CS273A Machine Learning

IMDB MOVIE REVIEW Project Report

Yanqi Gu

Zhijian Li

Ziyang Zhang

1. Introduction

In this project, we aim to apply different machine learning and deep learning methods to solve task of sentiment analysis of movie reviews. More specifically, it will try to input a movie review into a model and predict whether it is a positive or negative movie review. Our sentiment analysis project aims at using raw movie review text with associated labels from IMDB to precisely classify phrases on a scale of 2 classes: negative, positive. The challenging part of this task is to deal with obstacles like sentence negation, abbreviation, language ambiguity, and metaphors.

2. Data analysis

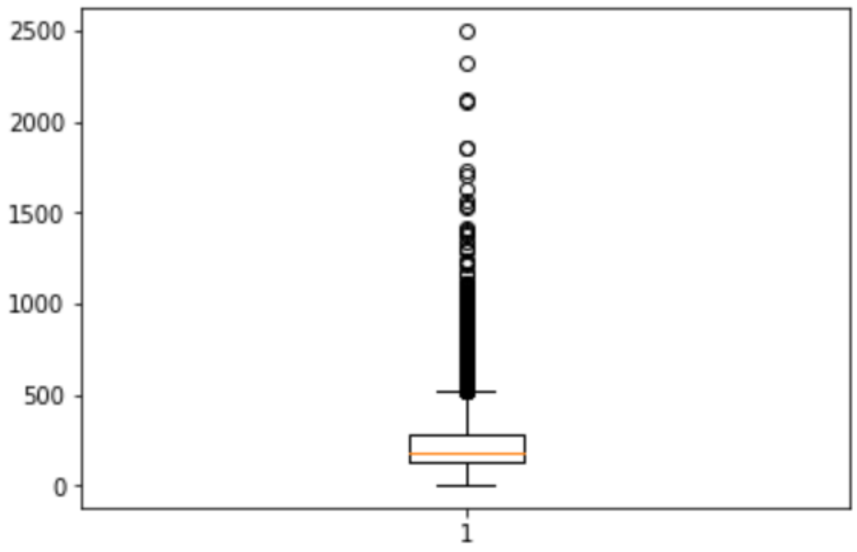
2.1 Data preprocess (using keras and numpy):

From the picture, we can see that although the green point tends to have a higher positive score while the red point tends to have a higher negative score, there are still many red points with higher positive scores and green points with higher negative scores. This is not a good evaluation.

On one hand, we replace the words with integers that indicate the absolute popularity of the word in the dataset. The sentences in each review are therefore comprised of a sequence of integers. On the other hand, we also provides additional arguments including the number of top words to load (where words with a lower integer are marked as zero in the returned data), the number of top words to skip (to avoid the "the"'s) and the maximum length of reviews to support.

Using numpy.concatenate(), we check the shape of the training dataset: x.shape: (50000,) y.shape: (50000,). Also we print the unique class values: numpy.unique(y): [0 1]We can get total number of unique words in the dataset with len(numpy.unique(numpy.hstack(X))): 88585, which means there are actually not many words used in dataset.Also we get information about review length (mean:234.76 std:172.91)

Take a look at the plot of review length in words for the dataset, we can see that an unobvious exponential distribution that we can cover the mass of the distribution with a clipped length of 400 to 500 words.



Review length in words for dataset

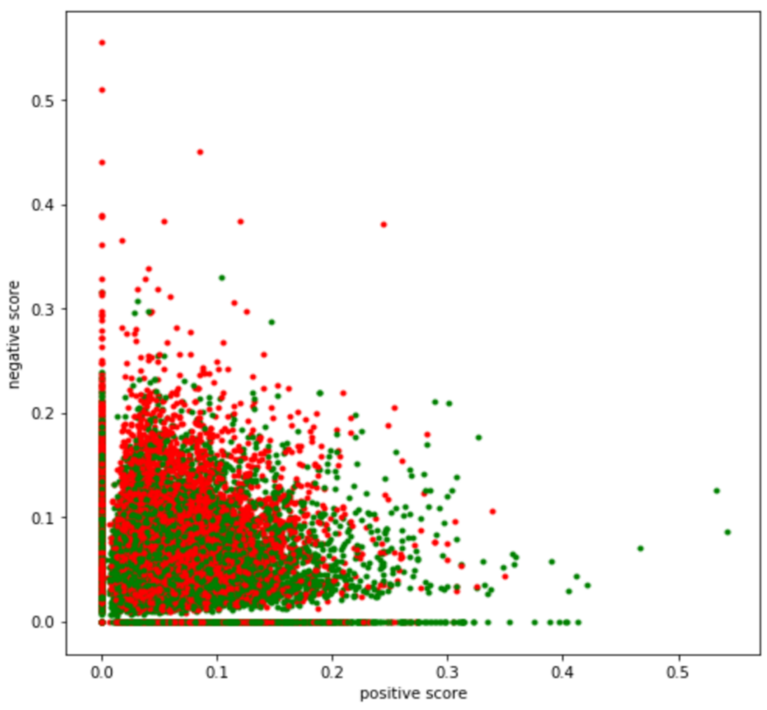
2.2 Text Topic Exploration

Since our data set contains two classes of movie reviews: positive ones and negative ones, we can assume that each one in a class have similarities with others. Because they are reviews so each class must share the same topic on their feelings about movies. Then we can evaluate the texts using high frequency words that are strongly correlated with the corresponding topic.

To quantify the topic of the text on positive or negative, firstly we calculate the frequency for each word in the data set and select some high frequency words with strong emotional characteristics that related to the topic, such as 'like', 'good', 'great', 'not', 'no', 'never'. Then we use term frequency–inverse document frequency (TF – IDF) to calculate the importance of all words with strong emotional characteristics in different reviews. By using TF-IDF, we can evaluate the importance of the words in each text and correct the word eigenvalues expressed only by words frequencies.

Since we have got the weight for each word in a review, we can calculate the scores on whether this review perform better as a positive review or negative review. We define the positive score for one review will be the sum of weight for words with positive feelings 'like', 'good', 'great', and the negative score equals the sum of weight for words with negative feelings 'not', 'no', 'never'. Hypothetically, positive score should be higher than negative score for a positive review while negative review will have higher negative reviews.

In the picture, all the green points represent positive reviews and red points represent negative ones.



Positive and Negative scores for Test Reviews

From the picture, we can see that although the green point tends to have a higher positive score while the red point tends to have a higher negative score, there are still red points with higher positive scores and green points with higher negative scores.

This indicates that the using review topics to classify the reviews in not a good option, which means that these film reviews are highly personalized and cannot be easily classified by the positive and negative emotion tendencies of high frequency words. We need some more advance method to classify the movie reviews.

3. Sequence Model

3.1 Word Embedding

Word embedding is a useful method in the field of natural language processing . It's a technique where words are encoded as real-valued vectors in a high-dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space. Discrete words are mapped to vectors of continuous numbers. This is useful when working with natural language problems with neural networks and deep learning models. Two popular examples of learning word embedding are Word2Vec and GloVe. We would like to use a word embedding representation for the IMDB dataset as part of a deep learning model.

3.2 Sequence Models

This indicates that the using review topics to classify the reviews in not a good option, which means that these film reviews are highly personalized and cannot be easily classified by the positive and negative emotion tendencies of high frequency words. We need some more advance method to classify the movie reviews.

I choose sequence models because:1) feedback networks and CNN perform poorly on sequence data since feedback networks and CNN take a fixed length as input while the sentence we're dealing here can't have same length. I don't want to pad all inputs to a fixed size.2) RNN can perform better since conventional models can't understand the context of input while sequence models can derive relations from the previous words.

In this section, to improve performance of model, we try different sequence models: RNN, Bidirectional RNN, LSTM, GRU. And we want to combine them together. We choose keras to run the backend.

We first try RNN, LSTM and GRU. The performance is actually pretty good, all three methods achieve precision over 80%. However, it turns out that they're all not as good as bidirectional RNN. The reason is simple, RNN, LSTM, GRU share a same problem that sometimes we not only want to learn representations from previous words, we also want to learn from future words to understand context more precisely. For example, "The old man said, Captain Kirk is awesome" and "The old man said, Captain America is dead in Avenger IV" have different meanings and if you only consider precious words of "captain", you can't understand whether the old man is talking about Star Trek or Marvel heroes.

The final model of bidirectional RNN we trained is as follow:

Sequential
Embedding(max_features=6000, embed_size=128)
Bidirectional(LSTM(32))
GlobalMaxPool1D
Dense(20, active_func="relu")
Dropout(0.05)

Dense(1,active_func=”sigmoid”)

Deep learning model based on embedding and bidirectional RNN

To compile, we choose adamboost as optimizer, use binary_crossentropy as loss. I also set batch_size=100, epochs=3 to train faster.

The reason to pick these parameters is based on the previous analysis of dataset and the training performance. The max_feature set to pick is between 1000 to 10000, the embed_size set to pick is 32, 64, 128, 256. Based on actual training performance, I pick 6000 and 128 as final choice.

The combination of Dense(“relu”) and Dense(“sigmoid”) layer is to make sure we introduce non-linear properties into neural network. I use Dropout layer to prevent overfitting. The precision of training dataset is 98%.

The final precision for test is 0.9593, which is the highest this model can achieve.

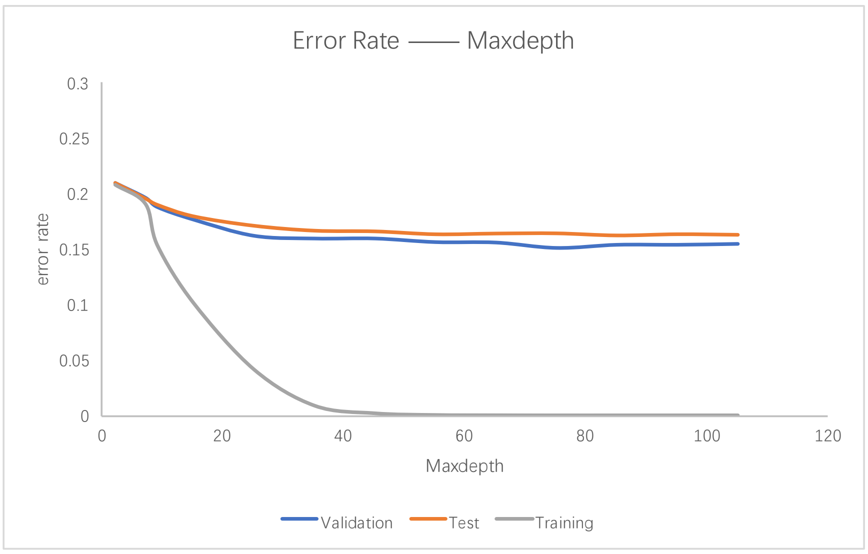
4. Random Forest

4.1 Random Forest Model

Random Forest is an algorithm that integrates multiple trees using the idea of integrated learning with decision trees as basic units. Random Forest algorithm belongs to Ensemble Learning, which is a branch of machine learning. As a basic unit, each decision tree is a classifier. For an input sample, N decision trees will have N classification results. The Random Forest algorithm integrates all the classification voting results, and use the category with the most votes as the final output. This is the simplest Bagging idea.

In this project, we use Keras to preprocess the data and transform the text data into numerical feature matrix. Beside features, there are some parameters will affect the accuracy of the prediction. First one is the maximum number of features that the Random Forests algorithm allows a single decision tree to use. The second one is the number of estimators in the Random Forest algorithm. The last one is max depth for a single decision tree in the forest.

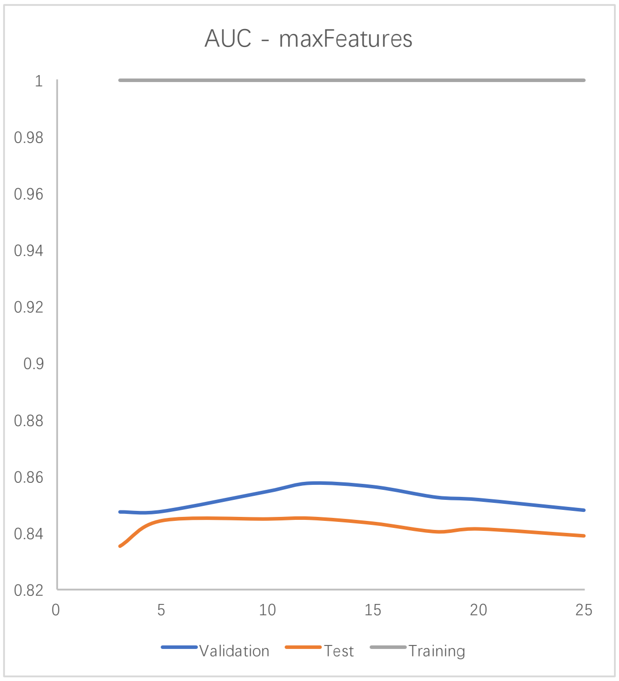
Firstly, we explore how max decision tree depth affect the RF model.



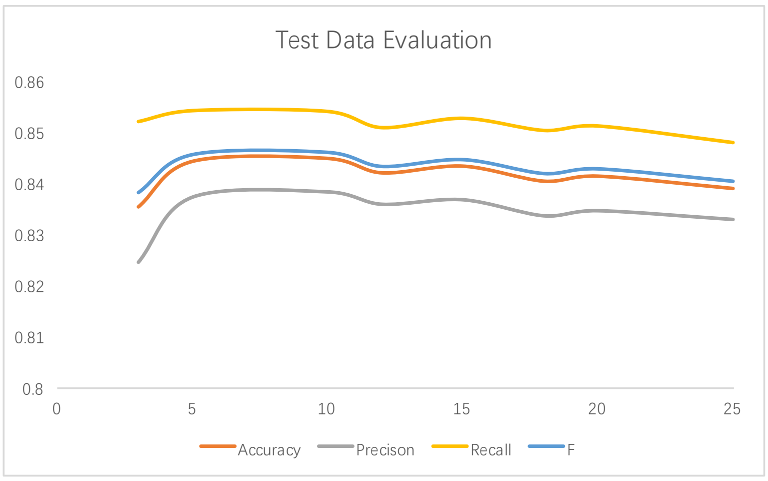
Error Rate of Training, Validation and Test with Different Max Tree Depth

As we can see, when the max decision tree depth is larger than 50, the error rate for both validation data and test data become very stable and the training error rate becomes zero. So, setting max decision tree depth to 60 will be a good option with low error rate while the complexity is not high.

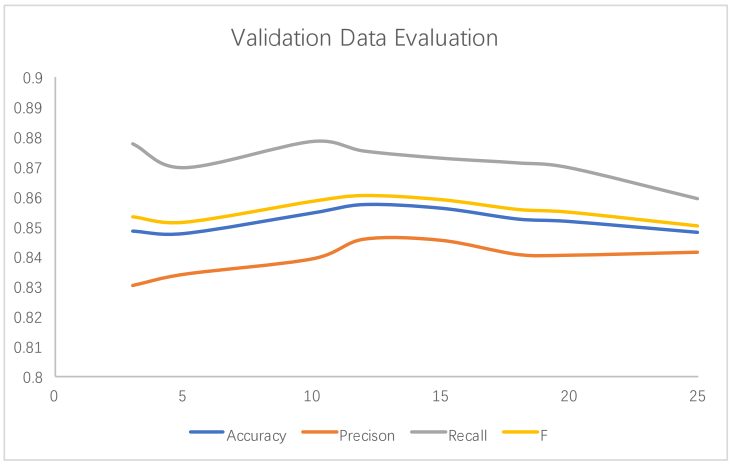
Then, the max feature number for the whole RF model is evaluated. The evaluation result is as follows:



AUC of Training, Validation and Test with Different Max Features



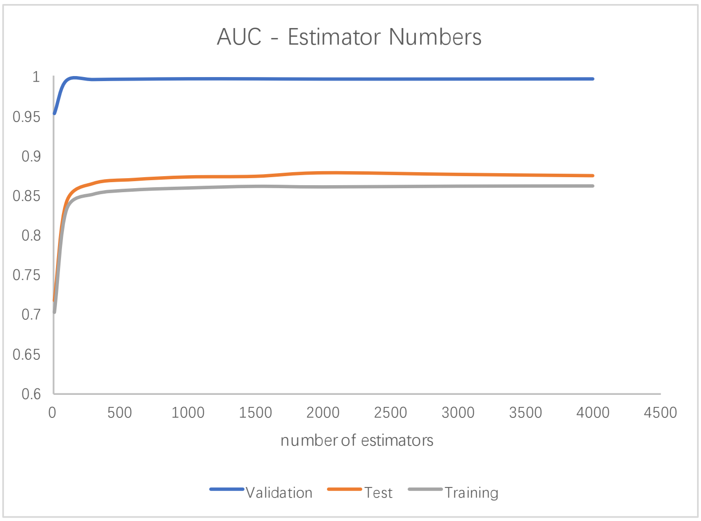
AUC, Precision, Recall and F of Test Data



AUC, Precision, Recall and F of Validation Data

Increasing max feature number generally improves the performance of the model because at each node we have more options to consider. However, sometimes, as it reduces the diversity of individual trees, which is the unique advantage of random forests. Also, more features will decrease the calculating speed. Therefore, an appropriate max feature is very necessary. Since movie review is a binary classification, Precision, Recall and F is also applied in the evaluation. From the evaluation, we can see that when the feature number increase from 3 to 25, the AUC of the validation data and test data begin to increase and then decrease which means it's over-fitting. So, setting max feature number as 12 is more appropriate.

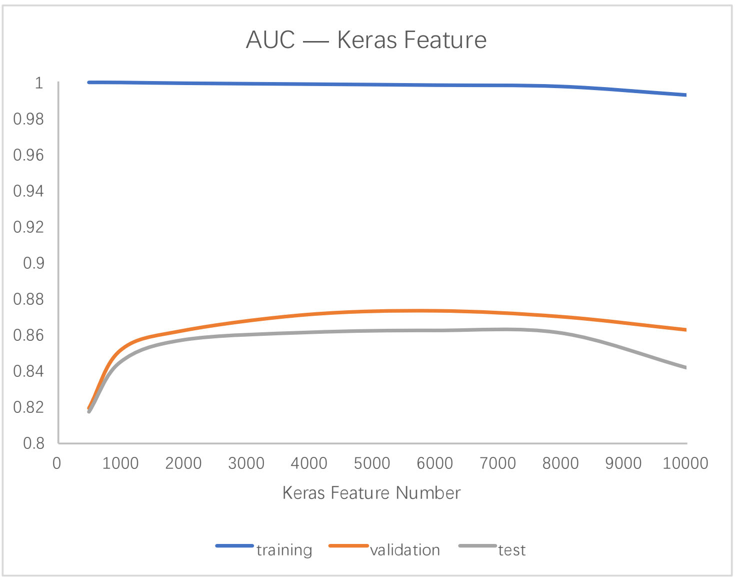
As for estimator number in the number, generally more subtrees can make the model perform better, but at the same time may make the code slower.



AUC of Training, Validation and Test with Different Number of Estimators

As the result shown in the picture, when estimator number is larger than approximately 700 to 800, the AUC of validation data and test data increase very little as the number of estimator increase. So, using 1000 estimators is a good choice.

4.2 Feature Optimization



Since we must set the feature numbers for Keras to preprocess the text data and the output is the matrix with the same feature numbers, this will also effect the prediction result of the Random Forest model. As shown in the picture, if there are too much feature in the matrix, there will be over-fitting. So the number of words we use as features will be around 5000 to 6000.

4.3 Overfitting

As we discuss in the model, a too high max feature number in RF model and too much feature for feature-document matrix will cause overfitting and we have chosen the more appropriate parameters.

4.4 Conclusion

In general, we use max tree depth 60, max feature number 12 and estimator number 1000 as parameters for the Random Forest model (RandomForestClassifier (n_estimators=1000, max_depth=65,max_features= 12,random_state=0))with 6000 words as features to convert text data into matrix. And the prediction AUC for validation data and test data are 87.2% and 86.3%.

5. Multilayer Perceptron Classifier

5.1 Document feature extraction

tf-Idf:

Tf means term-frequency, and Idf is inverse document-frequency.

$\mathrm{tfidf}(t, d) = \mathrm{tf}(t, d) \times \mathrm{idf}(t),\ \ \mathrm{idf}(t) = \log\frac{1+n_d}{1+\mathrm{df}(d,t)} + 1,$

Where $\mathrm{tf}(t, d)$is the number of times term t appears in document d, $n_d$is the total number of documents and $\mathrm{df}(d, t)$it the number of documents containing t.

Finally, we normalize our the vector: $v = \frac{v}{\sqrt{{v_1}^2 + \ldots + {v_n}^2}}$.

countvectorizer:

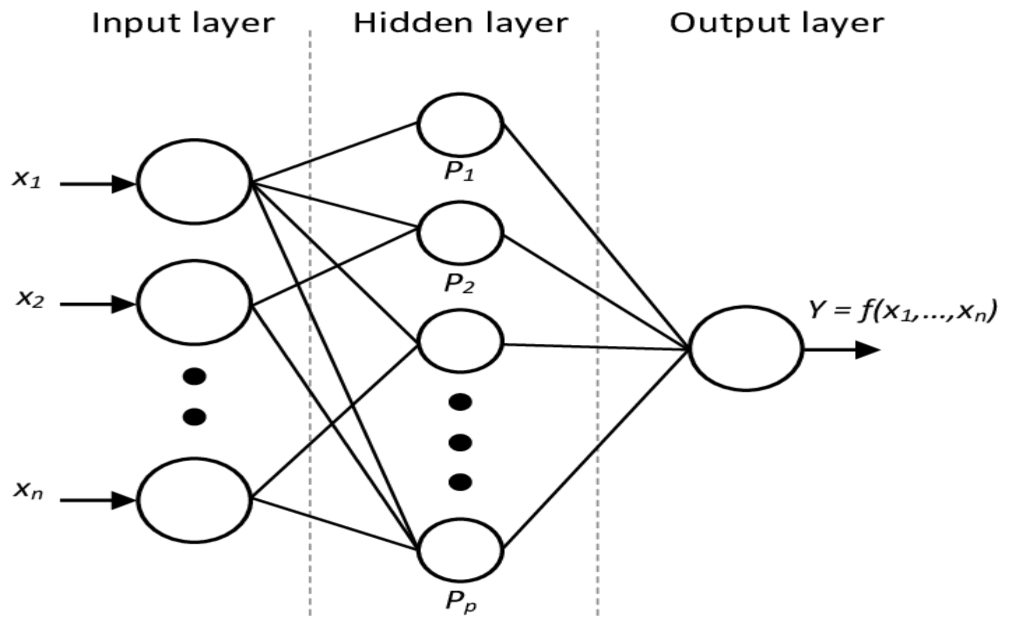
Scikit-learn also provides another vectorization method called countvectorizer. This method gets generates a list containing every distinct word in documents: feature=[word 1, word 2,.....,word n]. The it vectorizes every documents to a vector $v_d = [v_1, \ldots\ldots, v_n]$, where $v_i$is the number of times word i appears in document d.

Lemmatization: since the methods above generate each word as a feature, words like ‘is’, ‘are, ‘be’, and ‘was’ will be considered as different features. Therefore, we use Lemmatization to reduce the features. Our goal is not to reduce the time cost since it is already small under sparse computation. Instead, we hope lemmatization can reduce noise of our data and improve our results. However, we find lemmatization changes the result very little.

Both tf-idf and countvectorizer return a 25000 by 92715 scipy sparse matrix (csr format) with 3439788 nonzero elements, and with lemmatization countvectorizer reduces the matrix to 25000 by 85362 scipy sparse matrix with 3461902 nonzero elements. The matrices are very large. However, take advantage of sparse matrix computation, we can get the training done with low time cost. We compare the two vectorization methods, and it turns out that tf-idf gives a better performance.

5.2 Model: MLPClassifier

Multilayer perceptron classifier is a neural network. A visualization of a 3-layer MLPClassifier:



Scikit-learn MLPClassifier has loss function:

$$Loss(\hat{y}, y, w) = \sum(y\, ln\hat{y} - (1-y)ln(1-\hat{y})) + \alpha||w||^2$$

Where $y \in \{0,1\}$ is the true vary of class, and $\hat{y}$is the predicted probability that y_prediected=y. We use adamoptimizer to do stochastic gradient descent.

The following table shows our results of prediction, where CV is countvectorizer, CV+Lemma is countvectorizer+lemmatization

epoch	train(CV+Lemma)	test(CV+Lemma)	train(tf-idf)	test(tf-idf)	train(CV)	test(CV)
1	0.90308	0.86608	0.5	0.5	0.9562	0.87548
2	0.94504	**0.8829**	0.91112	0.8686	0.965	0.8814
3	0.97056	0.8816	0.95652	0.88052	0.98232	**0.88208**
4	0.98008	0.87668	0.9748	0.88364	0.98736	0.87792
5	0.98752	0.87324	0.98512	**0.88552**	0.99464	0.86952
6	0.99056	0.87188	0.9918	0.88264	0.99624	0.8648
7	0.99284	0.8664	0.99144	0.87964	0.99612	0.85416

6. Conclusion

Model	TrainingAUC	TestAUC
Sequence Model	0.98	0.9593
Random Forest	0.99	0.86
MLPC	0.9815	0.88552

As we can see, data is not linearly dependent on parameters so some models are not taken in consideration. Word embedding turns out to be very efficient in processing data. Also Bidirectional RNN stands out in sequence model family considering performance. Also it takes little time to train.

As for Random Forest, decreasing the number of features will reduce the correlation and classification ability of the tree. Increasing feature numbers will also increase the correlation and classification ability of the tree. The best input parameters will be RandomForestClassifier (n_estimators=1000, max_depth=65,max_features=12,random_state=0).

Also, we find that, in MLPClassifer, tf-idf slightly outperforms countvectorizer, and countvectorizer causes overfitting very quickly. Since the keras processing only keep top several thousand words, but the two methods in section 5.1 keep all the information in a sparse matrix, we find that the training of all information does not necessarily outperform that of partial information.

7. Appendix

https://scikit-learn.org/stable/modules/feature_extraction.html
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#inter
https://keras.io/
https://en.wikipedia.org/wiki/Word_embedding
Fan, Yuchen / Qian, Yao / Xie, Feng-Long / Soong, Frank K. (2014): "TTS synthesis with bidirectional LSTM based recurrent neural networks", In INTERSPEECH-2014, 1964-1968
https://arxiv.org/abs/1406.1078

8. Contribution

YanqiGu 1 2.1 3 6
Zhijian Li 5 6 Report Integration
Ziyang Zhang 2.2 4 6 Report Integration