PROPAGANDA DETECTION USING NLP MODELS

1. Introduction

In this report, we will see how different Natural Language Processing (NLP) techniques have been used in the given dataset of the statement to detect and classify the propaganda conveyed by each statement. We will investigate the performance of each technique while classifying different types of propaganda. We will examine the performance of several algorithms such as Bidirectional LSTM, and transformer-based models like BERT based on a dataset. These approaches will be used to solve two problems, they are:

- To identify whether the statement contains propaganda or not
- To detect which type of propaganda is conveyed by the statement

In this analysis, for categorizing propaganda and not-propaganda RNN, LSTM and BERT are used. The categorization of the propaganda label is done using LSTM and BERT, respectively. We will also discuss the hyperparameters in each model, along with the optimal value for these parameters, and how they affect the accuracy of each model. Furthermore, a comparison of the models is made based on accuracy, F1, precision and recall.

2. Dataset

The dataset used for the analysis is ‘’**propaganda_dataset_v2**”, which presents a dataset consisting of 2 files with identical formats; one is for testing, and the other is for training. The dataset consists of labelled sentences belonging to one of nine categories. Flag-waving, appeal to fear/prejudice, causal simplification, doubt, exaggeration/minimization, loaded language, name-calling/labelling, repetition, and not propaganda are the categories. The labels are represented in the first column and a sentence or chunk of text in the second column where propaganda or not-propaganda technique is identified.

3. Data pre-processing

A semi-structured or unstructured dataset may contain a great deal of information, but it may not play an important role in predicting the outcome of a given task, even though it may be interesting to analyze. Moreover, large datasets require long processing times and stop words reduce prediction accuracy. So, text preprocessing is both necessary to save computational resources and increase accuracy time. For the first requirement the following pre-processing tasks are done.

- Replacing <BOS> and <EOS> with an empty space since <BOS> and <EOS> doesn’t contribute to the meaning of the sentence.
- The labels are replaced with binary i.e., 0 for propaganda and 1 for not-propaganda for the purpose of binary classification.

- Removal of stop-words, multiple space, single character, punctuation, and number by invoking the stop-word library in NLTK.

The following preprocessing steps are carried out for the second requirement:

- We are splitting the statement into 3 parts and taking only a chuck of text that is present in between the <BOS> and <EOS> token.

4. Models for Text Classification

4.1. Recurrent Neural Network (RNN)

An RNN is a variation of a feed-forward network, in which the hidden layer neurons receive inputs with a delay. RNNs process sequences of data by passing information from one time step to another. The feedback loop allows them to use previous outputs as inputs for the current time step. RNNs are flexible and can be trained online, enabling them to handle sequential data in real-time for a variety of applications. These advantages have made them a powerful and versatile architecture for handling sequential data.

For the pre-processing of RNN we use the **preprocess_text().** In order to convert text into integers or vectors, a Tokenizer instance is created with **num_words** parameter set to 10000, that will take only the most frequently used 10000 words by limiting the vocabulary size, which will decrease memory usage and increase efficiency. To update the vocabulary based on our text, we use **fit_on_text()**. Each text passed is of varying length, and **pad_sequences** is used to convert them all to the same length. In the next step we create weighted matrix for the word embedding by assigning the word embedding from the glove model if the word of our vocab list is present in the glove word embedding whereas it is assigned with 0 otherwise using the pretrained **glove-wiki-gigaword-100** model. **Sequential()** is used to create RNN model and we are adding layers into the model, here four layers are added.

- The first layer is an **Embedding** layer, which maps the input sequences of integers (i.e., words) to dense vectors of fixed size (**EMBEDDING_DIM = 100**). The weights of this layer are pre-trained using a **gensim_weight_matrix**, which is a matrix of pre-trained word embeddings that are not trainable (**trainable = False**).
- The second layer is a **SimpleRNN** layer which will add 64 units to the model. The SimpleRNN layer will return output (**return_sequences= True)** at each time step in the input sequence, rather than just the output at the final time step.
- The third layer is **Flatten** layer, RNNs are designed to handle sequential data and can return output at each time step in the input sequence. However, fully connected layers require a fixed-size input, which means that the output of the RNN layer needs to be flattened or reshaped into a 2D tensor. This task is done by flatten layer.

- The fourth layer is a **Dense** layer with a sigmoid activation function and class_num as 2 since we are dealing with 2 classes. This layer will output a single probability value between 0 and 1, representing the predicted class probability.

The model is compiled using the binary **cross-entropy** loss function for calculating loss for binary classification along with **Adam** optimizer. Adam optimizer is a commonly used and well-performing stochastic gradient method. It has a learning rate of 0.001 by default. The model's performance will be evaluated using the accuracy metric. RNNs have the disadvantage of being slow to train, particularly if the input sequence is long. Additionally, RNNs are susceptible to the problem of vanishing gradients, which can make them difficult to train and result in poor performance.

4.1.1 Hyperparameter setting

Setting hyperparameters is essential to developing effective natural language processing (NLP) models. Hyperparameters can have a significant impact on model performance in NLP. A model's speed, accuracy, and generalizability can be affected by these factors.

- **Output_dim :** This parameter is passed in embedding layer. This parameter specifies the dimensionality of the dense vector of words **(output_dim = EMBEDDING_DIM).** In this model we are setting it to 100. The higher the value of output_dim it will capture more semantic information but lead to over usage of computational resources. Lower value of this parameter means it will capture poor information of words
- **No of hidden units:** This hyperparameter defines the number of nodes in the hidden layer. It is set to 64. More nodes mean it will capture complex patterns but can also lead to overfitting. Increasing the number of nodes helps models to learn intricate patterns.
- **Epochs:** is set to 10, this determines the number of times the model should iterate over the entire dataset.
- **Class_no:** This hyperparameter specifies how many classes the data must be classified. It is set to 2.
- **Activation function:** this specifies the activation function used in dense layers. Since we are doing binary classification, we are using sigmoid classification
- **Loss function:** this hyperparameters defines loss function used. Since we are doing binary classification, we are using binary_crossentropy.
- **Optimizer:** This hyperparameters determines which optimizer is used in model. We are using Adam optimizer which is very efficient.

4.2. Bidirectional LSTM model

Bidirectional LSTM models are recurrent neural networks (RNNs) that can process sequences of data both forward and backward. The bidirectional LSTM makes predictions based on information from both directions simultaneously. It can enhance sequence modeling and handle long-term dependencies in sequential data, increase robustness to noisy data, and provide flexibility in handling different types of input data.

In text classification, it is beneficial to use a pretrained word embedding provided by Genism called "glove-wiki-gigaword-100" that produces 100-dimensional GloVe embeddings trained on both Gigaword and Wikipedia. Thus, we create a weighted matrix for the word embedding by assigning the word embedding from the glove model to our vocab list if it is present in the glove embedding, otherwise we assign 0 to it. We create an LSTM model with Sequential(). The model contains different layers in it.

- In the first layer, there is a **Bidirectional** layer with an LSTM layer inside. Bidirectional layers create copies of LSTM layers that process input sequences in reverse order, letting the model capture context both ways. The layer has 100 units and returns a sequence of outputs for each input element (return_sequences=True).
- The second layer is a **Dropout** layer, which drops 20% of inputs randomly to prevent overfitting.
- The third layer is another **Bidirectional** layer with an **LSTM** layer inside. This layer has 200 units and returns a sequence of outputs for each input element.
- The fourth layer is another **Dropout** layer.
- The fifth layer is another **Bidirectional** layer with an **LSTM** layer inside. This layer has 200 units and returns a single output for the entire input sequence (**return_sequences=False**).
- Lastly, a **Dense** layer with a **sigmoid** activation function will output a single class probability value between 0 and 1. class_num is taken as 2 for binary class classification. For the multi-class classification instead of the sigmoid function a **softmax** function is implemented which transforms the output of the layer into vector of the probability. The multi-class classification has the class_num set to 8, which determines the dimensionality of outer space.

This model is compiled using the **binary_crossentropy** loss function for binary and **categorical_crossentropy** for the multi-class classification along with the Adam optimizer. The accuracy metric will be used to evaluate the model's performance. LSTM networks are more complex than other types of recurrent neural networks, which can make them more difficult to understand and implement. They are also computationally expensive and require a large amount of memory.

4.2.1 Hyperparameter setting

In order to build effective machine learning models, hyperparameters must be set by the user before training. Hyperparameters are parameters that do not change during training.

- **EMBEDDING_DIM (**value-100) – that is the dimensionality of the word embedding. Each word is represented in 100 dimensions. In the model the word embedding is represented in 100 dimensions. The higher the value it will capture the more semantic information, but it may lead to over usage of memory and computational resources.

Lower values will capture less useful semantic information and will result in poor word embedding.

- **Units in LSTM** – The number of LSTM units in each layer represents the complexity of the layer. The more number of units in the layer will lead to a complex problem and lead to overfitting. Increasing the number of nodes helps models to learn intricate patterns. In the first layer of Bidirectional LSTM is 100 and in the next 2 layers of Bidirectional LSTM it is 200.
- **Epochs:** is set to 12 for binary and 25 for multiclass, this determines the number of times the model should iterate over the entire dataset.
- **Dropout rate**- the value for dropout rate is 0.2. Usually, the dropout rate lies within the range of 0.2-0.5. It determines the number of that needs to be dropped during training. A higher dropout rate can give higher regularization, but it may result in underfitting of the model.
- **Loss function** – it is set to binary_crossentropy for binary and categorical_crossentropy for multiclass classification. This loss function is suitable to calculate loss for binary and multiclass classification.
- **Optimizer** – We are using Adam optimization algorithm which adapts learning rate during training. It also achieves good results fast.
- **Learning rate-** We are taking the default learning rate of the adam optimizer which is 0.001. The optimal learning rate means it will update the parameters optimally to reach local minima.
- **Batch size** – batch size refers to the number of training samples processed in each epoch. A larger batch size can provide computational efficiency but may also result in less noise in the gradient estimation, potentially causing the model to converge to a flatter minimum. Smaller batch sizes may introduce more noise but can allow the model to converge to sharper minima. The choice of batch size should consider the available computational resources and the properties of the dataset.

4.3 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a transformer-based neural network architecture pre-trained on large amounts of text data to learn contextualized word embeddings. BERT is unique in that it is trained bidirectionally, which means that it considers both the preceding and succeeding words in a sentence to predict the target word. As a result, BERT produces more accurate word embeddings by better capturing words' context. It is preferable to use the **bert_base_cased** model for propaganda dataset classification since it has a high level of accuracy and is capable of understanding complex word relationships. Machine learning using this method outperforms traditional methods when it comes to various NLP tasks, which makes it a popular choice for classifying propaganda datasets. This report aims to determine if a sentence contains propaganda and to identify the specific propaganda techniques used by the multiclass model.

The dataset is prepared for model training and evaluation by tokenizing it with the BERT tokenizer. A pre-trained BERT model is loaded on a dataset using the AutoModelForSequenceClassification class, and a classification layer is added to it. Input text is processed through multiple layers, and logits are obtained. The model is then fine-tuned on the dataset based on transfer learning techniques. After passing these logits through a softmax function, the model predicts probability levels for each class, which represents its confidence level.

4.3.1 Hyperparameter setting

- **max_length** determines the maximum length of a tokenized sequence to ensure that all input sequences are of the same length for efficient batch processing, and it is set to 140 to match the length of the input data.
- **num_labels** indicate the total number of labels or classes that the model needs to predict for a given classification task. The value is set to 2 for binary classification and 8 for multi-model.
- **Logging_steps** is the number of steps between two logs, and it is set to 100
- **num_train_epochs** determine the number of times the model should iterate over the entire dataset. Since the dataset used here is small, it is set to 5 and 7 which is considered reasonable.
- **per_device_train_batch_size** and **per_device_eval_batch_size** determines the number of samples in a batch during training and evaluation. Both are set to 1, to avoid any memory-related problems during training on a GPU.
- **learning_rate** determines the scale of the updates made to the model's parameters during each iteration of the optimization algorithm, specifically the gradient descent algorithm used for model training. In this case, it is set to the default value 5e-6 suitable for pre-trained model.
- **evaluation_strategy and eval_steps** is the strategy for the evaluation during the training and helpful to monitor the model performance during fine-tuning. In this scenario, the model evaluation is done after each epoch's completion, with a constraint of 100 evaluations at maximum.
- **early_stopping_patience** specifies the number of epochs to wait for before terminating the training process if the model's performance does not improve. When set to 1, as in this case, training will be stopped if there is no improvement in performance after a single epoch.

5. Evaluation and Analysis

5.1 Results of binary classification

Classifier	Accuracy	Precision	Recall	F1
RNN	0.64	0.66	0.66	0.66
LSTM	0.75	0.75	0.74	0.74
BERT	0.96	0.97	0.95	0.96

Table 1: Binary classification

The RNN model achieved an accuracy of 0.64 with a moderate ability to identify positive cases, while the LSTM model achieved a higher accuracy of 0.75 with a balanced performance in correctly identifying positive cases and avoiding false positives. The BERT model outperformed both RNN and LSTM with an accuracy of 0.96 and a high ability to correctly identify positive and negative cases while avoiding false positives and negatives. Overall, BERT showed the best performance across all metrics, followed by LSTM and then RNN.

5.2 Results of multi-class classification task

Classifier	Accuracy	Precision	Recall	F1
LSTM	0.31	0.48	0.31	0.36
BERT	0.57	0.56	0.57	0.56

Table 2: Multi-class classification

The evaluation shows that the BERT model outperforms the LSTM model in all metrics. The LSTM model achieved a low accuracy, precision, recall, and F1 score, indicating poor performance in identifying positive instances. In contrast, the BERT model achieved high accuracy, precision, recall, and F1 score, indicating better performance in correctly identifying positive instances. Therefore, the BERT model can be considered as a better classification model than the LSTM model based on the evaluation metrics.

5.3 Model Comparison

There are notable differences between binary and multi-class classification models based on the evaluation metrics. The binary classification models, especially BERT, performed better than the multi-class classification models in correctly identifying the positive instances. The LSTM model showed consistent performance in both binary and multi-class classification. Still, the overall performance of the models in multi-class classification was lower than binary classification, indicating the complexity of identifying multiple classes simultaneously. Binary classification involves categorizing instances into only two classes, whereas multi-class classification involves categorizing instances into more than two classes, so it is a simpler problem than multi-class classification. As a result of the simpler binary classification problem, the models are easier to train and optimize, resulting in better accuracy, precision, recall, and F1 scores. The higher complexity and increased number of classes in multi-class classification can lead to lower performance metrics.

6. Challenges faced

In natural language processing, the classification of propaganda and non-propaganda is a difficult problem. The main problem is that some sentences can be ambiguous and may be interpreted as propaganda or not propaganda. Due to the language's complexity and the dataset's bias, this can occur. Propaganda does not have a standard definition, so different interpretations can occur. Additionally, propaganda can be conveyed through subtle means, such as metaphors or rhetorical questions, making it difficult to detect. Sarcasm, irony, and figurative language can also pose challenges for automated systems to understand. As a final point, a lack of large, diverse, and labeled datasets can also make it difficult to develop effective propaganda detection models.

7. Future Scope

There are many potential research areas in natural language processing (NLP). In addition to developing better language understanding models, multilingual NLP models, explainable NLP models, and specialized NLP models for different fields, NLP has several significant future scopes. In the future, language processing will have new possibilities as models become more advanced and capable, changing the way we communicate and interact.

8. Conclusion

The classification of propaganda and not-propaganda is a challenging problem in natural language processing. As far as propaganda and not-propaganda texts are concerned, deep learning models like RNN, LSTM, and BERT have proven useful. In binary classification, Bert has shown the highest accuracy, followed by LSTM and RNN. Bert has also shown higher accuracy in multi-class classification than LSTM, though it is relatively lower than in binary classification. As a result of the challenges in detecting subtle and ambiguous language, sarcasm, irony, and a lack of a standard definition of propaganda, the classification of propaganda and not-propaganda texts remains a difficult task despite reaching high accuracy levels. The advancement of natural language processing techniques and the development of more extensive and diverse datasets, however, promise a bright future for propaganda classification and non-propaganda classification.

References:

- Sentimental analysis:https://onlinelibrary.wiley.com/doi/epdf/10.1111/coin.12415
- Simple LSTM :https://www.kaggle.com/code/kredy10/simple-lstm-for-text-classification
- BERT: https://www.kaggle.com/code/nayansakhiya/text-classification-using-bert
- Recurrent Neural network: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/