Learning to predict "stance" of news articles

COMP9417 Assignment 2

z5119297 Jiahui LANG
z5103300 Chen GUO
z5099147 Tao WAN

Introduction

There is a lot of misinformation seen each day in the news industry today, which is defined as a 'made-up store with intention to deceive'. According to a December Pew Research poll, 64% of US adults claimed that fake news made them really confused about the truth of the current news. The idea of the project is originally from the Fake News Challenge stage one called FNC-1 and this project is mainly focused on using the technologies from Artificial Intelligence, especially from Machine Learning and Natural Language Processing to solve the fake news problem **[1]**. The final well-trained model will be able to automatically determine if a story is real related to its headline or unrelated, furthermore it can detect its stance which the body text may agree, disagree, discuss or be unrelated to the headline.

Overview

Out stance detection system for FNC-1 consists of two main parts (two main stages). The first stage of our development is to identify whether the body text is relative to headline or not, and the second is to determine what the stance of the body text is against the headline, 'agree', 'discuss' or 'disagree'.

Representation and features

As we known, it has already been a time-consuming task to determine whether the article body is relative to the headline or not manually, not to speak of determining the stance of the body against the headline based on rich semantics. For NLP machine learning task, first we must find a way to transform the raw text data into numeric vectors that can be read by machine, which should also try its best to minimizes the loss of original features. Rather than the selection of different models, data pre-processing and transforming are actually the most important parts of our project.

Unfortunately, after experiment, we confirmed that the scale of provided training set (number of headlines and articles are both 1600+) is not enough to support word2vec method since word2vec itself is a deep learning procedure, and deep learning sometimes works not very well with small-scale and noisy training set like this, let alone the newest doc2vec method.

Since the rules of this challenge forbids us to do some extra data augmentation. Finally we decided to use more traditional method like TF-IDF**[2]**(Term Frequency-Inverse Document Frequency).

Stage 1

In our point of view, the task to determining whether the headline and the body is related or not is more about key words capturing than understanding the semantics. It's very likely that a headline and a body is related when they have similar composition of words. To generate the features vectors, we transformed each article into TF-IDF vector, and then executed some tricks on them. As following is a simple example:

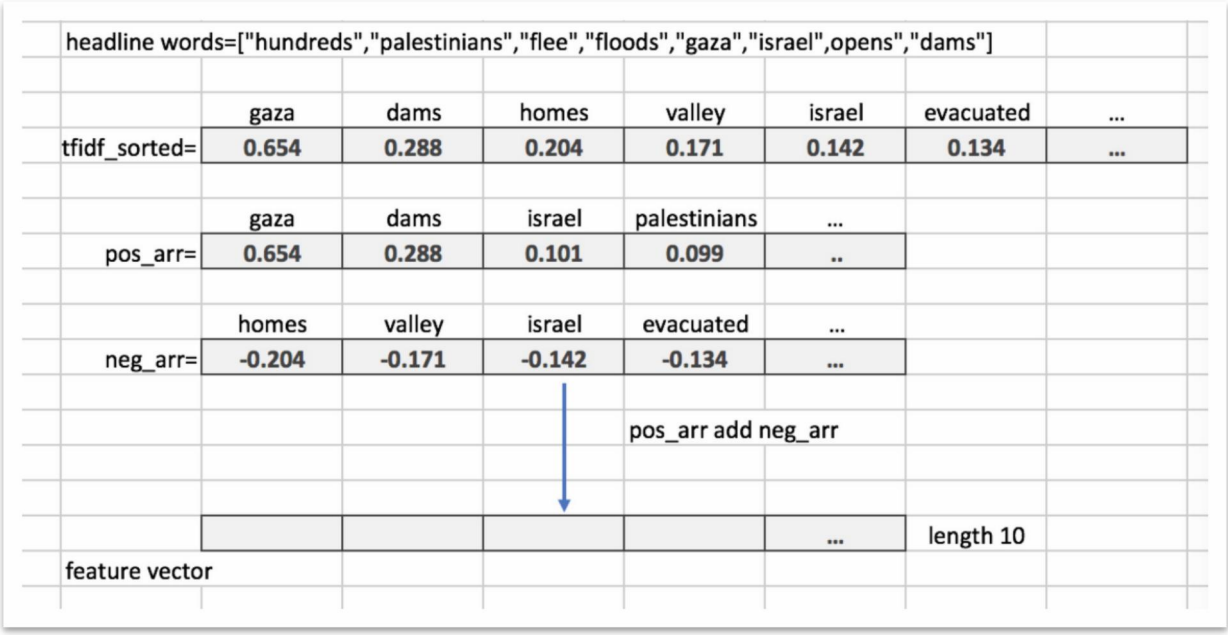

Figure 1

On this basis, we first defined two zero arrays of size 10, let's name them 'pos-arr' and 'neg-arr'. Next, for each pair of headline and body TF-IDF vector, we sorted the vector in descending order which then allowed us to find out the most representative words in the text body. After that, we pick out each word one by one from front to back, and checked if it appeared in headline text. if it did, we assigned its TF-IDF value to 'pos-arr'; otherwise we assigned its minus TF-IDF value to 'neg-arr'.

Keep doing this until we have filled both 'pos-arr' and 'neg-arr' or we have gone through all words in the body. Finally, the feature vector can be obtained applying ('pos-arr' adds 'neg-arr') and fed into the classifier.

Stage 2

We used BOW(bag of words) representations as inputs: term frequency(TF), the picture of architecture is shown in the following, and the TF vectors are extracted from headline and body text. As you can see, the inputs are combined by TF vector for Headline and TF vector for Body text. Before generating these input vectors, we need to specially pre-processed each text body for its corresponding headline.

Firstly, each text body were tokenised into separated sentences, and we computed TF-IDF for not only the headline but also these sentences. Secondly, the cosine similarity between the headline and each sentence was calculated, and the top 5 most similar sentences will be chosen to make up a new body text.

The intention of this pre-processing is that we only need representative sentences. if a sentence has a similar composition to the headline, it probably contains important message and makes greater impact on the stance of the whole article body .

For the TF vectors, we selected the most frequent 5000 words in the training set to build the vocabulary(stop words were excluded). These vectors were concatenated into a feature vector of total size 10000 and then fed into the classifier.

Architecture

Stage 1

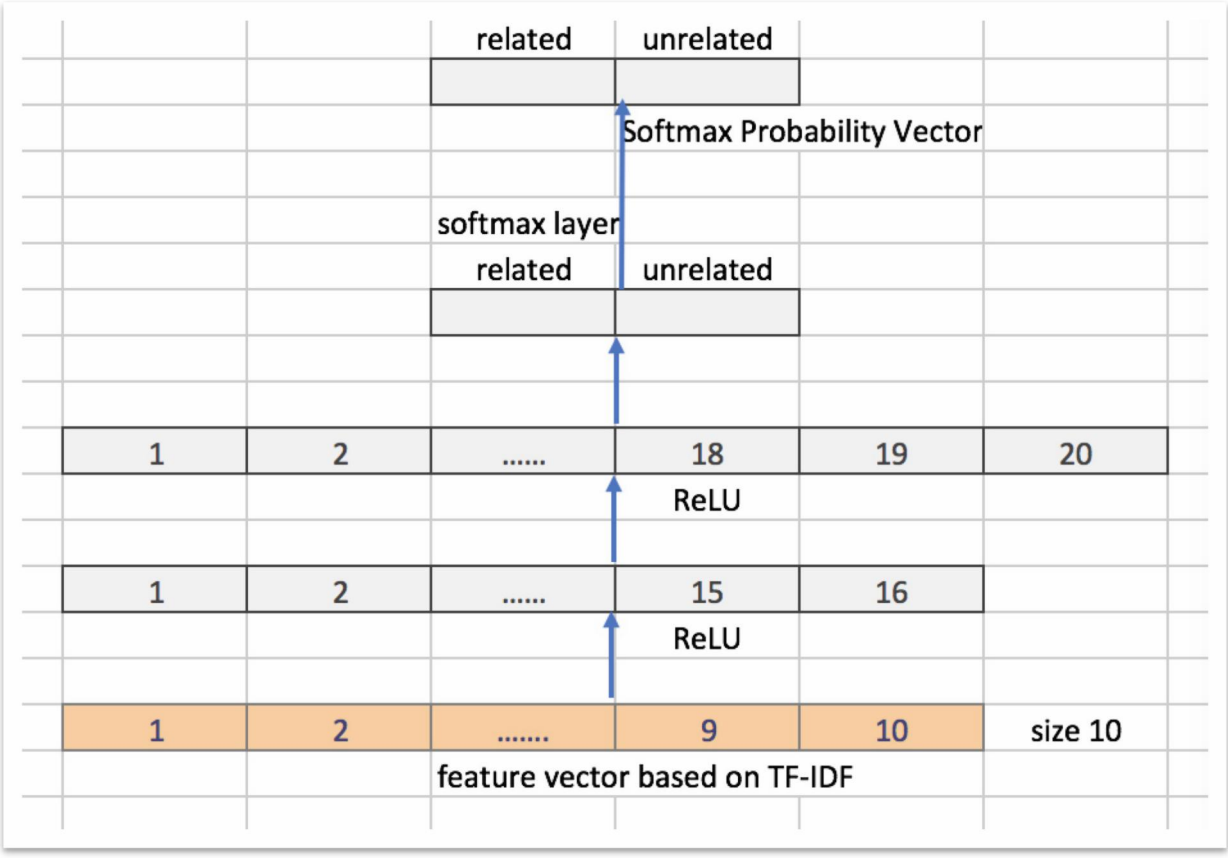

Figure 2

Stage 2

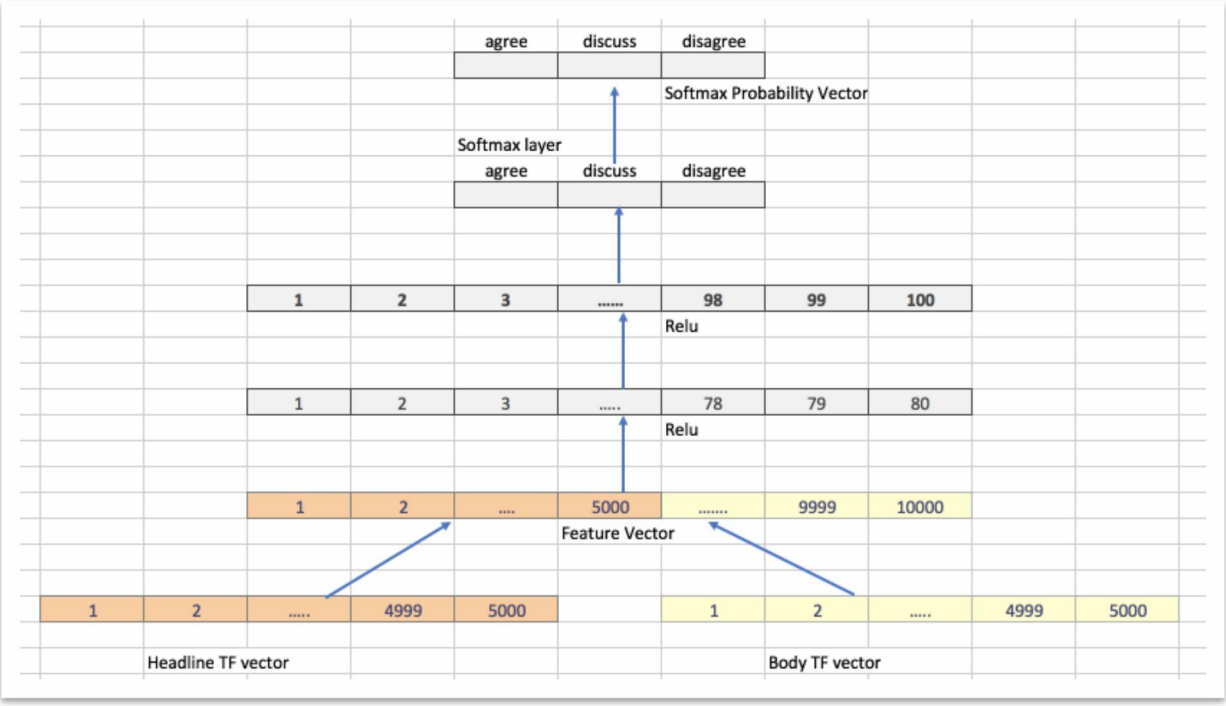

Figure 3

Classifier

Our final classifier is combined by two neural network models connected by some logical judgement, implemented with TensorFlow, NumPy and Pandas. The code of system and instructions including introduction and README file are available at public Github repository: https://github.com/WilliamPot/FakeNews.

Stage 1

The model is a multi-layer neural network model with two hidden layers, one is 16 units and the other is 20 units, both of them have 50% dropout rate, lastly a 2-unit Softmax layer generates the output of the final linear layer. ReLU(rectified linear unit) is selected as activation function. This model determines the relativeness between headline and body. We also had a try on SVM classifier, but the performance of NN was better.

Stage 2

This model is also a multi-layer neural network model with two hidden layers, one is 80 units and the other is 100 units, both of them have 50% dropout rate, after them is a linear layer with a 3-unit Softmax that generates the final output behind. The activation function we used here is ReLU. This model determines the stance distinction between headline and body.

Training and Predicting

In training, we trained these 2 models separately. Stage 1 model will easily get a high accuracy at early epochs, but we need more training to get a optimal result. On contract, stage 2 model just needs fewer epochs to get a lift in accuracy, but it won't get better by extra training time extension. After training, we combined them together for predicting work.

In predicting, for each testing data we generated both versions of feature vectors for stage 1 and stage 2, and saved them into csv files, then restore them from csv files and shuffle during training and predicting procedure. The reason to do this is that our machine didn't allow us to store the whole large number of giant-dimension(id. 10000 columns) arrays directly into memory space as it will probably exert harmful impact on efficiency.

After a stage 1 version vector of a data point is fed into first model, If it classifies this point as 'unrelated', then just label it with 'unrelated'; otherwise the stage 2 version feature vector of this data point should be fed into next model, and it will select a result from {'agree', 'disagree','discuss'} to label this data point.

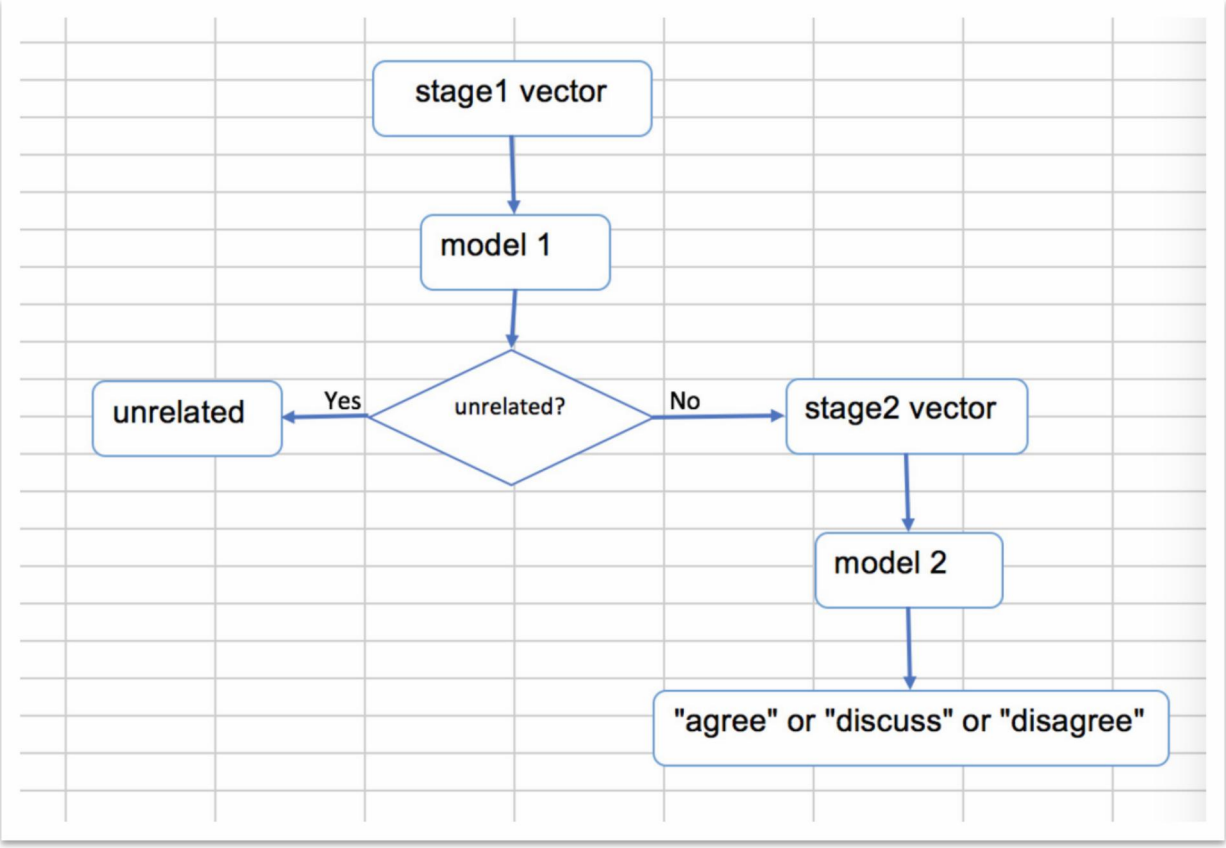

Figure 4

In fact, in real procedure we implement batch prediction which can be faster than predicting test data one by one. The description above is just the general principle.

Hyper Parameters

Table1: Details on Hyper parameters of the Model

Stage	Parameter	Description	Range	Optimised
Stage 1	learning_rate	Learning rate	[0.001, 0.1]	0.01
		Vocabulary size	default	default
	batch_size	Batch size	[32, 64]	32
	k	#Epochs	[2000, 10000]	10000
Stage 2	learning_rate	Learning rate	[0.001, 0.1]	0.1
		Vocabulary size	[4000, 10000]	5000
	batch_size	Batch size	[70, 600]	500
	k	#Epochs	[80, 300]	200

Results

Before using provided 'scorer.py' to test the performance of our model, we have tested it using 'competition_test_stances.csv' in 'fnc-1' folder. At stage 1, we got a pretty good testing accuracy at about 94%, but we only got about 68% in stage 2. For future improvement we should solve the problem caused by new strange words in test data.

The table below is the final result from provided file 'scorer.py'.

Table2: The result of our task

	agree	disagree	discuss	unrelated
agree	797	41	758	307
disagree	232	11	222	232
discuss	568	48	3031	817
unrelated	87	12	297	17953

Accuracy: 85.8%

Related Score: 75.5

Discussion

After we got our related score, we went to FakeNewsChallenge website and viewed the top 3 winners' works. They all got a score at about 82 which was far greater than ours. But one of them used GoogleNews pre-trained model to do word2vec on training text data, which seems like a covert training set augmentation, this is just my personal view. The other two really have done a splendid job, from their idea and code we still have much more to learn like building a decision system based on multi-model voting method.

The main problems we met this time were three below:

1. Struggling with words that didn' t appear in training set. The second half of testing set contains large number of new words, which resulted in the poor performance of stage 2 model. Methods like TF-IDF and word2vec turned to be almost invalid since we cannot use testing set for training or inference building. We still have no idea how to deal with it.
2. Word2vec is somewhat easy, but doc2vec is hard. Current methods only allow us to keep words frequency features, but it cannot capture the meaning of a certain sequence of words, more precisely the semantics. Gensim.doc2vec could be an option, but after experiment we believe that it still has much to improve for small-scale training set.
3. Hardware limitation. Our laptops cannot afford the workload when 3 or more models are running at the same time, also dealing with giant-dimension data is almost impossible, this could be solved after we purchase new equipments.

To summary, taking everything into consideration, this result just hit the baseline, we can do better next time with the experience of this group project.

Members roles

z5119297 Jiahui Lang

- resources gathering, partial implementation and advice

z5099147 Tao Wan

- partial implementation and advice

z5103300 Chen Guo

- plan establishment and full implementation

References

[1] FakeNewsChallenge.org. (2018). Fake News Challenge. [online] Available at: www.fakenewschallenge.org [Accessed 31 May 2018].
[2] Term Frequency-Inverse Document Frequency statistics.(2016) [online] Available at: https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF [Accessed May 2018].
[3] Gensim.models.tfidfmodel [online] Available at: https://radimrehurek.com/gensim/models/tfidfmodel [Accessed May 2018].
[4] Tensorflow. [online] Available at: https://www.tensorflow.org/ [Accessed May 2018].

[5] Dropout(neural networks). [online] Available at: https://en.wikipedia.org/wiki/Dropout_(neural_networks) [Accessed May 2018].

Comment

There are two main ways to access our full implementation code:

1. Recommended and simple, download from google drive and follow the instruction from README.md in our submission file. Google drive link: https://drive.google.com/open?id=120uxcc7Ae5E9wPKjhErepMRU_INdUWVJ;
2. Not recommended, the full implementation can also be accessed by our GitHub repository: https://github.com/WilliamPot/FakeNews.git .