Aarush Kartik*
InspiritAI,
Columbia, SC 29212 USA
(Dated: October 20, 2024)

The spread of fake news has become a prevalent issue in our contemporary world. The spread of fabricated news across different age groups has led our society to become misinformed, which in turn undermines the health of democracy and society as a whole. This study presents a predictive model designed to classify news articles as fake or real. Leveraging natural language processing techniques and machine learning algorithms, our model incorporates a diverse dataset named "LiarLiar" that is comprised of thousands of short political statements from various sources. The model was trained using supervised learning methods, achieving an accuracy of 24 percent in quantifying the scales for "truthiness" ranking. This low accuracy underscores the innate complexities of misinformation detection. Furthermore, the findings accentuate the need for improved feature selection and the integration of more robust datasets. This research contributes to the understanding of the challenges of combating misinformation, emphasizing the necessity for ongoing progress in this particular field.

I. INTRODUCTION

The rapid proliferation of misinformation, particularly during events such as the presidential election, has highlighted the need for effective fake news detection methods. Generally, during elections and events corresponding to it, misinformation about candidates, their background, and other affiliations tend to spread across social media and news outlets, complicating group and individual decision-making. For this issue, machine learning techniques have emerged as powerful tools for tackling the challenge of identifying and mitigating fake news.

II. DATA

A. Dataset Description

The *LIAR* dataset [1] is a publicly available resource comprising 12.8K manually labeled short statements from various contexts, such as political debates, TV ads, and social media posts. Each statement is annotated with one of six truthfulness ratings: `true`, `mostly-true`, `half-true`, `barely-true`, `false`, and `pants-fire`. In addition to the statements, the dataset includes metadata such as the subject, speaker, job title, state, and party affiliation of the speaker.

B. Data Acquisition

The dataset was obtained from the official repository provided by the authors [1]. An automated script was developed to download and extract the dataset, ensuring reproducibility and efficiency in data handling. The script checks for the existence of the dataset locally to avoid unnecessary downloads and proceeds to download and unzip the dataset if not found.

C. Data Preprocessing

To prepare the dataset for training the Long Short-Term Memory (LSTM) model, several preprocessing steps were performed. These steps included loading the data into a structured format, renaming columns for clarity, computing statement lengths, encoding labels numerically, and transforming textual data into numerical representations suitable for input into the neural network.

1. *Data Loading and Column Renaming*

The dataset was provided in tab-separated values (TSV) format and was split into training, validation, and test sets. We used the `pandas` library to read the TSV files into DataFrames for efficient data manipulation. The columns were renamed to meaningful names for better readability and ease of reference during data processing. The primary columns used in this study included:

- **ID**: Unique identifier for each statement.
- **Label**: The truthfulness rating assigned to the statement.
- **Statement**: The actual textual content of the statement.
- **Subject**, **Speaker**, **Job Title**, **State Info**, **Party Affiliation**: Additional metadata about the statement and speaker.

2. *Statement Length Computation*

We calculated the length of each statement to gain insights into the distribution of statement lengths within the dataset. This information could be valuable for understanding the complexity and variability of the textual data. The statement length l_i for the i-th statement was computed as:

* aarushkrtk@gmail.com

$$l_i = \text{len}(s_i) \tag{1}$$

where s_i is the textual content of the i-th statement, and $\text{len}(\cdot)$ denotes the number of characters or words in the statement.

3. Label Encoding

The truthfulness labels, originally categorical, were converted into numerical values to facilitate supervised learning. We defined a mapping from the textual labels to integer values representing the degree of truthfulness:

$$\begin{aligned} \texttt{true} &\rightarrow 0 \\ \texttt{mostly-true} &\rightarrow 1 \\ \texttt{half-true} &\rightarrow 2 \\ \texttt{barely-true} &\rightarrow 3 \\ \texttt{false} &\rightarrow 4 \\ \texttt{pants-fire} &\rightarrow 5 \end{aligned}$$

This mapping reflects an ordinal relationship among the labels, with lower values indicating higher truthfulness. The encoded label for the i-th statement is denoted as y_i.

4. Textual Data Transformation

To convert the textual data into numerical features suitable for input into the LSTM model, we employed the Bag-of-Words (BoW) model using the Count Vectorization technique. Each statement s_i was transformed into a vector $\mathbf{x}_i \in \mathbb{R}^V$, where V is the size of the vocabulary extracted from the training data.

The vector components x_{ij} represent the frequency of the j-th word in the vocabulary appearing in the i-th statement:

$$x_{ij} = \text{Count}(\text{word}_j, s_i) \tag{2}$$

where $\text{Count}(\text{word}_j, s_i)$ is the number of times the j-th word appears in the i-th statement.

5. Data Reshaping for LSTM Input

The resulting feature vectors were reshaped to fit the input requirements of the LSTM model, which expects data in a three-dimensional format: $(\text{samples}, \text{timesteps}, \text{features})$. In our case, we set the number of timesteps to 1, treating each statement as a single timestep with V features.

6. One-Hot Encoding of Labels

The numerical labels y_i were converted into one-hot encoded vectors $\mathbf{y}_i \in \mathbb{R}^6$, where each vector has a value of 1 at the index corresponding to the label and 0 elsewhere:

$$\mathbf{y}_i = [y_{i0}, y_{i1}, y_{i2}, y_{i3}, y_{i4}, y_{i5}] \tag{3}$$

with

$$y_{ij} = \begin{cases} 1 & \text{if } j = y_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

This representation is suitable for multi-class classification tasks using categorical cross-entropy loss.

D. Data Splitting

The dataset was already partitioned into training, validation, and test sets by the dataset authors, ensuring that the model's performance could be evaluated on unseen data. The splits are as follows:

- **Training Set**: Used to train the model parameters.
- **Validation Set**: Used for hyperparameter tuning and to prevent overfitting via early stopping.
- **Test Set**: Used to assess the final performance of the model.

E. Summary of Data Preparation Steps

In summary, the data preparation involved the following key steps:

1. **Data Acquisition**: Automated downloading and extraction of the dataset.
2. **Data Loading**: Reading the TSV files into structured DataFrames.
3. **Data Cleaning**: Renaming columns and ensuring data integrity.
4. **Feature Engineering**: Computing statement lengths and transforming textual data into numerical feature vectors using the BoW model.
5. **Label Encoding**: Converting categorical labels into numerical and one-hot encoded formats.
6. **Data Reshaping**: Adjusting the shape of the data to match the input requirements of the LSTM model.

These preprocessing steps ensured that the data was in an optimal format for training the LSTM network, facilitating efficient learning and improving the potential for accurate classification.

III. MODEL

A. Model Selection and Description

In this study, we employed a Long Short-Term Memory (LSTM) network for the task of fake news detection using the *Liar, Liar Pants on Fire* dataset [1]. The LSTM is a type of recurrent neural network (RNN) capable of learning long-term dependencies in sequential data, which makes it suitable for processing natural language text.

B. Model Architecture

The architecture of our LSTM model is designed to capture the sequential patterns in textual data effectively. The model consists of the following layers:

1. **First LSTM Layer**: An LSTM layer with 50 units and `return_sequences=True`, which returns the hidden state output for each input time step. This setting allows the subsequent layer to receive the entire sequence of outputs, capturing richer temporal information.
2. **First Dropout Layer**: A dropout layer with a dropout rate of 0.7 is applied to the outputs of the first LSTM layer. This high dropout rate helps prevent overfitting by randomly deactivating 70% of the neurons during training, forcing the network to learn more robust features.
3. **Second LSTM Layer**: Another LSTM layer with 50 units, processing the sequence output from the previous layer and summarizing it into a single hidden state vector.
4. **Second Dropout Layer**: Another dropout layer with a dropout rate of 0.7 is applied after the second LSTM layer.
5. **Dense Output Layer**: A fully connected dense layer with 6 units corresponding to the six classes in the dataset, using the softmax activation function to output probability distributions over the classes.

The model is compiled with the categorical cross-entropy loss function and the Adam optimizer [2] with a learning rate of 0.001. The model is trained for up to 30 epochs with early stopping based on validation loss to prevent overfitting.

C. Mathematical Formulation

The LSTM units in the network are defined by the following equations at each time step t:

$$\text{Input Gate:} \quad i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{5}$$

$$\text{Forget Gate:} \quad f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{6}$$

$$\text{Cell Candidate:} \quad \tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{7}$$

$$\text{Cell State Update:} \quad C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{8}$$

$$\text{Output Gate:} \quad o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{9}$$

$$\text{Hidden State Update:} \quad h_t = o_t \odot \tanh(C_t) \tag{10}$$

Where:

- x_t is the input vector at time t.
- h_{t-1} is the hidden state from the previous time step.
- i_t, f_t, and o_t are the input, forget, and output gates, respectively.
- C_t is the cell state at time t.
- $\tilde{C}_t$ is the candidate cell state.
- σ denotes the sigmoid activation function.
- tanh denotes the hyperbolic tangent function.
- $\odot$ represents element-wise multiplication.
- W and U are weight matrices for inputs and hidden states.
- b are bias vectors.

The use of LSTM allows the network to retain information over long sequences, which is essential for understanding the context in textual data.

D. Hyperparameters and Regularization

- **Number of LSTM Units**: 50 units in each LSTM layer to balance model capacity and computational efficiency.
- **Dropout Rate**: A high dropout rate of 0.7 is applied after each LSTM layer to mitigate overfitting due to the relatively small size of the dataset.
- **Optimizer**: Adam optimizer is chosen for its adaptive learning rate and computational efficiency.
- **Learning Rate**: Set to 0.001 as a standard starting point for Adam.
- **Loss Function**: Categorical cross-entropy is used as it is suitable for multi-class classification problems.

- **Early Stopping**: Training is halted if the validation loss does not improve for 5 consecutive epochs, which helps in preventing overfitting.

E. Model Training and Evaluation

The model is trained using mini-batch gradient descent with a batch size of 32. During training, the model's performance is monitored on a validation set, and the best model weights are restored using early stopping.

To evaluate the model, we use the following metrics:

- **Accuracy**: Measures the overall correctness of the model's predictions.
- **Confusion Matrix**: Provides a detailed breakdown of true positives, false positives, true negatives, and false negatives for each class.
- **Classification Report**: Includes precision, recall, and F1-score for each class, offering insights into the model's performance on individual classes.

F. Rationale for Model Selection

The choice of an LSTM-based model is motivated by its effectiveness in modeling sequential data and capturing long-term dependencies, which are crucial in natural language processing tasks. Fake news detection benefits from understanding the context and nuances in language, which LSTMs are well-suited to handle.

Furthermore, the addition of dropout layers serves as a regularization technique to prevent overfitting, especially important given the limited size of the dataset (12.8K statements). The model aims to generalize well to unseen data by learning robust features rather than memorizing the training data.

G. Implementation Details

The model is implemented using the TensorFlow Keras API. Key implementation aspects include:

- **Model Definition**: The `Sequential` API is used to stack layers linearly.
- **Callbacks**: Early stopping is implemented using the `EarlyStopping` callback, monitoring the validation loss.
- **Evaluation Functions**: Custom methods are defined for model evaluation, including `evaluate` which computes and prints accuracy, confusion matrix, and classification report.

IV. RESULTS

In this section, we present the findings from our experiments on fake news detection using the proposed LSTM model. The results were evaluated against several baseline models. We utilized standard metrics—accuracy, precision, recall, and F1-score—to gauge the performance of our models. Despite the marginal improvements in accuracy, the precision and recall values were significantly lower than anticipated. The LSTM model achieved a precision of 0.21 and a recall of 0.20. These figures indicate a substantial challenge in effectively identifying true positive instances of fake news, highlighting that the model struggled to minimize false negatives. The low precision suggests that many articles classified as fake were, in fact, legitimate news, pointing to a need for more nuanced feature extraction and possibly a re-evaluation of the training data. This performance underscores the complexity of the fake news landscape and the importance of refining our approach to enhance detection capabilities.

	true	mostly-true	half-true	barely-true	false	pants-fire
true	62	37	41	17	41	10
mostly-true	67	46	48	34	38	8
half-true	48	53	60	47	39	18
barely-true	30	28	46	46	45	17
false	45	32	46	36	74	16
pants-fire	13	9	20	15	17	18

TABLE I. Confusion Matrix: True Labels vs. Predicted Labels

V. CONCLUSION

In this study, we explored the effectiveness of an LSTM model for fake news detection. While our model showed some improvement in accuracy over traditional methods, the overall performance was disappointing, particularly in terms of precision and recall. With precision at 0.21 and recall at 0.20, the model's ability to accurately classify fake news was significantly hindered, revealing the complexities inherent in this challenging task. The results highlight the need for a more robust approach to fake news detection, particularly as misinformation continues to evolve and proliferate. Our findings indicate that relying solely on textual features may not suffice, necessitating a broader strategy that incorporates various data sources and model architectures. For example, experimenting with hybrid models that combine LSTMs with other architectures—like convolutional neural networks (CNNs) or graph-based models— may enhance the ability to capture both sequential and relational features inherent in news articles and their dissemination networks.

[1] W. Y. Wang, Liar, liar pants on fire: A new benchmark dataset for fake news detection, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)* (ACL, 2017) pp. 422–426.

[2] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015).