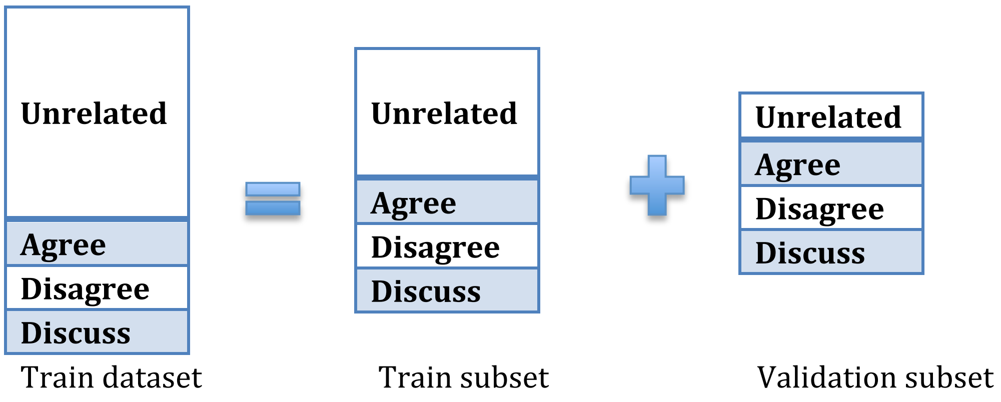
Information retrieval and data mining

The implementation of this project is used python, with some function being displayed by ipython. The purpose of this project is to detect the similarity as well as to classify different stances between headlines and bodies. The following section will discuss and explain the detailed implementation of each step in this project.

1. Split data:

The first step I did in this project is to separate the training data to training subset and validation subset. In order to make sure that the training data and validation data have the similar ratio of the four classes.

Firstly, I created four arrays to store data with ‘unrelated’, ‘discuss’, ‘agree’ and ‘disagree’ stances respectively. For example, in terms of stances with type ‘unrelated’, I put all the unrelated bodies and headlines into an array. Since the bodies is linked with headlines using body id, I used body id to find the corresponding article body of each headline. After creating arrays of four stances, I split the arrays to two parts. The portion of two parts of data is 9:1, with one part correlating to train subset and another to validation subset. The next step I did was to combine/contact the four arrays into one array. After doing the above steps, I have make sure that I have split the training set with the data number portion about 9:1. Also, the training set and validation set have similar ratio of four classes. The process of this step is as following:



2. Word to vector:

2.1 Split word:

In the project, the input is a list of bodies and headlines, and both of them are regarded as a list of documents. All the documents are consist of sentences. It is difficult to create vector based on sentences, so the first step I did is to split the word from the sentences in the documents. In addition, I removed some less important word including ‘the’, ‘a’ from the documents. Since these words are insignificant but may appear many times in each documents. After doing this step, a two-dimension matrix was generated with the representation of words in each document. Here is a simple example of the input and output of this step:

Input: ['the first sentence', 'the second sentence']

Output: [['first', 'sentence'], ['second', 'sentence']]

2.2 Word embedding

The purpose of the word-embedding step is to represent words with a low-dimension vector. As a result, the word in each document is represented as a tuple:

Tuple = (word token, word frequency)

The first value of the tuple is the word token and the second value is the frequency of the word appearing in the documents. The whole documents list is viewed as a corpus, and token is given to each distinct word in the corpus. The frequency of word is the number of times the word appearing in its correlating document. The implementation of this step is used the dictionary in the gensim corpora library in python. The input and output of the data can be represented as following:

Input: [['first', 'sentence', 'first'], ['second', 'sentence']]

Token: first: 0, sentence: 1, second: 2

Output: [[(0, 2), (1, 1)], [(2, 1), (1, 1)]]

By doing this step, the word in each document is represent as a list of tuple. And it is easier for the machine to recognize and build model on the representation of the words.

2.3 Tfidf model:

In the above step, I transferred the word to vector and plan to use the tifidf model to represent vector. Tfidf weight is composed by two terms, the Term frequency and inverse document frequency.

Term Frequency: it measures the number of times a term occurs in a document.

TF = (number of times term tappears in a document) / (Total number of terms in the document)

Inverse Document Frequency: it measures how important a term is. It can help to weight down the frequent terms while scale up the rare ones.

IDF = log (total number of documents/ number of document with term t int it)

Since the length of the headlines and bodies is not the same, the first step is to transfer headlines and documents to two vectors with same size.

The vector size is equal to the corpus size, each value in the vector representing a word in the corpus. The value in the vector is the tfidf result of the correlating word in the body or the headline. The headline is also represented as a vector in the same way. If the word do not exit in the headline, its correlating value will be zero. The following is a simple example of how to represent the vector using a

tfidf model:
Corpus_size: 3
Headline: [‘first’]
Bodies: [‘the’, ‘first’, ‘sentence’]

Tfidf vector of Headline: $\begin{array}{cccc} & \text{the} & \text{first} & \text{sentence} \\ [& 0 , & 0.\ 1\ 7, & 0 \quad] \end{array}$

Tfidf vector of Bodies: $\begin{array}{cccc} & \text{the} & \text{first} & \text{sentence} \\ [& 0\ .14, & 0\ .17, & 0 \quad .18] \end{array}$

3. Language model:

3.1 Unigram language model

In this project, I build the unigram language model with the dataset. Firstly, I calculate the frequency of each word in the dataset. Then I built a function to calculate the probability of each word, with the numerator being the frequency of word and the denominator being the number of words in the model. And the smoothing policy of the language model is to add 1 to the numerator and the denominator. Also, I calculated the probability of sentences with adding the probability of each word in each sentence and normalize the probability by dividing the number.

Probability of word = number of the same word in the document/ total number of words in the document

Probability of sentence = sum of probability of each word in the sentence

Smoothing = number of the same word in the document + 1/ total number of words in the document + 1

4.Similarity and Distance

In the project, I used various methods to calculate the similarity and distance between the body and headline. The following section will list the methods using in the project and compare the difference between different distances.

4.1 KL-divergence:

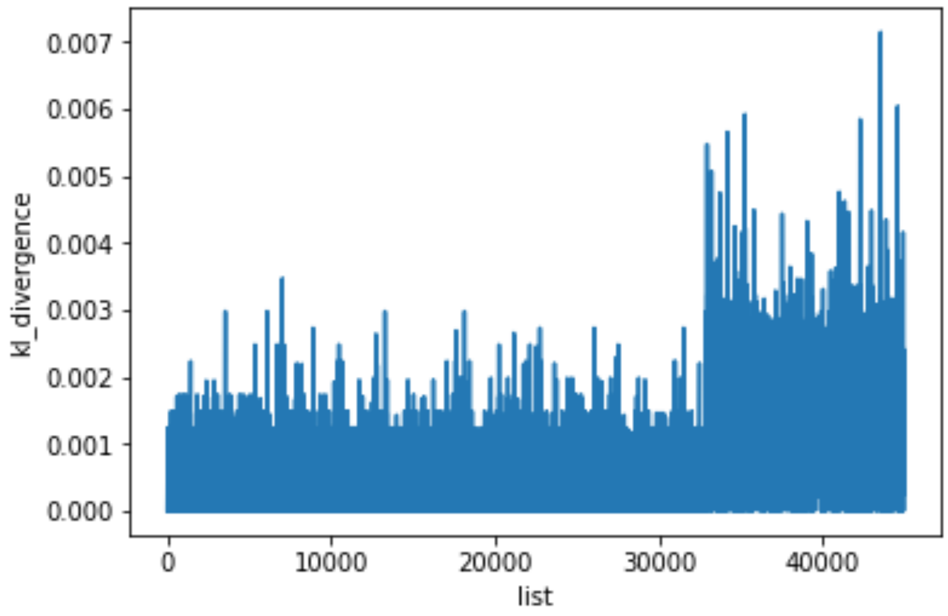
The kullback-leibler divergence is a measure of how one probability distribution diverges from a second. KL divergence is the expectation of the log difference between the probability of data in the original distribution with the approximating distribution.
Its formula is as following:

$$\mathbf{D_{KL}(p||q)=\sum^{N} p(x_i) \cdot (\log_{p}(x_i) - \log_{q}(x_i))}$$

The more common way to see KL divergence written is as follows:

$$\mathbf{D_{KL}(p||q)=\sum^{N} N_{p}(x_i) \cdot \log_{q}(x_i) p(x_i)}$$

The kl divergence in the training dataset is as following. Most of the unrelated data has the lower kl divergence, while the related data has the higher kl divergence.

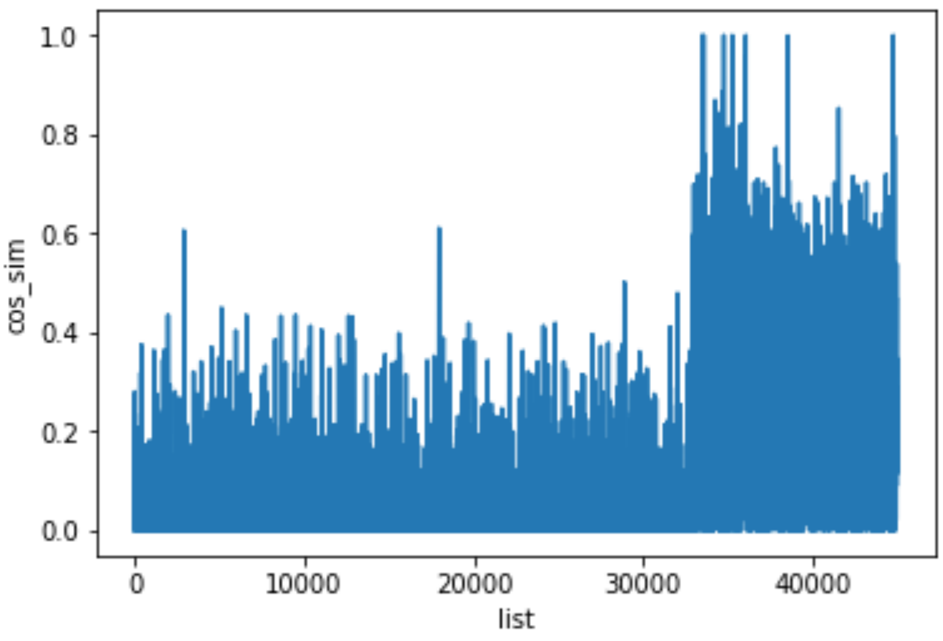


4.2 Cosine Similarity:

Cosine Similarity is used to measure the cosine angle between two non-zero vectors. It can imply the similarity between two vectors, and its value is between 0 and 1. Two vectors with the same orientation will have a cosine similarity of 1. The more similar the two vectors are, the higher the cosine similarity. The formula of the cosine similarity is:

Cosine similarity = A.B / ||A||*||B||

The following is the image of the cosine similarity in the training data. As the essay mentioned before, the training data is concated by four arrays with four different stances. The first part is the 'unrelated' stance, following by 'disagree', 'agree' and 'discuss' stances. According to the figure, all the 'unrelated' data has a relatively low cosine similarity. On the contrary, all the 'related' data has a higher similarity. However, it is difficult to distinguish the three type of related stances from the cosine similarity.

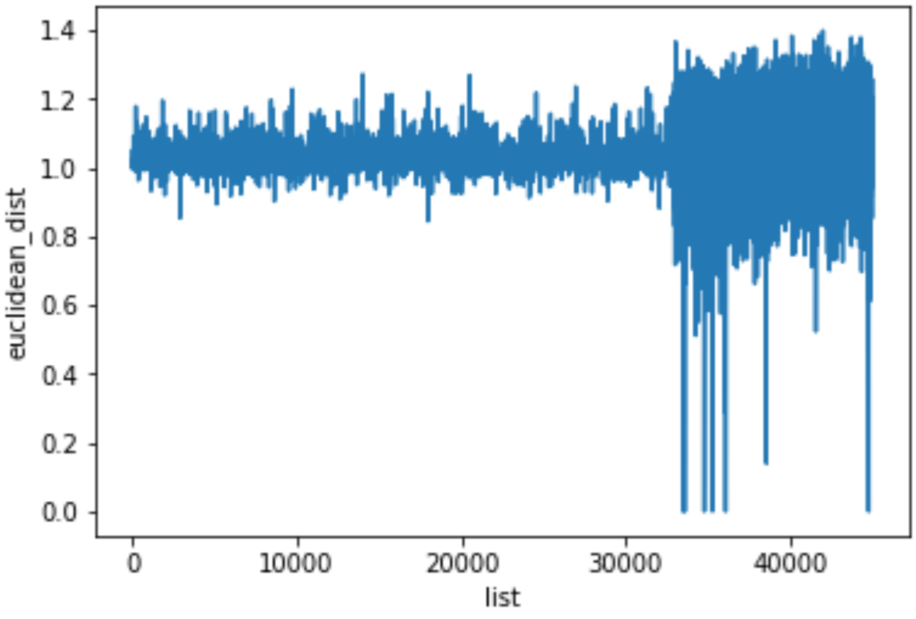


4.3 Euclidean distance:

Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space. The Euclidean Distance can be calculated as following:

$$\boldsymbol{d(p,q)} = \sqrt{\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} (\boldsymbol{q_i} - \boldsymbol{p_i})^2}$$

The following is the picture of Euclidean distance in the training dataset. Most of the unrelated data have the Euclidean distance between 0.8 – 1.2, while the related data has a wider variety of Euclidean distance.

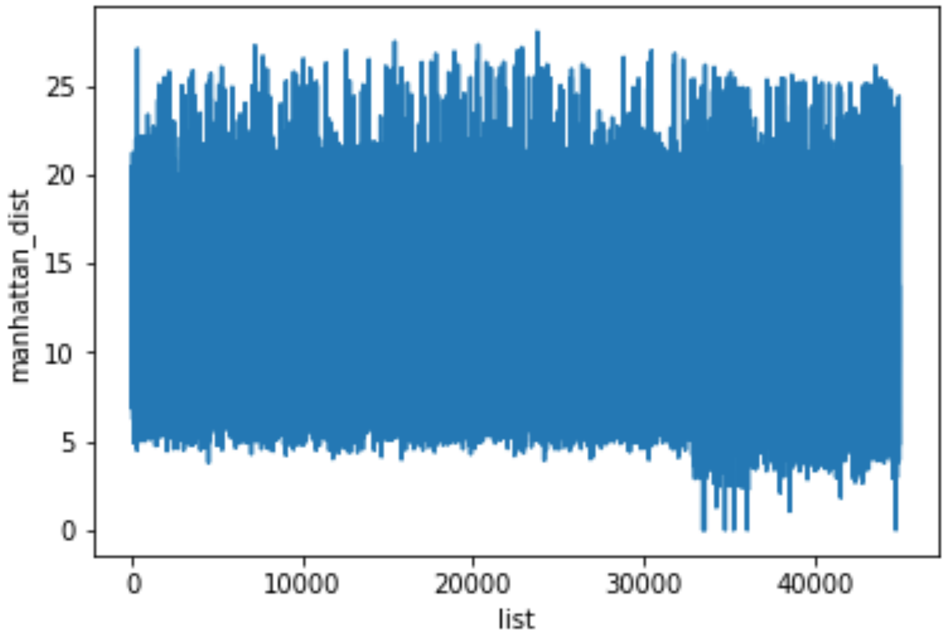


4.4 Manhattan Distance

Manhattan Distance is the distance between two points in a grid based on a strictly horizontal or vertical path. It is the simple sum of the horizontal and vertical components. The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2), the Manhattan Distance is:

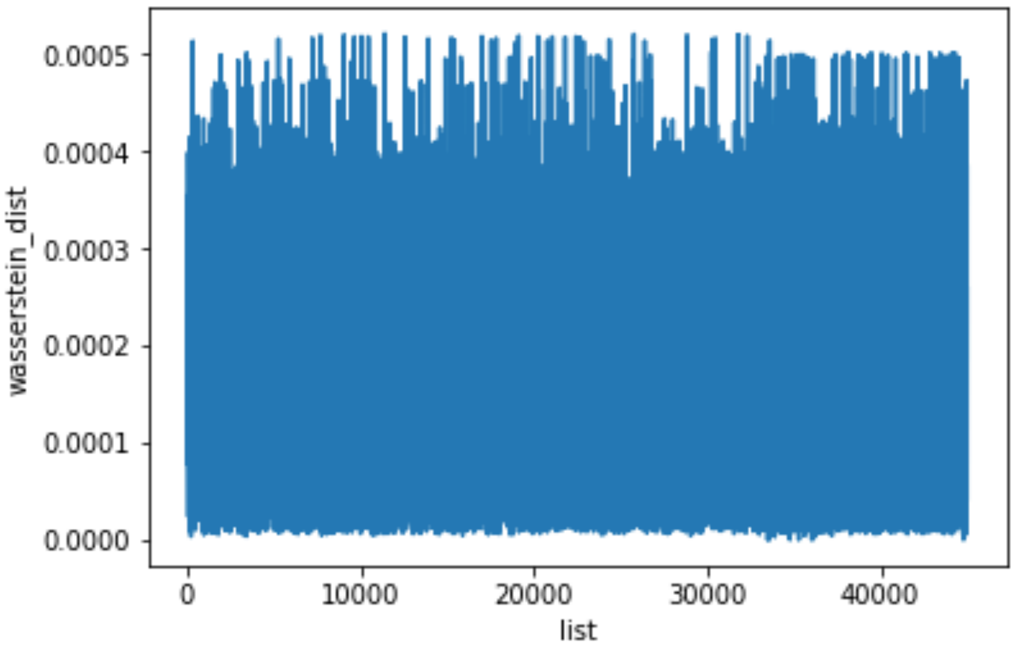
$$\textbf{Manhattan Distance} = |x_1 - x_2| + |y_1 - y_2|.$$

The following is the picture of Manhattan distance in the training dataset.



4.5 Wasserstein Distance

In mathematics, the Wasserstein distance is a distance function defined between probability distributions on a given metric space. The following is the picture of Wasserstein distance in the training dataset.



5. Linear Regression

Since the input with the dataset has multiple features, the linear regression used in the project is the multiple linear regression. In multiple linear regression, two or more independent variable are used to predict the value of a dependent variable.

The multiple linear regression model can be represented as:

$$Y=\beta_0+\beta_1x_1+\beta_1x_2+\ldots+\beta_nx_n$$

Xi is the i^{th} features of the input value. In this project, the features are the cosine similarity and Euclidean distance value. The equation can be converted to a matrix:

$$Y=\beta^TX \quad (\beta=[\beta_0\beta_1\beta_2..\beta_n]^T, \quad X=[x_0x_1x_2..x_n]^T)$$

The cost function of this equation is:

$$cost(\beta)=1/2m\sum i=1m(h\beta(x^{(i)})-y^{(i)})^2$$

Gradient Descent is used as an optimization algorithm to minimize the cost in multiple logistic regression. The gradient descent was calculated and updated in the following ways:

Hypothesis: $h_\beta(x)=\beta^Tx$

Loss: $(h_\beta(x)-y)$

Gradient: $(h_\beta(x)-y)xj$
Gradient Descent Updation: $\beta j:=\beta j-\alpha(h_\beta(x)-y)xj)$

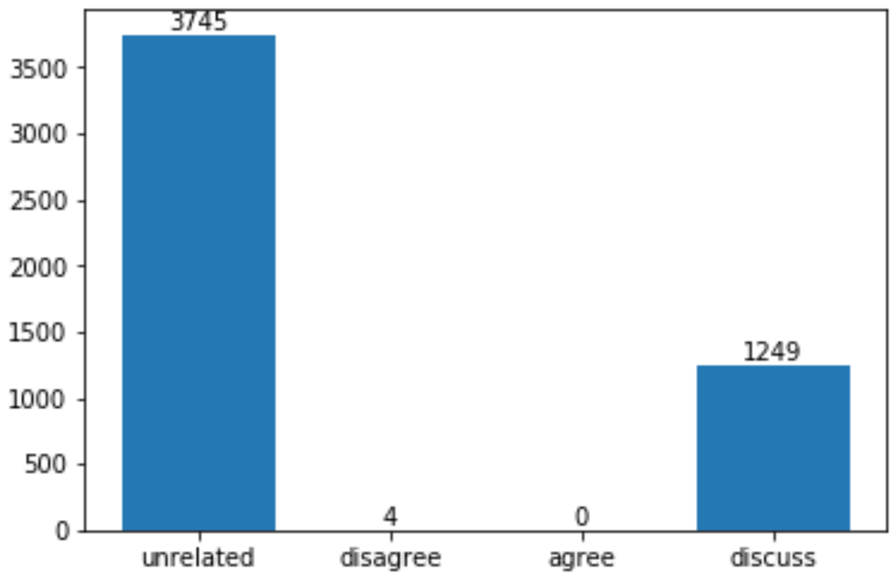
Since the project needed to predict four different stances, four linear regression model were built to predict different kind of stances. The result is the maximum probability of the four linear regression models.
The following is the training cost of the four different linear regression models after 3000 iterations.

Stances	Cost
Unrelated	0.05106
Agree	0.09925
Disagree	0.02932
Discuss	0.11125

The test accuracy of the multiple linear regression model is87.9%

Accuracy of validation data	0.879

The following is the distribution of four stances in the prediction of the validation set.



However, the model can’t tell the difference between relate stance.

6. Logistic Regression Model

Logistic regression is quite similar to linear regression model, and it is intended for binary classification problems. It will predict the probability of an instance belonging to the default class. Logistic regression uses a sigmoid function that generates outputs between 0 and 1. Sigmoid function is like this:

$$\textbf{Sigmoid}\ (z) = 1/\ (1+e^{-z})$$

The logistic regression model can be represented as:

$$Y = \mathrm{sigmoid}\,(W^T X)$$

With W representing the weight of each features. X is the list of feature values. The logistic model also used the gradient descent method to optimize the result:

Hypothesis: $h_\beta(x) = \mathrm{sigmoid}\,(W^T X)$
Loss: $\mathrm{sigmoid}\,(W^T X) - Y$
Gradient: $X^{T*}\,(\mathrm{sigmoid}\,(W^T X) - Y)$
Gradient Descent Updation: $W = W + \alpha\,(\mathrm{sigmoid}\,(W^T X) - Y)X)$

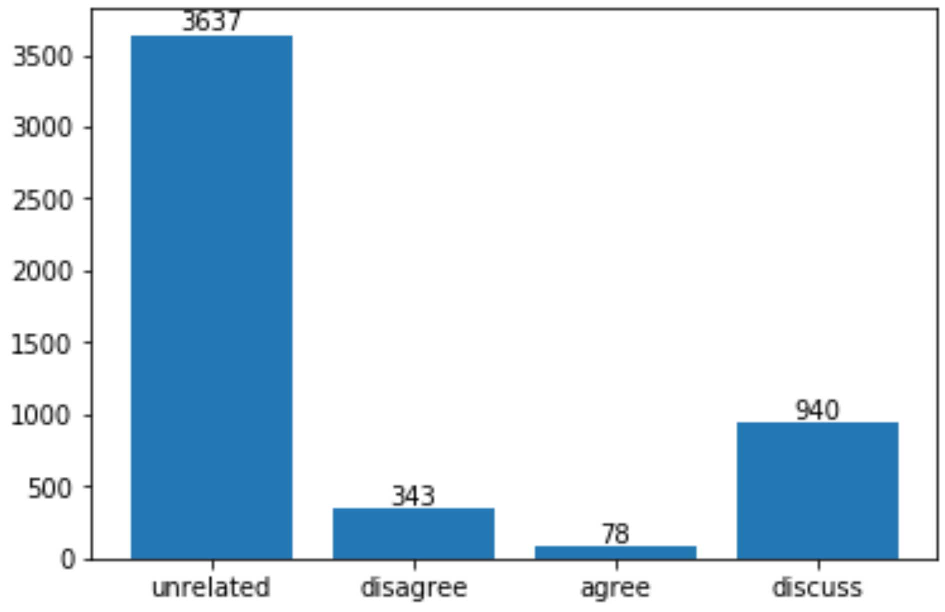
Four logistic regression model were built with four different stances. The training accuracy is as following

Stances	Accuracy of training data
Unrelated	0.96295
Agree	0.92653
Disagree	0.97998
Discuss	0.83659

The accuracy of the validation data is 83.9%.

Accuracy of validation data	0.839

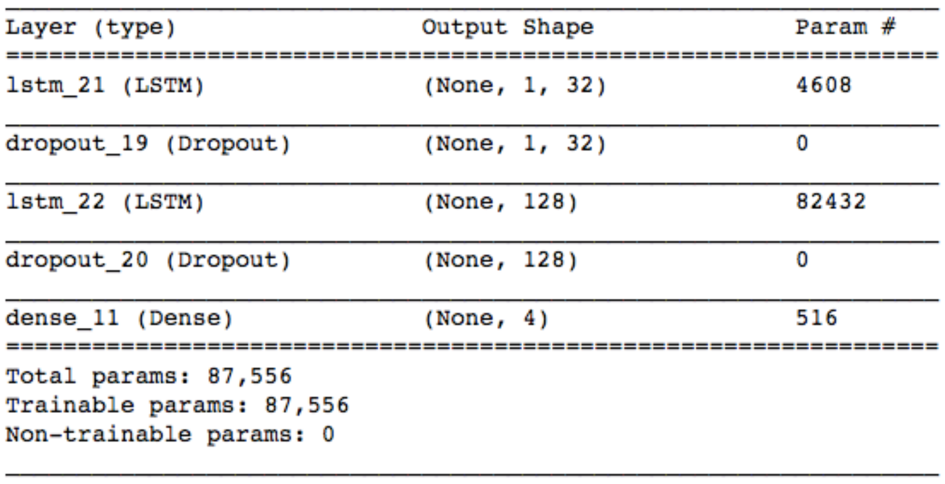
The following is the distribution of four stances in the prediction of the validation set.



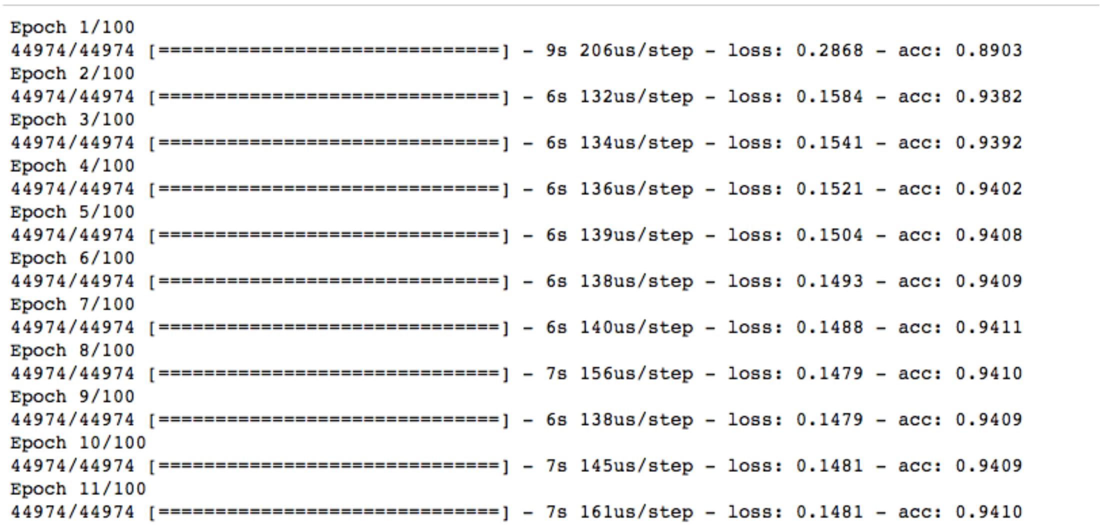
7. Comparison between features

8. LSTM machine learning models

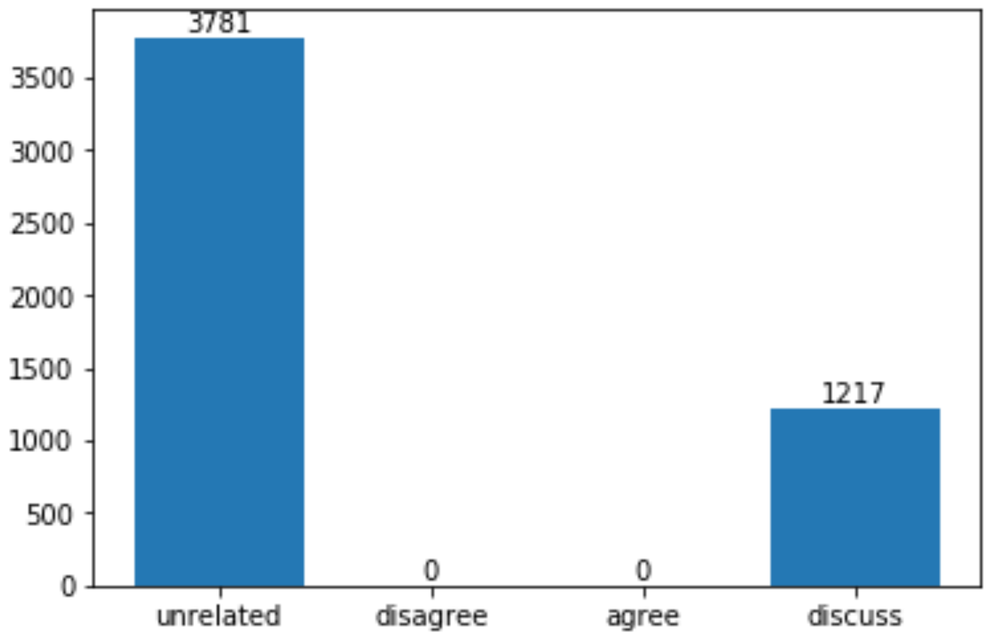
LSTM stands for Long Short Term Memory networks. It is a special kind of RNN, and is capable of learning long-term dependencies. Since the words in different sentences have dependencies with each other, LSTM is suitable for certain language process tasks. This project also used LSTM model to predict the stance. It can also reach an accuracy of 94% with the training data and 87.9% with the validation data. However, the training result is slightly overfitting, since it can't predict the disagree and agree stance correctly. Although certain methods including dropout the l1/l2 regression were used to avoid overfitting, the training result still cannot predict the result correctly.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_21 (LSTM)               (None, 1, 32)             4608
_________________________________________________________________
dropout_19 (Dropout)         (None, 1, 32)             0
_________________________________________________________________
lstm_22 (LSTM)               (None, 128)               82432
_________________________________________________________________
dropout_20 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 516
=================================================================
Total params: 87,556
Trainable params: 87,556
Non-trainable params: 0
_________________________________________________________________
```



```
Epoch 1/100
44974/44974 [==============================] - 9s 206us/step - loss: 0.2868 - acc: 0.8903
Epoch 2/100
44974/44974 [==============================] - 6s 132us/step - loss: 0.1584 - acc: 0.9382
Epoch 3/100
44974/44974 [==============================] - 6s 134us/step - loss: 0.1541 - acc: 0.9392
Epoch 4/100
44974/44974 [==============================] - 6s 136us/step - loss: 0.1521 - acc: 0.9402
Epoch 5/100
44974/44974 [==============================] - 6s 139us/step - loss: 0.1504 - acc: 0.9408
Epoch 6/100
44974/44974 [==============================] - 6s 138us/step - loss: 0.1493 - acc: 0.9409
Epoch 7/100
44974/44974 [==============================] - 6s 140us/step - loss: 0.1488 - acc: 0.9411
Epoch 8/100
44974/44974 [==============================] - 7s 156us/step - loss: 0.1479 - acc: 0.9410
Epoch 9/100
44974/44974 [==============================] - 6s 138us/step - loss: 0.1479 - acc: 0.9409
Epoch 10/100
44974/44974 [==============================] - 7s 145us/step - loss: 0.1481 - acc: 0.9409
Epoch 11/100
44974/44974 [==============================] - 7s 161us/step - loss: 0.1481 - acc: 0.9410
```



New feature:

The neural network model also used a new feature called word count. It is the same number of terms appeared both in the body and headline. The feature performs well with the training data and can increase the accuracy by percent of 2%.

Reference:

[1]Isabelle Augenstein, Tim Rockta¨schel, Andreas Vlachos, and Kalina Bontcheva. Stance detection with bidirectional conditional encoding. arXiv preprint arXiv:1606.05464, 2016.
[2] SamuelRBowman,GaborAngeli,ChristopherPotts,andChristopherD Manning. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326, 2015.