ADVANCED NLP : PROPAGANDA DETECTION

1. ABSTRACT

In this report our objective is to identify propaganda in text using natural processing techniques. Our main task is to identify whether propaganda exists in a given span of text and if yes to identify which propaganda technique was used. In order to do so we have used several methods to complete these two tasks like n-gram language models, word2vec and neural models. While doing so, it came out that for detecting the presence of propaganda and identifying it's type in a span of text specific models are performing better than other models which we will discuss in detail further in our report.

2. INTRODUCTION

The problem which we are addressing through this report is to detect the presence of propaganda in a given span of text and to classify the type of propaganda technique used. By presenting facts in a prejudiced or deceptive way, propaganda is a tactic used to influence public opinion. Propaganda may spread quickly and has the power to sway people's perceptions and choices, thus it is crucial to identify it. By examining text to find any manipulative or false information, propaganda may be found using natural language processing (NLP). In particular, for applications like social media monitoring, news analysis, and sentiment analysis, the use of NLP in propaganda identification has grown in significance. Making sure that individuals may base their judgements on correct and trustworthy information requires the development of an efficient system that can properly recognise propaganda and its specialised techniques. In order to address to our problem statement of identifying the presence of propaganda and it's specialised techniques we have employed several NLP models like n-gram language models, word2vec and neural models whose working and results we will further explore in our report.

3. PREVIOUS WORKS

Multiple investigations have been done to address the difficulties in identifying the deliberate dissemination of false or influenced information and the detection of propaganda in text is an important and active topic of research within the subject of natural language processing. In some researches, the use of domain-specific lexicons and ontologies to identify propaganda in particular settings, such as political speech or social media, has also been investigated. Propaganda detection is a difficult undertaking and research into novel strategies and procedures to increase accuracy and efficacy is ongoing utilising NLP methodologies. Some of the previous works in the field of propaganda detection are as follows:

- Zhang, J., Li, P., Peng, H., Zhang, X., & Liu, Y. (2019). A novel method for detecting propaganda based on word n-grams and semantic analysis. Journal of Intelligent & Fuzzy

Systems, 37(3), 3329-3336. [Link: https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs175364]

• Xu, Z., Xu, F., & Liu, K. (2016). An improved method of propaganda identification using n-gram model. In Proceedings of the 2016 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), pp. 1-5. IEEE. [Link: https://ieeexplore.ieee.org/document/7835932]

• Hossain, M. S., Mohammed, N., & Alelaiwi, A. (2018). Propaganda detection using word embeddings and convolutional neural network. Journal of Ambient Intelligence and Humanized Computing, 9(2), 417-428. https://link.springer.com/article/10.1007/s12652-017-0538-8

• Lai, V. T., Nguyen, T. T., & Pham, T. M. H. (2020). Propaganda detection using word2vec and machine learning. In Proceedings of the 2020 2nd International Conference on Advances in Computational Intelligence (pp. 21-27). ACM. https://dl.acm.org/doi/abs/10.1145/3425705.3425725

• Gupta, P., Khanna, A., & Joshi, J. (2020). Propaganda detection using LSTM with attention mechanism. In Proceedings of the 2020 9th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (pp. 776-779). IEEE. https://doi.org/10.1109/ICRITO49284.2020.9212235

• Gupta, S., Kumar, A., & Kumar, A. (2021). Propaganda detection in news articles using machine learning techniques. In Proceedings of the 2nd International Conference on Inventive Research in Computing Applications (pp. 103-108). IEEE. https://ieeexplore.ieee.org/document/9340736

4. THE DATASET

We are working on a propaganda dataset containing two columns: label and sentence. There are total 9 unique values of label column as follows:

1. flag_waving
2. appeal_to_fear_prejudice
3. causal_simplification
4. doubt
5. exaggeration,minimisatio
6. loaded_language
7. name_calling,labelling
8. repetition
9. not_propaganda

The first 8 labels are all propaganda techniques and are a subset of those identified in the Propaganda Techniques Corpus (Da San Martino et al., 2020). The final label not propaganda indicates that no propaganda has been identified in the text. The second column contains a sentence or chunk of text where the propaganda technique has been identified (or no propaganda has been identified in the case of not propaganda). Note the use of additional tokens <BOS> and <EOS> which indicate the beginning and end of the span of text (within the sentence) which is actually annotated with the given propaganda technique.

5. METHODS EMPLOYED

We have employed different NLP methods to tackle the two tasks. For the first task which is to identify propaganda is present or not in a given span of sentence (binary classification) we have used n-grams language model and word2vec methods . For the second task to identify what type of propaganda technique was present (multiclass classification) we have used n-grams language model and LSTM. Here we will talk about the methods and later we will talk about our code, implementation and accuracy of these methods to achieve our tasks.

- N-gram Model with Multinomial Naive Bayes
  This process can be used for both binary and multi class classification tasks and is able to process efficiently large chunks of texts that's why it is a very popular approach when it comes to propaganda detection. The CountVectorizer is used to first turn the text into a matrix of word counts, from which N-gram features are generated depending on the provided range. The Multinomial Naive Bayes classifier, which determines the probabilities of each label given the N-gram features, is trained using the resultant matrix. The classifier converts fresh text into a matrix using the same feature extraction technique as it receives new text, and it then predicts the label based on the determined probabilities.

- Word2Vec
  Word2Vec is also a popular choice for since it is very good in catching semantic relationships between words and propaganda in a given span of text. Using associations between words in a text corpus, the Word2Vec method creates vector representations of words. This approach is built on a neural network that has been trained on a sizable textual dataset, enabling it to learn the word co-occurrence patterns. A collection of dense vector representations that can identify the semantic and syntactic similarities between words is the end result of this training. For a variety of natural language processing applications these vector representations can be utilised.

- LSTM
  Long Short-Term Memory, often known as LSTM, is a sort of neural network architecture used in natural language processing. LSTM is helpful in propaganda detection because it can analyse sequential data and identify long-term relationships between words in a text corpus. By keeping a memory cell that can selectively retain or forget information based on the input and utilise it to understand the current

word in the sequence, LSTMs may recall crucial information from prior words. With each time step, the network receives an input, generates an output, and creates a hidden state to process the input data, which is how LSTMs work. The network is therefore able to recall data from earlier time steps and use it to influence predictions since the hidden state is passed on to the following time step along with the subsequent input.

5a. CODE IMPLEMENTATION

STEP1. In the first step, we import all the necessary libraries.

STEP2. In the second step, we load the dataset into pandas dataframe in training data and testing data. Then do basic checks with the dataset like seeing the first and last 5 rows of the training data and the testing data. Our data got 2 columns label and tagged_in_context so we rename the tagged_in_context as sentence.

STEP3. In the third step, we get the text in between the <BOS> and <EOS> tags and save it in a new column named span. Then we tokenize the span column and save it in a new column named tokens for both training dataset and testing dataset. Then in order to do the first task binary classification we convert the labels into propaganda or not propaganda and store it in a new column with the name propaganda_classification.

TASK 1

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

APPROACH 1 USING N GRAM LANGUAGE MODELS

First we split the data into training and testing sets using train_test_split function from the sklearn library. The text input is train_df['span'] and the corresponding label is train_df['propaganda_classification']. Then we convert the text data into n-grams using CountVectorizer function from the sklearn library. The ngram_range=(1,3) parameter specifies that we want to create n-grams of length 1 to 3. The fit_transform function is applied to the training set, and the transform function is applied to the testing set. This is done to ensure that the same vocabulary and n-gram representation is used for both the training and testing sets. Then we create an instance of the MultinomialNB classifier and fit it to the training data using the fit function. Then we predict the labels for the testing set using the predict function. Thereafter we create a confusion matrix using confusion_matrix function from the sklearn library. It takes the actual labels y_test and predicted labels y_pred as input and specifies the label names as ['propaganda', 'not_propaganda']. Then we visualize the confusion matrix as a heatmap using the heatmap function from the seaborn library. The annot=True parameter adds the numerical values to the heatmap and fmt="d" specifies the format of the values to be integers.

APPROACH 2 USING Word2Vec

_First we create training and test datasets by splitting the train_df and test_df data frames into feature and target variables. The X_train and X_test variables contain sentences, while y_train and y_test contain their corresponding propaganda classification labels. Then we define the maximum length of a sentence to consider in the model. Any sentence with a length greater than max_sequence_length will be truncated, while shorter sentences will be padded with zeros. Next we define a function called sentence_vector() that takes a sentence, a word2vec model, and a maximum sequence length as input parameters. The function checks if each word in the sentence is present in the word2vec model vocabulary. If so, the corresponding word vector is added to a list of word vectors for that sentence. If no word vectors are found for the sentence, it returns an array of zeros. The function then calculates the mean of the word vectors to obtain a sentence vector, and pads the vector with zeros or truncates it to the max_sequence_length. Next, we train a word2vec model on the train_df["tokens"] corpus. The vector_size parameter is set to 100, which means each word in the corpus will be represented by a vector of length 100. The window parameter is set to 5, which means that the model will consider the context of each word within a window of five words to generate word vectors. The min_count parameter is set to 1, which means that the model will consider all words in the corpus, regardless of their frequency. The workers parameter is set to 4, which means that the model will use four cores to train. Then we define the max_sequence_length variable as 100, which is the maximum length of the sentence vector that will be created. Next, we create sentence vectors for the training and test sets by applying the sentence_vector() function to each sentence in the dataset. These sentence vectors are then converted to numpy arrays. Then we define a logistic regression classifier using the LogisticRegression() function from the sklearn library and train it on the training set using the fit() method. The classifier is then used to predict the propaganda classification of the test set using the predict() method.Then we calculate the evaluation metrics (accuracy, precision, recall, and F1 score) using the accuracy_score(), precision_score(), recall_score(), and f1_score() functions from the sklearn library.Finally, we create a confusion matrix using the confusion_matrix() function from the sklearn library and visualize it using a heatmap from the seaborn library.

TASK 2

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

APPROACH 1 USING N GRAM LANGUAGE MODELS

In this task, a multi-class classification model is trained to classify propaganda techniques used in text. To prepare the data, the non-propaganda rows are removed from both the training and testing datasets, and a new column called propaganda_techniques is created with only the eight propaganda techniques as labels. Now we extract the newly created propaganda_techniques column from the training and testing data, respectively, and store them as train_labels and test_labels. Then we create a CountVectorizer object that will be used to convert the text data in the span column to numerical features. The fit_transform method is called on the training data to create the features, and the transform method is called on the testing data to create the same features based on the training data's vocabulary. We create a MultinomialNB object, which is a type of Naive Bayes classifier, and fit it to the training data using the features and labels. The predict method is called on the testing data to generate predictions based on the learned model. Thereafter we create a classification report that contains various metrics (precision, recall, F1-score) for each class (propaganda techniques) based on the predicted labels and the true labels from the testing data. Then we convert the classification report generated into a Pandas DataFrame for better visualization of the report.

APPROACH 2 USING LSTM

We first load the training and testing data, where the text data is stored in the 'span' column and the propaganda techniques labels are stored in the 'propaganda_techniques' column. The text data is then tokenized and converted to sequences of integers using the Keras tokenizer, and padded to ensure that all sequences have the same length. Using the scikit-learn OneHotEncoder the propaganda technique labels are one-hot encoded. Using the Keras functional API, a Sequential model is constructed with the embedding dimension set to 100. LSTM layer with 128 units, dense layer with softmax activation function for multiclass classification, and embedding layer make up the model.The model is then built using the categorical cross-entropy loss function and Adam optimizer, then trained using the training set for 50 epochs with a batch size of 20. The model is evaluated after training using the evaluate() function, which determines the test loss and accuracy. Using the predict() function, which produces the projected probability for each class label, the model makes predictions. The scikit-learn classification_report() function then creates the classification report by converting the predicted probabilities into class labels and returning several evaluation metrics, including precision, recall, and F1-score for each class label.

6. EVALUATION AND HYPERPARAMETERS

TASK 1

For task 1 where we need to identify whether a given span of text contains propaganda or not we used 2 models namely n-gram language_model and Word2Vec. For that task we can see that the accuracy of n-gram language model is greater than Word2Vec model. For n-gram language model the accuracy is 65% and for Word2Vec model the accuracy is 52%.

HYPERPARAMETER SETTINGS USED FOR N-GRAMS

ngram_range: set to (1,3) .

random_state: for same production of train-test split everytime, set to 42

MultinomialNB():Multinomial Naive Bayes classifier

WAYS TO IMPROVE ACCURACY FOR N-GRAMS

The accuracy of n-grams can be improved by experimenting with different n-gram ranges. By performing feature selection like chi-squared. By using a different classifier than Multinomial Naïve Bayes. Using pre-trained word embeddings and by modifying and exploring different parameters for Multinomial Naïve Bayes classifier.

<u>HYPERPARAMETER SETTINGS USED FOR Word2Vec</u>

vector_size = 100: The dimensionality of the word vectors in the Word2Vec model.

window = 5: The maximum distance between the target word and its neighboring words considered by the Word2Vec model.

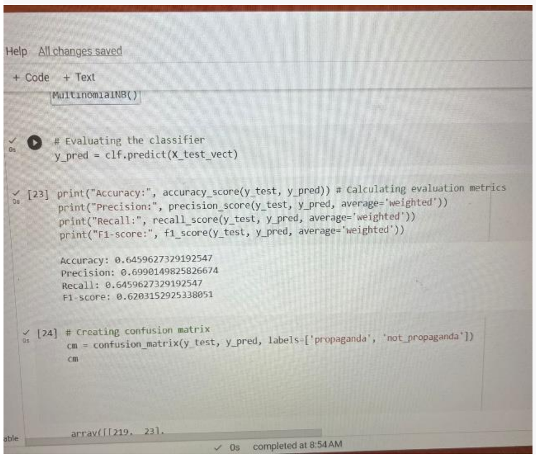
min_count = 1: The minimum frequency of a word in the corpus to be included in the Word2Vec model.

workers = 4: The number of worker threads used by the Word2Vec model to train the word vectors.

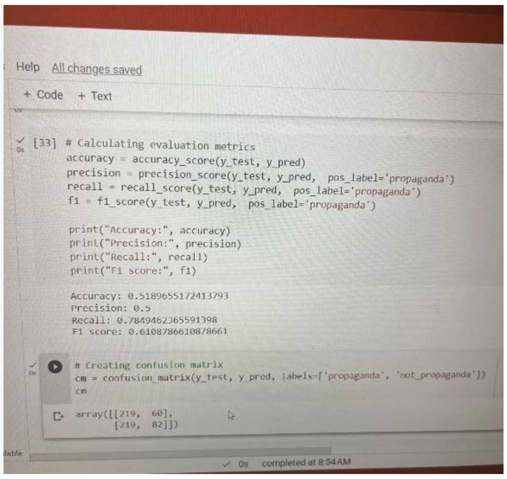
max_sequence_length = 100: The maximum length of a sentence vector after padding

<u>WAYS TO IMPROVE ACCURACY FOR Word2Vec</u>

The accuracy for Word2Vec could be improved by doing feature engineering, using some other classification model rather than logistic regression, increasing the size of the training data, adjusting and exploring different values for the hypermeters.



```
MultinomialNB()

# Evaluating the classifier
y_pred = clf.predict(X_test_vect)

[23] print("Accuracy:", accuracy_score(y_test, y_pred)) # Calculating evaluation metrics
     print("Precision:", precision_score(y_test, y_pred, average='weighted'))
     print("Recall:", recall_score(y_test, y_pred, average='weighted'))
     print("F1-score:", f1_score(y_test, y_pred, average='weighted'))

     Accuracy: 0.6459627329192547
     Precision: 0.6990149825826674
     Recall: 0.6459627329192547
     F1-score: 0.6203152925338051

[24] # Creating confusion matrix
     cm = confusion_matrix(y_test, y_pred, labels=['propaganda', 'not_propaganda'])
     cm

     array([[219,  23],
```



```
[33] # Calculating evaluation metrics
     accuracy = accuracy_score(y_test, y_pred)
     precision = precision_score(y_test, y_pred,  pos_label='propaganda')
     recall = recall_score(y_test, y_pred,  pos_label='propaganda')
     f1 = f1_score(y_test, y_pred,  pos_label='propaganda')

     print("Accuracy:", accuracy)
     print("Precision:", precision)
     print("Recall:", recall)
     print("F1 score:", f1)

     Accuracy: 0.5189655172413793
     Precision: 0.5
     Recall: 0.7849462365591398
     F1 score: 0.6108786610878661

     # Creating confusion matrix
     cm = confusion_matrix(y_test, y_pred, labels=['propaganda', 'not_propaganda'])
     cm

     array([[219,  60],
            [219,  82]])
```

<u>TASK 2</u>

For task 2 where we need to identify the type of propaganda technique present in the text we used n-grams model and neural model. For that task we can see the accuracy of neural model using LSTM is greater than n-grams model. The accuracy of Neural model is 41% whereas the accuracy of n-grams model is 38%.

<u>HYPERPARAMETER SETTINGS USED FOR N-GRAMS</u>

ngram_range: set to (1,3) .

WAYS TO IMPROVE ACCURACY FOR N-GRAMS

Data augmentation, feature engineering, hyperparameter tuning, preprocessing, ensemble learning or selecting a different model can improve accuracy for n-grams.

HYPERPARAMETER SETTINGS USED FOR NEURAL MODEL

max_words: The maximum number of words to keep, based on their frequency in the text. set to 10,000.

embedding_dim: The size of the word embeddings. It is set to 100 in this code.

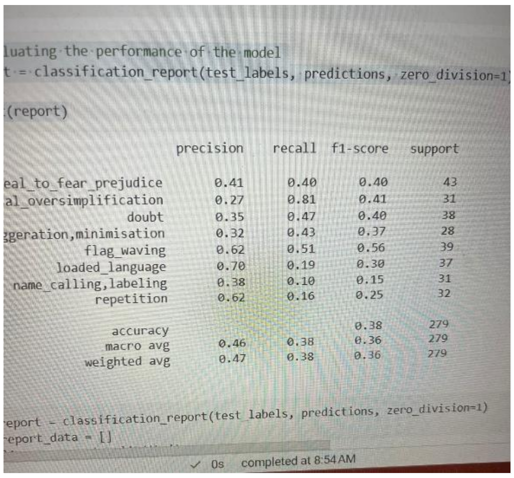
LSTM(128): The number of LSTM units in the layer. set to 128.

epochs: The number of epochs to train the model for. set to 50.

batch_size: The number of samples per gradient update. set to 20.

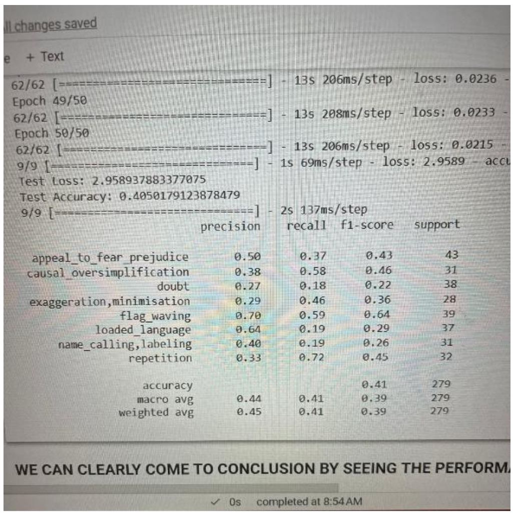
WAYS TO IMPROVE ACCURACY FOR NEURAL MODEL

Data augmentation, trying different model architecture, data preprocessing, modifying hyperparameters, regularization and to try to train own word embedding layer can result in improved accuracy for this model.



```
luating the performance of the model
t = classification_report(test_labels, predictions, zero_division=1

(report)

                          precision    recall  f1-score   support

eal_to_fear_prejudice         0.41      0.40      0.40        43
al_oversimplification         0.27      0.81      0.41        31
                doubt         0.35      0.47      0.40        38
ggeration,minimisation        0.32      0.43      0.37        28
          flag_waving         0.62      0.51      0.56        39
      loaded_language         0.70      0.19      0.30        37
  name_calling,labeling       0.38      0.10      0.15        31
           repetition         0.62      0.16      0.25        32

             accuracy                             0.38       279
            macro avg         0.46      0.38      0.36       279
         weighted avg         0.47      0.38      0.36       279

eport = classification_report(test labels, predictions, zero_division=1)
eport_data = []
```

✓ 0s completed at 8:54 AM



ll changes saved

e + Text

```
62/62 [==============================] - 13s 206ms/step - loss: 0.0236 -
Epoch 49/50
62/62 [==============================] - 13s 208ms/step - loss: 0.0233 -
Epoch 50/50
62/62 [==============================] - 13s 206ms/step - loss: 0.0215 -
9/9 [==============================] - 1s 69ms/step - loss: 2.9589 - accu
Test Loss: 2.958937883377075
Test Accuracy: 0.4050179123878479
9/9 [==============================] - 2s 137ms/step
                          precision    recall  f1-score   support

  appeal_to_fear_prejudice     0.50      0.37      0.43        43
 causal_oversimplification     0.38      0.58      0.46        31
                     doubt     0.27      0.18      0.22        38
exaggeration,minimisation      0.29      0.46      0.36        28
               flag_waving     0.70      0.59      0.64        39
           loaded_language     0.64      0.19      0.29        37
     name_calling,labeling     0.40      0.19      0.26        31
                repetition     0.33      0.72      0.45        32

                  accuracy                         0.41       279
                 macro avg     0.44      0.41      0.39       279
              weighted avg     0.45      0.41      0.39       279
```

WE CAN CLEARLY COME TO CONCLUSION BY SEEING THE PERFORM

✓ 0s completed at 8:54 AM

7. FUTURE WORKS

Even greater performance for propaganda identification may be attained by combining existing techniques or creating hybrid models.The generalisation and prediction capacities of these models might potentially be improved by experimenting with other model topologies, hyperparameters, and regularisation methods. Additionally, data augmentation and preprocessing techniques can increase the resilience of models to handle varied and noisy input. Generally speaking, the application of NLP for propaganda detection has shown encouraging results, but there is still more that can be done to increase the precision and potency of these systems. Our society's propagation of propaganda and false information may be halted by developing more resilient and adaptive models, which we can do with the support of ongoing research and development.

8. CONCLUSION

We tackled 2 tasks related to propaganda detection by the help of NLP methods. We got to know the positives and negatives about every model which we used that will help us in developing a more robust propaganda detection system in the future where we can explore some new models and even try to create hybrid models as well.

9. . REFERENCES

- Hassan, A., Li, J., & Xu, H. (2020). Fine-tuning BERT for Propaganda Detection. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1829-1838. https://doi.org/10.18653/v1/2020.emnlp-main.146
- de la Cunza, L., Iavicoli, M., & Tesconi, M. (2021). Propaganda Detection in Social Media: A Literature Review. ACM Transactions on Internet Technology (TOIT), 21(3), 1-36. https://doi.org/10.1145/3452448
- Shahsavari, S., Kolahi, S., & Khodabakhsh, A. (2021). Detecting Propaganda Techniques in News Articles Using a Hybrid Approach Based on Word Embeddings and Syntactic Dependencies. Expert Systems With Applications, 184, 115558. https://doi.org/10.1016/j.eswa.2021.115558
- J. Yang, H. Zhang, and J. Dong, "Propaganda detection using multi-level attention-based neural network," in Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, 2018. https://www.aclweb.org/anthology/P18-1223/
- Mollá, D., & Moreda, P. (2020). "The Propaganda Techniques Classification Challenge in Social Media". Proceedings of the 2nd Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda (ACL), 84-91. https://www.aclweb.org/anthology/2020.nlp4if-1.10/

- García-García, D., García-Sánchez, F., García-Cumbreras, M. A., & García-Santiago, R. (2021). "Deep Learning Approaches for Propaganda Detection: A Survey". Journal of Information Processing Systems, 17(2), 373-398. https://doi.org/10.3745/JIPS.04.0217
- Shu, K., Mahudeswaran, D., Wang, S., & Liu, H. (2021). "Fake News Detection as Natural Language Inference with Feature-based Attention". Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP), 2143-2153. https://www.aclweb.org/anthology/2021.acl-main.191/
- Hassan, N., Li, C., Arslan, F., & Carley, K. M. (2019). "Propagandist Personas on Twitter". Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 1164-1171. https://doi.org/10.1109/ASONAM.2019.8941945
- Zhou, J., Huang, J., Zhang, X., & Wang, Y. (2019). "A Survey of Deep Learning for Fake News Detection". Information, 10(6), 208. https://doi.org/10.3390/info10060208
- Xie, R., Liao, L., Zhang, J., & Wang, H. (2019). "Fake News Detection with Deep Learning". Proceedings of the 27th ACM International Conference on Multimedia (MM), 797-805. https://doi.org/10.1145/3343031.3351065
- Yang, Y., Sun, C., Wang, X., & Zhang, Z. (2021). "Fake News Detection with Multi-modal Learning: A Survey". Journal of Information Processing Systems, 17(4), 738-757. https://doi.org/10.3745/JIPS.04.0232
- Barrón-Cedeño, A., Mendoza, M., & Rosso, P. (2018). Detection of propaganda in news articles using NLP: the case of the 2016 US presidential elections. Information Processing & Management, 54(5), 807-825. https://doi.org/10.1016/j.ipm.2018.03.003
- Kumar, N., & Singh, S. (2021). A review on machine learning approaches for propaganda detection. In Advances in Computing and Communications (pp. 723-731). Springer. https://doi.org/10.1007/978-981-15-8759-4_68
- Lai, Y. C., & Tsai, M. F. (2021). A transformer-based approach to propaganda detection. In Proceedings of the Second Workshop on NLP for Internet Freedom: Censorship, Disinformation, and Propaganda (pp. 54-63). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.nlpcfi-1.6
- Yang, L., Xu, F., Guan, X., & Liu, X. (2019). N-gram analysis for detecting propaganda in news articles. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (pp. 6346-6352). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2019/881