# Fake News Challenge Stage 1 (FNC-1): Stance Detection

**Shabnam Hajian**

Department of Management Science, University of Waterloo

MSCI 641 – Text Analytics
Olga Vechtomova

August 5,2020

## Abstract

Fake news is made-up stories written with the intention to deceive (Tavernise, 2016). Fake news detection is a relatively new topic in machine learning. This project is based on the Fake News Challenge stage one (FNC-1)[1] as "stance detection". In the stance detection stage, each pair of headlines and news articles have to be classified as "unrelated", "discuss", "agree" or "disagree". For the input of the classification task in this project, the doc2vec model plus some of the baseline model features have been used. In this project, the advantages of neural models will be checked in comparison to the more prior models such as logistic regression. Since this projects NN model cannot bit the baseline model performance and the main topic of this project is to show improvements by using neural models, another model involved in this project as regression model. The goal of the regression model is not to compare with the baseline model, and it is only to compare and show improvement of neural model as the main topic of this project. Thus, the same input will be used for both simple cost-sensitive feed-forward neural model and Weighted Logistic Regression model and results have been compared.

## Introduction

To have a general idea about the topic, fake news is an allegation about some story that is misled by significant omissions or even false or lie and designed to deceive its intended audience (Dentith, 2016). The fake news detection is not easy even for humans. In early 2017 the "Fage News Challenge" organized to encourage using machine learning approaches for fake news detection (Chaudhry, Baker and Thun-Hohenstein, 2017). This project is focused on the first stage of fake news detection, named "stance detection". In this stage each pair of headlines and news has to be classified as "unrelated", "discuss", "agree" or "disagree". The data used in this project is the same as the data of the "FNC-1" challenge retrieved from challenge GitHub repository[2]. The organizers of the Fake News Challenge also provided the baseline model which used the "Gradient Boosting classifier" as the classification[3]. The goal is to find a suitable model for stance detection as the first step of the auto-detection of the fake news.

The reason of checking two classifiers instead of one was to experiment with the ability of the neural model of this project although it could not perform better than the baseline model. The main focus of this project was to use the ability of neural models in the classification tasks. The neural model of this project with the designed input did not hit the baseline model results. There are many possible reasons for this poor performance, such as choosing inappropriate input. The neural models have many hyper-parameters to be tuned plus the input of the classifier also affects the model performances. As a result, to show the advantages of the neural network model, it was needed to use the same input for two models and compare their results. Hence, the regression model has been designed to check the improvements in the classification by implementing the neural networks. Neither of these two models hit the

1 http://www.fakenewschallenge.org/

2 https://github.com/FakeNewsChallenge/fnc-1/tree/29d473af2d15278f0464d5e41e4cbe7eb58231f2

3 https://github.com/FakeNewsChallenge/fnc-1-baseline

baseline model performances. However, the neural model shows improvement in its results. On one hand, the neural model of this project, unfortunately, did not perform well. On the other hand, it is worth experimenting with new models and new approaches, though some of them cannot perform well by the nature of experimenting with new approaches.

In this project, some concepts of the baseline model used. Furthermore, the doc2vec model have been implemented to experiment with the advantages of this text vectorizer and its effect on the classification task. Concepts such as polarity, refuting words score plus overlapping scores are from the baseline model. Subjectivity is another score that has been added to the input of the model as a feature. The other important part of the input contains the doc2vec model which vectorizes the headline and news body articles.

For the classification, a weighted multinomial logistic regression and a cost-sensitive feed-forward neural network model have been used. Both models selected based on the nature of the training set data. The classification task has been done in one step which means that each pair of headlines and news articles directly classified as one of the four final class labels. Hence, the performances of both classifiers are tested based on this one step classification.

The final evaluation of the model is based on the official FNC-1 challenge score. Though, other traditional model evaluations have been used as well. The official score is a weighted score which 25% of its weight is to detect "unrelated" pair of data and 75% of the score weight is to detect the final label of related data as "discuss", "agree" or "disagree". The reason for this weighing is that determining unrelated/related labels is a more trivial task compared to determining other classes. This official metric described model performance for the unlabeled test set data and has been extracted from "CodaLab"[4]. Hence, more than traditional model evaluation terms, which will be used to evaluate models' performance on the validation set, the term "score" in this project used to address the official score for model performance on the test set.

## Background

A lot of valuable works have been done in the field of "stance detection". Following the FNC-1 Challenge, many published articles working on this stage of fake news detection. Many papers tried different approaches and published their results. From relatively easy approaches such as what Gupta and Kadam (2017) done by using a bag of the word and Support Vector Machines (SVM) and Random as a classifier, to some more complicated approaches by using LSTM-based bidirectional conditional encoding model plus applying pre-trained GloVe word has been done by Chaudhry, Baker and Thun-Hohenstein (2017). In some papers like Chopra, Jain and Sholar (2017), stance detection has been split into two sub-problems of defining related/unrelated stances and then defining the other three classes. Some of these papers also compared results of two-step classifiers with one-step classification (Chopra, Jain and Sholar, 2017). Borges, Martins and Calado, (2019) provided a good review of the past efforts in this topic.

In general, results show that using the LSTM-based bidirectional conditional encoding model with the attention layer provides a good score (Chopra, Jain and Sholar, 2017). However, checking new approaches is still valuable to find new better classifiers. Hence, this project experimented on two classifiers of logistic regression and neural networks by using some new features as the input of the model.

## Approach

The first thing to do is to understand data and find out how to use it. There are four files in the CSV format that be used in this project as data; for training purposes "train_bodies.csv" and "train_stances.csv" and for test model performance "test_bodies.csv" and "test_stances_unlabeled.csv" have been used. In this dataset, each news article body has been identified by a specific ID. The train_bodies and test_bodies files contain the body text of articles (the articleBody column) with corresponding IDs (Body ID). The train_stances file contains article headlines (Headline column) and article bodies id (Body ID column), referring to entries article body

4 https://competitions.codalab.org/competitions/24757?secret_key=d3211eb3-eb2e-4f18-aa10-967adf200a26

in the train_bodies file plus the label stances (the Stance column) for each pair of headline and body_id. The train_stances file similarly contains pairs of article headlines (Headline column) and article bodies id (Body ID column), referring to entries article body in the test_bodies file but is unlabeled and there is no stance column on that. In this paper, the first two train_bodies and train_stances CSV files merged based on the body id and called as training set; the two other files of test_bodies and test_stance_unlabed also merged based on the body id and called as test set. The training set used for training and validation and the test set only used for evaluating the model based on the CodaLab score. The important note about data is that some headlines paired with many body articles and vice versa. Now labelled training set in 49,972 rows and 4 columns and unlabeled test set in 25,413 rows and 3 columns are ready for further steps.

The input data of the stance detection task is a pair of headline and article body in text format. In order to work with the text data, it is important to do some text cleaning based on Natural Language Processing (NLP) concepts. In this project, a personalized function defined to clean text data. In this function, all texts first changed to a lowercase format, then “text regularization” has been applied and only alphabetic text data kept. Then the “stop words” removed from texts and word lemmatization performed to get the base words after removal of inflections. The “stop word” removal and “lemmatizing” process have been done by implementing an NLP package in python named “nlkt”. Since an important initial action in working with text data is doing to some text cleaning process, for all other steps in this project, the clean text data have been used.

As mentioned, two classifiers of this project used the same input data. The input is in the form of a vector of size 177 that used for the classification. This input “vector” can be discussed in three main parts as the headline data, features of headline and body and the news article body. Data of both headlines and article bodies are in the text format as mentioned. For modelling, it is needed to convert text data into a more understandable format for machines as numbers. This conversion named vectorization and there are many approaches for it, such as “bag of words” or “TFIDF”. This project has experimented with the Document to Vector (doc2vec) model for the text vectorization. Thus, the big part of the input vector contains doc2vec format of headline and article body. The combination of these doc2vec vectors, “subjectivity” plus some features from the baseline model created a general vector of size 177 that be used as input of classifiers.

| | Headline | Body Article |
|---|---|---|
| Minimum | 2 | 2 |
| Mode | 7 | 144 |
| 99 percentiles | 18 | 854 |
| Maximum | 28 | 2736 |

Table 1- Length information for "Headlines" and "Body Article"

As discussed, the biggest part of the input vector is the output of doc2vec models of each pair of headlines and body articles. The Doc2vec model is a derivation of the Word to Vector (word2vec) model, that first introduced by Mikolov, Chen, Corrado and Dean in 2013 as a method to convert text data into a vector format. The advantage of the word2vec approach over similar prior methods was its ability to keep semantic relation between words. The doc2vec is an approach introduced by Le and Mikolov (2014) a year after introducing the word2vec model. Similar to the word2vec, doc2vec method also has the ability to keep semantic relations, but it is useful for paragraphs. The doc2vec model has no limitation of same-sized input and this makes it a better model for document vectorization. The doc2vector model, gives a vector format of the inputted paragraph in a specific size, though the inputted paragraph might have different lengths. This fixed-size vector of the text as the output of the doc2vec model saved the semantic relations, just like what word2vec models do for words. The ability to keep semantic relations was the main reason for choosing the doc2vec text vectorizer for both headlines and body articles in this project as part of the input vector.

The important part in training a doc2vec model is to define the output vectors size. In order to choose the best vector size for the headlines and articles, it is needed to gain more knowledge about the length of the cleaned text of headlines and body articles. Table 1 shows the length of the headlines and article bodies in the training data set. Based on the provided information from table 1, for the headline doc2vec model size of 20 and for the article news body vector size of 150 has been set. By defining vector sizes, models can be trained on

the related data and then vectors of headlines and articles can be extracted from it.

The other feature in the input vector is the concept of "Polarity". The idea of using the polarity of the text is based on the baseline model but in this project, a different approach has been implemented to find this score. Polarity is a part of sentiment analysis that identifying the orientation of a sentence or text which can be positive, neutral, and negative (Liu, 2012). Polarity is one of the features that have been used in the baseline model of FNC-1. In this project, the polarity of headlines and article bodies have been extracted by using the "TextBlob" library. TextBlob is a python library that provided a simple API for diving into common natural language processing (NLP) tasks (Loria, 2018). The result of the TextBlob polarity score is a number in the range between [-1.0, 1.0]. The polarity score of headline and article news are two other features of the input vector.

The next feature of the input vector is the "Subjectivity" of each headline and news article. The subjectivity of a sentence expresses the amount of personal feelings, views, or beliefs in a text. A subjective text is expressed opinions, allegations, desires, beliefs, suspicions, and speculations, for instance, "I like this apple" (Liu, 2012). While objective texts, the opposite of subjective texts, are based on facts for example "This apple is red". Subjectivity is also a part of the sentimental analysis of the TextBlob library. The subjectivity of this library returns score as a decimal number within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective (Loria, 2018). The subjectivity score of both headlines and body articles have been used as two other features of the input vector.

There are two other concepts as refuting words score and overlapping score in the input vector which both have been extracted from the baseline model. Refuting words are a list of specified words such as "fake", "fraud", "hoax", "not", "denies" etc., that have the potential of providing very useful information based on the baseline model. Moreover, the overlapping ratio is calculated by counting the common token for each pair of headline-body, divided by the total number of tokens of both headline and body article. The overlapping ratio turned out to very different for unrelated stances from the other three classes (Gupta and Kadam, 2017). The refuting words for headlines and articles and the overlapping score of each pair of data are the last three features of the input vector.

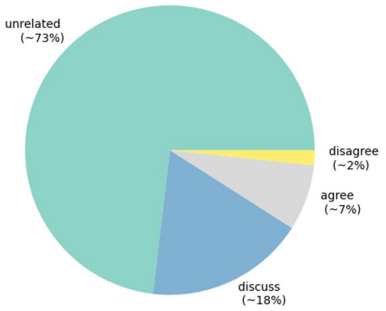



Figure 1- Share of the class label in training set

To sum up, the final vector which is a combination of features from both headlines and articles is the input used for both classifications. As explained in the introduction section, besides the main neural model of this project, the logistic regression model has been used to check model performances based on the same input data. This input vector is a combination of doc2vec vector of size of 20, polarity score, subjectivity score and refuting words scores of headline plus the overlapping score, followed by doc2vec vector of size of 150, polarity score, subjectivity score and refuting words scores of the article body. This input of size 177 is the one which used in classifiers.

Before modelling, there are two important points to be studied in the datasets. First is the amount of missing information and the other is the contribution of stances as the class value in the training set. About missing data, there is no need to concern as there is no missing information in neither of training-set nor the test-set. However, the contribution of the stance label in the training set has to be studied. Figure 1 shows the contribution of the stances as the target value in the training set and illustrates the data is severely imbalanced. As there is no missing data, imbalance feature should only be considered for the modelling step.

The imbalance class data in the dataset leads us to the "Accuracy Paradox" problem in this dataset. This simply means that when there is a large class imbalance in the dataset, further inspection is essential for model evaluation. Since the dataset has been used in this project have severely skewed class distribution, considering the model accuracy to evaluate and compare models is not enough. For instance, in this project, if a model falsely considered all stances as "unrelated", based on the contribution of the stances in dataset, the model

reaches the accuracy of 73%. However, in reality, this model performed badly and did not specify any relations between headlines and body articles. As a result, it is important to investigate model performance more accurately especially when the data set is imbalanced. Since it is possible that model with lower accuracy was preferred over the model with higher accuracy.

In this project, the accuracy paradox caused by the imbalanced class data exists. Hence, besides the score, both classification reports and confusion matrix used to evaluate models' performance in the validation set. The classification report is used to have more detailed information about the accuracy, precision and f1-score for each class. Moreover, the confusion matrix will present a clear and unambiguous picture of the results of the model. In addition to the score as the weighted measure for the test set which has been discussed before, both the confusion matrix and the classification report together can be used to evaluate models in the validation set and comparing the performances.

The unbalanced contribution of class data in training set should also be considered in the training step of the models. Most models cannot handle imbalance data for training. Thus, there are methods such as up-sampling examples from the minority classes or down-sampling examples from the majority classes to deal with imbalance data. These two approaches aim to balanced class data and then used balanced data to train the model. Based on Wang et al. (2016) if imbalance data had not been considered in the training step and model be trained with balance data, there is no guarantee for the good performance of the model on imbalance data after that. Hence, Cost-Sensitive Neural Network and weighted logistic regression models have been defined as a solution to deal with skewed class distribution in this project. Both models have armed to deal with imbalance data in the training stage.

The first model discussed is the "weighted logistic regression" model. This model can be regularized to train with imbalanced data by tuning its hyper-parameters. The logistic regression model is a relatively simple classifier. This model called simple as it has relatively few hyper-parameters to tune. In this project, there are two hyper-parameters that need to be changed from its default mode. The "class_weight" that needed to set as "balanced" and the "multi_class" that needed to be set as "multinomial". Based on the documentation of the model from the "sklearn" library in python[5], the "balanced" mode "*uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data*". This is what helps the model to train with imbalanced data. The "multi_class" is needed to define the loss function of the model. By setting loss as 'multinomial', the "*loss fit across the entire probability distribution*" minimized. After setting these two hyperparameters, the weighted logistic regression model is ready to be trained and evaluated in the validation and test set.

The next model is based on a relatively new concept in supervised learning as the Neural network model. The feed-forward neural networks can be imagined as a function approximation machines, that achieve some generalize insights from data (Goodfellow, Bengio and Courville, 2016, p. 169). Function approximation is a technique that uses available observation, and historical data to estimate an unknown function underlying in data. This process is somehow like the function that is known from the brain and the name of this technique is based on this similarity. Nowadays, this powerful new approach is used to solve complex classification. Hence, in this project, the feed-forward neural model has been chosen for classification.

The second model of this project is the cost-sensitive neural network model. The weighted neural network model is used in this project to deal with the problem of imbalanced class data. The standard neural network algorithm does not support imbalanced classification. While in this dataset, class data is skewed. A cost-sensitive learning method can be used to solve the imbalanced data problem (Wang, Liu, Wu, Cao, Meng and Kennedy, 2016). The "class_weight" is the hyperparameter of the neural model that maps class values to each class in order to reflect their importance. In general, the best practice of class weighting is to use the inverse of the class distribution present in the training dataset; to define these class weights, the "sklearn" library has been

5 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

implemented. Thus, the Cost-Sensitive Neural Network by heuristic approaches for defining weighting has been chosen as a solution to deal with skewed class distribution in this project.

The neural networks are new and powerful approaches in classification. Neural models are also complicated, since there are many hyperparameters that need to be tuned. Moreover, having large weights in the neural model means a more complex model and increases the risk of overfitting. The more model becomes overfitted, the less generalization ability it will have. This means that model might perform incredibly good on the training data by becoming overfitted, while it performs poor results for the unseeded test data. The model performance on the unseen new data shows the generalization ability of the model. Neural network models are powerful tools to define relations. As a result, these models need some regularizations to prevent overfitting.

The dropout layer helps the neural model to prevent overfitting and increases model generalization power. Dropouts help the model to prevent overfitting by the cost of creating a noisier training process (Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov, 2014). As Srivastava et al. (2014) mentioned, by applying dropout in some nodes, all incoming and outgoing connections of that nodes temporarily be removed from the network. Hence, dropout layers reduce model capacity and a wider model is needed when using dropouts based on the Srivastava et al. (2014) findings. Dropouts applied to a layer by using a fixed probability independently from other layers. Probabilities of 50% for dropout of hidden units and 20% for dropout of visible units give good results in a wide range of networks and tasks (Hinton, Srivastava, Krizhevsky, Sutskever and Salakhutdinov, 2012). However, it is essential to check other probabilities to find the best results based on the structure of the model. In this weighted neural model, dropout regularization applied to prevent overfitting.

Furthermore, the batch normalization layer can be implemented to speed up the training process of the neural model (Ioffe and Szegedy, 2015). The batch normalization layer can also act as regulator based on the paper of Ioffe and Szegedy (2015). This layer can be used before and after the activation function of the layer (Ioffe and Szegedy, 2015). The batch normalization is automatically standardizing the inputs to a layer. This technique

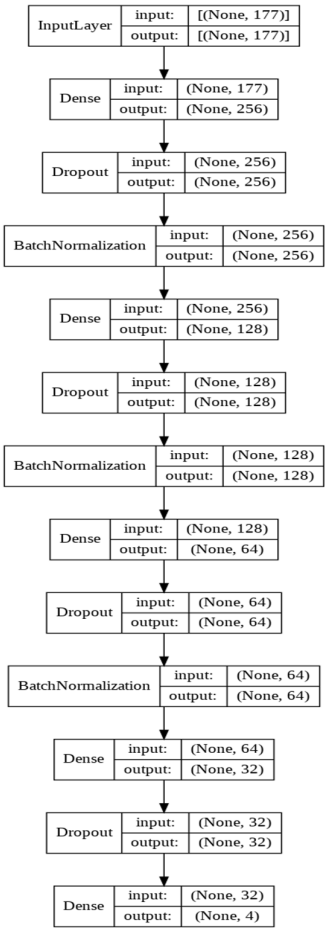



Figure 2- Architecture of the NN model

is designed to accelerate the learning of the neural model. In this project also, the batch normalization layer has been used to speed up learning process of the model.

The learning rate is another hyperparameter that affects neural model performance. Base on the Goodfellow et al. (2016, p. 429) the learning rate is the most important hyperparameter to tune. The learning rate is the amount that the weights are updated during training. On one hand, large learning rates result in unstable training and on the other hand, small rates lead to fail in training. As a result, the learning rate must define cautiously.

The last step is to study the neural model architecture. Figure 2 shows the neural model layers and the number of nodes in each layer. The model used in this project is the sequential model which accepts a vector of size 177 as input and classified data into 4 categories. The sequences of the model layers, the width of each layer and the type of layers have been shown in this picture are made the architecture of the model.

## Experiments

To experiment with data and evaluate models' performances both training set and test set is needed. The training set data has been split into two parts for training and validating the model. From the training set, 80% used to train models as training and 20% used in order to tune hyper-parameters of the model and check model performances as validation. The unlabeled test set also used to check model performance based on the official FNC-1 weighted score. To have this score it is needed to create a CSV file with three columns as "Headline", "Body ID" and predicted "Stance". The body id refers to the related news article from the other file. Thus, to evaluate models after training them, first the results of the validation set will be discussed and then the official score of the test set will be mentioned.

First, the logistic regression model performances have been evaluated. After training the model, the validation set has been used to predict data. Table 2 and figure 3 respectively show the classification report and confusion matrix of the logistic regression model on the validation set. Also, the score of the test data for the logistic regression model is 4073.75 based on the CodaLab. All these results show the poor performance of the weighted logistic regression model.

The next model to be evaluated is the neural model. The feed-forward neural network model of this project has been trained based on all mentioned notions of the neural networks and the validation set used to tune the hyperparameters of the model. After checking several models and components (activation functions and optimizers) and changing hyper-parameters, the final model has these properties: The optimizer used is SGD with learning rate of 1e-6, decay of 1e-2 and momentum of 0.99, the, the batch size is 32 and the model has been trained for 50 epochs. Moreover, the activation function of all hidden layers is relu and for the last layer is sigmoid.

The important note is the existing accuracy paradox in this data as a result of imbalance data. The model accuracy decreases per epochs in training process, but the model needs enough time to learn and modify weights. Table 3 shows the classification report and figure 4 is the confusion matrix of the neural model on the validation set. The official CodaLab score for this

| Classification report | precision | recall | f1-score | support |
|---|---|---|---|---|
| agree | 0.09 | 0.48 | 0.15 | 765 |
| disagree | 0.00 | 0.00 | 0.00 | 167 |
| discuss | 0.18 | 0.45 | 0.26 | 1,720 |
| unrelated | 0.74 | 0.15 | 0.26 | 7,343 |
| accuracy | | | 0.23 | 9,995 |
| macro avg | 0.25 | 0.27 | 0.17 | 9,995 |
| weighted avg | 0.58 | 0.23 | 0.24 | 9,995 |

Table 2- Logistic regression classification report

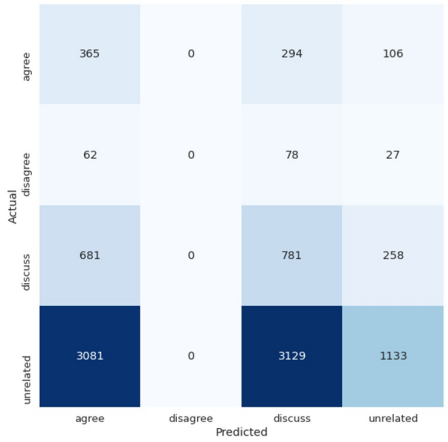



Figure 3- Logistic regression confusion matrix

| Classification report | precision | recall | f1-score | support |
|---|---|---|---|---|
| agree | 0.8 | 0.09 | 0.09 | 765 |
| disagree | 0.02 | 0.41 | 0.03 | 167 |
| discuss | 0.17 | 0.89 | 0.28 | 1,720 |
| unrelated | 0.74 | 0.76 | 0.75 | 7,343 |
| micro avg | 0.33 | 0.73 | 0.46 | 9,995 |
| macro avg | 0.25 | 0.54 | 0.29 | 9,995 |
| weighted avg | 0.58 | 0.73 | 0.61 | 9,995 |
| samples avg | 0.32 | 0.73 | 0.44 | 9,995 |

Table 3- NN model classification report

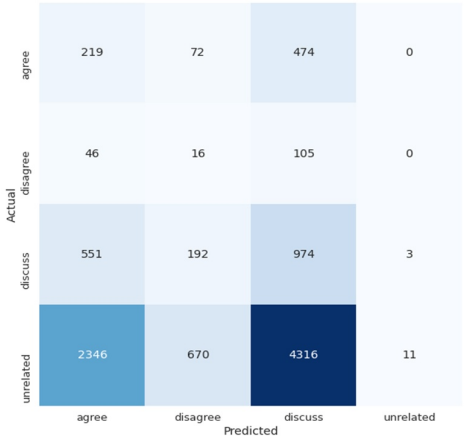



Figure 4- NN model confusion matrix

model is 5004.0. From all this information it can be concluded that this model performed better than the logistic regression model, although neither of them shows better "score" in comparison to the baseline model results.

## Conclusion

Sadly, the approaches used in this project and the designed neural model did not show better performance over the baseline model though several hyper-parameters have been checked. This can be as a result of the bad inputs or the structure of the models. In order to check the performance of the neural model without considering the effect of the input data, the logistic regression classifier has been trained and results have been compared.

Neural Networks models are powerful tools but there are a considerable number of hyper-parameters that needed to be tuned. The larger weights make the model also more complicated and even small changes can affect the model considerably. In this NN model, using decay to control learning rate considerably increase model performance. Moreover, the number of epochs is important. Train model more than it needed caused overfitting as weights are big.

*Area for improvements*

As Dai, Olah, and Le, (2015) mentioned, the vector size in the doc2vec models can affect the performance of the vectorization. Since a considerable amount of the input data comes from the doc2vec models, vector size of these models is important. Besides the other text vectorizers such as word2vec or TFIDF must be checked in this model.

The other point that can be experimented with is to use a functional model instead of sequential. Based on this single input, the sequential model performed better. However, by using a functional model it is possible to use three vectors of headlines doc2vec, features vector and article body doc2vec separately as input. And checked model performance.

## References


Borges, L., Martins, B., & Calado, P. (2019). Combining similarity features and deep representation learning for stance detection in the context of checking fake news. Journal of Data and Information Quality (JDIQ), 11(3), 1-26.

Chaudhry, A. K., Baker, D., & Thun-Hohenstein, P. (2017). Stance detection for the fake news challenge: identifying textual relationships with deep neural nets. CS224n: Natural Language Processing with Deep Learning.

Chopra, S., Jain, S., & Sholar, J. M. (2017). Towards automatic identification of fake news: Headline-article stance detection with LSTM attention models. In Stanford CS224d Deep Learning for NLP final project.

Dai, A. M., Olah, C., & Le, Q. V. (2015). Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998.

Dentith, M. R. (2016). The problem of fake news.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

GUPTA, Y. N. A., & KADAM, Y. (2017). Stance Detection for the Fake News Challenge.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In International conference on machine learning (pp. 1188-1196).

Liu, B. (2012). Sentiment analysis and opinion mining. Synthesis lectures on human language technologies, 5(1), 1-167.

Loria, S. (2018). textblob Documentation. Release 0.15, 2.

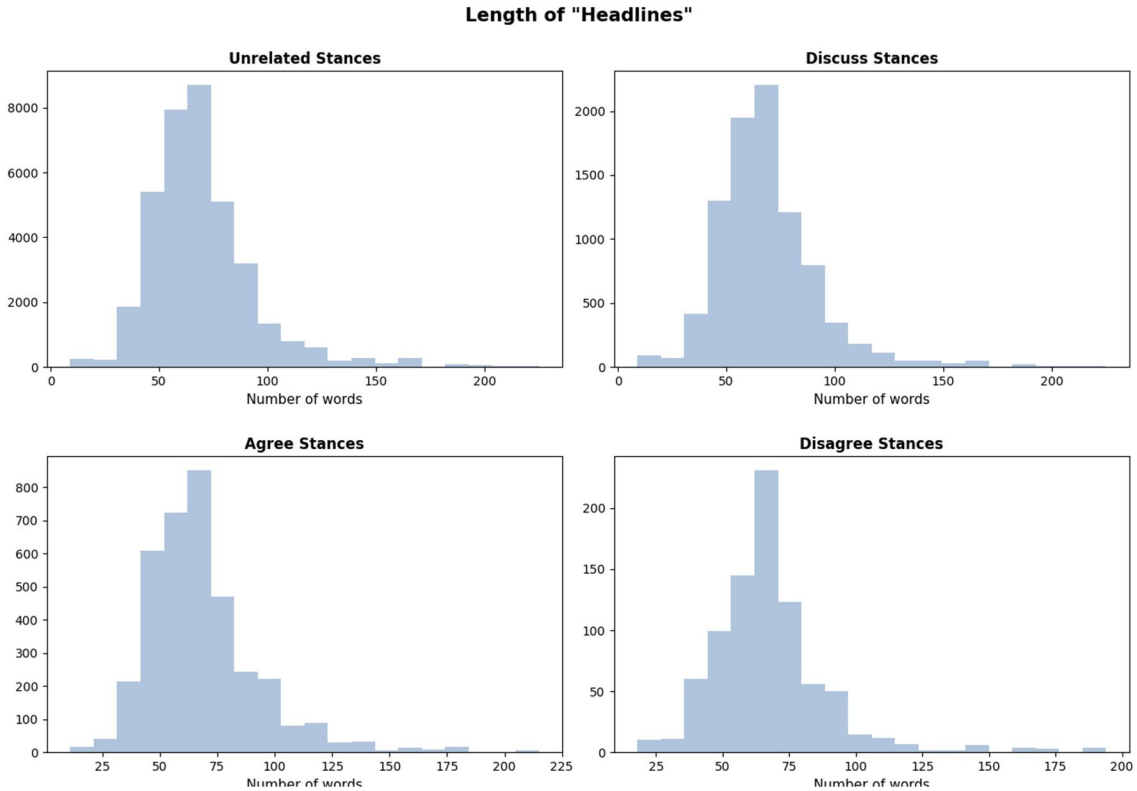
Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

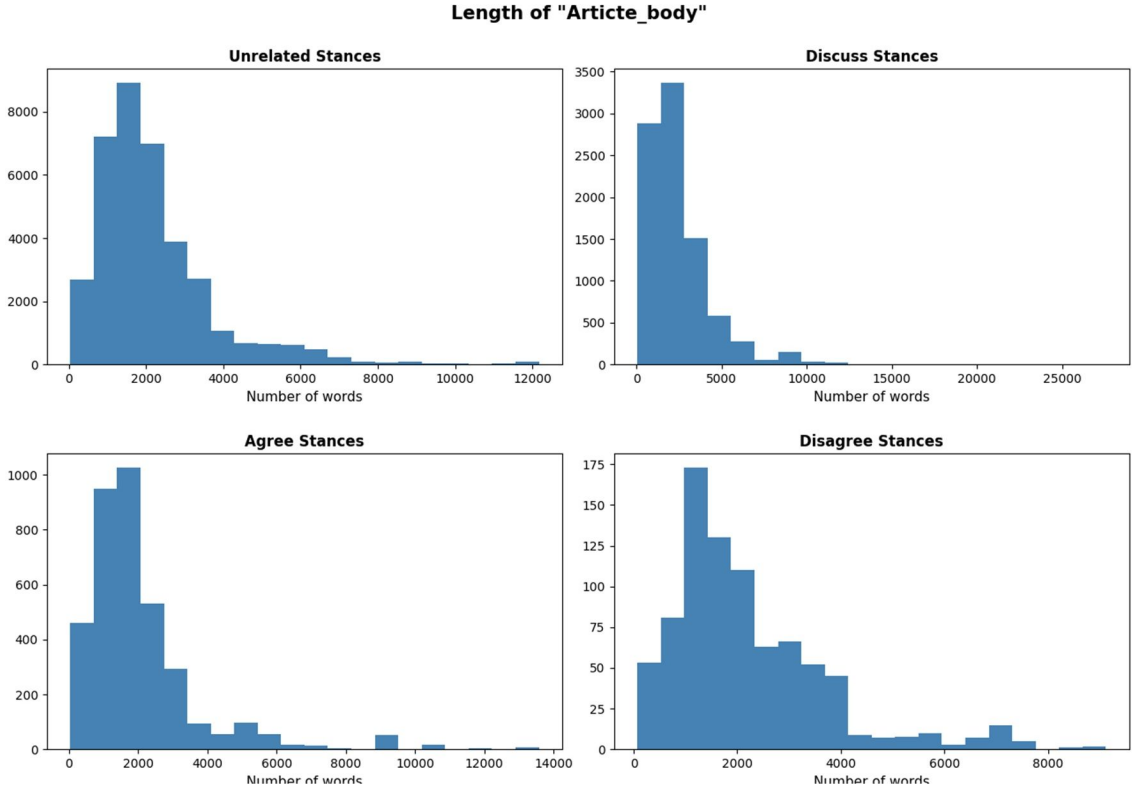
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

Tavernise, S. (2016). As fake news spreads lies, more readers shrug at the truth. The New York Times, 6.

Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q., & Kennedy, P. J. (2016, July). Training deep neural networks on imbalanced data sets. In 2016 international joint conference on neural networks (IJCNN) (pp. 4368-4374). IEEE.

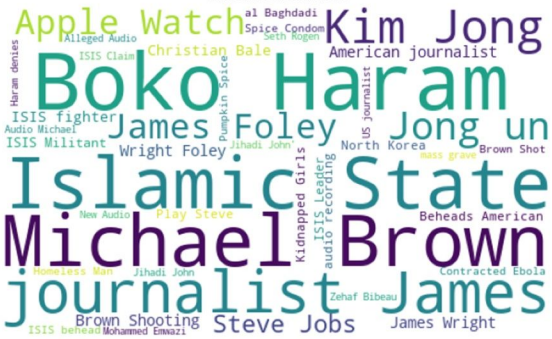
# Appendix 1


Length of "Headlines"
Unrelated Stances
Discuss Stances
Agree Stances
Disagree Stances
Number of words

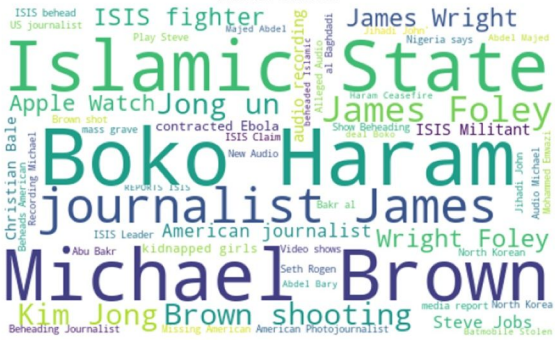

Length of "Articte_body"
Unrelated Stances
Discuss Stances
Agree Stances
Disagree Stances
Number of words

**Most frequent words in "Headlines"**

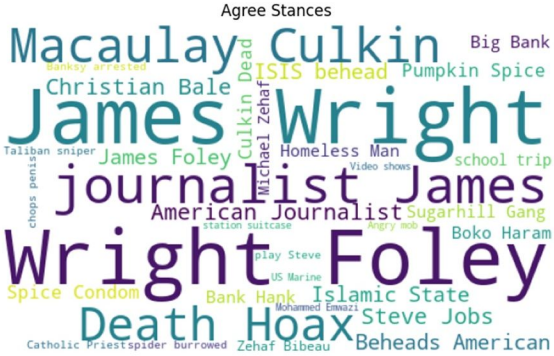
Unrelated Stances




Discuss Stances




Agree Stances

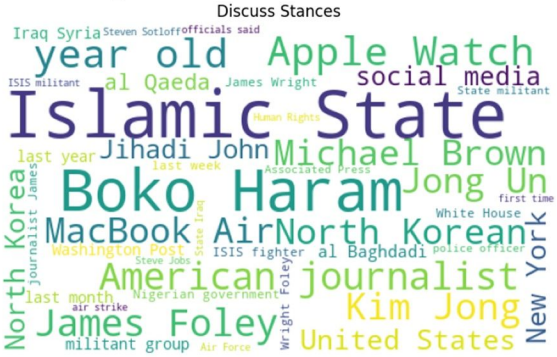



Disagree Stances

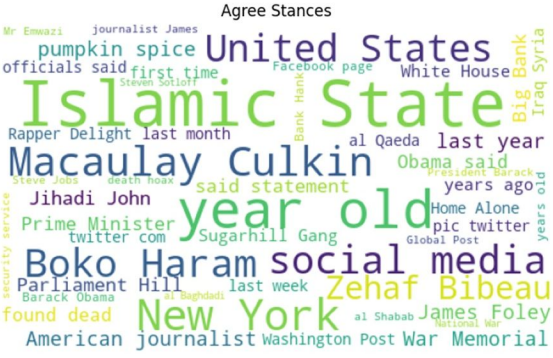

**Most frequent words in "Article_Body"**

Unrelated Stances


Discuss Stances

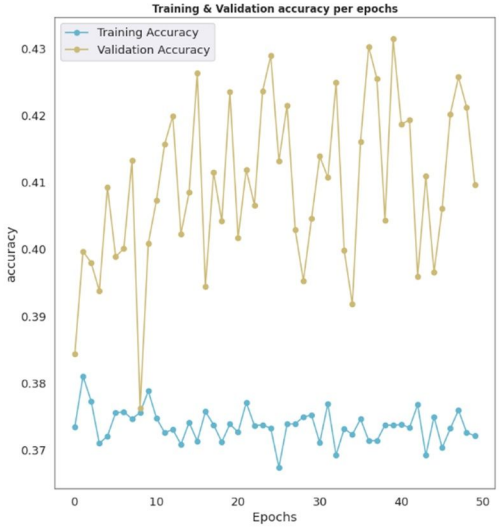


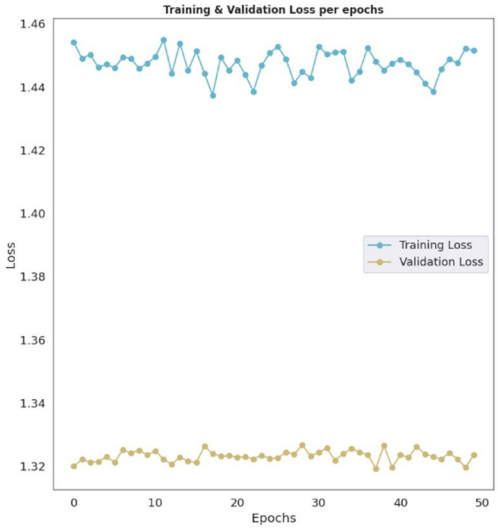

Agree Stances




Disagree Stances

**Fitting model history per epoches**



# Appendix 2

## MSCI 641 - Text analytics

### Final courese project

University of Waterloo - spring 2020.

## Preparing coding environment

These codes have been run in **Google Colab** and the coding environment has been set based on that.

1. Importing essential libraries and download some extra components of packages(need to upgrade tensorflow library)
2. Setting random states to have consistency in answers every time of running codes
3. Turning warnings off
4. Uploading data
5. Saving data as a dataframe
6. Setting a drive on google drive to be able to save models, pictures and graphs
7. Checking system GPU
8. Running personalized functions that will be used in the rest of the codes

```
# To upgrade tensorflow library
!pip install tensorflow
!pip install --upgrade tensorflow
!pip install tf-nightly
!pip install keras
```

```
# To import requred libraries
import io
import os
import warnings
import random
import statistics
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import re
import nltk
```

```
import keras
import gensim
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from nltk.corpus import wordnet
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from wordcloud import WordCloud
from textblob import TextBlob
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import plot_model

print('All required libraries were imported')
```

```
# To download essential components of nltk library and set them
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
stopwords = set(nltk.corpus.stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
print('\n', '*'*15, 'All extra packages downloaded and set', '*'*15)
```

```
# To set random seeds
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)
PYTHONHASHSEED=123
print('Random seeds set')
```

```
# To turn warnings off
warnings.filterwarnings("ignore")
print('All warnings turned off')
```

```
# To upload data file and wait untill become 100% done. This might takes some time
'''Select all 4 .csv files which are needed in text codes as:
"train_bodies.csv", "train_stances.csv","test_bodies.csv", "test_stances_unlabeled.csv"'''
from google.colab import files
uploaded = files.upload()
```

```
# To save data on dataframe
```

```
train_bodies = pd.read_csv(io.BytesIO(uploaded['train_bodies.csv']))
train_stances = pd.read_csv(io.BytesIO(uploaded['train_stances.csv']))
test_bodies = pd.read_csv(io.BytesIO(uploaded['test_bodies.csv']))
test_stances_unlabeled = pd.read_csv(io.BytesIO(uploaded['test_stances_unlabeled.csv']))
print('All 4 required datasets are stored in dataframes with same name as their .csv files
```

```
# To set a folder to save images and tabels
from google.colab import drive
drive.mount('/content/gdrive')
os.chdir("gdrive/My Drive/MSCI 641-Project")
print(os.getcwd())
print('MSCI641-Project in google drive set as repository')
```

```
# To check the GPU of running codes (If using google colab GPU the output should be "/devi
tf.test.gpu_device_name()
```

These functions will be used later on this project.

```
# To define some personalized functions

# This will combine two seperate .csv files and return a dataframe
def create_data_set ( headline_stance_file,  body_file):
  dataset = pd.merge(headline_stance_file, body_file ,on ='Body ID',how ='outer')
  dataset.drop(['Body ID'],axis=1,inplace=True)
  return dataset

# To make words Pos tags similar
def get_simple_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN


# This function clean text data and return it as text
def clean_text(text):
  text = text.lower()  # lower case the text
  text = re.sub(r"[^a-z]+", ' ', text)  # remove all signs and punctuation and numbers
  words = word_tokenize(text)  # Tokenize data
  wordsFiltered = []
  lemmalist= []
  for w in words:
    if w not in stopwords:
      wordsFiltered.append(w)
      pos = pos_tag([w])
      lemma = lemmatizer.lemmatize(w,get_simple_pos(pos[0][1]))
```

```
      lemmalist.append(lemma)
  return lemmalist
#  return '{}'.format(' '.join(lemmalist))


# This function remove characters and lower case and return data as text
def no_char(text):
  text = text.lower()  # lower case the text
  text = re.sub(r"[^a-z0-9_]+", ' ', text)  # remove all signs and punctuation and numbers
  return text


# This function returns the score of refutation words in text
def refutation_score(clean_text):
  score=0
  for words in clean_text:
    for w in refuting_words:
      score = score + 1
  return score


# This function gives each pair of headline and body and return the amount of overlapping
def overlapping(list_cleaned_headline , list_cleaned_body):
  overlap = len(set(list_cleaned_headline).intersection(list_cleaned_body)) / float(len(se
  return [overlap]


print('Personalized functions have been loaded')
```

## Exploratory data analysis (EDA)

Next codes are to explore the original data.

```
# To join training dataframe
train_df = create_data_set (train_stances, train_bodies)
train_df = train_df[['Headline','articleBody','Stance']]
train_df.head()
```

```
# To join test dataframe
test_df = create_data_set (test_stances_unlabeled, test_bodies)
test_df.head()
```

```
# To check data shape
print('Training dataset:')
print('# of columns:', train_df.shape[1])
print('# of rows:', train_df.shape[0])
print('\n', '*'*15, '\n')
print('Test dataset:')
print('# of columns:', test_df.shape[1])
print('# of rows:', test_df.shape[0])
```

```
# To check data types
print('Training dataset data types')
print(train_df.dtypes)
print('\n', '*'*15, '\n')
print('Test dataset data types')
print(test_df.dtypes)
```

```
# To check missing values
print('training dataset missing values')
print(train_df.isna().sum().T)
print('\n', '*'*15, '\n')
print('test dataset missing values')
print(test_df.isna().sum().T)
```

From now, only the training dataframe will be explored and test dataframe will be used only to evaluate the performance of the model.

```
# To calculate the share of data in training set
unrelated_count = (train_df['Stance']=='unrelated').sum()
agree_count = (train_df['Stance']=='agree').sum()
disagree_count = (train_df['Stance']=='disagree').sum()
discuss_count = (train_df['Stance']=='discuss').sum()
total=train_df.shape[0]
total_related = agree_count + disagree_count + discuss_count
print('# of rows:', total)
print('# of unrelated posts:',unrelated_count, '({:4.2f}'.format(unrelated_count*100/total
print('# of agree posts:',agree_count, '({:4.2f}'.format(agree_count*100/total),'% of tota
print('# of disagree posts:',disagree_count, '({:4.2f}'.format(disagree_count*100/total),'
print('# of discuss posts:',discuss_count, '({:4.2f}'.format(discuss_count*100/total),'% o
```

```
# To check dataframe
train_df.head()
```

```
# To visually show the share of data
fig, (ax1,ax2) = plt.subplots(ncols=2, figsize=(10, 5), dpi=100)
plt.tight_layout()

train_df['category'].value_counts().plot(kind='pie',colormap='Set3', ax=ax1 ,startangle=36
ax1.set_ylabel(' ')
ax1.set_title('Share of all four labels \n  in training data set', fontsize=11)

train_df[train_df['category'] != 3 ]['category'].value_counts().plot(kind='pie',colormap='
ax2.set_ylabel(' ')
ax2.set_title('Share of other three labels \n in "related"group of training data set', fon


fig.savefig('Images/datashare.png')
plt.show()
```

```
# To use wordcloud visualization to see the most frequent tokens in headlines
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

wc_unelated = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color =
ax[0][0].imshow(wc_unelated)
ax[0][0].set_title('Unrelated Stances')
ax[0][0].axis('off')

wc_discuss = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color ='
ax[0][1].imshow(wc_discuss)
ax[0][1].set_title('Discuss Stances')
ax[0][1].axis('off')

wc_agree = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color ='wh
ax[1][0].imshow(wc_agree)
ax[1][0].set_title('Agree Stances')
ax[1][0].axis('off')

wc_disagree = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color =
ax[1][1].imshow(wc_disagree)
ax[1][1].set_title('Disagree Stances')
ax[1][1].axis('off')

plt.axis("off")
plt.grid(b=None)
fig.suptitle('Most frequent words in "Headlines"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.05, hspace=0, top=0.95)

fig.savefig('Images/Hedlines_wordclouds.png')
plt.show()
```

```
# To use wordcloud visualization to see the most frequent tokens in article body
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

wc_unelated = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color =
ax[0][0].imshow(wc_unelated)
ax[0][0].set_title('Unrelated Stances')
ax[0][0].axis('off')

wc_discuss = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color ='
ax[0][1].imshow(wc_discuss)
ax[0][1].set_title('Discuss Stances')
ax[0][1].axis('off')

wc_agree = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color ='wh
ax[1][0].imshow(wc_agree)
ax[1][0].set_title('Agree Stances')
ax[1][0].axis('off')

wc_disagree = WordCloud(width = 500 , height = 300 , max_words = 1000 , background_color =
ax[1][1].imshow(wc_disagree)
ax[1][1].set_title('Disagree Stances')
```

```
ax[1][1].axis('off')

plt.axis("off")
plt.grid(b=None)
fig.suptitle('Most frequent words in "Article_Body"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.05, hspace=0, top=0.95)

fig.savefig('Images/body_wordclouds.png')
plt.show()
```

```
# To use visualization to compare the length of texts in headlines
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

ax[0][0].hist(train_df[train_df.category == 3].Headline.str.len(),bins = 20,color='lightst
ax[0][0].set_title('Unrelated Stances', fontweight='bold')
ax[0][0].set_xlabel('Number of words',fontsize=11)
ax[0][0].set_facecolor('white')

ax[0][1].hist(train_df[train_df.category == 2].Headline.str.len(),bins = 20,color='lightst
ax[0][1].set_title('Discuss Stances', fontweight='bold')
ax[0][1].set_xlabel('Number of words',fontsize=11)
ax[0][1].set_facecolor('white')

ax[1][0].hist(train_df[train_df.category == 0].Headline.str.len(),bins = 20,color='lightst
ax[1][0].set_title('Agree Stances', fontweight='bold')
ax[1][0].set_xlabel('Number of words',fontsize=11)
ax[1][0].set_facecolor('white')

ax[1][1].hist(train_df[train_df.category == 1].Headline.str.len(),bins = 20,color='lightst
ax[1][1].set_title('Disagree Stances',fontweight='bold')
ax[1][1].set_xlabel('Number of words', fontsize=11)
ax[1][1].set_facecolor('white')

fig.suptitle('Length of "Headlines"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.1, hspace=0.3, top=0.9)

fig.savefig('Images/headlines_length.png')
plt.show()
```

```
# To use visualization to compare the length of texts in article bodies
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

ax[0][0].hist(train_df[train_df.category == 3].articleBody.str.len(),bins = 20,color='stee
ax[0][0].set_title('Unrelated Stances', fontweight='bold')
ax[0][0].set_xlabel('Number of words',fontsize=11)
ax[0][0].set_facecolor('white')

ax[0][1].hist(train_df[train_df.category == 2].articleBody.str.len(),bins = 20,color='stee
ax[0][1].set_title('Discuss Stances', fontweight='bold')
ax[0][1].set_xlabel('Number of words',fontsize=11)
ax[0][1].set_facecolor('white')
```

```
ax[1][0].hist(train_df[train_df.category == 0].articleBody.str.len(),bins = 20,color='stee
ax[1][0].set_title('Agree Stances', fontweight='bold')
ax[1][0].set_xlabel('Number of words',fontsize=11)
ax[1][0].set_facecolor('white')

ax[1][1].hist(train_df[train_df.category == 1].articleBody.str.len(),bins = 20,color='stee
ax[1][1].set_title('Disagree Stances',fontweight='bold')
ax[1][1].set_xlabel('Number of words', fontsize=11)
ax[1][1].set_facecolor('white')

fig.suptitle('Length of "Articte_body"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.1, hspace=0.3, top=0.9)

fig.savefig('Images/body_length.png')
plt.show()
```

```
# To use visualization on number of unique words in headlines
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

ax[0][0].hist(train_df[train_df.category == 3].Headline.str.split().map(lambda x: len(x)),
ax[0][0].set_title('Unrelated Stances', fontweight='bold')
ax[0][0].set_xlabel('Number of words',fontsize=11)
ax[0][0].set_facecolor('white')

ax[0][1].hist(train_df[train_df.category == 2].Headline.str.split().map(lambda x: len(x)),
ax[0][1].set_title('Discuss Stances', fontweight='bold')
ax[0][1].set_xlabel('Number of words',fontsize=11)
ax[0][1].set_facecolor('white')

ax[1][0].hist(train_df[train_df.category == 0].Headline.str.split().map(lambda x: len(x)),
ax[1][0].set_title('Agree Stances', fontweight='bold')
ax[1][0].set_xlabel('Number of words',fontsize=11)
ax[1][0].set_facecolor('white')

ax[1][1].hist(train_df[train_df.category == 1].Headline.str.split().map(lambda x: len(x)),
ax[1][1].set_title('Disagree Stances',fontweight='bold')
ax[1][1].set_xlabel('Number of words', fontsize=11)
ax[1][1].set_facecolor('white')

fig.suptitle('Number of words in "Headlines"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.1, hspace=0.3, top=0.9)

fig.savefig('Images/headlines_no_words.png')
plt.show()
```

```
# To use visualization on number of unoque words in article bodies
fig,ax= plt.subplots(nrows=2, ncols=2 , figsize=(13, 9), dpi=100)
plt.tight_layout()

ax[0][0].hist(train_df[train_df.category == 3].articleBody.str.split().map(lambda x: len(x
ax[0][0].set_title('Unrelated Stances', fontweight='bold')
ax[0][0].set_xlabel('Number of words',fontsize=11)
ax[0][0].set_facecolor('white')
```

```
ax[0][0].set_facecolor('white')

ax[0][1].hist(train_df[train_df.category == 2].articleBody.str.split().map(lambda x: len(x
ax[0][1].set_title('Discuss Stances', fontweight='bold')
ax[0][1].set_xlabel('Number of words',fontsize=11)
ax[0][1].set_facecolor('white')

ax[1][0].hist(train_df[train_df.category == 0].articleBody.str.split().map(lambda x: len(x
ax[1][0].set_title('Agree Stances', fontweight='bold')
ax[1][0].set_xlabel('Number of words',fontsize=11)
ax[1][0].set_facecolor('white')

ax[1][1].hist(train_df[train_df.category == 1].articleBody.str.split().map(lambda x: len(x
ax[1][1].set_title('Disagree Stances',fontweight='bold')
ax[1][1].set_xlabel('Number of words', fontsize=11)
ax[1][1].set_facecolor('white')

fig.suptitle('Number of words in "Articte_body"',fontweight='bold',fontsize=15)
plt.subplots_adjust(wspace=0.1, hspace=0.3, top=0.9)

fig.savefig('Images/body_no_words.png')
plt.show()
```

## Data preperation

In this section, data will be cleaned and tokenized for further modelling.

```
# To keep the first data unchanged
df = train_df.copy()
print('Dataframe has been copied')
```

```
# To create a new numeric column for the target variable(stance)
df['Stance'] = pd.Categorical(df.Stance)
df['category'] = df.Stance.cat.codes
cat_dic = dict(zip(df.Stance.cat.codes, df.Stance))
print('Data stance have been change to categorical data based on dictionary as:\n',cat_dic
```

```
# To remove punctuations, stop words, lowercase texts and lemmatize it and create four new
df['tokenized_headline'] = df['Headline'].apply(lambda x: clean_text(x))
df['tokenized_body'] = df['articleBody'].apply(lambda x: clean_text(x))
df['clean_headline'] = df['Headline'].apply(lambda x: no_char(x))
df['clean_body'] = df['articleBody'].apply(lambda x: no_char(x))
print('New columns with clean and tokenized data have been created')
```

```
# To check the dataset
df.drop(['Headline','articleBody', 'Stance'], axis=1, inplace=True)
df = df[['clean_headline','clean_body','tokenized_headline','tokenized_body','category']]
df.head()
```

```
# To find the length of tokenized headline and tokenized body
```

```
# To count the length of tokenized_headline and tokenized_body
title_len = list(df.tokenized_headline.apply(lambda x : len(x)))
body_len = list(df.tokenized_body.apply(lambda x : len(x)))
print('"Clean headline length"')
print(' Minimun:', int(np.min(title_len)) )
print(' Mode:', int(statistics.mode(title_len)) )
print(' 99 percentile:', int(np.percentile(title_len, 99)) )
print(' Maximum:', int(np.max(title_len)) )
print('\n', '-'*15, '\n')
print('"Clean Article body length"')
print(' Minimun:', int(np.min(body_len)) )
print(' Mode:', int(statistics.mode(body_len)) )
print(' 99 percentile:', int(np.percentile(body_len, 99)) )
print(' Maximum:', int(np.max(body_len)) )
```

## Doc2vec model

For both title and news body, word to vector model will be applied to convert them into vectors.

```
# To combine two data and prepare it for the model
data = [ *list(df.tokenized_headline) , *list(df.tokenized_body) ]
tagged_data= [TaggedDocument(doc, [i]) for i, doc in enumerate(data)]
print('Data is ready to create doc2vec model')
```

```
# To Train two doc2vec models for data in headlines and article body (this might takes som
d2v_head = Doc2Vec(vector_size=20, min_count=1, epochs=10)
d2v_head.build_vocab(tagged_data)
d2v_head.train(tagged_data, total_examples=d2v_head.corpus_count, epochs=d2v_head.epochs)

d2v = Doc2Vec(vector_size=150, min_count=1, epochs=10)
d2v.build_vocab(tagged_data)
d2v.train(tagged_data, total_examples=d2v.corpus_count, epochs=d2v.epochs)

print('Two Doc2Vec models for headlines and article bodies trained')
```

```
# Create vectors for titles and article body (this might takes some times)
df['headline_vec'] = df.tokenized_headline.apply(lambda x: d2v_head.infer_vector(x) )
df['body_vec'] = df.tokenized_body.apply(lambda x: d2v.infer_vector(x) )

print('Both titles and article body converted to document to vector format')
```

```
# To check the dataset
df = df[['clean_headline','clean_body','tokenized_headline','tokenized_body','headline_vec
df.head()
```

## Polarity, Subjectivity, Refutation and Overlapping

Other four characteristics of headline and body that will be used in classification.
The overlap function and the list of refuting words has been extracted from the baseline model.

```
# To get the polarity and subjectivity of headlines and bodies by using textblob (this mig
df['headline_polarity'] = df.clean_headline.apply(lambda x: [TextBlob(x).sentiment.polarit
df['body_polarity'] = df.clean_body.apply(lambda x: [TextBlob(x).sentiment.polarity])
df['headline_subjectivity'] = df.clean_headline.apply(lambda x: [TextBlob(x).sentiment.sub
df['body_subjectivity'] = df.clean_body.apply(lambda x: [TextBlob(x).sentiment.subjectivit
print('Columns for polarity and subjectivity for both headline and article body added to t
```

```
# To get the refution score from headlines and article bodies(this might takes some times)
refuting_words = ['fake','fraud','hoax','false','deny','denies','refute','not','despite','
df['headline_refutation_score'] = df.clean_headline.apply(lambda x: [refutation_score(x)])
df['body_refutation_score'] = df.clean_body.apply(lambda x: [refutation_score(x)])
print('Columns for refutation score of headline and article body added to the dataftame')
```

```
# To get the amount of overlapping between each headline and article body(this might takes
df['overlapping_score'] = df.apply(lambda x: overlapping(x.tokenized_headline , x.tokenize
print('Column for overlapping scores added to the dataftame')
```

## Finalizing vectors

Now is time to combine all feature vectors and make it ready for the neural model.

```
# To create one vector for each pair of headline and body articles
df['vector']= df.apply(lambda x: [*x.headline_vec, *x.headline_polarity, *x.headline_subje
print('vector column created')
```

```
# To check the dataset
df = df[['clean_headline','clean_body','tokenized_headline','tokenized_body','headline_vec
df.head()
```

```
# To check vector column
vector_len = list(df.vector.apply(lambda x : len(x)))
print('The final vectors size is:', set(vector_len))
```

## Data pre_processing

```
# Splitting training data frame into two parts
x= np.array(list(df.vector))
y= to_categorical(np.array(df.category))

train_x , val_x ,train_y , val_y = train_test_split( x , y , test_size = 0.2)
print('Data has been split into training and validation parts')
```

```
# To create desirable format for the test dataframe(this will takes some times)
print('Preparing headline of test dataset')
test_df['tokenized_headline'] = test_df['Headline'].apply(lambda x: clean_text(x))
test_df['clean_headline'] = test_df['Headline'].apply(lambda x: no_char(x))
test_df['headline_vec'] = test_df.tokenized_headline.apply(lambda x: d2v_head.infer_vector
test_df['headline_polarity'] = test_df.clean_headline.apply(lambda x: [TextBlob(x).sentime
test_df['headline_subjectivity'] = test_df.clean_headline.apply(lambda x: [TextBlob(x).sen
test_df['headline_refutation_score'] = test_df.clean_headline.apply(lambda x: [refutation_

print('Preparing article body of test dataset')
test_df['tokenized_body'] = test_df['articleBody'].apply(lambda x: clean_text(x))
test_df['clean_body'] = test_df['articleBody'].apply(lambda x: no_char(x))
test_df['body_vec'] = test_df.tokenized_body.apply(lambda x: d2v.infer_vector(x) )
test_df['body_polarity'] = test_df.clean_body.apply(lambda x: [TextBlob(x).sentiment.polar
test_df['body_subjectivity'] = test_df.clean_body.apply(lambda x: [TextBlob(x).sentiment.s
test_df['body_refutation_score'] = test_df.clean_body.apply(lambda x: [refutation_score(x)

print('Creating combination of headline and body vectors')
test_df['overlapping_score'] = test_df.apply(lambda x: overlapping(x.tokenized_headline ,
test_df['vector']= test_df.apply(lambda x: [*x.headline_vec, *x.headline_polarity, *x.head

print('Creating test_x from test dataset')
test_x = np.array(list(test_df.vector))

print('Test dataset is ready to use as "test_x"')
```

```
# To check the test dataset
test_df = test_df[['clean_headline','clean_body','tokenized_headline','tokenized_body','he
test_df.head()
```

```
# To check vector column of test set
test_vector_len = list(test_df.vector.apply(lambda x : len(x)))
print('The final test_set vectors size is:',set(test_vector_len))
```

## Classification models

Now, it is time to use the vector and create two models, a multominal logistic regression and a sequential neural networks, for stance detection.

### Multinominal logistic regression

```
# To create a logistic regression classification model
lr_model = LogisticRegression(class_weight= 'balanced' , multi_class='multinomial')
lr_model.fit(train_x , np.argmax(train_y,axis=1) )
```

## To check the model performance in validation set

```
# To apply model on validation set
lr_pred= lr_model.predict(val_x)
lr_report = classification_report(np.argmax(val_y,axis=1), lr_pred, target_names = ['agree
print(lr_report)
```

```
# To graph the confusion matrix
lr_cm=confusion_matrix(np.argmax(val_y,axis=1), lr_pred)
lr_cm = pd.DataFrame(lr_cm)
lr_cm.index.name = 'Actual'
lr_cm.columns.name = 'Predicted'

fig, axis= plt.subplots(figsize=(10, 8))
plt.tight_layout()

ax_label = ['agree','disagree','discuss','unrelated']
sns.heatmap(lr_cm ,cmap= "Blues",annot = True, fmt='', xticklabels=ax_label, yticklabels=a
sns.set(font_scale=1.2)

fig.savefig('Images/lr_confusion_matrix.png')
plt.show()
```

## Creating .csv data to be evaluate in codaLab

```
lr_test_pred = lr_model.predict(test_x)
result= [cat_dic[int(x)] for x in lr_test_pred]
print(' Count of each class in result is as below:', '\n', { x : result.count(x) for x in
```

```
# give a csv file with answers(this code will overright existing folder!)
answers = test_stances_unlabeled.copy()
answers['Stance'] = result
answers.to_csv('data/answer.csv', index=False, encoding='utf-8')
print('"answers.csv" saved in data folder of the repository')
```

## Neural network model

```
# Defining Sequentional model
model = Sequential(name='Neural_Model')
model.add(Dense(256, activation= 'relu', input_dim=177 ))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(32,activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(Dropout(0.2))
model.add(Dense(4, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.SGD(lr=1e-6, decay= 0.01, momentum=0.99 , nest

model.summary()
```

```
# To plot the Neural Model
plot_model(model, to_file='Images/model_plot.png', show_shapes=True, show_layer_names=Fals
```

```
# To compute weights for the model based on the training set weights
train_y_int= [y.argmax() for y in train_y]
compute_weights = class_weight.compute_class_weight('balanced',np.unique(train_y_int),trai
weights = dict(enumerate(compute_weights))
# Cat: {0: 'agree', 1: 'disagree', 2: 'discuss', 3: 'unrelated'}
print(' The computed weights of the model will be set as:', '\n' , weights)
```

```
The computed weights of the model will be set as:
{0: 3.4180061559507524, 1: 15.006381381381381, 2: 1.3958449720670392, 3: 0.34195264
```

```
# To set weights and fit the model (this takes some times)
history = model.fit(train_x, train_y, batch_size=32, epochs=50, validation_data=(val_x, va
print('\n','*'*20,'Model trained','*'*20)
```

```
# To save sequentioal model
model.save('data/sequential_model')
print('Sequential model has been saved')
```

```
# Plotting the accuracy and loss of the training and validation sets during epochs
fig, (ax1,ax2)= plt.subplots(ncols=2, figsize=(10, 5), dpi=100)

epochs = [i for i in range(50)]
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax1.plot(epochs , train_acc , 'co-' , label = 'Training Accuracy')
ax1.plot(epochs , val_acc , 'yo-' , label = 'Validation Accuracy')
ax1.set_title('Training & Validation accuracy per epochs',fontsize=12, fontweight='bold')
ax1.legend()
ax1.set_xlabel("Epochs")
ax1.set_ylabel("accuracy")
ax1.set_facecolor('white')
ax1.spines['bottom'].set_color('0.5')
ax1.spines['top'].set_color('0.5')
ax1.spines['right'].set_color('0.5')
ax1.spines['left'].set_color('0.5')

ax2.plot(epochs , train_loss , 'co-' , label = 'Training Loss')
ax2.plot(epochs , val_loss , 'yo-' , label = 'Validation Loss')
```

```
ax2.set_title('Training & Validation Loss per epochs', fontsize=12, fontweight='bold')
ax2.legend( fontsize= 13)
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Loss")
ax2.set_facecolor('white')
ax2.spines['bottom'].set_color('0.5')
ax2.spines['top'].set_color('0.5')
ax2.spines['right'].set_color('0.5')
ax2.spines['left'].set_color('0.5')

fig.suptitle('Fitting model history per epoches',fontweight='bold')

fig.savefig('Images/learning_loss.png')
plt.show()
```

## Checking model performace in validation set

```
# To apply model on validation set
pred_prop =model.predict(val_x)
pred_class=np.around(pred_prop , decimals = 0)
report = classification_report(val_y, pred_class , target_names = ['agree','disagree','dis
print(report)
```

```
# To graph the confusion matrix of valodation set
cm=confusion_matrix(np.argmax(val_y, axis=1),np.argmax(pred_class,axis=1))
cm = pd.DataFrame(cm)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'

fig, axis= plt.subplots(figsize=(10, 8))
plt.tight_layout()

ax_label = ['agree','disagree','discuss','unrelated']
sns.heatmap(cm ,cmap= "Blues",annot = True, fmt='', xticklabels=ax_label, yticklabels=ax_l
sns.set(font_scale=1.2)

fig.savefig('Images/confusion_matrix.png')
plt.show()
```

## Creating .csv data to be evaluate in codaLab

To evaluate the model, a .csv file in a special format is needed.

```
# To apply model on test set and get results based on the "cat_dic" dictionary
pred_prop =model.predict(test_x)
pred_class = np.argmax(pred_prop, axis=-1)
result= [cat_dic[int(x)] for x in pred_class]
print(' Count of each class in result is as below:', '\n', { x : result.count(x) for x in
```

```
# give a csv file with answers(this code will overright existing folder!)
answers = test_stances_unlabeled.copy()
answers['Stance'] = result
answers.to_csv('data/answer.csv', index=False, encoding='utf-8')
print('"answers.csv" saved in data folder of the repository')
```

## Appendix 3

Default project supplementary material

1. CodaLab username:
   S2hajian
2. CodaLab submission details:
   weightedScore = 5004.0
   Date of Entry = 08/08/2020 15:13:31
   Submitter = s2hajian
   Zip file name = Shabnam_Hajian_model55_milestone2.zip