University College London

Information Retrieval and Data Mining

Coursework Report

FNC-1 News Stance Detection Report

Author:
Ziyang Liu

Student Number:
14026048

April 8, 2018

Contents

Chapter 1

Feature engineering

1.1 Data Splitting

I first joined the headlines and bodies datasets on the body ID in the training and testing datasets before splitting the training set. I subsequently implemented my own data splitting function, which can be used to split the training data set into a new training set and a validation set with the data ratio about 9:1. These two new datasets also have similar ratios of the four classes namely 'agree', 'disagree', 'discuss' and 'unrelated'.

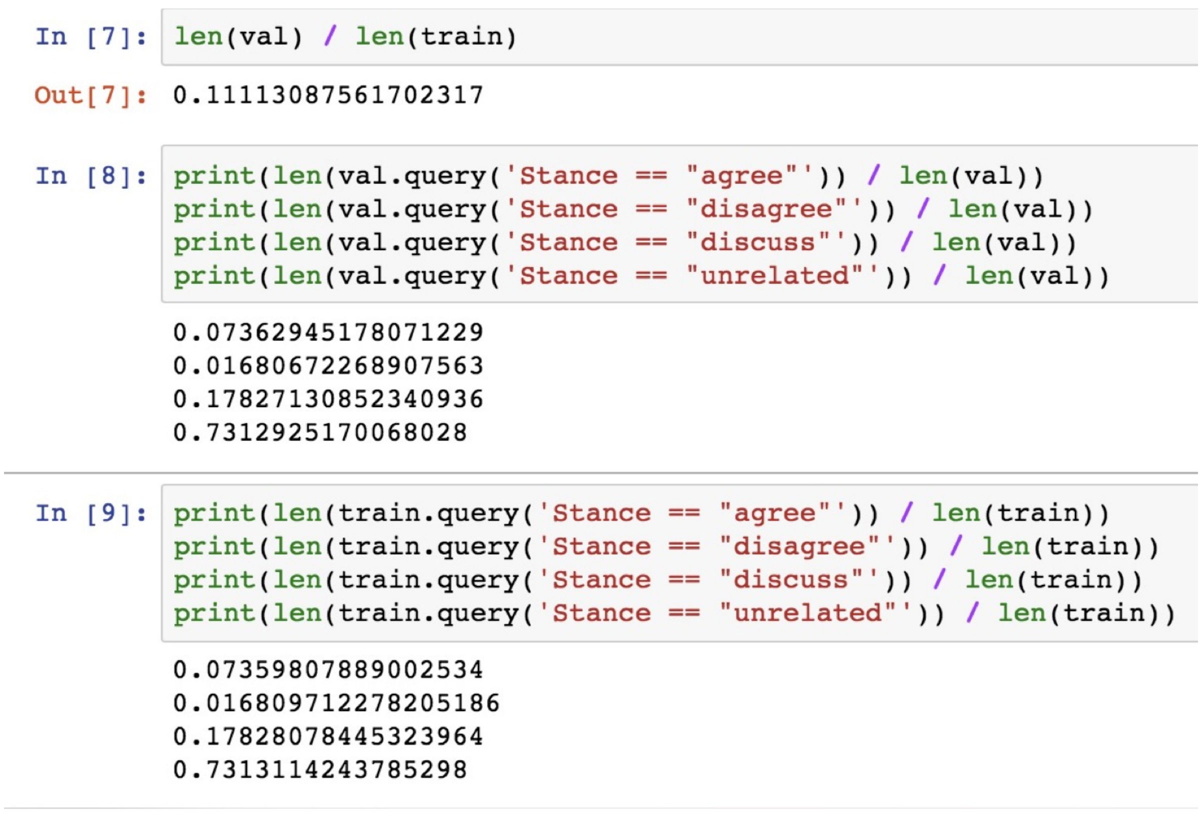
These statements can be clearly proved in Figure 1.1. Since the datasets are not time-related, I allocated the first 10% of the original training set as the validation set and assigned the rest as the new training set. I did not implement random splitting since it's more crucial to not lose any data after splitting.

1.2 Clean headlines and bodies in all the datasets

I implemented a tokenize function using the "nltk" library. The function essentially uses the "RegexpTokenizer" class to tokenize a sentence. After this, it converts each word to lower case stem word using the "PorterStemmer" class. This function also removes all the stop words using the "nltk" English stop words list. After implementing this, I applied this function to all the headlines and bodies to preprocess the sentences.

1.3 TF-IDF cosine similarities

I implemented my own 'tf_idf_vectorizer' class in the "bag_of_words.py" file, which can be used to convert all the headlines and bodies into vector representations. This class has two main functionalities namely "fit" and "transform". The "fit" function



```
In [7]: len(val) / len(train)

Out[7]: 0.11113087561702317

In [8]: print(len(val.query('Stance == "agree"')) / len(val))
        print(len(val.query('Stance == "disagree"')) / len(val))
        print(len(val.query('Stance == "discuss"')) / len(val))
        print(len(val.query('Stance == "unrelated"')) / len(val))

        0.07362945178071229
        0.01680672268907563
        0.17827130852340936
        0.7312925170068028

In [9]: print(len(train.query('Stance == "agree"')) / len(train))
        print(len(train.query('Stance == "disagree"')) / len(train))
        print(len(train.query('Stance == "discuss"')) / len(train))
        print(len(train.query('Stance == "unrelated"')) / len(train))

        0.07359807889002534
        0.016809712278205186
        0.17828078445323964
        0.7313114243785298
```

Figure 1.1: *Result of data splitting function*

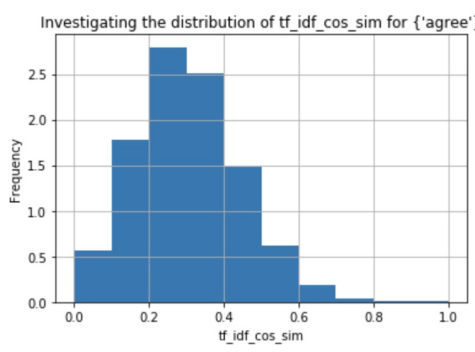


Figure 1.2: *Distribution of tf-idf cos sim for agree*

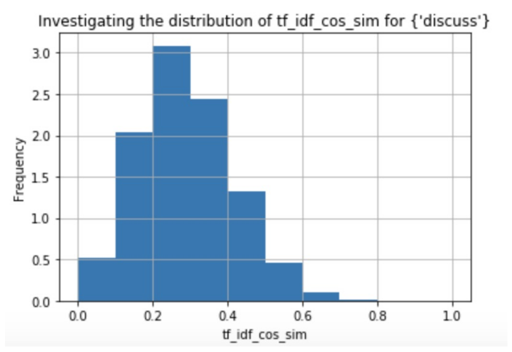


Figure 1.3: *Distribution of tf-idf cos sim for discuss*

can be used to learn the vocab from the training headlines and bodies. The function also adds an out of vocabulary - "oov" token to the vocab.

The "transform" function converts each headline and body to a vector, which is explained in details later. For each headline and body, the function converts all the unseen words to "oov" tokens. The data sets all have a very large dimension but the number of unique words in each document is very small. The function thus precomputed the IDF matrix for all the words. It subsequently loops through all the documents and then all the unique words in each document in order to compute "tf-IDF" for each word using the precomputed IDF matrix. Furthermore, I improved the speed of calculations by enabling the processing of each document concurrent with 5 threads using the python concurrent class.

Finally, I applied these two functions to calculate the tf-IDF matrix for all the documents in all the datasets. Please notice that the transform function was applied to the headlines and bodies separately. After these, I applied my own cosine similarity function to calculate the "tf_idf_cos_sim" feature for each headline and body pair. I also stored the vector representations of headlines and bodies in the result panda DataFrame.

1.4 KL divergence feature

I implemented a Unigram, a Ngram and an Interpolate language model class in the LM.py and Interpolate_LM.py files. The Unigram and Ngram classes both use the Lindstone correction discounting method with epsilon equal to 0.1. The Interpolate_LM class uses Jelinek - Mercer Smoothing method with alpha equal to 0.9 to interpolate with the background language model. The background lm is the headlines or body collection language model.

In addition, I created "inject_oovs" function and used this to inject an "oov" token to the corpus each time when encountering new words. This aids the language model to estimate the probability of unseen words. After processing the headlines and bodies, I created two collection language models for the headlines and bodies separately.

After these, I created an interpolated language model for each headline and body with the corresponding collection language model. I subsequently calculate the KL divergence using the formula 1.1 in the "KL_divergence" function. After my short investigations of suitable language models, I have found the Unigram language model actually performed best so I choose this as my basic language model.

$$KL(LM_H||LM_B) = -\sum_{w \in H} P(w|LM_H) \log P(w|LM_B) \tag{1.1}$$

where H means headlines and B means bodies.

1.5 Propose more features

I have also extracted more features from each headline and body pair. The most obvious features are the Euclidean and Manhattan distance between each vectors pair, which is called "tf_idf_eucliden_dis" and "tf_idf_Manhattan_dis" respectively. Furthermore, I counted the number of common words between each pair, which is called "common_words_count".

1.6 Analyze feature importance using distribution plots

I suspected the "tf_idf_cos_sim" and "common_words_count" features may be important for the learning process. I thus plotted the distributions of the feature data to explore the features' importance.

Figures 1.2 to 1.5 are tf-idf cos similarity distribution plots for the four stances. These four figures clearly indicate the probability of tf-idf cos similarity below 0.05 is very high(above 90%) in the unrelated stance. However, the probability is only below 2.5% in other stances. This provides a method to distinguish the unrelated and related stance for the learning process. The tf-idf cos similarity feature is thus very crucial to use when training.

Figures 1.6 to 1.9 are common words count distribution plots for the four stances. Similar to before, we will focus on the probability when the count is below 5. After some calculations, you can clearly find the unrelated stance result in a probability of 87.5% whereas the others result in a maximum of 39.4% (my calculation results: agree: 36.6%, discuss: 26.3%, disagree: 39.4%). These indicate the feature can be used to distinguish the unrelated and related stances as well. However, the differences in probabilities are smaller in four stances when using common words count feature. I thus suggest the tf-idf cos similarity is a much more useful feature.

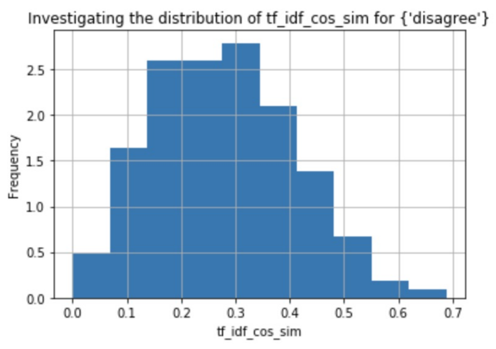


Figure 1.4: *Distribution of tf-idf cos sim for disagree*

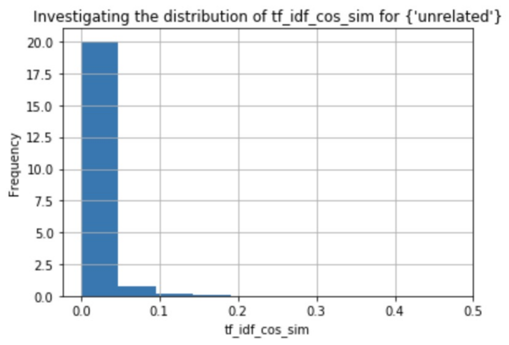


Figure 1.5: *Distribution of tf-idf cos sim for unrelated*

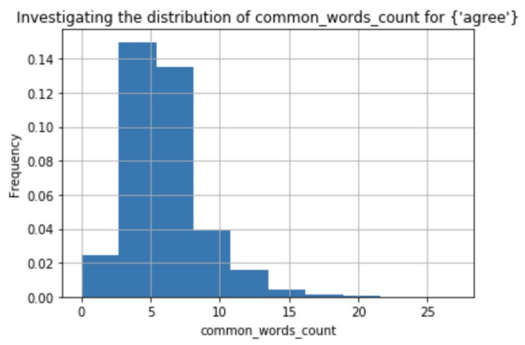


Figure 1.6: *Distribution of common words count for agree*

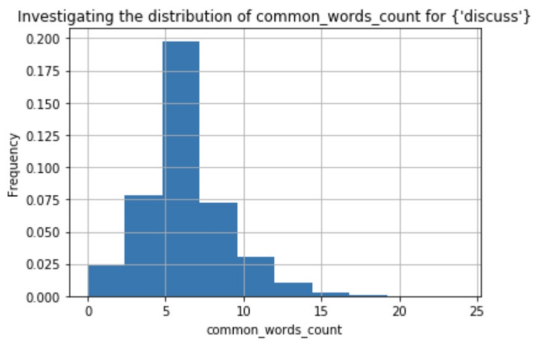


Figure 1.7: *Distribution of common words count for discuss*

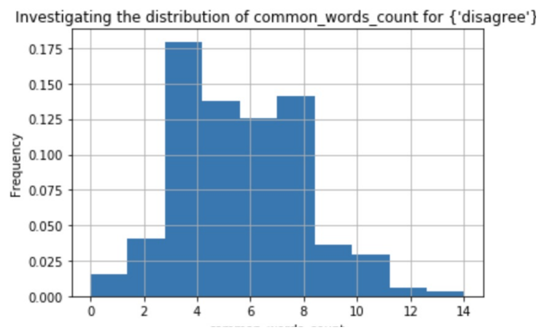


Figure 1.8: *Distribution of common words count for disagree*

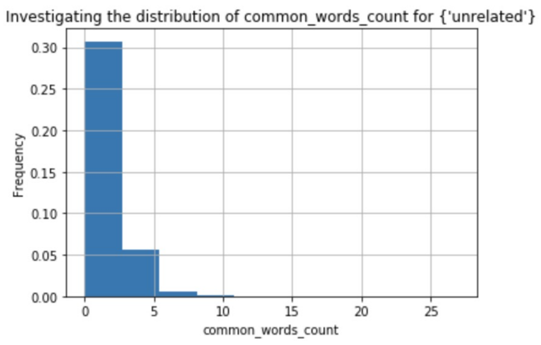


Figure 1.9: *Distribution of common words count for unrelated*

Chapter 2

Linear reg and logistic reg

2.1 Implementations

I implemented a linear regression and logistic regression model with gradient descent using "numpy" library. These two classes both have a fit and predict function. The fit function takes in x, y, epochs and learning rate so as to train the model.

2.2 Predict the stance

Firstly, I preprocessed the data again to remove useless features such as Euclidean distance. I also split the data into x and y. I also normalized the range of feature values in order to improve the performance of the gradient descent. This is because feature values with a large range can potentially make the models difficult to learn.

Since the task is a multi-class classification issue, I predicted the probabilities of all four stances using the one-vs-rest strategy. I subsequently classified the stance to be the one with the highest probability. I trained a linear regression and a logistic regression model. I then tuned the parameters on the validation set. The best learning rate is 0.061 and number of epochs is 3000 for the linear regression model. The logistic regression model uses 0.36 as the learning rate instead.

I implemented an accuracy score function as the evaluation metric since it is a multi-class classification task. The accuracy can be essentially calculated by using the number of correctly classified rows divided by the total number of rows. The training, validation and test accuracies are 87.04%, 88.60%, and 85.53% respectively for the linear regression model. The training, validation and test accuracies are 87.55%, 88.74%, and 85.83% respectively for the logistic regression model. The test accuracy is much higher when using the logistic regression model. I thus suggest the logistic regression model is better to use for this task.

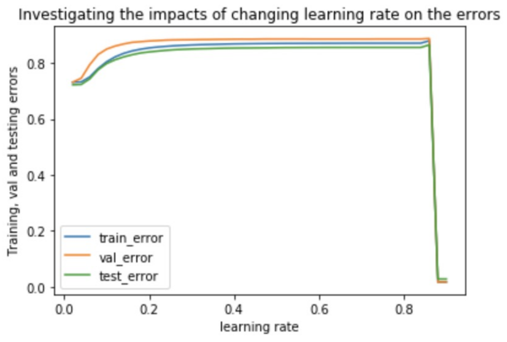


Figure 2.1: *Linear regression experiments*

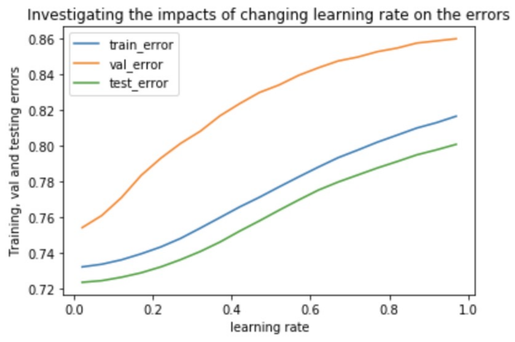


Figure 2.2: *Logistic regression experiments*

2.3 Investigation of effect of learning rate

Figures 2.1 and 2.2 are the experiment results of investigating the impact of changing learning rate on the performance of the model. These clearly demonstrate the training, validation and test accuracies all increase when the learning rate inclines. However, Figure 2.1 also reveals there is an optimal learning rate which results in optimal accuracies. If we increase the learning rate beyond this optimal value, the accuracies will decrease.

2.4 Feature importance

For each feature I used previously in the models, I calculated the accuracy reductions compared with using all the features. The feature with the highest reduction in performance will be the most important one. The results are listed in Table 2.1. The results clearly indicate the most important feature is "tf_idf_cos_sim" in linear regression whereas "KL_divergence" is the most crucial factor in logistic regression.

	tf_idf_cos_sim	common_words_count	KL_divergence
Linear regression	0.020	0.001	0.005
Logistic regression	0.022	0.001	0.279

Table 2.1: The reduction in accuracy when dropping each feature for linear regression and logistic regression

Chapter 3

Literature Survey

Linear regression and logistic regression models are not capable of learning non-linear features. In this section, I will thus discuss some improvement proposals found when searching the FNC-1 Stance competition literature. The literature suggests several additional features, which are not implemented in my models and are listed below. These also recommend we classify in two stages. Firstly, we distinguish the unrelated pairs of headline and body from related ones. We subsequently classify how each pair is related ie 'discuss', 'disagree' and 'agree'.[1][2] Furthermore, we need to implement a gradient boosting classifier especially "xg-boost" and a bidirectional lstm encoder model in order to predict the stance. The bidirectional lstm encoder can obtain the contextual representations of the words in headlines and bodies. [1][2][3]. We can also build the bidirectional lstm encoder with attention layers. All these models can automatically learn non-linear features in the text. According to my knowledge, a two-stage random forest model with same features lead to the highest accuracy for this task.[2]

- List of refuting word features extended with related words from both PPDB and WordNet
- Features for disagreeing words and discussing words from both PPDB and WordNet
- Google's pre-trained Word2Vec and Stanford's GloVe embeddings
- Average of vector representations of words in the embeddings
- Average embeddings weighted by tf-idf score
- Word Mover distance
- Body Text Clipping (Merely keep the first two and last two sentences in a body)

Chapter 4

Proposals

4.1 Replace the model with xgboost

In this section, I replaced the previous machine learning models with K fold "xgboost". The preprocessing steps remained identical except not normalizing the feature values. Similar to before, I predicted the probability of each stance using the one-vs-rest strategy. I subsequently classified the stance to be the one with the highest probability. The result validation and test accuracies are 89.18% and 87.05% respectively. The test accuracy increases approximately 1.22%, which is a huge improvement. Furthermore, I also obtained a test accuracy very close to the best accuracy reported in the literature.[1]

4.2 Xgboost with two stages classifier

We can also classify in two stages using xgboost K fold model. Due to a limitation of time, I have implemented such classifier with same features in two stages. The result validation and test accuracies are 89.18% and 86.95% respectively, which are very similar to one stage xgboost classifier. In the future, we can also investigate about features listed in the literature survey chapter.[1]

4.3 Bidirectional lstm - future work

Furthermore, we can compute the Glove embeddings for each word and subsequently compute the average embeddings for each word. We consequently obtain vector representations of headlines and bodies, which can then be plugged into a bidirectional lstm model. The model is able to automatically learn the features of each vectors pair such as similarity. We do not need to explicitly complete time-consuming features engineering process.[1]

Bibliography

[1] Dushyanta Dhyani Jayanth Reddy Regatti Meghan Day. *FNC-1: Stance Detection*. `https://dushyantadhyani.github.io/files/fnc-1-stance.pdf`. (Visited on 03/24/2018).

[2] Shanshan Xu Qi Zeng Quan Zhou. *Neural Stance Detectors for Fake News Challenge-Stanford University*. `https://pdfs.semanticscholar.org/0023/9e95f8a60054bf5a93019766385b4a901` `pdf`. (Visited on 03/27/2018).

[3] Andreas Hanselowski. *Team Athene on the Fake News Challenge*. `https://medium.com/@andre134679/team-athene-on-the-fake-news-challenge-28a5cf5e017b`. (Visited on 03/27/2018).