Propaganda Detection Model

Advanced Natural Language Processing 968G5

Felipe Martin CandNo: 260774

April 28, 2023

UNIVERSITY
OF SUSSEX

Contents

1 INTRODUCTION

The spread of false information through digital media such as social media is increasing. This can be seen, for example, in politically motivated messages that seek to persuade a certain group of people to act in a certain way according to the political purpose. Propaganda is a key element in the dissemination of this type of information, which is why there is a need for propaganda detection systems to filter out such messages.

In this scenario, natural language processing models have become increasingly popular to solve the problem of detecting textual propaganda, as one of their characteristics is that they are designed to understand and process human language in a similar way to a person [1]. This includes the ability to understand syntax, semantics and context.

The aim of this report is to show different ways of solving the problem of propaganda detection using NLP models. A description of each method, the hyperparameters used and how they were chosen, the results and conclusions are included.

2 Dataset

The dataset provided is a propaganda dataset. Two files with the same format are included: one for training and one for testing. Each file is in tabular (tsv) format with 2 columns.

The first column contains one label from a set of 9 possibilities that are propaganda techniques. The first 8 labels are propaganda techniques and are a subset of those identified in the Corpus of Propaganda Techniques (Da San Martino et al., 2020). The labels are the following:

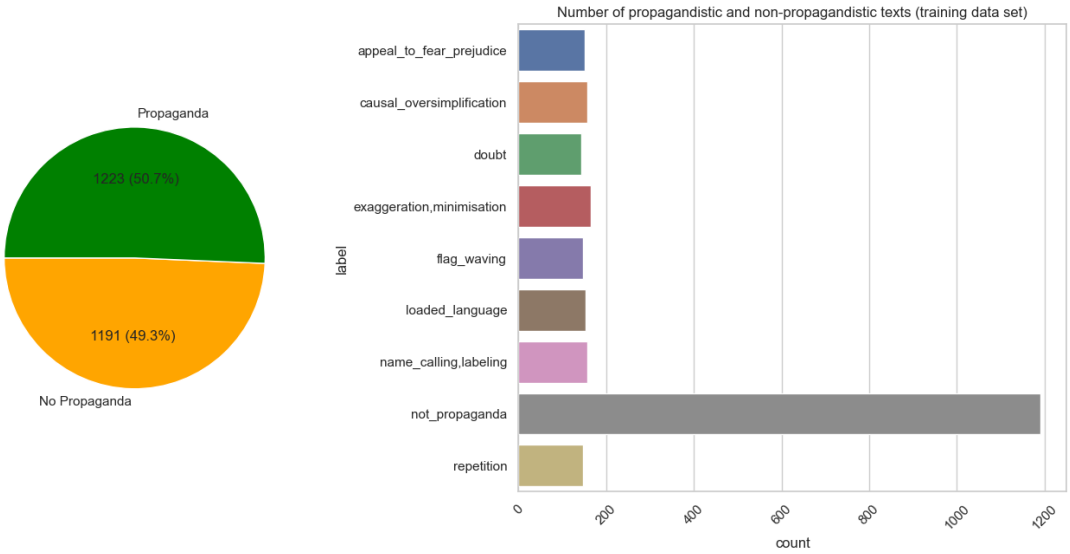
- Flag waving
- Appeal to fear and prejudice
- Causal simplification
- Doubt
- Exaggeration and minimisation
- Loaded language
- Name calling and labeling
- Repetition
- Not propaganda

The second column contains a sentence or text fragment where the propaganda technique has been identified (or not propaganda in the case of not propaganda).

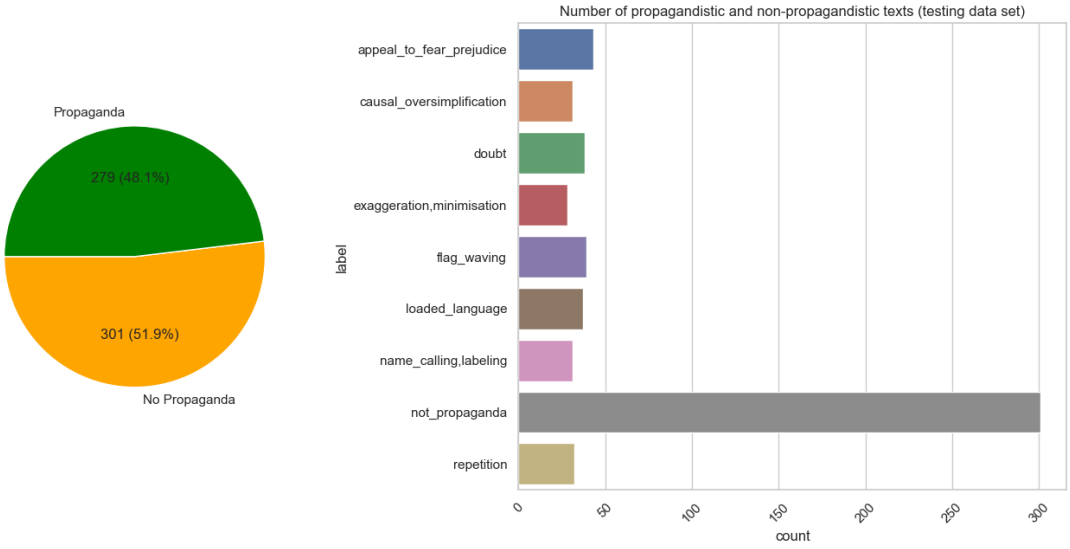
These two tables were loaded into a pandas dataframe, and a new column was added to identify whether the text is propaganda or not, as shown below.

	label	tagged_in_context	is_propaganda
0	not_propaganda	No, <BOS> he <EOS> will not be confirmed.	No Propaganda
1	not_propaganda	This declassification effort <BOS> won't make ...	No Propaganda
2	flag_waving	The Obama administration misled the <BOS> Amer...	Propaganda
3	not_propaganda	"It looks like we're capturing the demise of t...	No Propaganda
4	not_propaganda	<BOS> Location: Westerville, Ohio <EOS>	No Propaganda
...	...	...	...
2409	not_propaganda	<BOS> We support and appreciate <EOS> your bus...	No Propaganda
2410	not_propaganda	International Atomic Energy Agency (IAEA) Dire...	No Propaganda
2411	not_propaganda	What has been done: there has been work on for...	No Propaganda
2412	not_propaganda	This is <BOS> the law of gradualness not the g...	No Propaganda
2413	name_calling,labeling	In it, Jews are described as: "arrogant," "jea...	Propaganda

The training dataset contains 2413 texts, which is 68% of the total data. Where 1191 are classified as Non-propaganda and 1223 as propaganda. These 1223 texts that are classified as propaganda are distributed as follows:



The testing dataset contains 580 texts, which is 32% of the total data. Where 279 are classified as Non-propaganda and 301 are classified as propaganda. These 279 texts classified as propaganda are distributed as follows:



3 Proposed Methods

The first task is the development and evaluation of a model for classifying whether a text contains propaganda or not. This problem can be approached as a binary classification model and to solveit, we create two models, one in which we use the word2vec method together with logistic regression and the other in which we use BERT together with the Adamw optimiser.

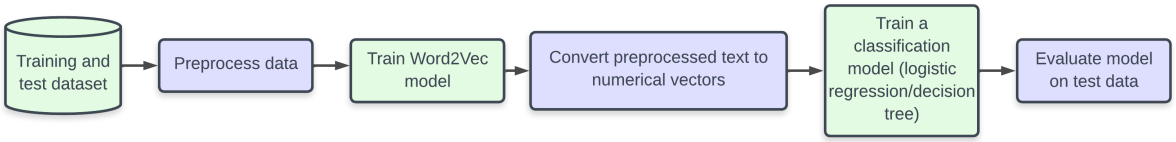
The second task is the development and evaluation of a model to classify the method of propaganda used in the text, so it is a multi-class classification problem wich can be solve with the word2vec method together with decision trees and the BERT model together with the Adamw optimiser.

3.1 Word2Vec and Classification Models

Word2vec is a set of algorithms used to create word vectors, also known as word embeddings. These vectors represent words and are created from the context of surrounding words. These algorithms are based on shallow neural networks and their purpose is to produce the word vectors for use in natural language processing tasks [2][3].

To solve the propaganda detection problems we combine Word2Vec with a classification model as proposed in [4]. It is proposed to first use Word2Vec to transform the texts into numerical vectors and then use these vectors to extract the input features to be used by the classification model

The general diagram of the process is shown below.



3.1.1 Text Preprocessing

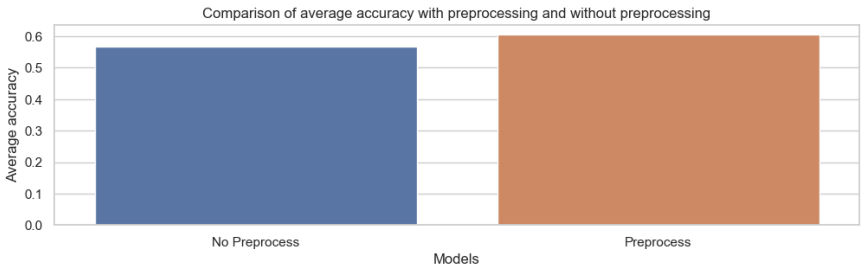
The preprocessing function obtained from the code in [5] is applied to the texts of each record in the training and test set, and the results are stored in two new columns in the DataFrames. This step is important to reduce noise in the text data and make it more suitable for modelling.

The preprocessing function removes the stop words OR commonly used word (such as "the", "a", "an", "in") [6] from the texts by importing the English stop words from the NLTK library and saving them in the stop word list which is cross-referenced with the texts of our training and test data to remove the stop words.

3.1.2 Comparison of the model with preprocessing and without preprocessing using K-fold Cross Validation

A comparison of the model without preprocessing and with preprocessing was performed. For this, we used the K-fold cross validation technique with 5 folds (see figure), which allows us to evaluate the model with different training and test sets to reduce the probability of overfitting the model to a particular dataset.

The result of the mean value of accuracy considering the 5 folds is 0.61 for the model with preprocessing and 0.57 in the model without preprocessing, as shown in the following figure.



This result indicates that pre-processing the texts by deleting stop words is an effective method to improve the performance of the model.

3.2 Word2Vec Model

A Word2Vec model was trained on the pre-processed texts of the training set. The Word2Vec model converts each word into a fixed-length numeric vector representing its meaning in the context of the surrounding words.

Next, a function is defined to convert the preprocessed texts into numeric vectors using the trained Word2Vec model. The function takes a list of words and returns a numeric vector representing the text as a whole.

Finally, the vectorisation function is applied to the pre-processed texts of each record in the training set and tested and the resulting numerical vectors, are stored in two new columns in the DataFrames file that will be used as input features to train the LR model.

3.3 Logistic Regression (Propaganda/non-propaganda binary classification model)

Logistic regression is a classification algorithm used to predict a binary outcome based on a set of independent variables [7].

Input features are extracted from columns of numeric vectors created by word2vec, while class labels are extracted from the label column of each DataFrame. The class labels are coded as 0 for propaganda and 1 for non-propaganda.

3.4 Decision Trees (Propaganda technique multi-class classification model)

Decision trees are a supervised learning approach used as a predictive model to make conclusions about a set of observations, with classification and regression trees being the two main types [8][9].

Input features are extracted from columns of numeric vectors created by word2vec, while class labels are extracted from the label column of each DataFrame. The class labels are coded as follows:

```
label_map = {'flag_waving': 0, 'appeal_to_fear_prejudice': 1, 'causal_simplification': 2,
             'doubt': 3, 'exaggeration,minimisation': 4, 'loaded_language': 5, 'name_calling,labeling': 6,
             'repetition': 7, 'causal_oversimplification': 9}
```

This turns the problem into a multi-class classification problem.

3.5 Hyper-parameter Tuning

To adjust the hyperparameters, we use the grid search method which involves selecting specific values for each hyperparameter and evaluating the model for each combination of values.

The hyperparameters that can be modified in this code to obtain a better result are:

3.5.1 Word2Vec Hyper-parameters

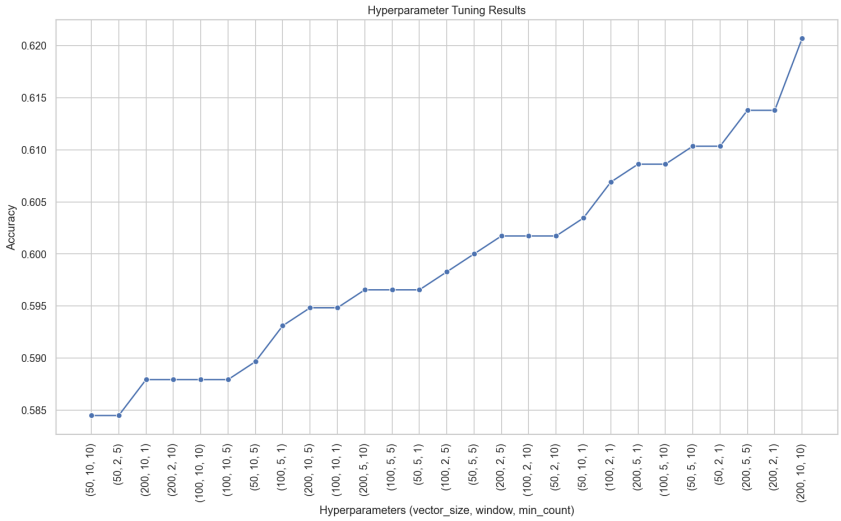
- min_count : specifies the threshold for ignoring all words that have a frequency of occurrence less than the specified value.
- vector_size : specifies the dimension of the word vectors generated by Word2Vec.
- window : This is the parameter that determines the number of words in each context.

In order to find the combination of these three parameters that gives the best model performance, we first created a list of 3 values for each parameter: (50, 100, 200) for vector_size, (2, 5, 10) for window, and (1, 5, 10) for min_count.

```
params_w2v = {
    'vector_size': [50, 100, 200],
    'window': [2, 5, 10],
    'min_count': [1, 5, 10] }
```

Then the product function from the itertools library was used to perform all the possible combinations between these variables and, finally, each of these possible combinations is evaluated through a for loop that stores the result of accuracy for each scenario.

The final result can be seen in the following graph.

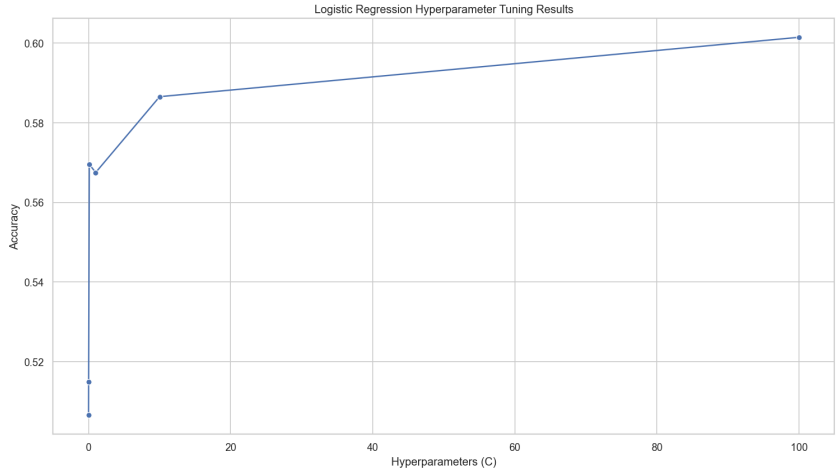


As can be seen, the best result in accuracy is obtained with the combination vector_size: 200, window: 10 and min_count: 10. So it will be this combination of parameters that we will use in our final model.

3.5.2 Logistic Regression Hyper-parameters

- C: The parameter C controls the strength of the regularisation in the model.

In order to find the best value of C, we create the following matrix of C values: [0.001, 0.01, 0.1, 1, 10, 100] and test the model with each value of this matrix. The results are shown in the graph below.



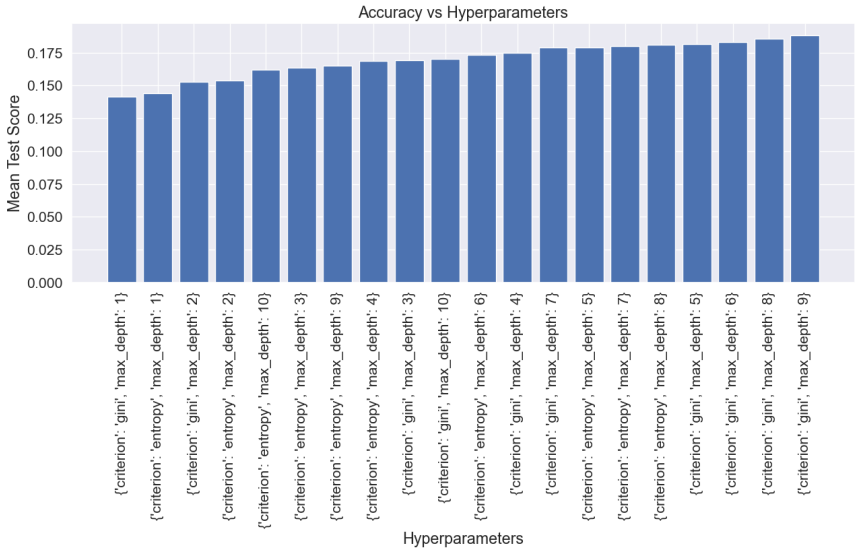
As the graph shows, the best Accuracy value (0.6) is obtained with a value of C equal to 100, therefore, this will be the value of the hyper parameter C that we will use in the final model.

3.5.3 Decision Trees Hyper-parameters

- Criterion: The available criteria are "gini" for Gini impurity and "entropy" for information gain.
- max_depth: It controls the complexity of the model and therefore helps to prevent overfitting of the model to the training data.

In order to find the combination of these two parameters that gives the best model performance, we first create a list of values between 1 and 11 for Max_depth and another list with the "gini" and "entropy" values for the criterion. Then we use the GridSearchCV function to perform all combinations between these two lists of hyperparameters. The results are shown in the graph below:

```
param_grid = {'max_depth': range(1, 11),
              'criterion': ['entropy', 'gini']}
```



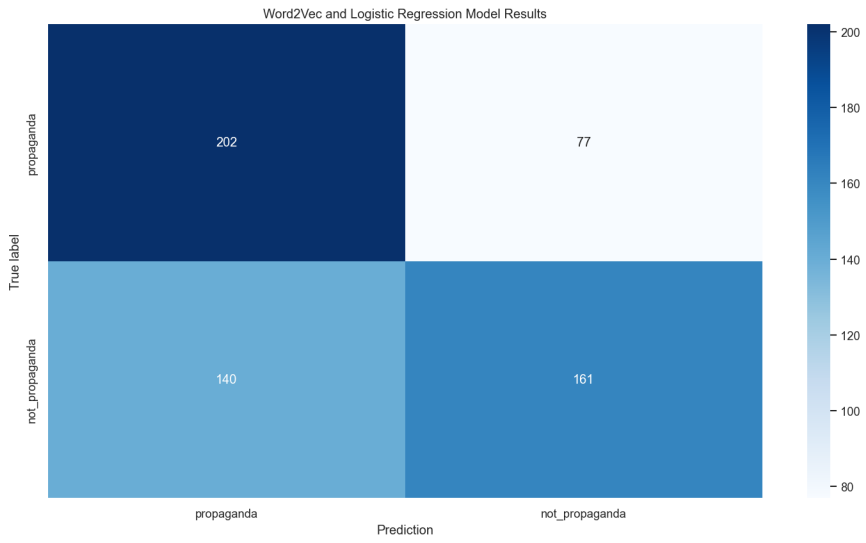
We find that the best average accuracy score result is with the combination criterion: gini and max_depth: 9. Therefore, we will use these hyper parameters in the final model.

3.6 Evaluation

3.6.1 Propaganda/non-propaganda Binary classification model (Word2Vec + Logistic Regression)

The classification report and the confusion matrix when evaluating the model gives the following values:

	precision	recall	f1-score	support
0	0.59	0.72	0.65	279
1	0.68	0.53	0.60	301
accuracy			0.63	580
macro avg	0.63	0.63	0.62	580
weighted avg	0.64	0.63	0.62	580



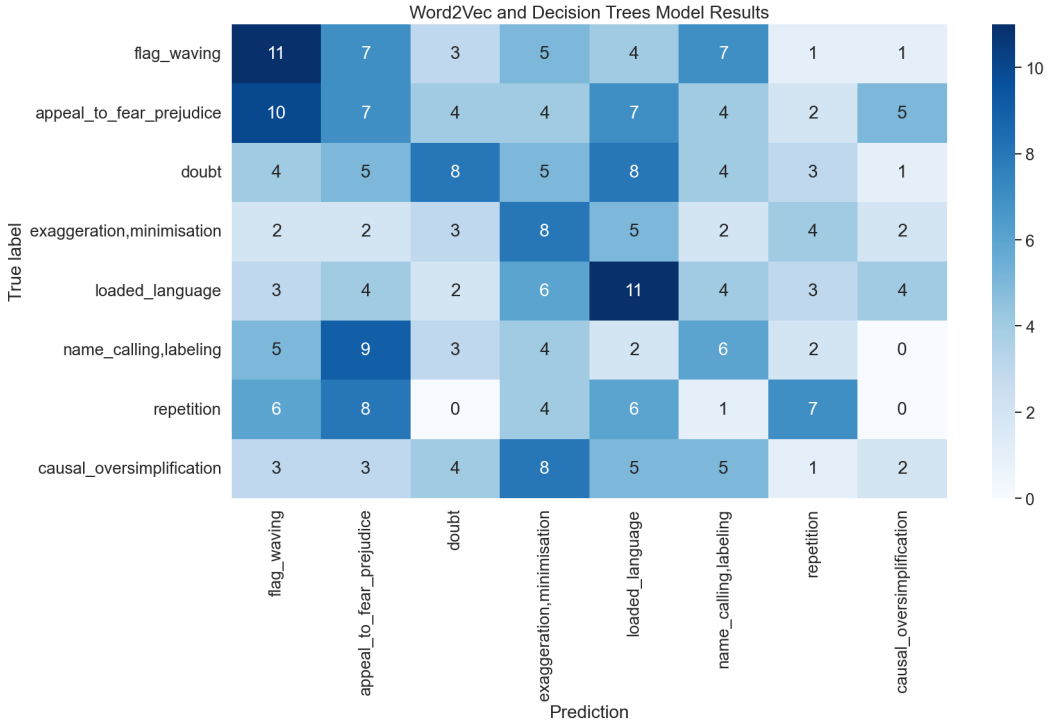
The model has an accuracy of 64%, meaning that 64% of the positive predictions are correct. It has a recall of 63%, meaning that the model correctly identifies 63% of all positive cases. The F1-score of the model is 62%, which is

the harmonic mean of accuracy and recall. The model's accuracy is 63%, indicating that the model correctly classified 63% of the cases. Overall, The model is moderately accurate and has a reasonable balance between precision and recall.

3.6.2 Propaganda technique multi-class classification model (Word2Vec + Decision Trees)

The classification report and the confusion matrix when evaluating the model gives the following values:

	precision	recall	f1-score	support
0	0.25	0.28	0.27	39
1	0.16	0.16	0.16	43
3	0.30	0.21	0.25	38
4	0.18	0.29	0.22	28
5	0.23	0.30	0.26	37
6	0.18	0.19	0.19	31
7	0.30	0.22	0.25	32
9	0.13	0.06	0.09	31
accuracy			0.22	279
macro avg	0.22	0.21	0.21	279
weighted avg	0.22	0.22	0.21	279



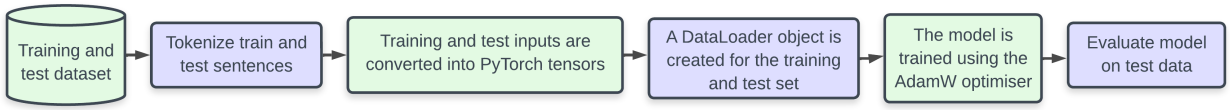
The decision tree model used to detect the different propaganda methods performs poorly in terms of precision, recall, f1-score and accuracy (0.2), This may be because improvements are still needed in the design or in the data used to train it.

4 Pretrained large language models BERT

BERT is a pre-trained language model that uses a Transformer architecture and a self-directed learning method to train a language model on large amounts of unlabeled text [10][11]. This model has previously been used in the propaganda/non-propaganda detection task using a variant of the pre-trained BERT model, finely tuned with a multi-label linear classification layer [12][13].

To solve the problem of propaganda detection, we use the BERT model together with the AdamW optimiser, based on the codes of [14][15][16][17]. First, we define the maximum length of the sequences and tokenise the sentences using the batch encode plus function of the BERT tokeniser. Then, training and test datasets are created,

and PyTorch dataloaders are used to load the data into the model. Finally, the model is trained using the AdamW optimiser and evaluated on the test dataset to calculate the accuracy.



4.1 Preprocessing data

4.1.1 Tokenize Sentences

The process of text tokenisation consists of breaking a text into smaller units, called tokens. The max_length variable is used to define the maximum length of the input tokens to the neural network This is done to ensure that all inputs have the same length, which is necessary to process the data in batches and speed up training.

It is important to note that a value of max_length that is too low can result in a significant loss of information, while a value that is too high can significantly increase the training time.

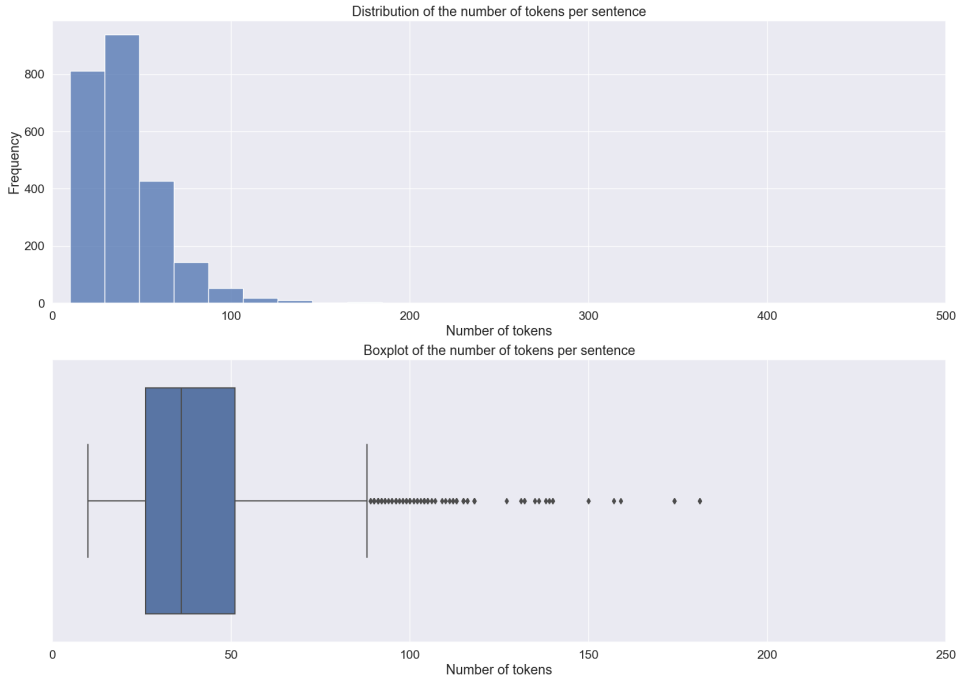
Below is a sentence from the training set transformed into tokens:

```
Britain doesn't need more hate even just for a <BOS> few days." <EOS>

['britain', 'doesn', "'", 't', 'need', 'more', 'hate', 'even', 'just', 'for',
'a', '<', 'bo', '##s', '>', 'few', 'days', '.', '"', '<', 'e', '##os', '>']

num_tokens:  23
```

To select an optimal value of max_lenght, we obtain the number of tokens in each text and plot the distribution of the number of tokens per sentence in a histogram and a boxplot considering all the texts. We observe that with a value of 100 tokens there is no major loss of information.



4.1.2 Word Embeddings

After adding these special tokens, the token sequence is fed into BERT's Transformers architecture which encodes the sequence information in word embedding, where each token is mapped to a high-dimensional vector.

The resulting matrix is shown below:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
0	101	2053	1010	1026	8945	2015	1028	2002	1026	1041	...	0	0	0	0	0	0	0	0	0	0
1	101	2023	11703	27102	9031	3947	1026	8945	2015	1028	...	0	0	0	0	0	0	0	0	0	0
2	101	1996	8112	3447	28616	3709	1996	1026	8945	2015	...	0	0	0	0	0	0	0	0	0	0
3	101	1523	2009	3504	2066	2057	1521	2128	11847	1996	...	0	0	0	0	0	0	0	0	0	0
4	101	1026	8945	2015	1028	3295	1024	2225	2121	3077	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2409	101	1026	8945	2015	1028	2057	2490	1998	9120	1026	...	0	0	0	0	0	0	0	0	0	0
2410	101	2248	9593	2943	4034	1006	24264	5243	1007	2472	...	1055	4517	3450	1012	102	0	0	0	0	0
2411	101	2054	2038	2042	2589	1024	2045	2038	2042	2147	...	0	0	0	0	0	0	0	0	0	0
2412	101	2023	2003	1026	8945	2015	1028	1996	2375	1997	...	0	0	0	0	0	0	0	0	0	0
2413	101	1999	2009	1010	5181	2024	2649	2004	1024	1523	...	0	0	0	0	0	0	0	0	0	0

4.1.3 PyTorch tensors

A tensor in PyTorch is similar to a multidimensional array in NumPy, but with the ability to run on CPU or GPU [18]. This GPU computing capability significantly accelerates the training of deep learning models.

The training and test inputs are transformed into PyTorch tensors, then, we use Torch's DataLoader function that allows to load and process large datasets in an efficient and scalable way, define the batch size and create a DataLoader object for the training and test set.

```
1 train_inputs_tensor
```

```
tensor([[  101,  2053,  1010,  ...,     0,     0,     0],
        [  101,  2023, 11703,  ...,     0,     0,     0],
        [  101,  1996,  8112,  ...,     0,     0,     0],
        ...,
        [  101,  2054,  2038,  ...,     0,     0,     0],
        [  101,  2023,  2003,  ...,     0,     0,     0],
        [  101,  1999,  2009,  ...,     0,     0,     0]])
```

These tensors will be the input data for our Adamw optimisation model.

4.2 AdamW optimisation

To get the best results with Bert, it is necessary to use a suitable optimisation method. Adamw is a variant of the Adam optimisation algorithm that introduces decay weight regularisation to improve model performance and prevent overfitting [19]. The steps we follow are listed below:

- First, the learning_rate and the number_epochs are defined.
- A list is defined to store the training loss for each step and a loop is started to train the model for a given number of epochs.
- The model is trained using the training data and the loss is calculated for each iteration.
- The model is evaluated using the test data and the accuracy of each iteration is calculated.

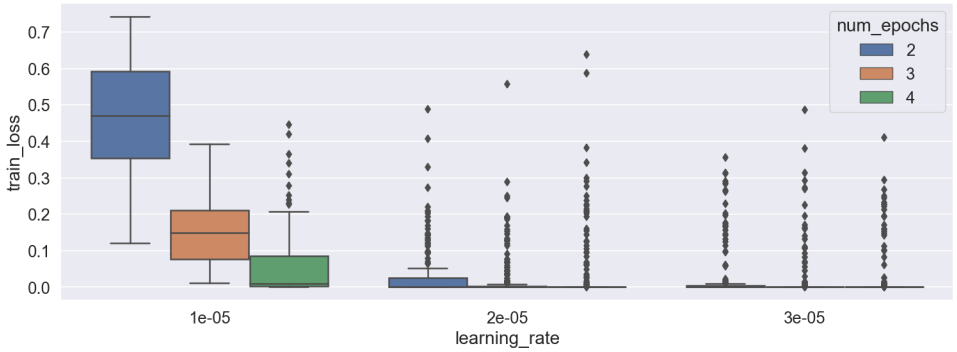
4.3 Hyper-parameter Tuning

The hyperparameters used in this code are:

- learning_rate: Controls the size of the changes that are made to the model parameters during training.
- num_epochs: The number of times the model will iterate through the entire training data set.

To find the optimal hyper-parameters, the developed code uses two nested for loops to test different values of learning_rates [1e-5, 2e-5, 3e-5] and num_epochs [2, 3, 4]. Then a loop is performed on the number of epochs and the model is trained on each epoch. The training loss is added to the epoch_train_losses list and plotted at the end of each combination.

The following boxplots show the distribution of train_loss over the different combinations of learning_rates and epochs.



As observed from the boxplots, the combination that generates the best result (lowest loss values) is learning_rate: 3e-5 and num_epochs: 4, therefore, these are the values used in our final model.

4.4 Evaluation

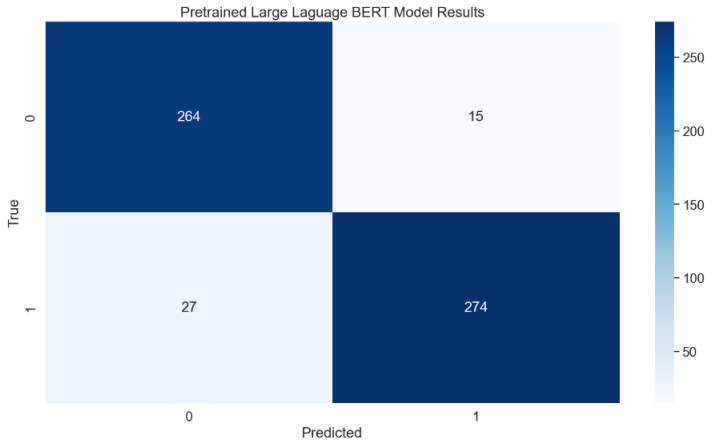
4.4.1 Propaganda/non-propaganda binary classification model (BERT model)

The classification report and the confusion matrix when evaluating the model gives the following values:

```
1 = Not propaganda
0 = Propaganda
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       279
           1       0.95      0.91      0.93       301

    accuracy                           0.93       580
   macro avg       0.93      0.93      0.93       580
weighted avg       0.93      0.93      0.93       580
```

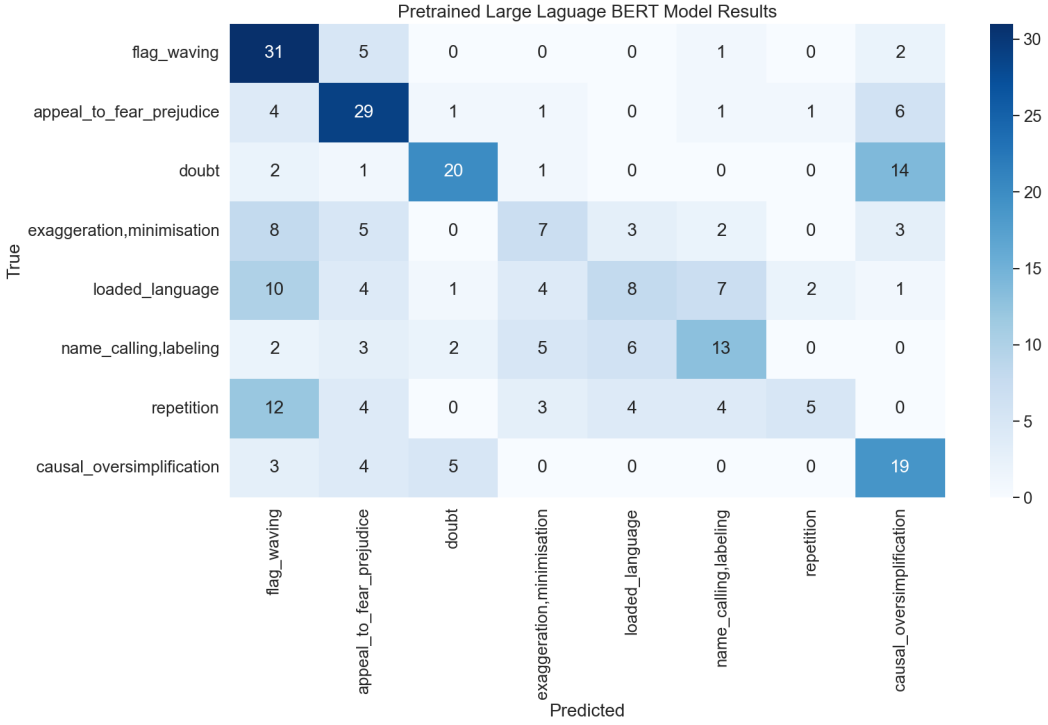


The precision, recall, f1-score and accuracy results, which are all equal to 0.93, indicate that the model is able to accurately identify both propaganda and non-propaganda texts. Therefore this model is highly accurate and reliable.

4.4.2 Propaganda technique multi-class classification model (BERT model)

The classification report and the confusion matrix when evaluating the model gives the following values:

	precision	recall	f1-score	support
0	0.43	0.79	0.56	39
1	0.53	0.67	0.59	43
3	0.69	0.53	0.60	38
4	0.33	0.25	0.29	28
5	0.38	0.22	0.28	37
6	0.46	0.42	0.44	31
7	0.62	0.16	0.25	32
9	0.42	0.61	0.50	31
accuracy			0.47	279
macro avg	0.48	0.46	0.44	279
weighted avg	0.49	0.47	0.45	279



5 Conclusion

After a thorough analysis of the results obtained with the Word2Vec + Logistic Regression, Word2Vec + Decision Trees and BERT models for the detection of propaganda in texts, it is observed that the models for classifying propaganda/non-propaganda have good results. The BERT model has an accuracy, precision, recall and f1-score of 0.93, therefore, it can be considered a reliable tool for identifying propaganda in texts. Whereas, the Word2Vec + Logistic Regression model is moderately accurate (63%).

In contrast, the models designed to detect different propaganda techniques did not perform well. The Word2Vec + Decision Trees model for detecting different propaganda methods showed poor precision, recall, f1 score, and accuracy. Similarly, the BERT model has a F1-score value of 0.45 that suggests an imbalance between the model's accuracy and sensitivity, resulting in an overall accuracy of only 0.47.

To improve models designed to detect different propaganda techniques, it is recommended to increase the size and quality of the training dataset. This will give the models access to a wider variety of propaganda examples of different styles and approaches, which will improve their performance in the future. In addition, more sophisticated text pre-processing techniques should be considered, such as removing irrelevant words and exploring different feature selection techniques to improve the quality of the input data and the accuracy of the model.

6 References

[1] Lutkevich, B. and Burns, E. (2023) What is natural language processing? an introduction to NLP, Enterprise AI. TechTarget. Available at: https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP .

[2] Understanding word vectors: A tutorial for "Reading and writing electronic text," A class I teach at ITP. (python 2.7) Available at: https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469 .

[3] Johnson, D. (2023) Word embedding and word2vec model with example, Guru99. Available at: https://www.guru99.com/word-embedding-word2vec.html .

[4] Propaganda detection using sentiment aware ensemble deep ... - IEEE xplore. Available at: https://ieeexplore.ieee.org/document/9596654/ .

[5] Eea (2018) Eea.corpus/stopwords.py at 6638A3D63D06C209710D22A84F2842CD0FB94F4D · EEA/Eea.corpus, GitHub. Available at: https://github.com/eea/eea.corpus/tree/6638a3d63d06c209710d22a84f2842cd0fb94f4d/src/eea.corpus/eea/corpus/processing

[6] Removing stop words with NLTK in python (2023) GeeksforGeeks. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/ .

[7] Rawat, A. (2019) Binary logistic regression, Medium. Towards Data Science. Available at: https://towardsdatascience.com/implementing-binary-logistic-regression-in-r-7d802a9d98fe .

[8] Yadav, P. (2019) Decision tree in machine learning, Medium. Towards Data Science. Available at: https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96 .

[9] Kingsford, C. and Salzberg, S.L. (2008) What are decision trees?, Nature biotechnology. U.S. National Library of Medicine. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2701298/ .

[10] Yu, W. et al. (2022) Dict-Bert: Enhancing language model pre-training with dictionary, arXiv.org. Available at: https://arxiv.org/abs/2110.06490 .

[11] Joshi, P. (2022) Transfer learning for NLP: Fine-tuning bert for text classification, Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/

[12] Yoosuf, S. and Yang, Y. (no date) Fine-grained propaganda detection with fine-tuned Bert, ACL Anthology. Available at: https://aclanthology.org/D19-5011/ .

[13] QCRI/PropagandaTechniquesAnalysis-en-BERT · Hugging Face. Available at: https://huggingface.co/QCRI/PropagandaTechniquesAnalysis-en-BERT .

[14] fine-grained propaganda detection with fine-tuned Bert - Researchgate. Available at: https://www.researchgate.net/publication/336999573_Fine-Grained_Propaganda$_D$etection_with_$Fine{-}Tuned_BERT$.

[15] Fine-tuning a pretrained model - transformers 4.7.0 documentation. Available at: https://huggingface.co/transformers/v4.8.2/training.html .

[16] Briggs, J. (2021) How to train bert, Medium. Towards Data Science. Available at: https://towardsdatascience.com/how-to-train-bert-aaad00533168 .

[17] Bert fine-tuning tutorial with pytorch (2019) BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick. Available at: https://mccormickml.com/2019/07/22/BERT-fine-tuning/ .

[18] Geekgirldecodes (2022) Pytorch quick reference-tensors, Medium. HowsOfCoding. Available at: https://medium.com/howsofcoding/pytorch-quick-reference-tensors-2e9569ce8699 .

[**19**] Brownlee, J. (2021) Gentle introduction to the adam optimization algorithm for deep learning, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ .