News Stance Detection Report **Qiuru Dai 17011048**

This report follows the order of the code in the coursework file IRDM_Qiuru.ipynb which is slightly different from the order in the assignment description. Pandas and numpy are widely used in this project, while some self-defined functions are written in five python files.

1. Prepare data

First, training set and test set are loaded by using read_csv function from pandas and are stored as dataframe. There four dataframes being used to store train bodies, train stances, test bodies and test stances respectively. The bodies data include body ID and the article bodies. The stances data contain headlines, the corresponding body ID and the stance label of each pair of headline and body.

Then, a process is designed to clean the data which is written as a function called clean_sentence in clean.py. The nltk package and regular expression operations for python are used during this process. This function processes one document (headline or article body) at a time. First, to remove special characters, all characters which are not English letters are replaced by space. Second, the document is tokenized using nltk word_tokenize function. Then, nouns, verbs, adjectives and adverbs in the documents are lemmatized using the WordNetLemmatizer. Finally, all stop words are removed according to the stopwords list in nltk corpus. After this cleaning process, datasets with cleaned documents are stored in dataframes.

By observing the datasets and using the unique function of dataframe, I find that there are 1669 unique article bodies and 1648 unique headlines in the training set. A variable is built to store these 3317 unique documents as the unique collection.

After that, the dataframes store headline and the dataframes store bodies are merged on body ID. In the merged dataframes, each row contains a pair of headline and body and its stance. There are two merged dataframes store training set which has 49972 records and test set respectively.

2. Vector representation and the cosine similarity

For this subtask, three vector representations are built based on term frequency–inverse document frequency (tf-idf) and Word2Vec which are commonly used for vector representation in Natural Language Processing (NLP). The tf-idf value of a word in certain document equals to tf multiplied by idf which can reflect how important a word is to a document in a collection. Word2Vec models can represent each word in documents with a fixed-dimensional vector. All representation models are built based on training set only.

The first two kinds of representation based on tf-idf which represent each document with a 19102-dimensional vector and a 3338-dimensional vector respectively. The unique collection mentioned before is used for the tf-idf calculation. Functions used in this process are written in tfidf.py.

First, the vocabulary of training set is built containing 19102 unique words and each word is assigned an index in the range of 0 to 19101. Then, the idf of each word is calculated on the unique collection using the self-defined function calculate_idf and is saved in the file idf.json. With the idf and the indexed vocabulary, we can use the calculate_tfidf_vector function to get the vector representation. The input of this function is a document, and it returns a vector, whose length is the same as the length of the vocabulary, in which the value of each dimension equals to the tf-idf value of a word. In this case, the length of the vocabulary is 19102, so we have the 19102-dimensional vector representation which is the first kind of representation.

The vectors of the first kind of representation can be very large and sparse, so I select some words with high tf-idf as key words and only use these key words as features. First, the unique collection is looked through. For each headline, words with top 3 tf-idf is selected and for each article body, words with top 30 tf-idf is selected. Second, by using the set function, we have a new key-word vocabulary contains 3338 unique words, and the index is added. Then, the calculate_tfidf_vector_key function is used to give vector representation of documents with the idf and the key-word vocabulary. In this function, the tf-idf value is calculated and put into the vector only when the words are in the key words vocabulary.

To do the representation based on Word2Vec, the open-source modeling tool Gensim is used. For each record in the training set, I connect the headline with the corresponding article body to generate a complete document. There are 49972 complete documents in the training set which are used as training data for the Word2Vec model. The min_count and size, which are parameters of model, are set to 1 and 200, which means the model would consider all words appear in the training collection and give a 200- dimensional vector to represent each word. The vectors of words in the documents are added together to generate the vectors of document and all document vectors are divided by their norm to be normalized. In this process, only the words in the model vocabulary are counted, and the word2vec_vector function is defined to complete this process.

The **cosine similarity** is the cosine of the angle between two vectors which can show the similarity between two non-zero vectors. The formula of cosine similarity is $\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$.

Based on the three kinds of vector representation proposed above, the cosine similarity of each pair of headlines and bodies is calculated using cos_similar function in feature.py. In figure 1, two examples are showed. We can see that the first example whose stance is unrelated has very low cosine similarity on three kinds of vector representation, while the second example whose stance is agree has much higher cosine similarity which suggests the headline and the body are very similar.

	Headline	Body ID	Stance	articleBody	cos_sim	cos_sim_key	cos_sim_w2v
0	[police, find, mass, graf, least, body, near, ...	712	unrelated	[danny, boyle, direct, untitled, film, seth, r...	0.000000	0.000000	-0.137995
1	[hundreds, palestinians, flee, flood, gaza, is...	158	agree	[hundreds, palestinians, evacuate, home, sunda...	0.525937	0.658249	0.775353

Figure 1 Examples of Cosine Similarity

3. Language model based representations and the KL-divergence

The language models used in this subtask are built based on the n-gram language model and interpolation smoothing. The n-gram model is a widely used probabilistic language model, which predicts the next word according to the n-1 words (history) before. To solve the data sparsity problem which is common in NLP area, smoothing technique is need when building n-gram language model. The interpolation smoothing used in this project is a simple and effective smoothing technique, which interpolate between n-gram model and n-1 gram model (MacCartney, 2005).

Two kinds of language models are built in this project which are unigram language model and the interpolation model based on the unigram model and bigram model. To build these models, the class Ngram_lm and Inter_lm are defined in the language_model.py. The class Ngram_lm allow us to create an instance of a n-gram language model based on the training data and a given n order. The probability function of this class can return the appearance probability of a certain word given a certain history. The class Inter_lm allow us to create an instance of an interpolation language model based on two n-gram language model. Two functions, create_unigram and create_bigram, are defined to create the unigram language models and interpolation language models for the programming convenience.

For each pair of headlines and bodies, these two kinds of language models are built. Then, both headline model and body model are used to predict the probability of words in the headline using function lm_vector in language_model.py and the results are returned as vectors. Vectors whose length is the same as the length of headlines are returned for the unigram model, while the vector length of interpolation models equals to the length of headline minus one because the length of history is one. I also tried to use the models to predict bodies to get vectors, but this method is abandoned due to bad performance.

Since we have two pairs of vectors for each pair of headlines and bodies, the KL-divergence can be calculated. The KL-divergence can measure the divergence between two probability distributions.

The formula of KL is $D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i) \ln \frac{Q(i)}{P(i)}$. In this project, i corresponding to words in headline. We can use KL to measure the difference between the headline's language model and the body's language model. The kl_div function is defined in the language_model.py to calculate the KL following the formula. For each pair of headlines and bodies, I calculate two KLs based on unigram and bigram interpolation model respectively. In figure 2, the first two records are shown as examples. High KL value shows larger difference between headline and body.

	Stance	kl_head_uni	kl_head_bi
0	unrelated	12.274821	107.867233
1	agree	3.443514	20.116824

Figure 2 Examples of KL-divergence

4. Propose and implement alternative features

Three kinds of features are proposed and implemented for this subtask, which are Euclidean distance, Spearman correlation and number of same words. Functions used to calculate these features are written in feature.py.

The Euclidean distance measures the straight-line distance between two points in Euclidean n-dimensional space where a n-dimensional vector corresponding to a point. The formula of Euclidean distance is $d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}$. Following this formula, the euc_dist function is defined. It receives two vectors and returns the Euclidean distance.

The Spearman correlation is a non-parametric measure of non-linear correlation which could describe the relationship between two variables. The formula of Spearman correlation is shown below where the x and y corresponding to the rank of original data X and Y.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

The spearman_corr function is built to calculate the Spearman correlation of two vectors based on corr function from pandas.

The number of same words is a self-defined feature, which is the number of unique words that both in the headline and the body. I assume that headlines and bodies tend to be more similar if they have more words in common. The same_word function is written to calculate the same-word value. It receives a pair of headline and body and returns the result.

Since Euclidean distance, Spearman correlation are calculated based on three vector representations from part 2, in total, there are seven features calculated in this part. There are some examples shown below. We can see that the second example, whose stance is agree, gets higher score on Spearman correlation and same-word value and lower value on Word2Vec based Euclidean distance, which suggests higher similarity. These features seem be able to show the difference between different stances. It also gets higher value on tf-idf based Euclidean distance, which is unexpected.

	Stance	euc_dist	euc_dist_key	euc_dist_w2v	spear_corr	spear_corr_key	spear_corr_w2v	same_word
0	unrelated	1.110189	0.956684	1.505255	-0.001715	-0.005329	-0.148394	0.0
1	agree	1.508805	1.388314	0.685887	0.193569	0.368627	0.733944	7.0

Figure 3 Examples of other Features

5. Analysis of two important features

The cosine similarity based on tf-idf with key words and KL-divergence based on unigram are plotted and analyzed in this part. By observing all features' distributions, I find that all features based on Word2Vec show similar distributions and cosine similarity and Spearman correlation based on both kinds of tf-idf show similar distribution. The KL based on unigram and same-word are relatively unique. Most of the features seem be able to show the difference between unrelated and other kinds of samples. The Euclidean distance based on both kinds of tf-idf and KL based on bigram interpolation model seem not be able to show the difference between four kinds of stances.

Therefore, the cosine similarity based on key-tf-idf is chosen as a representative of these similar features, which seems to have slightly better performance than others, and the unigram-KL is chosen as a unique good performance feature.

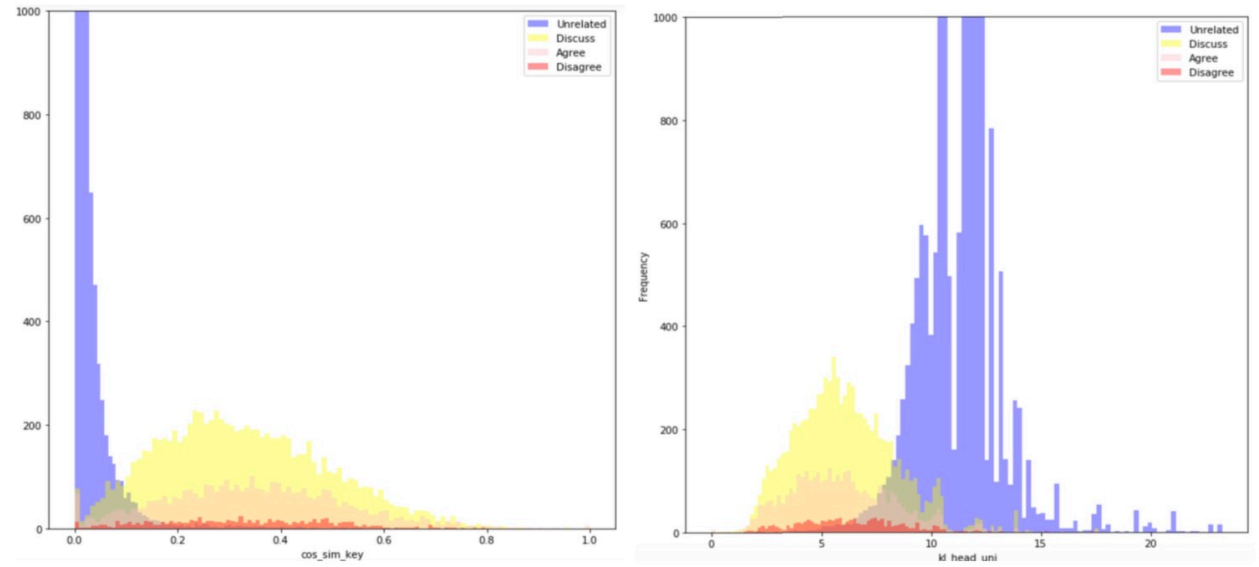

Figure 4 the Distributions of cosine similarity based on tf-idf with key words (left) and KL-divergence based on unigram (right)

From figure 4, we can see that, for both features, the distribution of unrelated samples shows significant difference with other three distributions, while the discuss, agree and disagree samples may not be able to be distinguished by either feature. Since these two features are generated by different methods, it may be good to combine the features.

6. Split the training set into a training subset and a validation subset

The training set is split in to a training subset and a validation subset before building stance classification models. The training data is shuffled and cut at the 0.9 percentile to make two subsets. The ratio of the four classes of is similar as shown below.

	agree	disagree	discuss	unrelated
Training set	0.07360121668134155	0.016809413271432	0.1782798367085568	0.7313095333386697
Training subset	0.07321845469705392	0.01698721511951084	0.17881045025013897	0.7309838799332963
Validation subset	0.07704622773664198	0.015209125475285171	0.17350410246147688	0.7342405443265959

7. Implementation of linear regression and logistic regression

The linear regression and logistic regression (LR) models are implemented using gradient descent. Gradient descent is an optimization algorithm to find the minimum of a function. To optimize a model, first, we need to calculate the cost function and the gradient of cost function. Then, apply the gradient descent to update the parameters of model and minimize the cost function. When the cost function is minimized, usually the performance of a model tends to be optimized and we can have the final parameters. This function below shows how can we update the parameters θ for linear regression and LR using gradient descent in which h(x) corresponding to the linear function and sigmoid function respectively.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

The functions used to implement linear regression and LR are defined in the model.py, including the preprocess of input (normalize the input and add bias item), cost and gradient calculation of both regressions and the gradient descent process for both regressions.

Moreover, the original linear regression and LR can only deal with binary classification. Since we have four classes to be classified, the models are implemented in one vs all method. For each class, a linear and a logistic model are trained using labels that only show if certain record belongs to this class. Combining the prediction results of models for four classes, we can choose the class with highest prediction result as our final prediction. The function onevsall is defined to complete this process.

8. Model evaluation

To evaluate the models, accuracy, precision, recall and f1-score are used as evaluation metrics. Accuracy show the rate of correct prediction, which can provide an intuitive feeling of model performance, but it may not be a good measure of model performance due to the class imbalance in the datatset. To deal with this problem, precision, recall and f1-score are used. The precision is the fraction of relevant instances among the retrieved instances, while recall which is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Usually higher precision means lower recall, and the f1-score combines these two metrics can show the overall model performance.

On the test set, the LR model achieves 0.82 accuracy and the linear model achieve 0.83 accuracy. The rest of evaluation results is shown below.

	Precision(linear)	Recall(linear)	F1-score(linear)	Precision(LR)	Recall(LR)	F1-score(LR)
agree	0	0	0	0	0	0
disagree	0	0	0	0	0	0
discuss	0.63	0.68	0.65	0.65	0.57	0.61
unrelated	0.88	0.99	0.93	0.85	0.99	0.91
avg / total	0.75	0.83	0.79	0.73	0.82	0.77

We can see the although the linear model achieves slightly higher accuracy, the LR model shows better performance on precision, recall and f1-score. Due to the class imbalance problem, precision, recall and f1-score can better reflect the classification ability of model. Therefore, the LR model have better performance than linear model.

Both models can work properly when the learning rate is set in the range of 0.1 to 0.5. When the learning rate is larger, the less iterations are needed to optimize the model. However, when we set the learning rate to a quite large number like 50, the gradient descent cannot work properly and the cost function cannot be minimized.

9. Features importance

In total, 12 features mentioned before are used for the models. The parameters value for these features are plotted in the figure 5 and 6.

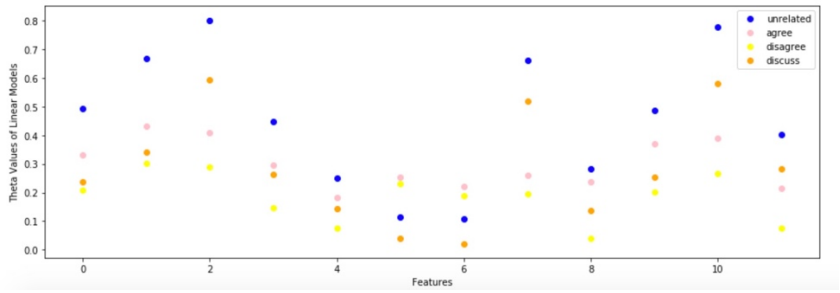

Figure 5 the Parameters of Linear model

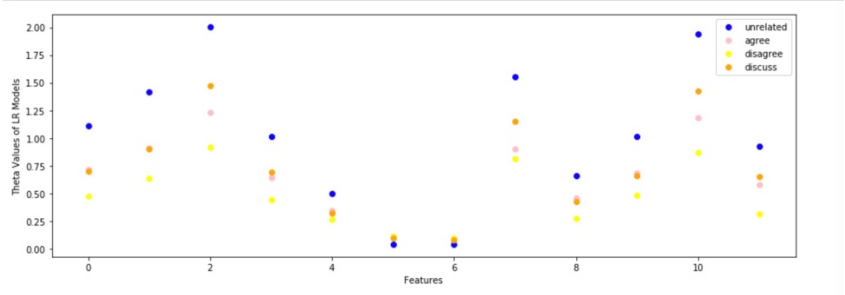

Figure 6 the Parameters of LR

The weights or parameters of features show their importance for the models. We can see that the distributions of parameters for linear and LR model are very similar. For both model, the features whose index are 2, 7, 10 have high weights, which are Word2Vec cosine similarity, Word2Vec

Euclidean distance and Word2Vec Spearman correlation. The features 4, 5, 6, which corresponding to bigram-KL, tf-idf Euclidean distance and key tf-idf Euclidean distance, have the lowest weight. This result is similar to the result from distribution observation. Also, Word2Vec seems to have better performance than tf-idf.

10. Literature review

Support vector machines (SVM) is most commonly used in the stance detection task, for example, it is used by (Küçük, D., 2017) and (Mohammad, S., 2016). Moreover, random forest, and gradient boosting decision trees (GBDT) are proposed by Liu et al. (2016). Thorne et al. (2017) propose an ensemble system which can show better performance than each individual model.

11. The improvement of models

Due to the class imbalance, the classes with very little samples are hard to be predicted. To deal with this, I give different threshold to each model when doing the classification, for example, we can predict all samples whose predicted results larger than 0.3 as positive samples instead of 0.5 for LR. The final results tend to predict the sample as a minority class. The threshold values are tuned using validation and the model is tested on the test set. This process is implemented by the onevsall_new function and tends to be able to improve the f1 score and decrease accuracy slightly.

The down-sampling method is used to try to deal with the same problem, which means the negative samples (majority samples) are sampled at given rate and all positive sample are kept. This sample will be used as new training data. After trying different down sampling rate on the validation set, the rate is set to 0.3 for agree and 0.05 for disagree. From the table below, we can see the linear model shows better ability in classification of minority classes, and the performance is improved.

	Precision(linear)	Recall(linear)	F1-score(linear)
agree	0.31	0.03	0.06
disagree	0.09	0.31	0.14
discuss	0.71	0.38	0.50
unrelated	0.89	0.99	0.93
avg / total	0.79	0.79	0.77

Moreover, GBDT model is implemented using XGBoost which is an open-source package providing gradient boosting framework. Being different from the linear and LR model, GBRT is a non-linear model which could learn the non-linear features, which is difficult to use the feature engineering to achieve. Initially, the XGBoost model is trained using all training data and has bad performance. Then, the downsampling method is used, which means 20% unrelated samples and all other classes samples are used as training data. After downsampling, the XGBoost model shows good performance. The evaluation results are shown below. We can see that Xgboost model shows better classification ability in general and have higher overall performance than linear and LR model.

	precision	recall	f1-score	support
agree	0.32	0.37	0.34	1903
disagree	0.09	0.03	0.04	697
discuss	0.58	0.54	0.56	4464
unrelated	0.94	0.96	0.95	18349
avg / total	0.81	0.82	0.81	25413

Figure 7 performance of XGBoost

Moreover, the feature importance of XGBoost is quite different from linear and LR models, for example, the Word2Vec Euclidean distance becomes unimportant and tf-idf Euclidean distance becomes very important. This may show the difference between linear and non-linear models. The feature importance is not shown due to space limitation.

Reference

MacCartney, B. (2005). NLP Lunch Tutorial: Smoothing. [online] Nlp.stanford.edu. Available at: https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf [Accessed 9 Apr. 2018].

Liu, C., Li, W., Demarest, B., Chen, Y., Couture, S., Dakota, D., ... & Steimel, K. (2016). Iucl at semeval-2016 task 6: An ensemble model for stance detection in twitter. In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016) (pp. 394-400).

Küçük, D. (2017). Stance detection in Turkish tweets. arXiv preprint arXiv:1706.06894.

Mohammad, S., Kiritchenko, S., Sobhani, P., Zhu, X., & Cherry, C. (2016). Semeval-2016 task 6: Detecting stance in tweets. In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016) (pp. 31-41).

Thorne, J., Chen, M., Myrianthous, G., Pu, J., Wang, X., & Vlachos, A. (2017). Fake news stance detection using stacked ensemble of classifiers. In Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism (pp. 80-83).