First: combine text files into one csv file

Firstly, I Combine all text files into one csv file and give each review a sentiment value (pos->1, neg->0). Then shuffle the order of the reviews.

	review	sentiment
2	Hard to categorize the film - perhaps it's an avant garde spy thriller? Mother Ni	1
3	too predictable for spoilers, but i'll not be cagey below, so don't read it if you c	0
4	You can't really blame the movie maker for glorifying Che because the industr	0
5	The only reason to give this movie even a single star is how much the ending r	0
6	After reading the first 5 reviews on IMDb I was very enthusiastic about this mc	0
7	I like Wes Studi & especially Adam Beach, but whoa is this movie a load of pr	0
8	It is playing on SHOWTIME right now but is going to be released as a movie c	0
9	Not only is this film entertaining, with excellent comedic acting, but also intere	1
10	We were excited to rent this one after reading a few reviews and seeing that it s	0
11	Imagine you have the opportunity to see yourself again as a kid. Now think wh	1

Fig.1: The combination of reviews with randomly shuffled.

Second: data cleaning

Clean the data. That is, I remove some unimportant or disturbing elements (for example, punctuations, URLs, other symbols) in reviews but keep emoticons which are helpful for learning the emotion of the review. Then I converte all the text to lower case, and extra blank spaces are removed.

Furthermore, in the preprocessing step, I implement stemming process. Stemming techniques allows us to consider in the same way nouns, verbs and adverbs that have the same radix. For example, in row 6, verb “reading” is changed to “read” after stemming. By calling the PorterStemmer library from Natural Language Toolkit(NLTK), we could realize the stemming process. However, this method may take a long time to work.

Finally, I filter out stop words (e.g., “After”, “the, “on” on row 6 are removed from the content.) because they can lead to a less accurate classification. By doing this, I can also obtain smaller size of dataset. The stops words I use are defined and downloaded from NLTK.

	review	sentiment
2	['hard', 'categor', 'film', 'perhap', 'avant', 'gard', 'spi', 'thriller', 'mother', 'night', 'veri', 'good', 'vonnegu	1
3	['predict', 'spoiler', 'cagey', 'read', 'care', 'few', 'dull', 'scriptwrit', 'togeth', 'half', 'afternoon', 'even', 'rur	0
4	['can', 'realli', 'blame', 'movi', 'maker', 'glorifi', 'che', 'becaus', 'industri', 'money', 'stori', 'hear', 'thi', 'fr	0
5	['onli', 'reason', 'give', 'thi', 'movi', 'even', 'singl', 'star', 'how', 'much', 'end', 'made', 'laugh', 'had', 'higl	0
6	['read', 'first', '5', 'review', 'imdb', 'wa', 'veri', 'enthusiast', 'thi', 'movi', 'realli', 'aw', 'movi', 'total', 'time'	0
7	['like', 'studi', 'especi', 'adam', 'beach', 'whoa', 'thi', 'movi', 'load', 'pretenti', 'ponder', 'slow', 'overli', 'c	0
8	['play', 'showtim', 'right', 'go', 'releas', 'movi', 'call', 'three', 'ha', 'releas', '2006', 'mess', 'includ', 'suppo	0
9	['onli', 'thi', 'film', 'entertain', 'excel', 'comed', 'act', 'also', 'interest', 'polit', 'wa', 'made', 'end', 'soviet', '	1
10	['excit', 'rent', 'thi', 'one', 'read', 'few', 'review', 'see', 'score', 'highli', 'well', 'got', 'home', 'could', 'belie	0
11	['imagin', 'have', 'opportun', 'see', 'again', 'a', 'kid', 'think', 'would', 'happen', 'chanc', 'speak', 'younge	1

Fig.2: Review content after doing data cleaning.

Third: data vectorization

Finding vectors of texts because computer cannot read “texts” but “numbers”. In this preprocessing phase I use CountVectorizer combine with TfidfTransformer to act as my preprocessor. CountVectorizer tokenizes every content and every text in every review by comma. Then it counts the number of occurrences of every token. TfidfTransformer reweight every token in the corpus. Some words (e.g., “have”, “could”, etc.), are not filtered out in the data cleaning process as stop words, may appear many times but does not contribute much for sentiment analysis. This may further occur the ignorance of some terms which are important but show up less. TfidfTransformer normalizes every term and calculates weight of each feature base on the result. Tf is for term-frequency and idf means inverse document-frequency, tf-idf denotes tf multiplies idf.

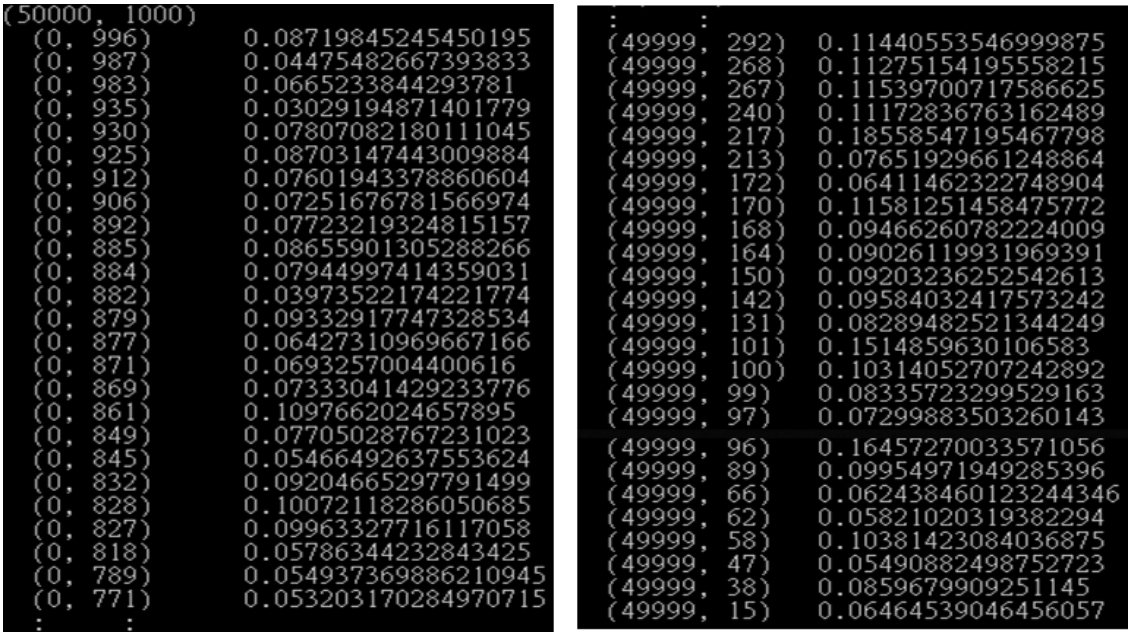

Fig. 3: The output sparse matrix of TF-IDF with max_features = 1000.

The output of CountVectorizer and TfidfTransformer is a sparse matrix, which can make computation faster and data size smaller. I save the output in a npz file. Firstly, I got 73698 features (there are 73698 tokens) from the corpus and this make me to wait very long time when I do the following machine learning process. Therefore, I set the max_features = 5000 and max_features = 1000 to reduce the computation cost. Setting the max_features means that I choose the top 5000 or 1000 features which appear frequently.

Forth: build model and train dataset

For classifying whether the review is negative or positive, I adopte two kind of machine learning methods, which are deep learning and classification.

For deep learning method, I build model with multi-layer perceptron and recurrent neural network (RNN). I encode the target label (i.e., y, sentiment value) into binary category. For both models, I give them train dataset, validation dataset and test dataset with 28,000 data, 7,000 data and 15,000 data, respectively. And use categorical_crossentropy as a loss function and adam as an optimizer.

For classification method, I adopte support vector machine(SVM). The number of data for training is 35,000 and for testing is 15,000.

	Train_score	Val_score	Test_score	Model	Note
ANN (keras)	0.9986	0.8741	87.3%	128 relu 256 relu 128 relu 2 softmax batch_size=200 epochs=10	max_ features = 5000
	0.96	0.86	85.4%		max_ features = 1000
LSTM(keras)	0.96	0.86	86.3%	64 relu 32 relu 2 softmax batch_size=100 epochs=30	max_ features = 5000
	0.97	0.86	85.3%		max_ features = 1000
SVM(sklearn)	0.86	-	85.1%	kernel="linear" C=0.025	max_ features = 1000
	0.86	-	85.5%	kernel="linear” C=0.025	max_ features = 5000
	0.92	-	88.3%	kernel="linear" C=0.8	max_ features = 5000
	1.0	-	50.6%	kernel='rbf'	max_ features = 1000

				C=0.8 gamma=20	
	0.50	-	49.7%	kernel='rbf', C=0.025, gamma=20	max_ features = 1000

Table 1: Results of different methods to prediction or classification.

As we can see in multi-layer perceptron, so-called ANN, I have utilized four fully connected layers. I give the model the datasets with max_ features = 5000 and max_ features = 1000, separately. The computation time of model with each dataset took respectively are close and fast. Therefore, there is no need to tradeoff the accuracy to the computation cost. I observe that the more epochs it goes, the higher the train_score gets but lower val_score gets. This means the phenomenon of slightly overfitting.

Following is RNN method. I adopt long short term memory network (LSTM) to realize the method. For LSTM model, the first two layers are using LSTM and ends with one fully connected layer. Different from how ANN works, LSTM works obviously much slower with the max_ features = 5000 dataset so the tradeoff between the accuracy and the computation cost is worthy. Similar to ANN, I observe that the more epochs it goes, the higher the train_score gets but lower val_score gets.

The only classification method I use is SVM. One of the biggest cons of SVM is that it took so much time to build and learn model and classify data. Especially when the kernel is rbf, which means that it need to additionally calculates each feature with Gaussian.

However, as we can see in the table 1, ones with kernel in linear got much better results. The ones with kernel in rbf got very bad classification result. Especially when C=0.025, the model just classifies every data to positive (i.e., sentiment = 1) no matter in training process or testing process. This may because the parameter C is too small so that the classifier is too rough to classify which could also be called underfitting. Unlike the one with C=0.025, the one with C=0.8 has higher C so it got 100% classification in training dataset but due to overfitting, when giving testing data into the model, it only got 50.6% accuracy.

Ones which kernel in linear are working faster than in rbf but still slow. Therefore, according to the test result of test_score of both max_ features = 5000 and max_ features = 1000. It is doubtlessly worthless to use max_ features = 5000 dataset to build the model. Moreover, when I change parameters C from 0.025 to 0.8 it did not result in overfitting and get better test_score instead.

References:

- Data cleaning:
  1. http://ceur-ws.org/Vol-1748/paper-06.pdf
  2. https://loperntu.gitbooks.io/ladsbook/content/cleaning.html
  3. https://hk.saowen.com/a/68ecb0527ba9a3332d4f60e466d95f955fb7182cbdcd8234507df65fd9c4a084
- Data vectorization:
  1. https://blog.csdn.net/m0_37324740/article/details/79411651
  2. https://hk.saowen.com/a/68ecb0527ba9a3332d4f60e466d95f955fb7182cbdcd8234507df65fd9c4a084
  3. http://d0evi1.com/sklearn/feature_extraction/
  4. https://blog.csdn.net/m0_37324740/article/details/79411651
- SVM:
  1. https://scikit-learn.org/stable/modules/svm.html#svm-kernels
  2. https://www.cnblogs.com/luyaoblog/p/6775342.html
  3. https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
  4. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
- ANN, LSTM:
  1. https://www.jianshu.com/p/6b16b592b08d
  2. https://www.liip.ch/en/blog/sentiment-detection-with-keras-word-embeddings-and-lstm-deep-learning-networks
  3. https://keras.io/models/sequential/
  4. https://keras-cn.readthedocs.io/en/latest/layers/embedding_layer/