# Propaganda Detection

## 1. Introduction:

Propaganda is a technique employed to manipulate or influence people's opinions and beliefs. Detecting propaganda is a significant challenge in natural language processing (NLP), and several machine learning algorithms are developed to identify and classify propaganda in text data. This essay discusses the propaganda dataset, data preprocessing, and text classification models, namely Bidirectional Encoder Representations from Transformers (BERT), Bidirectional LSTM and Naïve Bayes. Propaganda dataset is challenging for classification models due to non-literal language, subtle techniques, intentional evasion, biased/incomplete data. These factors make it difficult for models to accurately interpret and classify propaganda texts.

Using the aforementioned techniques, our goal is to determine whether a given sentence contains propaganda and not propaganda and to classify the type of propaganda technique employed. This report involves a series of steps to analyze a given dataset. First, the data undergoes preprocessing to ensure it is ready for analysis. Then we use three binary classification models - BERT, LSTM, and Naive Bayes - to determine whether a sentence contains propaganda or not. Two additional multi-classification approaches are employed using BERT and LSTM to categorize the type of propaganda into nine categories. Hyperparameter settings are established for all models to optimize their performance. The models are then evaluated based on various performance metrics, including accuracy, precision, recall, and F1-score. Their performance is compared to determine their effectiveness. Finally, we discuss future work possibilities for improving the models' accuracy and overall effectiveness.

## 2. Dataset:

The propaganda dataset is comprised of two files, namely training and testing, which contain labeled sentences. Each file has two columns - one for the sentence and the other for its corresponding label. The label column includes a set of nine categories: flag-waving, appeal to fear/prejudice, causal simplification, doubt, exaggeration/minimization, loaded language, name-calling/labelling, repetition, and not propaganda. The first eight labels are propaganda techniques, while the last label, not_propaganda, denotes that no propaganda has been detected in the text. The tokens <BOS> and <EOS> have been included to mark the start and end of the text span that has been annotated with the identified propaganda technique within the sentence.

## 3. Data Preprocessing:

Preprocessing the data in the propaganda dataset is crucial as it enables the conversion of raw data into a format that is appropriate for analysis and training machine learning models. Before training a model, the preprocessing steps include removing the additional tokens <BOS> and <EOS> and replacing them with a comma. The process of converting the labels from strings to binary values, specifically 0 or 1, is implemented. The function

"convert_tokens" takes a label as input and returns 1 if the label is "not_propaganda," indicating that the sentence is not propaganda. Otherwise, it returns 0.
Eliminating punctuations, numbers, single characters, multiple spaces, and stop words from the text. This is accomplished by utilizing the stop words library in NLTK.

## 4. Text Classification models:

## 4.1 Bidirectional Encoder Representations from Transformers (BERT):

We are using the pre-trained natural language processing model bert_base_cased developed by Google. The bert_base_cased model is preferred for propaganda dataset classification due to its accuracy and capability to understand intricate word relationships. It surpasses traditional machine learning methods in various NLP tasks, making it a popular choice for propaganda dataset classification. The aim of this report is to determine whether a sentence contains propaganda or not, and to identify the specific propaganda techniques employed using multiclass model.

The dataset is being prepared for model training and evaluation by tokenizing it with the BERT tokenizer. The pre-trained BERT model is loaded using the AutoModelForSequenceClassification class, and a classification layer is added on top of it. The model is then fine-tuned on a dataset using transfer learning technique. The input text is processed through multiple layers, and the logits are obtained. These logits are passed through a softmax function to obtain predicted probabilities for each class, which represent the model's confidence level in each class. These predicted probabilities are then used to make the final classification decision.

### 4.1.1 Hyperparameter settings:

Selecting the right hyperparameters plays a crucial role in fine-tuning a pre-trained BERT model for a specific task as it can greatly impact the model's performance. Proper hyperparameter tuning is essential for achieving high accuracy and generalization on a given task. Below are the hyperparameters for the BERT model

a. **max_length** determines the maximum length of a tokenized sequence to ensure that all input sequence is of the same length for efficient batch processing and it is set to 140 to match the length of the input data
b. **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.
**c. Logging_steps** is the number of steps between two logs, and it's set to 100
d. **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 12 which is considered reasonable.
e. **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 3, to avoid any memory-related problems during training on a GPU.

f. **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-5 suitable for pre-trained model
g. **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 evaluation of the model is carried out after completion of each epoch, with a constraint of 100 evaluations at maximum.
h. **early_stopping_patience** specifies the number of epochs to wait 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.

## 4.2. Naïve Bayes Model for Binary Classification:

The Naive Bayes binary classification is a type of machine learning algorithm utilized for classifying text data into two categories based on their attributes or features. In this scenario, the algorithm is employed on a dataset that consists of both propaganda and non-propaganda text data. Initially, the data is preprocessed through tokenization and vectorization of the text data.

The TfidfVectorizer is used to vectorize the text data and the parameter ngram_range=(1,2) generates a vector for both unigrams and bigrams. The apply method is applied to the "tagged_in_context" column of both the training and test datasets to convert the text data to string format using the str function. The data is separated into training and testing sets and fed into the model. Lastly, the shape of the training and testing data is displayed. As the "n_grams" range was increased in this classifier, the performance improved but there was a risk of overfitting or excessive generalization. The resulting numerical features were then fed into the built-in "MultinomialNB" Naive Bayes model for classification.

## 4.3 Bidirectional LSTM model:

Bidirectional LSTM models are a type of recurrent neural networks that are capable of processing sequences of data in both forward and backward directions. This approach utilizes information from both directions simultaneously, which can improve sequence modeling and address long-term dependencies in sequential data. Additionally, it can enhance the robustness of the model against noisy data and increase flexibility in handling different types of input data.

In the context of text classification, it is advantageous to use a pre-trained word embedding provided by Genism, which is known as "glove-wiki-gigaword-100". This embedding produces 100-dimensional GloVe embeddings that are trained on both Gigaword and Wikipedia. To utilize this embedding, we create a weighted matrix for the word embedding, where the word embedding from the glove model is assigned to our vocabulary list if it is present in the glove embedding. If not, a value of 0 is assigned to it.

Finally, we create an LSTM model using the Sequential() function.

The architecture of the model incorporates various layers within it

a. **The first layer** of the model comprises a Bidirectional layer with an LSTM layer embedded within it. The Bidirectional layer duplicates LSTM layers, which process input sequences in reverse order, allowing the model to capture context from both directions. This layer consists of 100 units and produces a sequence of outputs for each input element with the return_sequences parameter set to True
b. **The second layer** is a Dropout layer that randomly drops 20% of the input units to mitigate the risk of overfitting.
c. **The third layer** of the model involves another Bidirectional layer that contains an LSTM layer. This layer encompasses 200 units and returns a sequence of outputs for each input element.
d. **The fourth layer** is comprised of a Dropout layer, which randomly drops 20% of the inputs to prevent overfitting in the model**.**
e. **The fifth layer** is comprised of a Bidirectional layer with an LSTM layer inside, with 200 units and is designed to generate a single output for the complete input sequence, wherein return_sequences is set to False.
f. **Finally** the model is a Dense layer with a sigmoid activation function that produces a single probability value between 0 and 1 for binary class classification, where the class_num is set to 2. For multi-class classification, a softmax function is used instead of the sigmoid function to produce a probability distribution over all the classes, with each class having a probability value between 0 and 1 that adds up to 1. The class_num for multi-class classification is set to 8.

### 4.3.1 Hyperparameter setting

To develop accurate machine learning models, it is crucial to specify hyperparameters before training. Hyperparameters refer to the parameters that remain constant throughout the training process. Configuring these parameters is essential to optimize the model's performance.

a. **EMBEDDING_DIM** One of the hyperparameters that needs to be set for building effective machine learning models is EMBEDDING_DIM, which is set to a value of 100 in this case. This parameter determines the dimensionality of the word embedding, i.e., the representation of each word in 100 dimensions. A higher value can capture more semantic information but may lead to excessive memory and computational resource usage. Conversely, lower values may capture less useful semantic information, resulting in a poor word embedding
b. **Units in LSTM** – The layer's complexity is determined by the number of LSTM units. Increasing the number of units can help models learn intricate patterns, but it can also lead to overfitting and make the problem more complex. The Bidirectional LSTM's first layer has 100 units, while the following two layers have 200 units.
c. **Dropout rate**- The dropout rate is set to 0.2, which falls within the typical range of 0.2-0.5. This rate specifies the fraction of nodes to be randomly dropped during training. While a higher dropout rate can provide stronger regularization, it may also increase the risk of underfitting the model.

d. **Loss function** – The loss function used for binary classification is binary_crossentropy, while for multiclass classification it is categorical_crossentropy. These loss functions are appropriate for computing the loss in both binary and multiclass classification scenarios.
e. **Optimizer** – The Adam optimization algorithm, which adjusts the learning rate during training and provides fast and satisfactory results, is being utilized.
f. **Learning rate-** The default learning rate of 0.001 is being utilized for the Adam optimizer. An optimal learning rate is crucial for updating the parameters effectively and reaching a local minima.
g. **Batch size** – Batch size denotes the count of training samples that are processed per epoch. The use of a larger batch size can enhance computational efficiency, but it may also lead to reduced noise in the gradient estimation, which could cause the model to converge to a flatter minimum. On the other hand, smaller batch sizes may introduce more noise but can allow the model to converge to sharper minima. While selecting a batch size, one should take into account both the computational resources available and the characteristics of the dataset.

The model has been compiled utilizing the binary_crossentropy loss function for binary classification and categorical_crossentropy for multi-class classification, in conjunction with the Adam optimizer. The evaluation of the model's performance will be based on the accuracy metric. LSTM networks are more complex than other types of recurrent neural networks, making them more difficult to understand and implement. Moreover, they are computationally expensive and require a significant amount of memory.

# 5 Results and evaluation:

### 5.1 Results of binary classification

| Classifier | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Naïve Bayes | 0.66 | 0.68 | 0.67 | 0.65 |
| Bert | 0.70 | 0.66 | 0.85 | 0.74 |
| LSTM | 0.71 | 0.73 | 0.71 | 0.71 |

Based on the results, the three classifiers (Naïve Bayes, Bert, and LSTM) were evaluated in terms of their accuracy, precision, recall, and F1-score.

Starting with the accuracy metric, it measures the overall performance of the model in terms of the correct predictions made by the classifier. The Naïve Bayes classifier scored the lowest accuracy of 0.66, while Bert and LSTM performed better with 0.70 and 0.71, respectively. The precision metric, it measures the proportion of true positive predictions among all the positive predictions made by the classifier. The LSTM classifier achieved the highest precision score of 0.73, followed by Naïve Bayes with 0.68, and Bert with 0.66. The recall metric measures the proportion of true positive predictions among all the actual positive instances. The Bert classifier

achieved the highest recall score of 0.85, followed by LSTM with 0.71, and Naïve Bayes with 0.67.

Finally, the F1-score metric is the harmonic mean of precision and recall, and it is a measure of the overall balance between the two metrics. The LSTM classifier achieved the highest F1-score of 0.71, followed closely by Bert with 0.74, and Naïve Bayes with the lowest score of 0.65.

Overall, the results show that the LSTM and Bert classifiers performed better than Naïve Bayes in terms of accuracy, precision, recall, and F1-score.

**5.2 Results of Multiclass classification:**

| Classifier | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Bert | 0.54 | 0.55 | 0.54 | 0.51 |
| LSTM | 0.31 | 0.52 | 0.30 | 0.36 |

The results indicate that both the Bert and LSTM classifiers on various evaluation metrics. Bert achieved an accuracy score of 0.54, while LSTM with a score of 0.31. When examining the precision metric, Bert had a score of 0.55, while LSTM achieved a slightly better score of 0.52. In terms of recall, with Bert and LSTM achieving 0.54 and 0.37, respectively. Finally, the F1-score, which measures the balance between precision and recall, was 0.51 for Bert and 0.36 for LSTM, indicating that Bert had a slightly better balance between the two metrics. Overall, the results suggest that Bert perform better than LSTM across multiple evaluation metrics, including accuracy, precision, recall, and F1-score.

## 5.3 Evaluation:

Experimental testing has shown that Bert is more efficient than LSTM in multiclass classification tasks due to its transformer architecture that allows it to process entire input sequences, capture relationships between words and sentences, and make more precise predictions. In contrast, LSTM's sequential processing can struggle with long-term dependencies and context from both directions.

The increased number of possible outcomes in multi-class classification compared to binary classification can make it more challenging for models to learn the correct decision boundary.

Overall, Bert has performed well in both binary and multiclass classification tasks. While it has shown similar performance to LSTM in binary classification tasks, it outperforms LSTM in multiclass classification tasks due to its ability to capture contextual relationships between words and sentences.

## 6 Further work

Further work is exploring other pre-trained models, such as GPT-3, and comparing their performance with BERT. Additionally, different vectorization techniques can be employed to improve the model's accuracy and generalization, such as Word2Vec, GloVe, and Doc2Vec. Furthermore, data augmentation techniques can be used to increase the dataset's size, improving the model's performance. Finally, exploring the interpretability of the models and determining which features contribute to the classification decision can help gain insight into the propaganda techniques employed.

## 7 Conclusion

In conclusion, this report discusses three different models used for text classification: Bidirectional Encoder Representations from Transformers (BERT), Naïve Bayes Model, and Bidirectional LSTM. BERT is a pre-trained natural language processing model that has shown high accuracy and performance in both tasks. The Naïve Bayes Model is a simple yet effective algorithm that is utilized for binary classification. Lastly, Bidirectional LSTM is a type of recurrent neural network that is capable of processing sequences of data in both forward and backward directions and it shown better accuracy in both tasks. Each model has its own advantages and limitations depends on the specific task and dataset. Additionally, the essay also emphasizes the importance of proper hyperparameter tuning, which can greatly impact a model's performance. Overall, these models provide effective solutions for text classification tasks, and their applications continue to grow as natural language processing technology advances.

## References:


1) Detecting Propaganda Techniques using BERT Pretrained Model https://aclanthology.org/2020.semeval-1.229.pdf
2) Sentiment analysis of tweets using a unified convolution neural network https://onlinelibrary.wiley.com/doi/epdf/10.1111/coin.12415
3) Fake New accuracy using Naïve Bayes Classifier https://www.ijrte.org/wp-content/uploads/papers/v8i1C2/A11660581C219.pdf
4) Hyperparameter settings https://medium.com/grabngoinfo/hyperparameter-tuning-for-bertopic-model-in-python-104445778347
5) Bert-base-cased https://huggingface.co/bert-base-cased