Propaganda Detection using Word2Vec Classifiers and BERT Transformer Models

by **Pranov Sarath**

April 2023

1. Introduction

Propaganda is communication that is primarily used to influence or persuade an audience to further an agenda, which may not be objective and may be selectively presenting facts to encourage a particular synthesis or perception, or using loaded language to produce an emotional rather than a rational response to the information that is being presented (Wikipedia, n.d.) (Smith, 2016). Propaganda may be used as a means to spread false narrative and foster division in society by state and non-state actors, media, large corporations or politicians. Hence, it becomes imperative to detect cases of propaganda in any medium with a certain degree of accuracy so that efforts to influence and persuade people do not fructify. This report will discuss 2 different methods of identifying propaganda within a span of text. Furthermore, it will also discuss how to apply these methods to classify a span of text that is known to contain propaganda into one of the different types of propaganda.

2. The Dataset:

Propaganda detection is a form of text classification and is an advanced natural language processing task. The text classification models will be trained on the training and validation dataset provided specifically for this experiment, namely the tab delimited text files – 'propaganda_train.csv' and 'propaganda_val.tsv'. Each file consists of 2 columns – 'label' and 'tagged_in_context'. The 'tagged_in_context' field contains a span-identified string of varying length which may contain an instance of propaganda and the 'label' field contains one of 9 values – indicating the type of propaganda in the string or indicating it is not an instance of propaganda. The 9 different possibilities are:

1. flag waving
2. appeal to fear prejudice
3. causal simplification
4. doubt
5. exaggeration,minimisation
6. loaded language
7. name calling,labeling
8. repetition
9. not propaganda

The first 8 of these possibilities are a subset of the 14 different propaganda types (Giovanni Da San Martino, 2020). The distribution of data in the training and validation datasets are shown below.

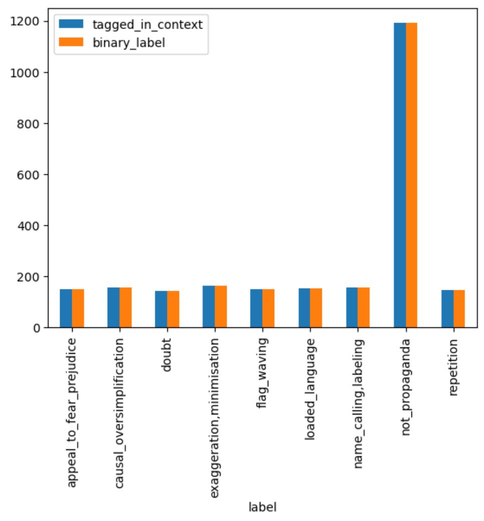

Figure 1) Training Dataset

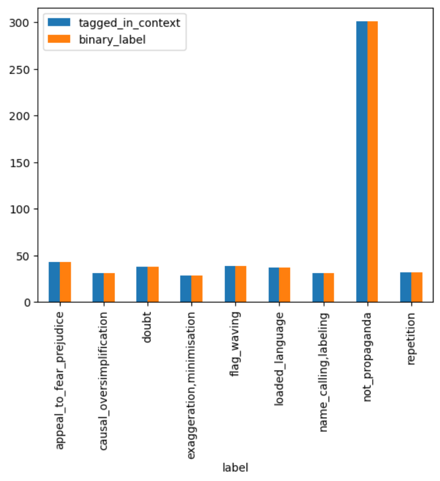

Figure 2) Validation Dataset

The training dataset contains 2414 records in total distributed across the 9 different possible labels. Though at first glance, the datasets look unbalanced, they are distributed evenly in terms of the task we are about to perform which will be explained in the next section.

The validation dataset 580 records in total distributed across the 9 different possible levels.

3. Propaganda Detection - Tasks:

To detect the type of propaganda in the text and assign a label to indicating the type of propaganda that is contained within the text, is by its very nature a text classification task. There are 2 different tasks to perform as part of this propaganda detection assignment.

Task 1: Build and evaluate at least 2 approaches to classify whether a sentence contains propaganda or not.

Task 2: Given a snippet or span of text which is known to contain propaganda, build and evaluate at least 2 different approaches to classifying the propaganda technique which has been used.

Task 1 can be reduced to a binary text classification task. As the labels it has to predict will be ‘propaganda’ and ‘not_propaganda’. To perform this task, a new label will be created in the dataset ‘binary_label’ which will have 1 of these 2 values. All the strings known to contain propaganda will have the value – ‘propaganda’. This also makes the dataset very balanced for the binary classification task.

Task 2 can be considered to be a multiclass text classification task. As the model which will be built needs to predict one of 8 labels. This is because this model will only use text known to contain propaganda and all the strings not containing instance of propaganda will have to be filtered out. In doing so, the dataset becomes balanced and can be used for training the classification models.

In order to perform these tasks, we will built classifiers using 2 very different approaches. They are:

1) A pretrained GloVe (Global Vectors for Word Representation) word embedding model will be used to generate a vector representation of text, which will be then passed into a neural network classifier.

2) A pretrained large language model BERT (Bidirectional Encoder Representations from Transformers) will be used to generate a contextualised embedding, which will be used downstream to perform text classification tasks by a neural network.

These are vastly different methods, and we will discuss how Task 1 and Task 2 will be implemented by these 2 methods in detail in the next section. However, we will have a brief overview of the GloVe and BERT models before this.

3.1. GloVe (Global Vectors for Word Representation):

GloVe is a model for distributed word representation. The model produces vector representations for words through an unsupervised learning algorithm. The words are mapped into a meaningful space where the distance between words is related to semantic similiarity (Abad, et al., 2016). It aims to capture the meaning of words by analysing the co-occurrence statistics of the word in a corpus of text.

GloVe constructs a weighted co-occurrence matrix of words in a corpus, where each element of the matrix represents the number of times a word co-occurs with another word in a certain context window. The matrix is then factorised to obtain word vectors that encode the distributional semantics of words based on their co-occurrence patterns. Unlike word2vec models that uses local context windows, GLoVe models explicitly take into account the global co-occurrence statistics, which can lead to more meaningful representations of words and better performance in downstream tasks (Jeffrey Pennington, 2014).

For obtaining the word embeddings for the text in datasets, we will be using the ‘glove-wiki-giga-300’ pre-trained model, provided by the gensim package which is trained on a large corpus of text from Wikipedia. This will take the text as input and return a vector representation where each word is represented as a dense vector of 300 dimensions.

3.2. BERT (Bidirectional Encoder Representations from Transformers)

BERT is a pre-trained transformer-based neural network architecture introduced by Google researchers in 2018. It is designed to address the limitations of earlier language models by capturing the bi-directional context of words in a text. It is trained on massive amounts of text data. During the training, BERT learns to represent a given input sequence by predicting masked words within that sequence and predicting the next sentence in a pair of sentences (Jacob Devlin, 2018). Thus, the relationship between the words and their context within a text is captured.

The BERT based classifiers and the GloVe vector representation-based classifiers we have built are very different from each other. While the GloVe model only captures the distributional semantics of a word based on co-occurrence and uses this to generate word embeddings, BERT can generate contextualised word embeddings. These embeddings can capture the meaning of words in different contexts.

For performing the propaganda detection tasks, we will use the pre-trained BERT model – ‘bert-base-uncased’ model, with 12 transformer blocks, 110 million parameters and an embedding dimension of 768. The model was trained on uncased text, which means all of the text will be converted into lowercase. This model will ignore case differences in the words. The model is provided for use by the Hugging Face Transformers library.

4. Task 1:

4.1. Binary Classification of text using GloVe vector representation based Neural Network:

We have obtained the dataset for binary classification, containing two labels – ‘propaganda’ and ‘not propaganda’ as described earlier in this report. Before we being training, we split the training dataset into separate train (80%) and test(20%) sets using the test_train_split function provided by the sklearn module. The aim is to use the GloVe sentence vectors as inputs to a simple neural network to perform the binary classification.

The first step is to generate the sentence vector representation. This is achieved by first tokenising the sentence by performing lemmatisation, followed by tokenisation. During the tokenisation step, all stop words and punctuation symbols are removed. After the pre-processing steps are complete, we generate the sentence vector for each text sentence using the ‘glove-wiki-giga-300’ pre-trained embedding model. The word vector for each word in the sentence is retrieved and appended to the sentence vector. The output is a dense vector of 300 dimensions.

Before we define the architecture of the neural network, we will have to build a Dataset class that will transform the dataframe object and the vectors and labels within them into a tensor object. The PyTorch library trains the models by converting the inputs into the form of tensors which can then be loaded onto the CPU, or GPU for model training, validation, testing and prediction. If the platform on which these tasks are carried out has a GPU, we can make use of the CUDA cores (only NVIDIA GPU’s) to make the process faster. By creating a custom Dataset class by inheriting from the – ‘torch.utils.data.Dataset’ class we can obtain a dataset that can be indexed like a Python list or array enabling easy loading and preprocessing of data for training or evaluation of data. This step will not be discussed in further sections, as it is the same for all models.

4.1.1. NN Architecture:

The neural network architecture for performing binary classification as part of Task 1 utilises a simple NN consisting of an input layer, hidden layer and an output layer. The neural network has 2 fully connected (dense) layers and a ReLU (Rectified Linear Unit) activation function.

Using 2 fully connected layers allows the classifier to learn complex representations of the input data. The input layer (fc1) has 300 neurons and an output size of 150, which matches the dense vector input size. Matching sizes are required between layers to avoid errors. The hidden layer (fc2) has an input size of 150 and an output size of 2 and is responsible for learning complex patterns in the training data.

We apply the ReLU activation function to the output produced by fc1. This is done to introduce non-linearity into the network. Neural networks will be limited to modelling only linear relationships between the inputs and outputs, if non-linearity would not be introduced into the network. It is this non-linearity that allows a neural network to learn almost and complex pattern in data theoretically. In the real world this would be limited by the size of the network, data, compute power, time etc. ReLU function is defined as,

$$f(x) = \max(0,x)$$

Applying the ReLU activation function between the fc1 and fc2 layers allows for introducing non-linearity that will allow the model to capture the complex patterns and relationships in the data.

We also add a dropout layer soon after applying the ReLU activation function. This is to prevent the model from overfitting to the training data. The drop out layer will cause some of the neurons in the layer to randomly turn off during training. This prevents the model from relying too much on a single neuron or feature and the model is forced to learn useful features and generalise easily to new data.

It is to be noted that the size of the hidden layer is not efficiently determined. We will have to apply various techniques like pruning, L1 or L2 regularisation etc which has not been performed. After applying these techniques, we may get a better model that has fairly good complexity, does not overfit the training data and is still fast enough to train and predict data well.

The hyperparameters in this Neural Network:

1) Size of the hidden layer
2) Dropout: The probability of a neuron being turned off in the dropout layer.
3) Learning Rate: This determines how much the model parameters are updated with respect to the gradient of the loss function.
4) Epochs: The number of iterations used to train the model
5) Batch Size: The size of the batch of training data.

After training for 5 epochs, with a learning rate of 1e-04 and batch size of 4, we get training accuracy of 72% and report testing accuracy of 71%, which is quite good.

4.2. Binary Classification of propaganda text using BERT-based Binary Classifier

As described above, we use the pre-trained BERT language model to generate a contextualised word embedding, which is provided as input to a simple neural network to perform the binary classification.

4.2.1. NN Architecture:

The first layer of this network is the 'BertModel' layer which uses the pre-trained model to generate the word embeddings. The next is a dropout layer, which uses the pooled output provided by the BERT model as input and is used to prevent overfitting. The next is a linear layer that has an input size of 768 since the BERT model has an embedding dimension of 768 and an output size of 8 (number of data labels). The linear layer takes the pooled output of the BERT model as its input since this layer contains the contextualised word embeddings that capture the contextual information of each word in the sentence. A ReLU activation function is then applied on the linear output to get the final output which is the predicted label. The ReLU function also adds non-linearity to the network.

The hyperparameters in this Neural Network:

1) Dropout: The probability of a neuron being turned off in the dropout layer.
2) Learning Rate: This determines how much the model parameters are updated with respect to the gradient of the loss function.
3) Epochs: The number of iterations used to train the model
4) Batch Size: The size of the batch of training data.

After training for 1 epoch, with a learning rate of 1e-05 and a batch size of 4, the model displays a training accuracy of approx. 77% and a testing accuracy of 93% which is quite good.

5. Task 2:

5.1. Multiclass classification of propaganda text using GloVe vector representation based NN

For performing multiclass classification, a new derived column called ‘prop_span’ was created which contains only the span of text which contains propaganda. This column, along with the original column containing the entire sentence was passed as input to the model. The assumption was that the span of text containing propaganda would provide extra semantic meaning to the classifier and may help to increase the performance of the classifier to predict the different propaganda labels.

5.1.1. NN Architecture:

The classifier takes 2 inputs, x1 and x2. x1 is the tensor containing the pre-trained GloVe word embeddings for each token in the input sentence and x2 is the tensor containing the propaganda identified span of text. This multiclass neural network consists of 4 fully connected layers. The first fully connected layer takes the concatenation of tensors x1 and x2. And hence, the input size is 600, since both sentence vectors have a size of 300.

Fully Connected Layer	Size of Input Layer	Size of Output Layer
fc1	600	512
fc2	512	256
fc3	256	128
fc4	128	8

Table 1) Input and Output size of linear layers (GloVe-Multiclass-NN)

In this network the first three connected layers are followed by a batch normalisation layer. This layer will help to stabilise the training process and speed up the convergence of the model during training. The other benefits include reduced overfitting and improved generalisation of the model. Batch normalisation is applied to address the problem of vanishing gradients.

After each batch normalisation layer, a LeakyReLU activation function layer is added to reduce the probability of “dead” neurons where the ReLU function may turn off too many neurons by setting their outputs to 0. LeakyReLU helps to overcome this problem and may also lead to better performance. A dropout layer is also added after each ReLU layer to reduce the problem of overfitting to the training data, better understand the features in the training data and help the model generalise easily to unseen data (PyTorch, n.d.). After the final fully connected layer, fc4, we do not apply any LeakyReLU activation layer or dropout layer. The output from this final linear layer is fed into a softmax function which generates a probability distribution over the output labels. Since this is a multiclass classification problem, this helps to generate a probability for each output label and select the highest probability as the predicted label.

The hyperparameters in this Neural Network:

1) Size of the hidden layer
2) Dropout: The probability of a neuron being turned off in the dropout layer.
3) LeakyReLU: The LeakyReLU function parameter can be varied to increase performance
4) Learning Rate: This determines how much the model parameters are updated with respect to the gradient of the loss function.
5) Epochs: The number of iterations used to train the model
6) Batch Size: The size of the batch of training data.

Training the model with 20 epochs, and learning rate = 1e-03, we get a training accuracy of 53%. The accuracy on the test set stands at 64-68%

5.2. Multiclass Classification of propaganda text using BERT-based Binary Classifier

It was attempted to pass in the 2 features – the entire sentence and the propaganda span of text as input features to the BERT model but eventually failed to materialise. The pre-trained BERT model did not accept the concatenated tensors containing the entire sentence and span as inputs. Therefore, the BERT model was trained using the 'tagged_in_context' field alone.

5.2.1. NN Architecture:

The neural network uses the BERT model to extract contextualised embeddings from the input text. The pooled output from the BERT model is then passed as input to a linear layer with input size 768. There are 3 fully connected (dense) layers in the network, with their sizes described below.

Fully Connected Layer	Size of Input Layer	Size of Output Layer
fc1	768	400
fc2	400	100
fc3	100	8

Table 2) Input and Output size of linear layers (BERT-Multiclass-NN)

The output of each fully connected layer is passed onto a leaky ReLU activation layer. This introduces non-linearity in the network. The activation layer is followed by a dropout layer to limit overfitting the model to the training data and let the model generalise well to unseen data. The last layer in the network is a Softmax layer which produces a probability distribution over the number of classes/outputs in the network to predict the label.

The hyperparameters in this Neural Network:

1) Size of the hidden layer
2) Dropout: The probability of a neuron being turned off in the dropout layer.
3) LeakyReLU: The LeakyReLU function parameter can be varied to increase performance
4) Learning Rate: This determines how much the model parameters are updated with respect to the gradient of the loss function.
5) Epochs: The number of iterations used to train the model
6) Batch Size: The size of the batch of training data.

With 3 epochs, learning rate set to 1e-6 and number of batches set to 4, we obtain a training accuracy of 11% and a test accuracy of 15%.

This indicates the model has not been trained well, is underfitting the training data and will lead to false predictions. The model hyperparameters need to be tuned to get good performance and the epochs need to be increased. Since the model is being trained on a Windows Laptop with a NVIDIA GPU, GPU memory and CUDA cores are proving to be a limitation.

6. Loss and Optimisation:

While training all the below models, we use the Cross Entropy Loss to calculate the loss during each epoch. Cross Entropy loss is a commonly used loss function in neural networks which can be used for both binary classification and multiclass classification problems.

We use the Adam (Adaptive Moment Estimation) algorithm to update the weights of the neural network during training. It used a Stochastic Gradient Descent (SGD) optimiser that computes adaptive learning rates for each parameter in the network.

The models are trained, validated and tested in the Jupyter Notebook provided by the Anaconda Distribution. Google Colab was also used initially, but it failed to provide a stable environment to do multiple iterations of the model training, validation and testing cycles due to limits on using a GPU enabled kernel.

7. Conclusion

The models (except BERT Multiclass Classifier) performed really well during the training, validation and testing phase. Two different approaches were investigated.

1) Building a binary and multiclass classifier based on a uncontextualized word embeddings based on co-occurrence obtained from a pre-trained GloVe model
2) Building a binary and multiclass classifer based on contextualised word embeddings obtained from a pre-trained large language model – BERT.

In the binary classification task, the BERT classifier displayed excellent performance over very little epochs over a GloVe embedding based classifier trained over larger epochs. This is potentially due to the BERT model providing a contextualised word embedding that retains the meaning of the words in different contexts. Also, while the GloVe classifier used a tokenised, lemmatised, stopwords and punctuation removed string as input to produce the embeddings, the BERT classifier did not need any such pre-processing steps and used the raw text sentences as input.

Since BERT language models are already pre-trained on an enormous corpus of textual information, these models do have such pre-processing requirements and can directly work on the raw text, which is a big advantage. They also need very less iterations/epochs for training as the embeddings produced by BERT are tuned for a lot of downstream language processing tasks.

In the case of multiclass classifiers, the GloVe model displayed a higher accuracy and other performance metrics. This might partially be due to the fact that this model utilised 2 features to make predictions, while the BERT model was unable to do so. This is not a limitation of the BERT model, but is due to the lack of proper training and shortcomings in the building of the classifier. If this issue is fixed and with sufficient compute power, the BERT model is likely to do better than the GloVe model.

Futhermore, hyperparameter tuning can be done to increase the performance of all models. A function for this has been built, but cannot be run as this requires the models to be executed multiple times with varying batch sizes, epochs, learning rate etc, which is leading to OutofMemory exeception.

8. Code Appendix

See file – ‘APNLP_Code_Appendix_PranovSarath.ipynb’

9. References

Abad, A. et al., 2016. *Advances in Speech and Language Technologies for Iberian Languages.* Lisbon, Portugal, s.n.

Giovanni Da San Martino, A. B.-C. H. W. R. P. P. N., 2020. SemEval-2020 Task 11: Detection of Propaganda Techniques in News Articles. p. 1379.

Jacob Devlin, M.-W. C. K. L. K. T., 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Jeffrey Pennington, R. S. a. C. D. M., 2014. *GloVe: Global Vectors for Word Representationm Conference on Empirical Methods in Natural Language Processing (EMNLP),*. s.l., s.n., pp. 1532-1543.

PyTorch, n.d. *LeakyRELU.* [Online]
Available at: https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html
[Accessed 2023].

Smith, B. L., 2016. Propaganda. In: *Encyclopædia Britannica, Inc.* s.l.:Encyclopædia Britannica, Inc.

Wikipedia, n.d. *Propaganda.* [Online]
Available at: https://en.wikipedia.org/wiki/Propaganda
[Accessed 28 April 2023].