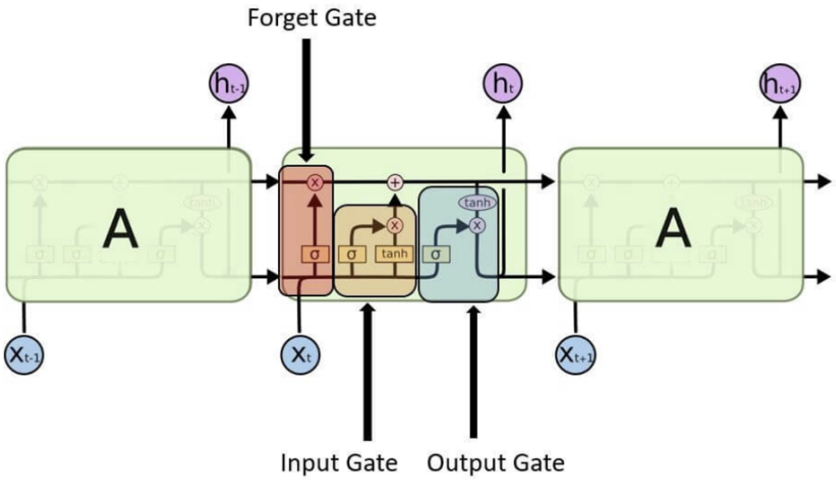

Figure 7: General Framework of LSTMs

By utilizing these components, LSTMs can effectively model sequential data [8] like news articles, where understanding the temporal patterns is crucial in distinguishing between reliable and unreliable information. In order to perform this classification task, we utilized Keras, which is a high-level API for TensorFlow, and NLTK (Natural Language Toolkit) for data processing.
In addition, Logistic regression, Naïve Bayes, and BERT models were also trained using the same dataset to compare the performance of different models in terms of how efficiently they learn underlying patterns in text data.

Data Cleaning and Preprocessing: In order to feed our data into the LSTM deep learning model, we pre-processed the data by handling all missing data instances, removing irrelevant characters using RegEx and converting all text to lowercase. We then tokenized the text into individual words/sub-words using NLTK's word_tokenize and removed all stop words like "a", "the", "is", "are", "at", "which" etc. Additionally, we applied stemming to normalize word forms using PorterStemmer, which removes the commoner morphological and inflectional endings from words for e.g., "connected", "connection", and "connecting" get reduced to the stem "connect" [9].

Tokenization and Sequencing: In order to capture semantic relationships between words and represent them meaningfully to our deep learning model, we leveraged the Keras Tokenizer to create a vocabulary of unique words in the training data and map each token to a unique integer value [10]. This transformation enables the LSTM model to work with numerical representations of the text rather than raw text data. Subsequently, we converted each article into a sequence of integers, where each integer represents a specific word from the vocabulary.

Padding: Before feeding the sequences into the LSTM model, we addressed the challenge of varying sequence lengths. LSTMs expect fixed-length sequences as input [11], but the articles in our dataset had different lengths depending on the number of words they contained. To handle this, we employed padding, a technique where sequences are padded with zeros (padding tokens) to ensure they all have the same length. We set a maximum sequence length of 500 tokens based on the dataset distribution (covering 95% of the training sequences) and pre-padded/truncated shorter/longer sequences to match this length. This transformation allowed us to create uniform input data that the LSTM model could process.

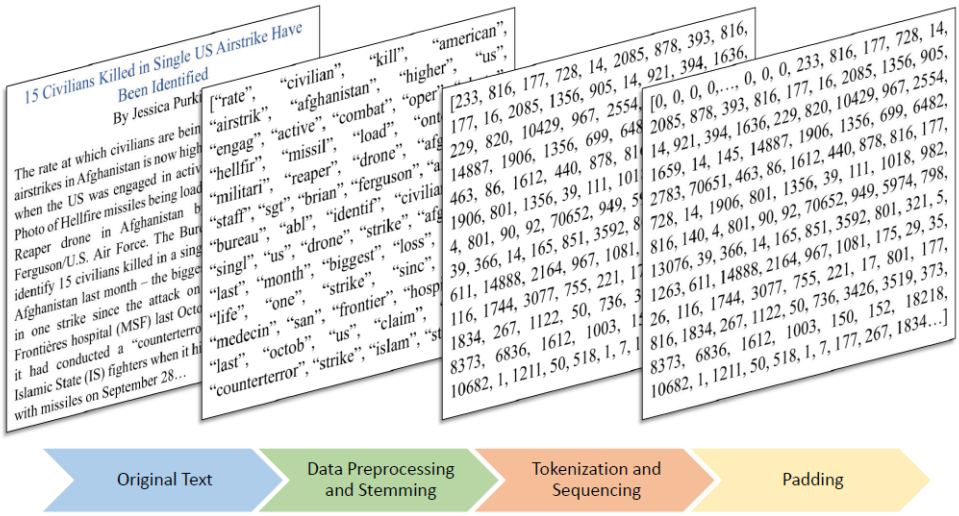

Figure 8: Data Processing Steps

Modeling: After splitting our data into 80% training and 20% validation sets, we built multiple models to classify whether a given news article is reliable 0 or unreliable 1. In the course of our analysis, we experimented with the following machine learning models:

LSTM:
After exploring multiple model architectures and performing hyper-parameter tuning, we utilized LSTM model with the following architecture and hyper-parameters:

1. Embedding Layer: This layer is responsible for learning word representations or embeddings from the tokenized sequences. It converts each integer-encoded word into a dense vector of size 128, allowing the model to represent words in a continuous and meaningful way.
2. LSTM Layer: Following the Embedding layer, we have an LSTM (Long Short-Term Memory) layer. This LSTM layer with 128 units processes the input sequences by updating and maintaining an internal memory state over time. It selectively retains and discards information, allowing it to learn complex patterns and dependencies in the text data. The use of LSTM layers in the architecture makes it particularly effective for capturing long-range dependencies making it well-suited for sentiment analysis on our text data.
3. Dense Layer: After the LSTM layer, we add a dense layer with 64 units with a Rectified Linear Unit (ReLU) activation function. This fully connected layer serves as a feature extractor, capturing high-level patterns and representations from the output of the LSTM layer.
4. Dropout Layer: To prevent overfitting and improve generalization, we include a dropout layer with a dropout rate of 0.5. During training, this layer randomly drops out 50% of the units in the previous dense layer. This regularization technique helps in reducing the model's reliance on certain neurons too much and improves its ability to generalize to new data [12].

5. Output Layer: The final layer in our model is a dense layer with a single unit and a sigmoid activation function. This layer is responsible for producing the binary classification output, where the model predicts the probability of an article being reliable or unreliable. The sigmoid activation function ensures that the output is in the range [0, 1], representing the probability of the article being real news (closer to 0) or fake news (closer to 1). We then convert this probability value to 0 or 1 using a threshold value of 0.5.

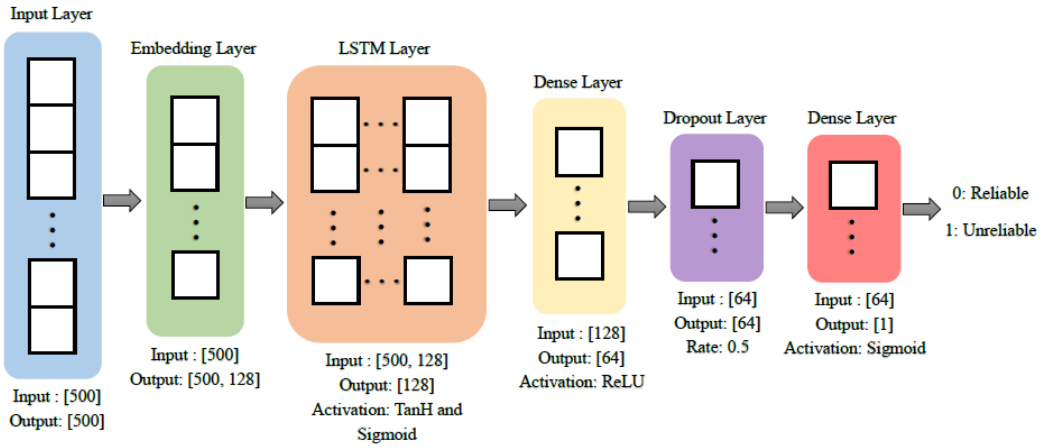

Figure 9: LSTM Model Architecture

6. Model Compilation and Training: After defining the model architecture, we compile the model using the binary_crossentropy loss function, which is appropriate for binary classification problems. We use the nadam (Nesterov-accelerated Adaptive Moment Estimation) optimizer and the model's performance during training is evaluated using the accuracy metric.
To prevent overfitting and to find the optimal model, we utilize the EarlyStopping callback. This monitors the validation loss during training and stops training early (with patience set to 3) if the validation loss does not improve for three consecutive epochs. The model's weights are restored to the best performing epoch. The model is then trained with a batch size of 32 and for a maximum of 10 epochs.

Bidirectional LSTM:
Additionally, we explored an alternative LSTM model architecture by incorporating a Bidirectional LSTM layer in place of the unidirectional LSTM. The Bidirectional LSTM model consists of two separate LSTM layers – one processing the sequences in the forward direction, and the other processing them in backward direction [13]. The outputs from both directions are then concatenated before being passed to the subsequent dense and dropout layers. The rest of the architecture remains the same, with the dense and dropout layers performing feature extraction and regularization, respectively. The final dense layer with a sigmoid activation function predicts the probability of the news article being reliable or unreliable, just like in the previous LSTM model (Figure 9).
Like the previous LSTM model, the Bidirectional LSTM model is compiled using the binary_crossentropy loss function, nadam optimizer and accuracy metric for validation. Additionally, we implemented the EarlyStopping callback to prevent overfitting during training. The model is then trained with a batch size of 32 and for 10 maximum epochs.

Logistic Regression:
Logistic Regression is one of the simplest yet effective classifiers available and is well-known for its efficiency with high-dimensional data. Our Logistic Regression modeling approach leveraged the Bag of Words technique. This technique employs a vast sparse matrix to store word counts for all words present in the text. To process the data, we utilized CountVectorizer from scikit-learn, which effectively tokenizes the text and assigns a unique identifier to each word.

To ensure effective validation of our model, we implemented the stratified k-fold technique. This method creates k balanced folds ensuring that each class is well-represented in each fold. We then trained the model on k-1 folds and performed validation and evaluation on the remaining fold. The performance metrics of our best-trained model can be seen in Table 1.

Naïve Bayes:
After applying the same data processing steps and validation technique as we did for the Logistic Regression model, we trained a Naïve Bayes model. Naïve Bayes is a probabilistic classifier commonly used in text classification tasks. It operates under the assumption that the features, which in our case are individual words, are conditionally independent given the class label. This enables the model to efficiently handle high-dimensional data. The Naïve Bayes model then employs Bayes' theorem to calculate the probability of each class given the observed features (words). Using these probabilities, the model determines the most likely class label for a given set of words in a text sequence. See Table 1 for performance evaluation.

BERT:
BERT (Bidirectional Encoder Representations from Transformers) is a transformer based language model, which is able to handle long-term dependencies. Unlike LSTMs, which can only consider the previous word when predicting the next one, BERT can simultaneously look at all the words in a sentence capturing long-term dependencies more effectively. However, this increased capability comes at the cost of greater computing power and longer training times compared to all previous models. BERT requires a significant amount of resources to train due to its complex architecture.
The model was trained with a batch size of 8 for the training set and a batch size of 4 for the validation set. Batch size determines the number of data samples processed together in each training iteration. For parameter updates, AdamW was used as the optimizer which is a stochastic optimization method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the bias terms [14]. We trained the model for a maximum of 10 epochs and to measure the performance of the model, we utilized the binary cross-entropy loss with logits. This loss function first applies a sigmoid function to the model's output logits, converting them to probabilities, and then calculates the binary cross-entropy loss between these probabilities and the true labels [15]. See Table 1 for performance metrics.

Evaluation and Final Results

In order to assess the performance of our trained models on unseen data, we utilized performance metrics like accuracy, precision, recall, F1 score and ROC/AUC. We analyzed these metrics to determine the model's ability to accurately detect reliable and unreliable news articles.

Model	Class	Precision	Recall	F1-Score	Accuracy	ROC/AUC	Kaggle Score
BERT	0	1.00	0.99	0.99	1.00	0.99591	0.99487
	1	0.99	1.00	0.99			
Log. Reg.	0	0.99	0.98	0.99	0.99	0.98602	0.98269
	1	0.99	0.99	0.99			
Naïve Bayes	0	0.92	0.98	0.95	0.95	0.94977	0.95192
	1	0.98	0.92	0.95			
LSTM	0	0.95	0.94	0.94	0.94	0.94403	0.94972
	1	0.94	0.95	0.94			
BiLSTM	0	0.92	0.95	0.94	0.94	0.93589	0.93397
	1	0.95	0.92	0.93			

Table 1: Model Performance Metrics

Our **LSTM** model demonstrates consistent and balanced performance in predicting both classes (0: reliable, 1: unreliable) with accuracy, precision, recall and F1-score all about 94%. This model achieved

a decent ROC/AUC score of 94.4% which highlights the model's ability to rank the probabilities correctly irrespective of the threshold set for classification. We also submitted our results on Kaggle for evaluation and received a public score of 94.97% which was based on our model's accuracy on Kaggle's test data. Meanwhile, the additional complexity introduced by the Bidirectional LSTM did not result in any significant performance enhancements. Both these models took around 15 minutes to train.

Logistic Regression was surprisingly accurate, with all performance scores around 99%. Despite its speed and efficiency, it's important to note that in the training process, the model did not reach the optimal solution within the maximum number of 100 iterations. To improve the convergence of the model, we increased the maximum number of iterations to 1000 to help the model get closer to the optimal solution. However, this improvement comes at a cost of increased training time taking approximately 13 minutes to train.

Naïve Bayes, despite its simplicity, demonstrates competitive performance in most metrics compared to all other models. However, it falls behind in precision for class 0 (reliable news articles). It took approximately 45 minutes to train this model.

BERT presents the best performance among all the models, with an accuracy of 100% and a precision near 100% for unreliable news articles. However, it takes a significant amount of time to train, approximately 3 hours for each training epoch. Resulting in a total elapsed time of 30 hours for a total of 10 epochs.

In the context of research and experimentation, it is essential to explore a variety of models including novel techniques like LSTM and BERT which can often lead to cutting-edge results and provide valuable insights to solve the problem at hand. However, when transitioning from research to real-world applications, time and resources play a critical role. Complex models often demand extensive training times which may not be feasible or cost-effective in production settings.

In the real world, model accuracy is only one aspect among several to consider. Practicality and simplicity usually take the center stage. In our case, a simple model like Logistic Regression offered satisfactory results while being computationally efficient. Striking the right balance between model complexity and performance is crucial. While complex models may give impressive accuracy, for practical deployment we should always consider model interpretability, scalability, resource requirements and training times.

Work Breakdown All the three team members contributed equally in this project as of ideas and workloads.

References

[1] Cambridge Dictionary. *Fake news*. In: *Cambridge Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/fake-news (visited on 07/04/2023) (cit. on p. 1).

[2] K. Peren Arin, Deni Mazrekaj, and Marcel Thum. "Ability of detecting and willingness to share fake news." In: *Scientific Reports* 13.1 (2023), pp. 1–12. ISSN: 20452322. URL: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib&db=a9h&AN=164397131&site=ehost-live&scope=site&custid=git1 (cit. on p. 1).

[3] Soroush Vosoughi, Deb Roy, and Sinan Aral. "The spread of true and false news online". In: *Science* 359.6380 (2018), pp. 1146–1151. DOI: 10.1126/science.aap9559. eprint: https://www.science.org/doi/pdf/10.1126/science.aap9559. URL: https://www.science.org/doi/abs/10.1126/science.aap9559 (cit. on p. 1).

[4] Gordon Pennycook and David G. Rand. "The Psychology of Fake News". In: *Trends in Cognitive Sciences* 25.5 (2021), pp. 388–402. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2021.02.007. URL: https://www.sciencedirect.com/science/article/pii/S1364661321000516 (cit. on p. 1).

[5] *Kaggle Fake News datasource*. URL: https://www.kaggle.com/competitions/fake-news/overview (cit. on p. 1).

[6] *LSTM Networks: A Detailed Explanation*. URL: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 (cit. on p. 5).

[7] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. on p. 5).

[8] *Long short-term memory (LSTM) RNN in Tensorflow*. URL: https://www.javatpoint.com/long-short-term-memory-rnn-in-tensorflow#:~:text=A%20general%20LSTM%20unit%20is,and%20out%20of%20the%20cell. (cit. on p. 5).

[9] *The Porter Stemming Algorithm*. URL: https://tartarus.org/martin/PorterStemmer/ (cit. on p. 5).

[10] *tf.keras.preprocessing.text.Tokenizer*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer (cit. on p. 5).

[11] *Understanding Masking Padding*. URL: https://www.tensorflow.org/guide/keras/understanding_masking_and_padding (cit. on p. 6).

[12] *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. URL: %5Curl%7Bhttps://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/%7D (cit. on p. 6).

[13] *Differences Between Bidirectional and Unidirectional LSTM*. URL: https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm (cit. on p. 7).

[14] *AdamW Explained*. URL: %5Curl%7Bhttps://paperswithcode.com/method/adamw#:~:text=AdamW%20is%20a%20stochastic%20optimization,decay%20from%20the%20gradient%20update.%7D (cit. on p. 8).

[15] *BCEWITHLOGITSLOSS*. URL: %5Curl%7Bhttps://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html%7D (cit. on p. 8).