Fake News Detection 2023

Anonymous Author(s)
Affiliation
Address
email

Group 11
Dharma Karan Reddy Gaddam
Rakshith Venkatesh Murthy Gowda
Thrishal Reddy Desam

Abstract

This report presents a comprehensive study on the development of a fake news detection system, utilizing the widely recognized LIAR dataset—a comprehensive collection of labeled political statements and media articles. Our approach integrates advanced natural language processing techniques and machine learning algorithms to observe patterns which indicate false information. This model is our approach towards identifying the small patterns that distinguish truth from fiction.

In this project we begin with the very crucial process of data preprocessing, which allows us to set the stage for effective feature extraction. This step is very crucial and ensures that the data used for our algorithms is the perfect fit and can give very good results. After data preprocessing we explored and experimented with various classifier models, highlighting the unique challenges posed by the nuanced nature of fake news. We used hybrid cnn models to integrate text and metadata. The models we tried are BiLSTM, CNN-LSTM, CNN-BiLSTM. For these three models we are using Text+All metadata along with our proposed metadata of Label+ subject + job + party + location.

We evaluated their performance in terms of accuracy and loss per epoch and fine tuned the hyperparameters to give the best accuracy possible. Through rigorous testing and validation, our model demonstrates a promising capability in identifying fake news.

1 Dataset

1.1 Overview of the LIAR Dataset

The LIAR dataset forms the backbone of our fake news detection system. It was published by william yang in 2017 and contains a comprehensive collection of over 12,000 statements collected from 2007 to 2016 by speakers and have been labeled for truthfulness. The statements are extracted from PolitiFact.com, a fact-checking website. These statements have been categorized into six truthfulness ratings: pants-fire, false, barely-true, half-true, mostly-true, and true. The dataset encompasses a diverse range of topics, primarily from the political arena, and includes statements made by politicians, political organizations, and media figures. Each entry in the dataset contains a label of truthfulness, a statement, its speaker, the speaker's job title, state of origin, party affiliation, the context or location where the statement was made and a credit history count (excluding true counts) of the speaker.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

1.2 Data Engineering and Preprocessing

When we conducted data analysis, we discovered many interesting aspects of the data. For instance, speaker affiliations were primarily republican, democrat, and others, with most speakers' jobs and their states being blank. To address this, we performed data cleaning to remove these blanks, followed by preprocessing, as direct feeding of statements to a deep learning model is not feasible. Below are the steps we performed before sending the final data to the model:

- **Data Cleaning:** We concatenated data from multiple columns and removed blanks to ensure consistency and quality in the data.
- **Text Normalization:** The text was cleaned using regular expression operations to eliminate unnecessary characters (such as special characters) and extra spaces. The cleaned data was then input to a tokenization function.
- **Tokenization:** The input data was converted to lowercase and split into tokens using the `word_tokenize()` function of the NLTK library.
- **Vocabulary and Mapping:** All unique tokens in the dataset were assigned a numerical value (index). We then padded shorter sequences with zeroes to ensure uniform length.
- **Label Mapping:** The label values were mapped to integers and converted into tensors.
- **Data Loaders:** The preprocessed data and labels were wrapped into a `TensorDataset` object, and subsequently into `DataLoader` objects.

These `DataLoaders` are utilized in our model for training, validation, and testing functions.

2 Model Description

2.1 List of Models Tried

We experimented with various models, including:

- **BiLSTM:** A model using only Bidirectional LSTM layers.
- **CNN-LSTM:** A combination of Convolutional Neural Networks and LSTM layers.
- **CNN-BiLSTM (Best Model):** A combination of Convolutional Neural Networks and Bidirectional LSTM layers.

The models were tested with combinations of Text+All metadata and Text+Proposed metadata. According to the LIAR research paper, models yield better results when text is paired with metadata. The proposed metadata includes Label, Subject, Job, Party, and Location.

2.2 BiLSTM Model

The BiLSTM model leverages Bidirectional Long Short-Term Memory layers to capture context and dependencies in sequential data like text.

- **Model Structure:**
  - **Embedding Layer:** Transforms words into dense vector representations.
  - **BiLSTM Layers:** Processes data in both forward and backward directions.
  - **Fully Connected Layer:** Maps the learned features to the desired output size.
  - **Output Layer:** Typically uses softmax or sigmoid activation for the final prediction.

2.3 CNN-LSTM Model

Combines Convolutional Neural Networks for feature extraction with Long Short-Term Memory networks for sequence modeling.

- **Model Structure:**
  - **Embedding Layer:** Converts words into vector representations.

- **CNN Layers:** Applies learnable filters to extract local features.
- **LSTM Layer:** Captures temporal dependencies and contexts in the data.
- **Fully Connected Layer and Output Layer:** Similar to the BiLSTM model.

2.4 CNN-BiLSTM Model

Incorporates a combination of Convolutional Neural Networks and Bidirectional Long Short-Term Memory networks.

- **Model Structure:**
  - **Embedding Layer:** Similar to the above models.
  - **CNN Layers:** Uses multiple convolutional layers with various filter sizes.
  - **BiLSTM Layer:** Processes data in both forward and backward directions.
  - **Fully Connected Layer and Output Layer:** Maps extracted features to the final output and produces the final prediction.

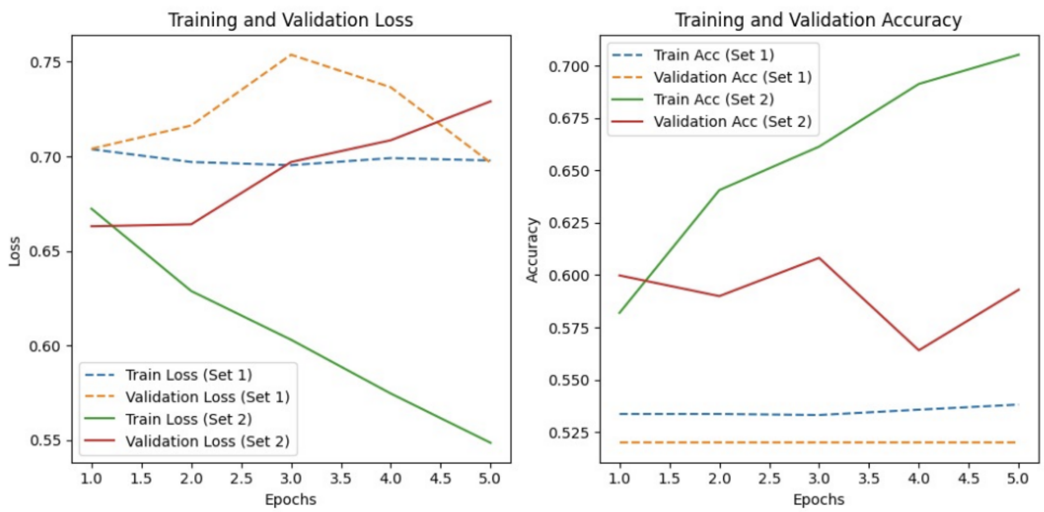

Figure 1: Hyperparameter Tuning Results

Hyperparameters: The graph illustrates the results after fine-tuning the parameters. We observed the best results with set-2 parameters, which included embedding dimensions of 100, 100 filters with sizes of [2, 3, 4], LSTM hidden dimensions of 256, dropout of 0.5, and a learning rate of 0.001. Set-1 had a dropout of 0.3 and a learning rate of 0.01.

3 Loss Function

3.1 Chosen Loss Function

For our project, we utilize two different loss functions depending on the nature of the classification problem:

3.1.1 Binary Classification

For binary classification tasks, `BCEWithLogitsLoss` was chosen. This loss function combines a Sigmoid layer and Binary Cross-Entropy (BCE) loss in one single class. It is more numerically stable than using a plain Sigmoid followed by a BCE loss. The reason for this stability is the potential problems that can arise from the imprecision of floating-point arithmetic when using the separate components. This function calculates the binary cross-entropy loss after applying the sigmoid function, making it particularly suitable for binary classification problems.

3.1.2 Multi-class Classification

For multi-class classification tasks, `CrossEntropyLoss` was used. This function first applies a softmax function to the output and then calculates the loss. This approach makes `CrossEntropyLoss` a standard choice for tasks involving multi-class classification, as it effectively manages the computation involved in multi-class output scenarios.

4 Optimization Algorithm

For our model, we chose the Adam optimizer as it is widely recognized as the best optimization technique available for NLP models. Adam is often the preferred choice for training deep learning models due to several key advantages it offers.

4.1 Rationale Behind Choosing Adam

The Adam optimizer stands out for the following reasons:

- **Adaptive Learning Rates:** Adam dynamically adjusts the learning rate throughout the training process. This feature is particularly beneficial for models like Transformers, which deal with large combinations of features.
- **Combination of Momentum and RMSprop:** Adam incorporates the advantages of both Momentum and RMSprop. Momentum is useful for handling sparse gradients, while RMSprop is known for accelerating the gradient descent process in the right direction.

We found that using the Adam optimizer with a learning rate of 0.001 yielded the best results for our model.

4.2 Other Optimization Techniques Explored

We also explored other optimization techniques, including:

- Stochastic Gradient Descent (SGD)
- RMSprop
- SGD with Momentum

However, these were ultimately not chosen. We considered incorporating Learning Rate Scheduling as an innovation in optimization, but this was deemed unnecessary as the Adam optimizer already includes an effective learning rate scheduling mechanism.

5 Metrics and Experimental Results

5.1 Proposed Metadata

The proposed metadata used in our experiments consists of Label, Subject, Job, Party, and Location.

5.2 Results for 6 Class Classification

We tested three models with both Text+All and Text+Proposed metadata. The following table presents our findings:

Models (6 class classification)	**Features Used**	**Validation**	**Test**
BiLSTM	Text+All	26.68%	23.73%
CNN LSTM	Text+All	25.15%	22.95%
CNN BiLSTM	Text+All	26.30%	23.15%
BiLSTM	Text+Proposed Metadata	24.70%	23.34%
CNN LSTM	Text+Proposed Metadata	26.22%	24.61%
CNN BiLSTM	Text+Proposed Metadata	**28.05%**	**26.44%**

Table 1: The evaluation results on the LIAR dataset for 6 class classification.

Discussion: Table 1 has the results for the 6 class classification we obtained for our three models with Text+All and Text+proposed metadata. In the LIAR research paper, the best test result obtained(27.4%) was for the Hybrid CNN model with Text+All. **The highest test result we got is 26.4% for the CNN BiLSTM model with our proposed metadata**. For all of the above results we ran the models for 5 epochs. .

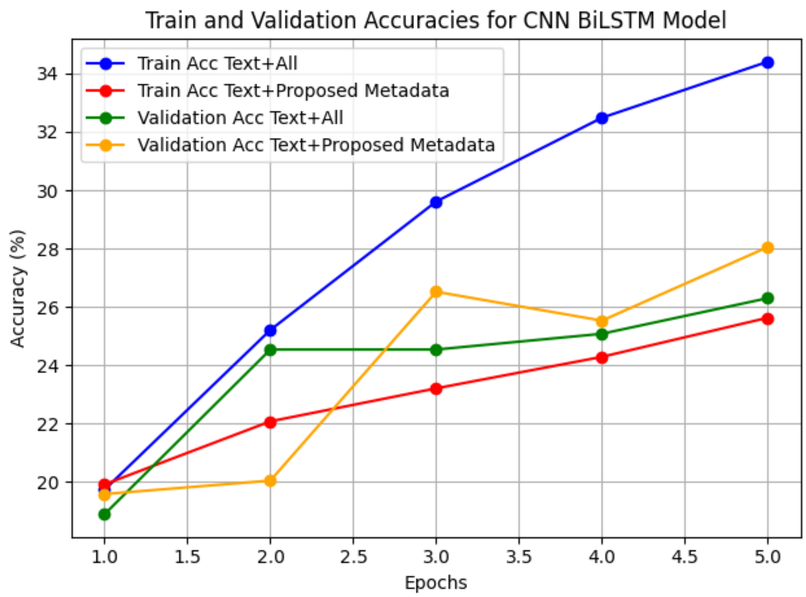

Figure 2: Training and validation accuracy graph for 6 class classification of the CNN BiLSTM model

5.3 Results for 2 Class Classification

Similarly, we conducted experiments for 2 class classification, and the results are as follows:

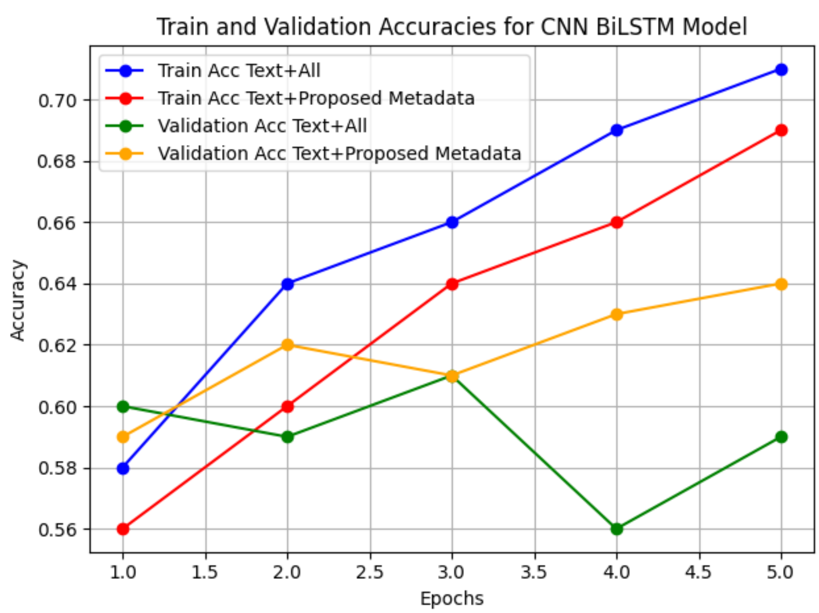

Figure 3: Training and validation accuracy graph for 2 class classification of the CNN BiLSTM model

Models (2 class classification)	Features Used	Validation	Test
BiLSTM	Text+All	64%	64%
CNN LSTM	Text+All	61%	62.38%
CNN BiLSTM	Text+All	59%	60.25%
BiLSTM	Text+Proposed Metadata	64%	62.72%
CNN LSTM	Text+Proposed Metadata	62%	64.10%
CNN BiLSTM	Text+Proposed Metadata	64%	**65.25%**

Table 2: The evaluation results on the LIAR dataset for 2 class classification.

Discussion: As shown in Table 2, the CNN BiLSTM model with Text+Proposed Metadata provided the best accuracy of 65.25% in 2 class classification tests.

5.4 Metrics Used

The primary metric used to evaluate the models was Accuracy, specifically focusing on validation and test accuracy. This metric is essential in classification tasks as it indicates the proportion of correctly predicted instances.

5.5 Comparison of Experiments

Our experiments revealed that, among all models tested, the CNN BiLSTM model with Text+Proposed Metadata outperformed the others in both 6 class and 2 class classifications. This indicates that the combination of CNN and BiLSTM architectures, along with the proposed metadata, is highly effective for the task of fake news detection using the LIAR dataset.

5.6 Conclusion

To conclude, our project shows promising results, especially using CNN-BiLSTM and text+proposed metadata methods. Although effective, model accuracy can definitely be improved by expanding the dataset(example: 100,000 samples) and fine-tuning its parameters. In the future, we will focus our efforts on incorporating more diverse samples and advanced natural language processing techniques. This will help adapt the models to get more accurate with fake news detection.

6 Contributions

The contributions to this project were as follows:

- Dharma Karan Reddy Gaddam - 34%
- Rakshith Venkatesh Murthy Gowda - 33%
- Thrishal Reddy Desam - 33%

7 GitHub and Web Link

The project's code and additional resources are available on GitHub and our web link:

- GitHub: `https://github.com/thrishal/DL_Group11`
- Web Link: `https://junior-tall-sponge.anvil.app`

8 References

The following references were instrumental in our research and development:

1. `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`
2. `https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0`

3. https://www.mdpi.com/2673-4591/39/1/33
4. https://machinelearningmastery.com/using-optimizers-from-pytorch/
5. https://arxiv.org/pdf/1705.00648v1.pdf
6. https://towardsdatascience.com/identifying-fake-news-the-liar-dataset-713eca8af6ac