Name: Amil Salihi

Class: MATH 475 Statistical Machine Learning

GitHub link to the repository: https://github.com/amilsalihi00/math475-final

Final Project

1. Introduction

For the final project, I chose a bit more challenging task, as we have not faced this kind of problem in class yet. The project's goal is to develop a model to classify text into clickbait and non-clickbait based on the data from this Kaggle repo: https://www.kaggle.com/datasets/amananandrai/clickbait-dataset. To do the classification, I used common RNN models such as LSTM and GRU. The dataset itself contains 16000 clickbait and non-clickbait headlines, where the non-clickbait headlines come from sources such as The Guardian, New York Times, etc., while clickbait headlines come from sources like UpWorthy, ViralStories, etc.

2. Data Exploration & Preprocessing

The first thing we do is load the dataset and explore the distribution of clickbait and non-clickbait dataset, which here is approximately 50/50 with around 16000 headlines per class. The next step would be to learn more about the data itself. With text data, that is not a simple task, and the tools to do so are limited. First, we check if there are any missing data, and there is no missing data, so we can proceed with other tools. What we can do is check the distribution of words in clickbait and non-clickbait headlines. The word distribution is similar in both classes, left-skewed histograms. It seems however, that clickbait data contains more of the texts centered around 9 words, while the non-clickbait data has a wider spread of word counts, and with the highest bin at 8 words (see Figure 1). As next step, we can look at the word cloud, which shows the frequency of words appearing for each class. From Figure 2 and Figure 3, we can see that the clickbait data has more sensational words in their headlines, while non-clickbait data has more serious words.

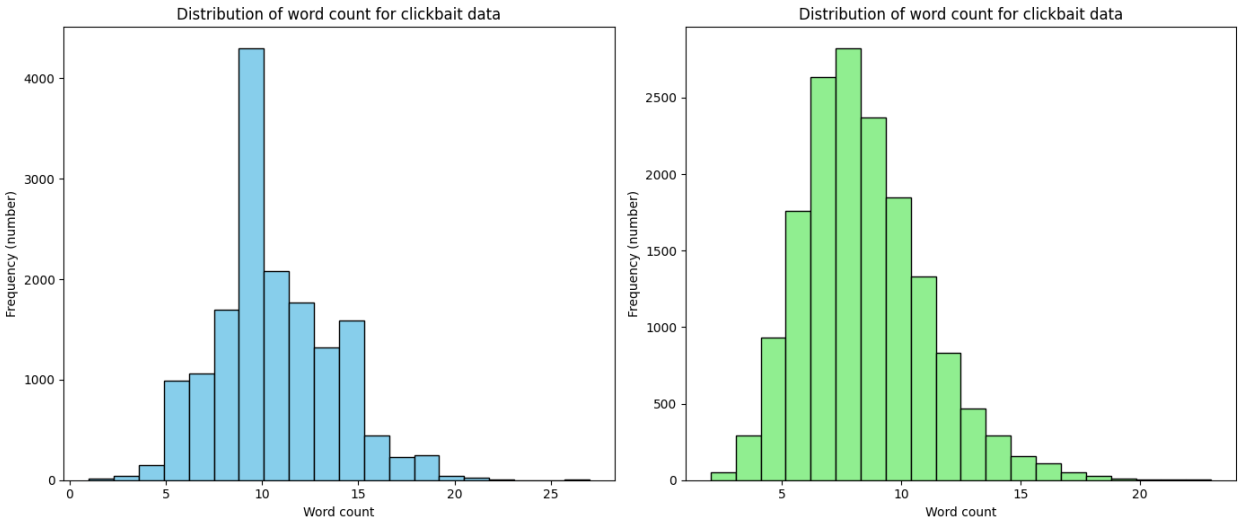

Figure 1 Word distribution count in non-clickbait and clickbait data.

Figure 2 Word cloud for non-clickbait data.

Figure 3 Word cloud for clickbait data.

For text data, preprocessign is different than for tabular data and is used for models to learn the text parts better. For the headlines, we do lemmatization and stemmatization, alongside removing punctuation and similar common steps (e.g. remove trailing whitespace, tabs, etc.). After that, we split the data in train/test and we create a vectorizer to prepare the data for the model in its understandable way

3. Model Selection & Training

For selecting the models, I started with a basic LSTM model with no regularizers, in order to create a baseline. I chose this architecture because LSTMs are effective in capturing temporal dependencies in sequences, and using a bidirectional layer helps learn from both past and future contexts, which is especially helpful for sequential data. The model uses a vocabulary-based embedding layer, followed by a bidirectional LSTM with 128 units and two dense layers to predict the output. I used a learning rate of 0.001 for the Adam optimizer, which is generally effective for training RNNs. The model is evaluated using accuracy, which is suitable for binary classification tasks.

For subsequent models, I changed small parts of the architecture, such as adding additional layers or regularizers. For model 2 for example, I added a dropout of 0.1 (I experimented with different dropout of 0.1 to 0.5 with that model, however, 0.1 got the best results). In other models, I also add additional layers, which should help the LSTM understand better, increased the size of neurons from 128 to 256, and also added L1 and L2 regularizers, as well as comibing L2 regularizers with dropout.

The next step was to find a suitable vectorizer vocabulary size, which would be optimal for the models and the data. The vectorizer with vocabulary size 2000 and max size of 30 was selected. With that, we could proceed with training the models.

For training the models, in my function I used epochs between 10 and 300, however, there was no significant advantages to more than 10 epochs, and there was a drawback of computational power. I also added early stopping. Batch size was 64 in all trainings.

After LSTM, I used GRU models for comparison, following a similar approach (starting from a simple baseline, then adding regularizers). Training setup remained the same. Selected models and their metrics are presented in the next section.

4. Performance Evaluation

Selected models and their metrics can be seen in the table below:

Model name	**Architecture**	**Train accuracy**	**Train loss**	**Val accuracy**	**Validation loss**
Model_1 (baseline)	LSTM	0.9986	0.0169	0.9865	0.1241
Model_l1005 (L1 regularizer)	LSTM	0.9721	0.7087	0.9844	0.4483
Model_l2001Dropout1 (L2 regularizer + dropout)	LSTM	0.9657	0.5460	0.9875	0.1935
Model_Dropout01	LSTM	0.9950	0.1948	**0.9900**	0.2609
Model_13_GRU (baseline, 2 layers)	GRU	0.9991	0.0137	0.9869	0.1024
Model_12_GRU (4 layers, no regularizers)	GRU	0.9992	0.0131	0.9818	0.1704
Model_11_GRU_L1L2dropout (Regularizers + dropout 0.25)	GRU	0.9858	0.3062	7.8125e-04	16.088

Table 1 Models and their performances.

The main evaluation metric was accuracy, which is a simple and clear way to measure how well the model classifies the data. The BinaryCrossentropy loss function was used to ensure that the model not only made correct predictions but also gave high-confidence predictions. By monitoring both accuracy and loss, especially on the validation set, we could ensure the model performed well on unseen data. From there, we can see that all models excelled in performance, however, the Model_Dropout01 had the best performance of all models with a training accuracy of 99.50% and a validation accuracy of 99.00%. The training loss was 0.1948, and the validation loss was 0.2609. These results show that the model is learning effectively and generalizes well to new data. The validation loss is slightly higher than the training loss, the small difference suggests only a minor risk of overfitting.

The best model, in this case Model_Dropout01, was chosen to be trained on the full data and later on tested with additional metrics. We received the train accuracy of 0.9972 and loss of 0.0357, with the test accuracy of 0.9875 and loss of 0.1554. Next, confusion matrix for the model was plotted, to see the ratio of the false positives and false negatives, as seen in Figure below.

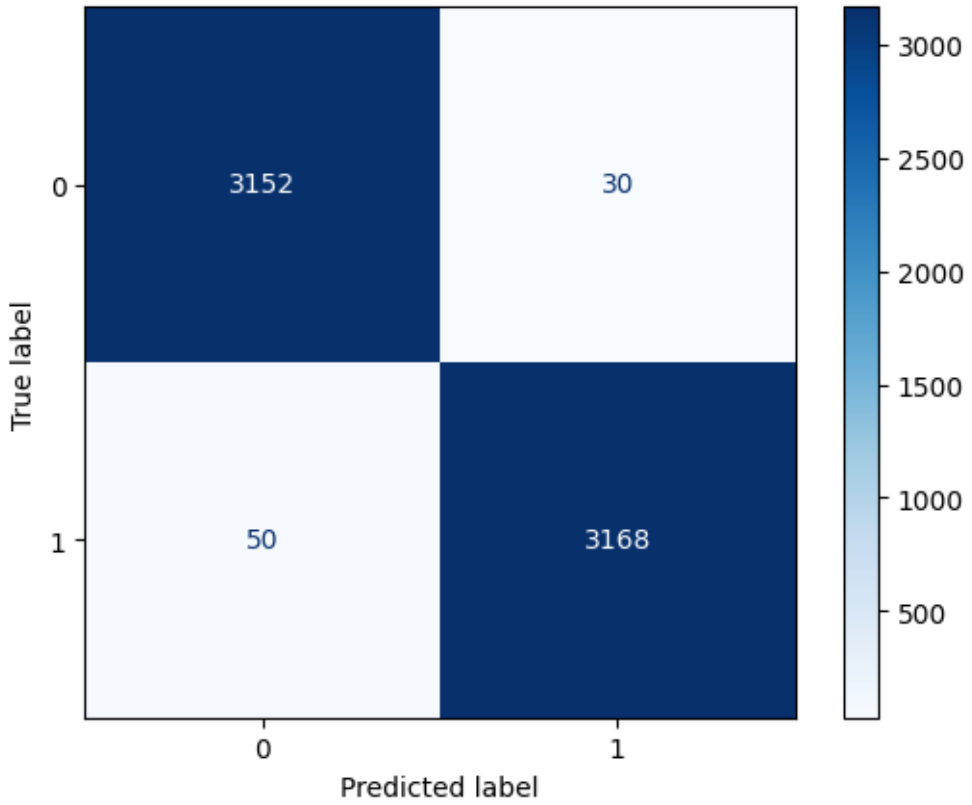

Figure 4 Confusion matrix on the test data of the chosen model.

From the confusion matrix we can also see that the model performs the classification well, with only a few misclassified samples.

5. Challenges & Future Work

During working with these models, I have faced several challenges. First and main challenge was computational limitations, as RNNs can be heavy models and both locally and on Google Colab it took some time for a model to be trained. However, there was not much I could do about this challenge, as it simply required time for training.

Another challenge was finding the right vectorizer, specifically vocabulary and max size for it. For that I simply took a baseline model and trained it on several vectorizers, choosing the best one. Next, it was slight overfitting with the baseline model, which I've overcome by adding regularizers or dropout. Choosing the right model architecture has also been challenging. Deciding on the depth of the model, the number of LSTM layers, and whether to include features like Bidirectional LSTMs required careful consideration. Hyperparameter tuning was another main challenge. Selecting appropriate values for learning rates, batch sizes, and patience for early stopping have been done manually, hence it took some time.

In the future, I would implement a grid-search for hyperparameter optimization, so that I would not have to tune them manually. I would also be more careful in spotting the overfitting at the beginning so not to

waste time with other parts, but rather to implement the regularizers from the beginning. Apart from that, now I have more experience with RNNs, and in the future I would experiment with combining RNNs with CNNs, as well as trying out Transformers, which are more power-consuming to train, but they promise better results when it comes to handling text, including my scenario of classifying text.