News Stance Detection

By: Bagus Maulana

For this course project, I developed a Python script capable of performing the stance detection task: given two pieces of text, estimate their relative perspective ('stance'), or whether the two pieces of text agree, disagree, or just discuss the same topic, or are unrelated.

Stance detection has various potential uses, from search engines (matching queries to documents) to recommending similar articles. For this project, we will focus on the specific problem of 'matching' an article's headline with its text. Quite often, news articles may have an 'attractive' headline that do not necessarily match the content of its body, which may be about a different subject or have a different claim than the headline. The goal of this project is, given an article's headline and body text, predict whether that body's text agrees with the headline, disagrees with the headline, discusses the same topics/claim as the headline, or is unrelated to the headline.

The dataset for this project comes from a publicly-available challenge, Fake News Challenge or FNC-1 (https://github.com/FakeNewsChallenge/fnc-1). This dataset provides a total of 75385 articles (headline-body pairs) (split into 49972 articles in the training dataset and 25413 articles in the test dataset). Each article (or headline-body pair) is labelled with either 'unrelated', 'discuss', 'agree', or 'disagree'.

Train-Validation split

The first step required was to split the training dataset into a train subset (used to train models for feature-engineering (e.g. idf) and machine learning) and a validation subset (used to optimise hyper-parameters for these models independently of the test set). The training dataset was split 9:1 (90% training subset, 10% validation subset).

While doing this train-validation split, I had to ensure that the training and validation subsets have similar, proportional ratios for the four labels. This was achieved by first splitting the training dataset to four arrays for each label, splitting each array 9:1 (to train_unrelated, val_unrelated, train_discuss, val_discuss, etc.), combining all the train subsets into one array and validation subsets into one array, then shuffling them.

After this split, the number of articles, for each label, on the training and validation subsets are:

	Unrelated	Discuss	Agree	Disagree	All
All	36545	8909	3678	840	49972
Train	32890	8018	3310	756	44974
Validation	3655	891	368	84	4998

Pre-processing

The next step (before calculating features) is to pre-process the raw text files. This is done via the tokenise(text) function. This converts raw text into a list of words, after stripping the text of all non-alphanumeric characters.

Additionally, common, meaningless words ('stop words') in English (such as 'the' or 'an') are removed from the returned list of words to reduce noise in the features (e.g. remove meaningless words from term-frequency vectors). This is shown to improve the accuracy of my learning model, as if I do not filter out these stop words, the accuracy of my model on the validation set decreases from 0.870 to 0.818.

Vector Representation (Cosine Similarity)

The first feature that I am using is the cosine similarity of the vector representations of the headline and the article body. This is a measure of how similar in meaning the sum of words in the headline and article body are (after excluding stop-words).

Initially, each document (headlines and article bodies) are vectorised into tf-idf form. Before this is possible, I need to learn the idf weights of every word in the collection. This is done by going through all documents and building a dictionary of (word -> in how many documents does that word appears). This is the document frequency (df) score of each word. To get the idf score of each word, its df score is inverted and smoothed with the following function:

$$idf_{word} = \ln\left(\frac{1 + N\ (no.of\ docs\ in\ collection)}{1 +\ df_{word}\ (no.of\ docs\ in\ which\ word\ occurs)}\right) + 1$$

Then, we can compute the tf-idf values of each word in a document. First, the term frequency of the word is counted (e.g. if the word 'glove' appears twice in a document d, the tf of glove is 2), then this tf value is multiplied by the idf weight of said word (e.g. if the idf of 'glove' is 1.25, then its tf-idf value in d is 2*1.25=2.5). This tf-idf representation of each document is saved as a (word->tf-idf) value dictionary (e.g. 'glove'->2.5).

$$tfidf_{word,doc} = tf_{word,doc}\ (no.of\ times\ word\ occurs\ in\ doc) \times idf_{word}$$

Then, to make these tf-idf representations comparable, I used GLoVe pre-trained word vectors (https://nlp.stanford.edu/projects/glove/) to convert these tf-idf representations to fixed-length vectors based on the learned 'meaning' of each word. GLoVe provides a mapping of six billion English words to a 50-dimensional vector, trained on Wikipedia and Gigaword. To convert a document to a GLoVe vector, the (scalar) tf-idf value of each word in the document is multiplied by the GLoVe vector associated with the word, which is summed together and normalised for document length:

$$glove_{doc} = \frac{\sum(glove_{word} \times tfidf_{word,doc})\ (for\ each\ word\ in\ the\ document)}{\sum(tfidf_{word,doc})\ (for\ each\ word\ in\ the\ document)}$$

(Note that $glove_{word}$ is a vector, while $tfidf_{word,doc}$ is a scalar value)

This GLoVe vector representation of documents is used as a feature by calculating the cosine similarity of the GLoVe vector representation of each article's headline and body. The cosine similarity of two similar-length vectors are computed with:

$$cosine\_similarity = \frac{H \cdot B}{|H||B|} = \frac{\sum_{i=0}^{len(glove)}(H_i \times B_i)}{\sqrt{\sum_{i=0}^{len(glove)}({H_i}^2)} \times \sqrt{\sum_{i=0}^{len(glove)}({B_i}^2)}}$$

Where H = GLoVe vector representation of headline, B = GLoVe vector representation of body, and len(glove) is the dimensionality of the GLoVe vector representation used (in this case 50).

This feature returns a real value between -1.0 and 1.0 (higher = words in headline and body are more similar, lower = words in headline and body are more different).

Language Model Representation (KL-Divergence)

Another feature that I am using is the KL-Divergence of the language model representations of the headline and the article body. This is a measure of how divergent (i.e. different) are the language (i.e. words) being used in the headline and the body.

To convert a document to a (simple, unigram) language model, we compute the probability of each word occurring in the document, by using the occurrence of the word:

$$LM_{doc}(word) = \frac{tf_{word,doc}\ (no.of\ times\ word\ occurs\ in\ doc) +\ eps}{|doc|\ (no.of\ words\ in\ doc) + eps \times |V|\ (no.of\ unique\ words\ in\ headline\ AND\ body)}$$

Whereas eps is a small value (I used 0.1) used to 'smoothen' the language model and add a small value to every word that occurs on either the headline or article body. Words that do not occur in either the headline or article body is ignored. The denominator |doc| + eps * |V| normalises the distribution LM_{doc} such that the sum of LM_{doc}(word) for all words in V is 1.

KL-divergence is a measure of how different two probability distributions are. In this case, KL-divergence is used to measure the divergence between the language model of each article's headline and body. This formula is defined as:

$$kl\_divergence\ = \sum\left(LM_{headline}(word) \times \ln\left(\frac{LM_{headline}(word)}{LM_{body}(word)}\right)\right)\ (for\ each\ word\ in\ V)$$

This feature returns a positive real value (in practice between 0.0 and ~3.0, but in theory uncapped), where the higher the value is, the more divergent the language model (i.e. words used) in the article's headline and body are.

Additional Features

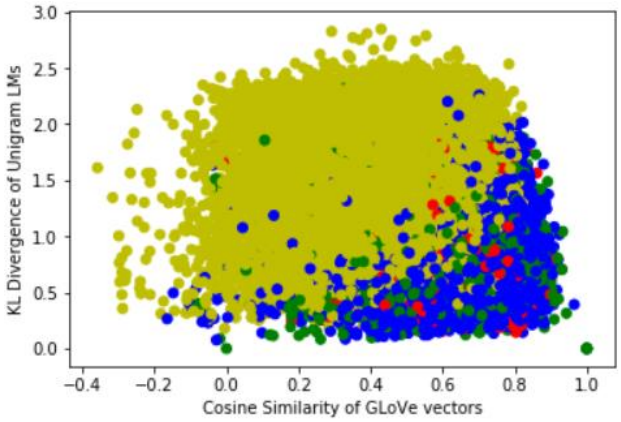
N-gram overlap is a measure of how many times n-grams that occur on the article's headline re-occur on the article's body. For each article (headline-body pair), I counted n-gram overlaps up to 3-grams (i.e. count no. of words in headline that re-occur on body + no. of sequence of 2-words in the headline that re-occur in body + no. of sequence of 3-words in the headline that re-occur in body). This value is then normalised for the article's length, and then raised to the power of 1^-e to 'even out' the bottom-heavy distribution along 0.0-1.0:

$$3gram\_overlap = \left(\frac{no.of\ 1/2/3grams\ in\ headline\ that\ reoccur\ in\ body}{no.of\ words\ in\ body\ (article\ length)}\right)^{\frac{1}{e}}$$

N-gram overlap returns a real value between 0.0 and 1.0 (higher = words in headline and body are more similar, lower = words are more different).

Distance Distribution Plot

To show the distance distribution of the four stances, I plotted a scatterplot of each article's KL-divergence and cosine similarity scores, alongside its label, using matplotlib:



Where yellow = unrelated, blue = discuss, green = agree, and red = disagree.

From this plot, a trend where unrelated articles tend to have higher KL-divergence scores and lower cosine similarity scores between its headline and body can be seen. This would be useful for the task of detecting unrelated articles – the regression model will learn to give higher scores for 'unrelated' where KL-divergence is high and cosine similarity is low, and higher scores for other labels where KL-divergence is low and cosine similarity is high.

However, while these two features seem very useful for classifying unrelated articles vs other labels, they appear to be less useful to distinguish between discuss, agree, and disagree.

Linear Regression & One-vs-All Classifier

By this point, we have converted each article (headline-body pair) to a feature vector over these features (Cosine similarity of word vectors, KL-divergence of language models, 3-gram overlap). This is represented in a matrix X, where e.g. the first ten rows of X (representing the first ten articles or headline-body pairs) are:

```
[0.56364406 1.55180116 0.        ]
[0.51647332 1.36692173 0.        ]
[0.67602961 1.17998136 0.        ]
[0.76712691 1.8089405  0.        ]
[0.75218343 0.91965695 0.12857061]
[0.87951867 1.83223888 0.42424795]
[0.50234723 2.00101174 0.        ]
[0.80303191 2.31892739 0.        ]
[0.61099341 2.32704144 0.        ]
[0.62774508 0.62197213 0.25717938]
```

Where column 1 correspond to cosine similarity of word vectors, column 2 correspond to KL-divergence of language models, and column 3 correspond to 3-gram overlap scores. Each row corresponds to a label in the output vector Y (e.g. the first ten labels in Y):

```
[3 3 3 3 3 2 3 3 3 2]
```

Where the expected output for [0.56364406, 1.55180116, 0.], is 3, the expected output for [0.62774508, 0.62197213, 0.25717938] is 2, etc. (0 = agree, 1 = disagree, 2 = discuss, 3 = unrelated)

A one-vs-all classifier using linear regression models was used to train a model to predict the output values of each row in X. The One vs All classifier will train four linear regression models (one for each label) by transforming the domain of Y to {0, 1} (0 if y_i != the label, 1 if y_i = the label) to train each linear regression model. Each linear regression model would then predict a real value y'_i for each feature vector x_i in X, in which y'_i corresponds to the likelihood of the feature vector x_i having the label associated with that model.

For prediction, for each feature vector x_i, the one-vs-all classifier simply selects the label associated with the model that returns the highest y'_i as the predicted label y''_i for x_i, and returns the vector of predicted labels Y'' as output.

The linear regression model predicts scores y'_i for each feature vector x_i, based on the following model:

$$y'_i = t_0 x_{i0} + t_1 x_{i1} + t_2 x_{i2}$$

The values of t_0, t_1, t_2 is trained using gradient descent to minimise the mean-squared error, using the following algorithm over 1000 iterations (with initial values $t_0 = t_1 = t_2 = 0.0$, and lrn_rate = 0.1):

$$t_j := t_j - \frac{lrn\_rate \times \sum_{i=0}^{|X|} \left((y'_i - y_i) \times x_{ij} \right)}{|X|}$$

Where Y' = predicted values of Y' in the previous iteration, Y = actual (gold) values of Y, and |X| = no. of rows in X.

Performance Analysis

Once the linear regression models for the one-v-all classifier has been trained, it can now be used to predict the labels of new or 'unseen' data.

To test the performance of this model, the one-v-all classifier was used to predict the labels of the validation subset (after converting each article in the validation set to a feature vector using a similar approach). These predicted labels are then compared to the actual labels associated with each article. The results are then fed into the score_submission function provided in the FNC-1 dataset (scorer.py in https://github.com/FakeNewsChallenge/fnc-1), which returns as follows:

Validation subset	Pred. Agree	Pred. Disagree	Pred. Discuss	Pred. Unrelated
Gold. Agree	0	0	302	66
Gold. Disagree	0	0	65	19
Gold. Discuss	1	0	741	149
Gold. Unrelated	0	1	46	3608

Accuracy: 0.870, Weighted Score: 1735.0 / 2256.75 (0.7688)

Then, I loaded the 25413-article test set, converted each article to feature vectors, and predicted the labels of each article, with the following results:

Test set	Pred. Agree	Pred. Disagree	Pred. Discuss	Pred. Unrelated
Gold. Agree	0	0	1504	399
Gold. Disagree	0	0	433	264
Gold. Discuss	1	0	3417	1047
Gold. Unrelated	0	0	160	18188

Accuracy: 0.850, Weighted Score: 8448.25 / 11651.25 (0.7251)

As we can see from these results, the model performs well in predicting 'unrelated' articles correctly, but is unable to distinguish between 'agree', 'disagree', and 'discuss' labels – in fact, it just predicts every article with a low

‘unrelated’ score as ‘discuss’ as it is the second most common label, and with one or two exceptions do not predict anything as ‘agree’ or ‘disagree’.

The learning rate of 0.1 is ideal for this model. If the learning rate is increased to 1.0, the model will diverge (with a very high mean-squared error) and the accuracy on the validation set decreases from 0.870 to 0.017. On the other hand, if the learning rate is decreased to 0.01, the accuracy on the validation set decreases from 0.870 to 0.825.

I have also tested the script using a logistic regression model instead of the linear regression model, by applying a sigmoid function to the linear regressions’ output. However, this model is shown to perform worse than the linear regression model on the validation set, as the logistic model performed with an accuracy of 0.844 while the linear model performed with an accuracy of 0.870.

Feature Importance Analysis

To analyse the importance of each feature for the model, I re-trained the one-v-all linear regression model, omitting one feature at a time, and tested the performance of each missing-feature model on the validation set.

- Without cosine similarity of word vectors, the validation set accuracy decreases from 0.870 to 0.870 (seemingly no difference).
- Without KL-divergence of language models, the validation set accuracy decreases from 0.870 to 0.842.
- Without 3-gram overlap, the validation set accuracy decreases from 0.870 to 0.809.

Literature Review

Various models have been proposed for the task of stance detection with regards to the domain of news headlines and article text, for example:

- The paper ‘Towards Automatic Identification of Fake News: Headline-Article Stance Detection with LSTM Attention Models’ by Chopra, S., Jain, S. and Sholar, J.M. (2017) proposed a two-step approach to stance detection on the FNC-1 dataset: using a SVM classifier on TF-IDF cosine distance to distinguish ‘unrelated’, and using an attention-based LSTM deep learning model on GLoVe word vectors to distinguish between ‘discuss’, ‘agree’, and ‘disagree’. It reported a weighted score of 0.8658 on the test set.
- The paper ‘Fake news stance detection using stacked ensemble of classifiers’ by Thorne, J., Chen, M., Myrianthous, G., Pu, J., Wang, X. and Vlachos, A. (2017) proposed a model using an ensemble of five ’slave’ classifiers based on multi-layer perceptron (MLP) with ReLU activation function, using features such as word2vec and TF-IDF vectors. Then, a master classifier that takes the output of the ‘slave’ classifiers as an input vector to predict a single output using a gradient-boosted decision tree model. It reported a weighted score of 0.7804 on the test set.
- The paper ‘Fake News, Real Consequences: Recruiting Neural Networks for the Fight Against Fake News’ by Davis, R. and Proctor, C. (2017) proposed a bag-of-words multilayer perceptron (BoW MLP) model. It reported a weighted score of 0.89 on the test set.

Further Improvement

Currently, the model trained by the script I have developed performs well in predicting ‘unrelated’ labels vs other labels, but is unable to classify between ‘discuss’, ‘agree’, and ‘disagree’. The main reasoning behind this is because the features extracted in this model (cosine similarity of word vectors, KL-divergence of language models, 3-gram overlap) performs well in distinguishing ‘unrelated’ labels vs other labels, but are less useful to distinguish between ‘discuss’, ‘agree’, and ‘disagree’, as shown in the distance distribution plot.

To distinguish between ‘discuss’, ‘agree’, and ‘disagree’, features that look beyond word-similarity would be required. For example, aspect-based sentiment models could be used to measure the divergence in sentiment between the headline and the article body towards common topics. However, this was infeasible as these sentiment models are quite complex.

Additionally, the one-v-all linear regression model is not fully suitable for classification tasks such as this. If it was possible to use existing machine-learning libraries for this task, I would try using more sophisticated classification models, such as random forest, SVM, or Naïve Bayes based classifiers, instead of adapting a regression algorithm for a classification task due to linear regression’s ease-of-implementation.