Fake News Classification

Author
Prajusha Reddy
Boston University
prajusha@bu.edu

1 Goal

The goal of this project is to be able to classify whether a given article is fake or real news, and also to determine which type of Natural Language Processing (NLP) model performs the best for this classification task. By analyzing the text, the models in this project should be able to determine the credibility of the information. I chose this as my project because I believe that it is more important now than ever before to be able to distinguish between fake and real news; in today's digital age, misinformation spreads rapidly through the use of the internet and different social media platforms (I have seen this happen first-hand through conversations with family, friends, peers, etc.), which is why it is essential to develop tools to identify accurate information for people to make fully informed and smart decisions.

2 Methods

For this classification task, I implemented and compared three different models.
Naïve Bayes Model: The Naïve Bayes model is a simple probabilistic model that uses a counting Bag-of-Words approach with smoothing to calculate class-conditional probabilities.
Long Short-Term Memory (LSTM) Model: My LSTM model is pretty simple; it consists of an embedding layer, followed by an LSTM and linear ouput layer to predict the label of the article.

In order to train the LSTM model, I created a custom PyTorch `Dataset` class called `NewsDataset`. It prepares each text-label pair for training by converting each tokenized sentence into a sequence of word indices using a vocabulary dictionary, assigning a special index to unknown words (`<UNK>`), and zero-padding shorter sequences up to a fixed maximum length (`max_len`), and returning each text as a tensor of fixed length and its corresponding label as a tensor.

Bidirectional Encoder Representations from Transformers (BERT) Model: I also implemented the BERT model (specifically the BERTForSequenceClassification model) using the HuggingFace Transformers library (Wolf et al., 2020), which is a pre-trained Transformer model. In my implementation, I fine-tuned the model on my labeled news dataset.

2.1 Existing Approaches

There has been a lot of work done to explore logistic regression, support vector machines (SVMs), recurrent neural networks (RNNs), Transformer models, and many other machine learning models for this fake news classification task. I simply wanted to research the progression from a simpler model, such as Naïve Bayes, to more modern, sophisticated models, like the LSTM and BERT models.

3 Experiments

3.1 Data

My experiments were conducted using a publicly available Kaggle dataset used for fake news classification (Singh, 2019). The dataset contains over 40,000 unique news articles and is split into three smaller datasets, news_train, news_val, news_test (as named in my project directory), which are used for training, validation, and testing, respectively. The testing and validation sets both consist of 8,117 articles, whereas the training set consists of 24,353 articles, which I cut down to 10,000 in my code since it was causing memory issues. The data has columns for the article's ID, title, text, and label (0 for false, 1 for true).

3.2 Preprocessing

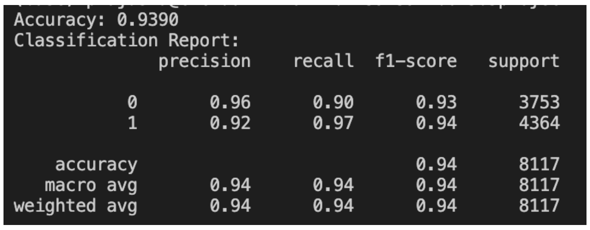
To preprocess the "text" column from the datasets for the Naïve Bayes and LSTM models, I created a function, `preprocess_text` in the `train_lstm.py` and `train_nb.py` files which takes the text from the article, removes unwanted characters (like punctuation) using regex, and tokenizes it. I also used the `NewsDataset` class to prepare the datasets for training the LSTM model, which I explain in the **Methods** section. For the BERT model, I used the `bert-base-uncased` (Turc et al., 2019) tokenizer with padding and truncation to a maximum sequence length of 128.

3.3 Metrics

The evaluation metrics I used for each of the models were accuracy, precision, recall, and F1-score. I chose these because they are the metrics most commonly used for evaluation for classification purposes. In order to get these evaluation metrics, I used `scikit-learn`'s accuracy_score and classification_report, and compared the true labels to the predictions of each model on the test set.

3.4 Baseline (Naïve Bayes)

The first model that I began with was the Naïve Bayes model. This model is used as the main baseline since it is the simplest model for text classification. For this model, I achieved pretty good accuracy, precision, recall, and F1-score, which can be seen in Figure 1. As you can see, the accuracy for this model is approximately 93.9%, which is pretty good for a baseline model, but it can be increased using the two other models. This is because Naïve Bayes assumes conditional independence, and tends to ignore structure and semantic relationships.



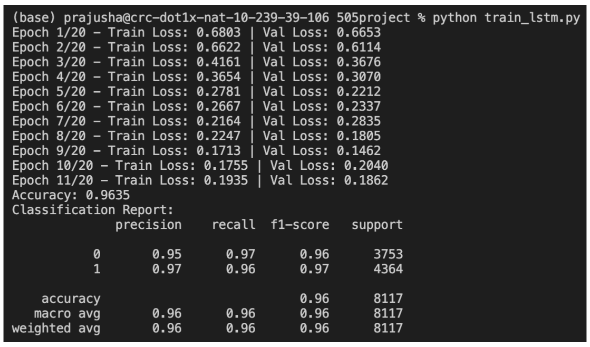
```
Accuracy: 0.9390
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.90      0.93      3753
           1       0.92      0.97      0.94      4364

    accuracy                           0.94      8117
   macro avg       0.94      0.94      0.94      8117
weighted avg       0.94      0.94      0.94      8117
```

Figure 1: Naïve Bayes Metrics

3.5 LSTM

My second model, which is the LSTM model, yielded better results than my Naïve Bayes model. This was to be expected because unlike Naïve Bayes, an LSTM is a deep learning model that is able to capture word order and context using LSTM units.

In the beginning, I was playing around with the number of epochs and learning rate for training. At first, I used 2 epochs and a learning rate of 1e-5 for training, which resulted in low accuracy (approximately 53.84%). This was expected since the number of epochs was low. After that, I increased the number of epochs to 10 with the same learning rate, which resulted in an accuracy of around 57.92%. After that, I increased the number of epochs to 20 and learning rate to 1e-4, and also employed an early stopping method by tracking the validation loss. The early stopping method stopped training if the validation loss continued to increase for 3 epochs. With this, the LSTM model stopped training after 11 epochs, and resulted in much higher metrics than before, as shown in Figure 2. As you can see, the accuracy is much higher (96.35%) when implementing these changes. There may have been other issues with the model which I fixed in between, but I attributed it mainly to the number of epochs and learning rate.



```
(base) prajusha@crc-dot1x-nat-10-239-39-106 505project % python train_lstm.py
Epoch 1/20 - Train Loss: 0.6803 | Val Loss: 0.6653
Epoch 2/20 - Train Loss: 0.6622 | Val Loss: 0.6114
Epoch 3/20 - Train Loss: 0.4161 | Val Loss: 0.3676
Epoch 4/20 - Train Loss: 0.3654 | Val Loss: 0.3070
Epoch 5/20 - Train Loss: 0.2781 | Val Loss: 0.2212
Epoch 6/20 - Train Loss: 0.2667 | Val Loss: 0.2337
Epoch 7/20 - Train Loss: 0.2164 | Val Loss: 0.2835
Epoch 8/20 - Train Loss: 0.2247 | Val Loss: 0.1805
Epoch 9/20 - Train Loss: 0.1713 | Val Loss: 0.1462
Epoch 10/20 - Train Loss: 0.1755 | Val Loss: 0.2040
Epoch 11/20 - Train Loss: 0.1935 | Val Loss: 0.1862
Accuracy: 0.9635
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      3753
           1       0.97      0.96      0.97      4364

    accuracy                           0.96      8117
   macro avg       0.96      0.96      0.96      8117
weighted avg       0.96      0.96      0.96      8117
```

Figure 2: LSTM Metrics

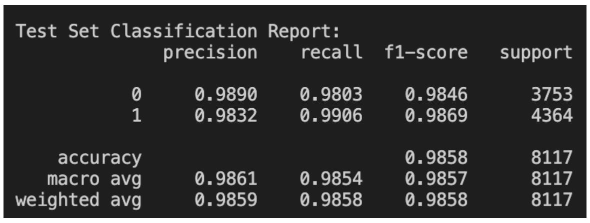
Although this model does very well, I wanted to see if a pre-trained Transformer model like BERT would perform even better, which it did.

3.6 BERTForSequenceClassification

For my BERT model, I achieved really high evaluation metrics, which I expected since it is a pre-trained Transformer model, meaning it uses a self-attention mechanism to capture relationships between words in a sentence, making it really good for text classification and semantic tasks. Figure 3 shows the evaluation metrics for the BERT model.

As you can see, the metrics are all around 98%, with the accuracy being 98.58%, which is extremely high.

When fine-tuning the BERT model, I used 3 epochs to train it. However, after being able to track the training and validation loss for the duration of the training (using wandb), I would probably train for 1 epoch. In Figure 4, you can see the



```
Test Set Classification Report:
              precision    recall  f1-score   support

           0     0.9890    0.9803    0.9846      3753
           1     0.9832    0.9906    0.9869      4364

    accuracy                         0.9858      8117
   macro avg     0.9861    0.9854    0.9857      8117
weighted avg     0.9859    0.9858    0.9858      8117
```

Figure 3: BERTForSequenceClassification Metrics

training loss decrease during training. However, in

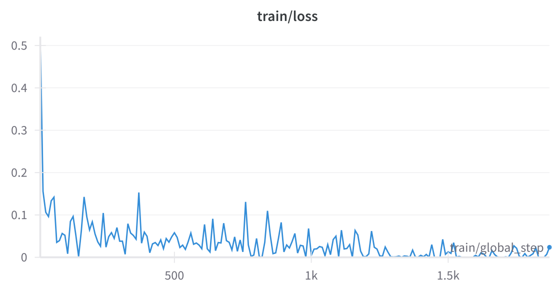


Figure 4: Training Loss (3 Epochs)

Figure 5, you can see the validation loss increase (although the difference is very small) by the end of the training. This shows me that perhaps 1 epoch of training is enough for my fine-tuned BERT model, and may actually lead to higher accuracy on the test set.

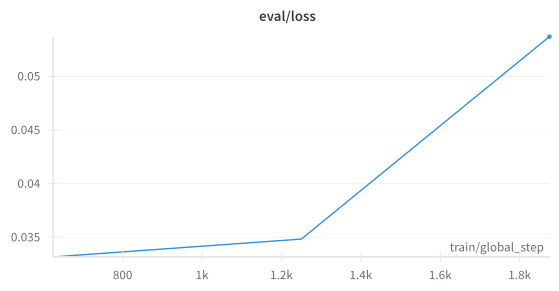


Figure 5: Validation Loss (3 Epochs)

4 Conclusions

Through my experiments, I learned that while a simple model like Naïve Bayes can achieve decent results with minimal computation, deep learning models, such as the LSTM and BERT models, significantly outperform it in terms of text classification, such as fake news classification. The BERT model especially, being a Transformer model, yields very good results. Additionally, I also observed that the BERT model achieved very high accuracy with very few training epochs, and it may have even better results with just one epoch, which showcases its dominance in text classification tasks.

References

Aadya Singh. 2019. Fake news classification dataset. https://www.kaggle.com/datasets/aadyasingh55/fake-news-classification. Accessed: 2025-04-30.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.