Climate Change Misinformation Detection

Minh Trung Nguyen
1016151 / Master of Data Science / The University of Melbourne
trnguyen@student.unimelb.edu.au

Abstract

Many Natural Language Processing related techniques have been developed to enable the computer machine to understand and extract useful information from raw texts. In this paper, I present my machine learning approaches to detect misinformation about climate change. I performed several text pre-processing steps and then apply both supervised and unsupervised learning methods to classify the problem.

1 Introduction

In today's era of 4.0 technology, web blog and social media platform have emerged and become an indispensable part in our everyday life. A piece of information can be disseminated easily to the world by anyone. As a consequence, unverified and inaccurate news have increased tremendously which might cause lots of confusions and even fear. For example, during the ongoing event of the Corona virus, fake news have lead the world to a global panic while people were rushing to buy toilet papers. These inaccurate information can even come from influencers and celebrities, making people to believe in its truthfulness. In this paper, I discuss about a similar problem on the climate change aspect. Fake news are published to disseminate fear about the global warming event as well as discrediting the science behind climate change. As a result, it is very important to develop a system which can detect the misinformation articles among the others on the web. In this report, I describe the pipeline system that I design to tackle the classification task. The report consists of several sections:

- **Text Pre-processing**: Given a set of articles as raw text, we need to prepare and transform the text to suitable format before having the machine to learn and discover useful patterns from it.
- **Learning methods**: Here I illustrate different machine learning models used to learn the pre-processed data.
- **Error Analysis**: Discover any weak points in the model that lower its performance and suggestion to enhance.
- **Evaluation and Comparison**: Evaluate and compare the performance of several models used on the focused F-score metric. Choosing the best model.
- **Conclusion**: Summarize the experimental results and propose future improvement.

2 Text Pre-processing

We need to perform several cleaning steps before we can feed the article text to the models. At first, I used the simple split() function in python which is equivalent to a tokenizer using space as a separator. Later, I realised that if using this approach, the system will not be able to separating punctuation from the word (e.g.'climate,'). Therefore, I decided to use the word tokenize function in the NLTK package that is able to split the punctuation as a single token. Now I am able to further filter all the stop words and punctuation from the list of extracted tokens. Before filtering out the stop words, the system has to lower the case of the token since the 'stopwords' list in NLTK package only contains the lowercase version. I also consider to stem and lemmatize the tokens.

Now after filtering out all the function words and punctuation, I decided to remove the URLs using regular expression matching. A token which has only a single alphabetic letter will also be removed. Numbers or word phrases which contain numeric value are kept in the list (i.e. probably fake news are more likely to contain numbers or measures

since they are hard to verify).

These are the main pre-processing steps applied to the raw texts prior to learning. Each learning model will need to perform a final step to transform the pre-processed tokens into useful features that can be appropriately feed to the model (e.g. vectorizing). Since this step depends on the model used, I will describe it in detail while explaining each model in the following section.

3 Learning methods

In this section, I illustrate the various models that I used to learn the pre-processed text data.

3.1 One-class classification

Since the data only includes positive (fake news) labels, I will firstly try to implement a semi-supervised model. It is called a semi-supervised because we only use the available positive label as supervision during training. Using the trained model, we can then classify a new data item to negative label if it looks different from the positive example. This is similar to anomaly detection problem. The model learnt the features of normal examples and later detect anomalies which do not look 'normal'.

I utilized the powerful scikit-learn package in python to develop this method. The package contains 'OneClassSVM' and 'Isolation Forest' that is created for the purpose of one class classification problem. As I mentioned in the previous section, we have to perform a final transformation step which converts the list of pre-processed tokens to a useful features representation for use in our models. The method is TF-IDF representation. This is also done by scikit-learn using TfidfVectorizer function.

3.2 Supervised approaches

Usually, the anomaly detection method is more useful when the class labels are unbalanced. For example, in banking industry, outliers detection method can be applied to detect fraud transactions which occur very infrequently. Within thousands of transactions, only a few can be fraud. But in our problem, fake news can appear as many as validated news. Therefore, I think it would be better if we collect more data and expand the training set to also have negative examples and then conduct a supervised learning approach.

Web crawling: In order to get additional data, I develop a python script with the help of the Beautiful Soup package. Initially, I went to the Internet and look for the high credited news website. I then search for topics about climate change which leads me to a page consists of all the articles talking about climate change. The Beautiful Soup then allowed me to extract the text part in each article. As a result, I managed to collect 883 news articles from `https://www.theguardian.com/au` and `https://climate.nasa.gov/` and assign to them a '0' label representing for negative or real news. Combining to the given positive examples, we now have a corpus of 2051 articles.

Successfully expanded the training dataset, we now apply the vectorizer using scikit-learn to the whole corpus again and ready for the interesting part, model learning.

3.2.1 Machine Learning model

Scikit-learn provides a wide range of different machine learning models for supervised tasks. The three popular models that I used to train are:

- Naive Bayes
- Logistic Regression
- Random Forest

Naive Bayes is a very simple model which offers a fast training time but it depends on the independence assumption of the data which might not be entirely true in most cases. Logistic regression model doesn't need the independence assumption and it also has more options for the hyperparameters such as the regularization term C and different solvers to choose for the optimization process. Random forest is an ensemble method which can deal with non-linear problem and the result can sometime be interpreted to better understand the patterns.

3.2.2 Deep Learning model

Deep Neural Network architecture has proved to be state-of-the-art technique for various NLP tasks. For this reason, I experimented building a deep network architecture to address the task using the Keras package.

For neural network, we will need a different

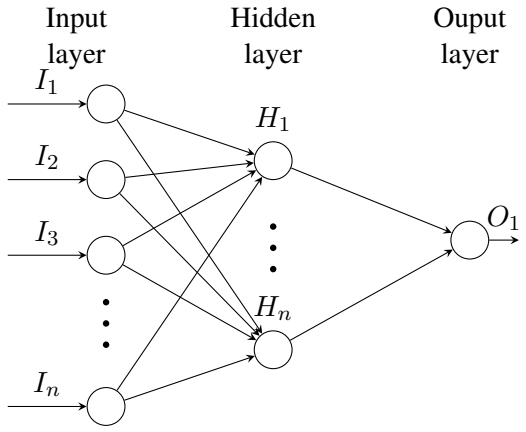

Figure 1: Feed forward network architecture

representations for the articles, called word embedding. An embedding represents a word as a D-dimensional vector in space. It can capture the similarity between words in a corpus rather than pure count like in TF-IDF vectorizer. Another advantage of using word embed is its ability to represent the document as a dense vector rather than sparse vector as in TF-IDF, which reduces the training time on neural network tremendously. Moreover, we can also put an embedding layer to the network and let it learns the embeddings itself during training. Since we don't have much data to create word embeddings ourselves, we will take the pre-trained word embed model that has been trained on a huge dataset. The first model is called word2vec which contains 300-dimensional embeddings for 3 million words over the Google News dataset. The second one is GloVe vectors developed by a group of researchers at the University of Stanford.

Figure 1 illustrate the architecture of the neural net model. The input layer takes the N-dimensional vectors as input. It computes the weighted sum of the inputs which is then being applied to a RELU activation function and stored in the hidden layer nodes. The hidden layer has 100 nodes which are weighted and put through a sigmoid function to give the final prediction output.

I create the input vectors by utilizing the pre-trained GloVe model in the Spacy package. The model is trained on a Common Crawl corpus and represents each word by a 300 dimensional vector. It can calculate the mean vector for each article.

I also tried another pre-trained model which is trained on the GoogleNews dataset. For this model, we have to use the Gensim package. This model doesn't have a method to directly compute the mean vector for the entire article, therefore we have to manually get the word embed of each token and then compute the mean using Numpy.

The neural network will then be trained to minimize the binary cross-entropy loss using the Adam optimizer in Keras. The training is repeated for 20 epochs and on a batch size of 8.

3.2.3 Embedding layer

The above method used entirely the pre-trained model to map tokens in our corpus to word embeds. But we can use the pre-trained embedding matrix only as a baseline by having an embedding layer as the input layer and allow it to tune the weights when training on our own corpus.

I first load the pre-trained embed to an embedding index dictionary where the keys are the words in GoogleNews corpus and the values are their 300 dimensional vectors. Then, using the tokenizer in Keras, the vocab for our corpus is created. The system iterates through each word in our vocab and if the word is also in the GoogleNews corpus, we get the corresponding vector and add it to the embedding matrix that we will train later on. The embedding matrix has a shape of (vocab size, embedding dim). The input to the model is now a list of indices where each index represents a word token in the article. For example, the sentence "I am good" is represented as [3, 55, 2] where "I" is mapped to 3 in our corpus. Since each article has a different length, we need to perform padding to ensure all the inputs can have a fixed length.

Finally, we now can run the model which has a similar structure as before with 1 hidden layer using RELU activator. Since deep neural network is prone to overfit, we also add a Dropout node which exclude 10% of nodes when training.

4 Evaluation and Comparison

Table 1 shows the results for different machine learning models using the development dataset. The one class approach is only above average while the Isolation Forest gave very poor prediction. This is reasonable since these models are mostly

Model	Fscore	Prec	Recall
1 class SVM	66.7%	60.7%	74.0%
Iso Forest	31.4%	55.0%	22.0%
NB (stem)	84.1%	78.9%	90.0%
NB (lemma)	83.0%	78.6%	88.0%
LR	**90.7**%	**93.6**%	**88.0**%
RF (stem)	82.2%	77.2%	88.0%
RF (lemma)	79.2%	75.0%	84.0%
LR (remove URLs)	88.2%	95.3%	82.0%
LR (gloVe)	83.3%	87.0%	80.0%
LR (word2vec)	84.6%	81.5%	88.0%

Table 1: Results and comparison between different models

Model	Fscore	Prec	Recall
NFF (gloVe)	84%	84%	84%
NFF (word2vec)	80%	73%	90%
Emb layer (gloVe)	82%	82%	82%
Emb layer (word2vec)	82%	72%	94%

Table 2: Results for deep learning models

used for an outlier detection problem where the number of unusual examples is very small. In other word, it can be used when the data is unbalanced. But in our problem, the number of misclassified and verified article are equal (50:50), therefore the one class model will not perform well.

Turning to supervised approaches with additional data, the stemming tokenizer gives slightly better results for both Naive Bayes (NB) and Random Forest (RF) models. Meanwhile, Logistic Regression (LR) model gives the best performance overall. Stemming and Lemmatization give the same results for LR model. The updated tokenizer which removes URLs token does not give better result.

For the word embedding representation, gloVe vector performs better on precision while word2vec gives better result on recall. But both of them still can not beat TF-IDF representation.

Table 2 shows results for deep learning methods. Overall, we can see that Logistic Regression model with hyperparameters (C=10000, solver=liblinear) with lowercasing and stopword removal give the best result. When predicting the test set on CodaLab, OneClass methods give around 49%, other supervised methods give around 64% and the best LR model gives 71%.

5 Error analysis

In this section, I describe the error analysis over the development dataset to discover any potential improvement for the system.

Looking at the wrongly classified examples, the best model mostly predicts some fake news as real news, thus reducing the precision metric. I have seen that the misclassified articles often include question mark or exclamation tokens. Therefore, I consider keeping the three symbols ('?', '!', '"') in our TF-IDF to see if it can give a better result. The performance after allowing these symbols actually increases a little bit (recall from 0.88 to 0.90 and precision from 0.936 to 0.9375).

Moreover, there is a misclassified positive example (5th document) which contains a phrase with all uppercase letters ("BUT NOT IN THEIR FAVOUR!"). Therefore, I created some hand-made features like "uppercase_percentage" and "number of question marks" and then combine them with the TF-IDF sparse matrix using hstack. But the result was not improved. I also try sentiment score for each article (maybe fake new is more subjective and express more emotion to spread fears while real news is more about fact). But again, it turns out that the performance is worse.

6 Conclusion

Overall, we take the Logistic Regression on TF-IDF features as our best model and get an F-score of 92% after allowing question mark and exclamation mark tokens. In the future, we can improve the model by collecting data from social media and do fact-checking to label. Since news from verified website may contain a lot of scientific information which might cause the system to incorrectly classify real news in the test set as fake news, therefore reducing the precision metric. Moreover, we can train our own word embed matrix if we have more data so that it can capture numeric information that may distinguish between fake and real news.