Final report of sentiment analysis

Group members: Lei Xu, Zijie Xue

Introduction of the problem:

With the rapid development of the Internet, the way users use the Internet has changed. Users are becoming information publishers from information acquirers. This transformation makes a large amount of information with personal emotions spread in the network, and act on the real society. More and more users can express their opinions and comment on hot events more conveniently through various social media. The social network has become an important platform for people to express their opinions and express their emotions. Its involvement and penetration in various fields of society is increasing day by day, and its social influence is increasing.

Sentiment analysis, also known as tendency analysis, opinion extraction, opinion mining, sentiment mining, subjective analysis, it is the process of analyzing, processing, inducing and reasoning about subjective texts with emotion. Analyzing and mining the emotional information hidden in the text can understand the public's attention to hot events and the changes in emotions, which can assist in identifying hot events and grasping the development of public opinion and emergencies. Sentiment analysis from text is currently widely used for customer satisfaction assessment and brand perception analysis, among others. In addition, the research of text sentiment analysis is also one of the important research directions of natural language processing and text mining.

Related Works:

We read some papers about sentiment processing.TextCNN is proposed by Yoon Kim in the paper (2014 EMNLP) Convolutional Neural Networks for Sentence Classification. In this paper, the convolutional neural network CNN is applied to the text classification task, and multiple kernels of different sizes are used to extract the key information in sentences (similar to the ngram of multi-window size), so as to better capture the local correlation. We also read some papers on other methods, such as Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks by RC Staudemeyer and E Rothstein Morris. We mainly use the research methods of these two papers and some classical classification algorithms to carry out this project.

Discussion of Methods:

Traditional machine learning sentiment analysis methods can be summarized into three categories: supervised learning, unsupervised learning and semi-supervised learning. Although most current sentiment analysis research based on supervised learning has achieved good results, supervised learning relies on a large amount of manually labeled data, which limits the promotion of such methods in other fields and across languages.

In recent years, with the development of deep learning related technologies, researchers are more inclined to use deep learning to solve this task. Deep learning technology is more and more popular among researchers to solve text classification problems. The powerful feature learning function of deep learning can capture basic features of data from a small group of samples. In related work, feature extraction based on traditional machine learning, affective dictionary resources and artificial rules often relies on corpus field, and the extracted text feature dimension increases linearly with the increase of extraction rules. Compared with the former, the neural network model considers the application of single model or improved model in the analysis of text sentiment.

In order to compare the performance between traditional machine learning algorithms and deep learning in snetiment analysis, we chose KNN as the representative of traditional machine learning algorithms, and TextCNN and LSTM as the representatives of deep learning method.

Dataset:

We choose IMDB (Internet Movie Database) dataset to train and test our model, This data set contains 25,000 movie data, including 12,500 positive data and 12,500 negative data. We used 23,000 of the 25,000 text comments as a training set and the remaining 2,000 text comments as a test set.

Data processing

1. Dictionary construction

   We build our dictionary based on these corpus, by the segmentation of the text and removing the duplicate.

2. Context conversion

   We convert the original text into machine recognizable encoding.

3. Load pre-trained word vector

   We use the word vectors which are pre trained in Glove as the word embedding of the model.

Experiment Setup:

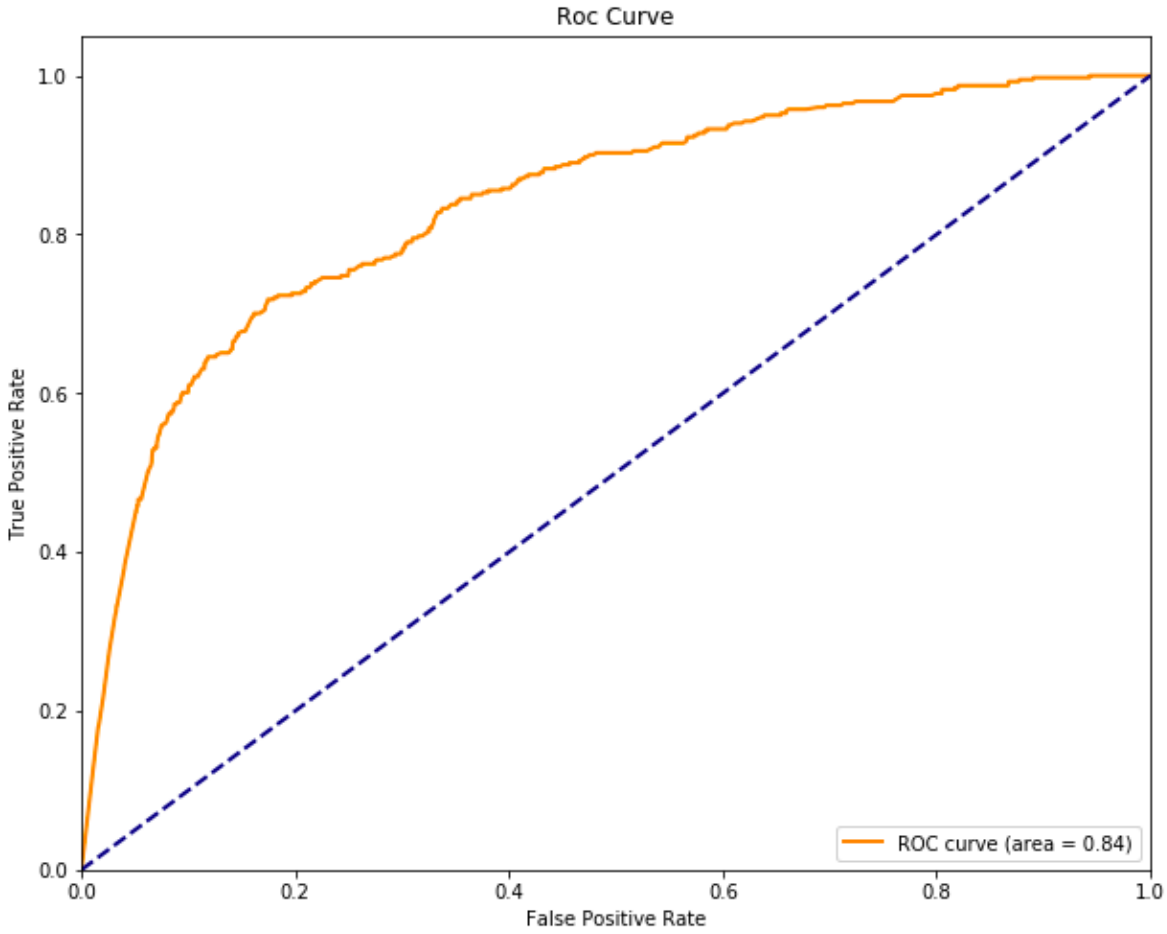
1. KNN

The idea of KNN classification algorithm is: if most of the k most similar samples in the feature space belong to a certain category, then the sample also belongs to this category.

For KNN, the hyperparameter are n_neighbors, weight and p. n_neighbors is the number that how many point we choose for the prediction, which is actually the K. Weight means Weight of distance; It can be Uniform: A uniform weight; Distance: The reciprocal of the distance serves as the weight. P is the way we calculate the distance, it can be European distance or Manhattan distance. In order to simplify the experimental process, we only change one of the hyperparameters which is n_neighbors to observe the experimental results.

```
1  best_score = 0.0
2  best_k = -1
3  for k in range(1,11):
4      knn = KNeighborsClassifier(n_neighbors=k)
5      knn.fit(x_train,y_train)
6      t = knn.score(x_test,y_test)
7      if t>best_score:
8          best_score = t
9          best_k = k
10
11 print(best_k)
12 print(best_score)
```

Performance Metrics



2. TextCNN

We all know that CNN has achieved very good results in image processing, because its convolution and pooling operations can capture the local features of the image. In the same way, CNN is used in text processing and can also capture local information in the text.

(1)Parameter initialization

We need batch_size, heights, widths and channels in convolution operation.

Width = embedding size, height = sentence length, channel = 1

(2)convolution

Since we use a variety of filters (filter size=2, 3, 4, 5, 6), the convolution pooling operation must be divided into filter processing. Firstly, pooled_outputs is defined to store the output results of the convolutional pooling operation of each filter.

For each filter, conv is first obtained through convolution, and then pooled after activation by the relu function to obtain max_pooling. Since we have 100 filters per filter, we can finally get a 100*5=500 dimensional vector after flattening, which is used to connect the fully connected layer.

(3)output

we try to Add dropout to improve model performance. Adding dropout before fully connected layer,

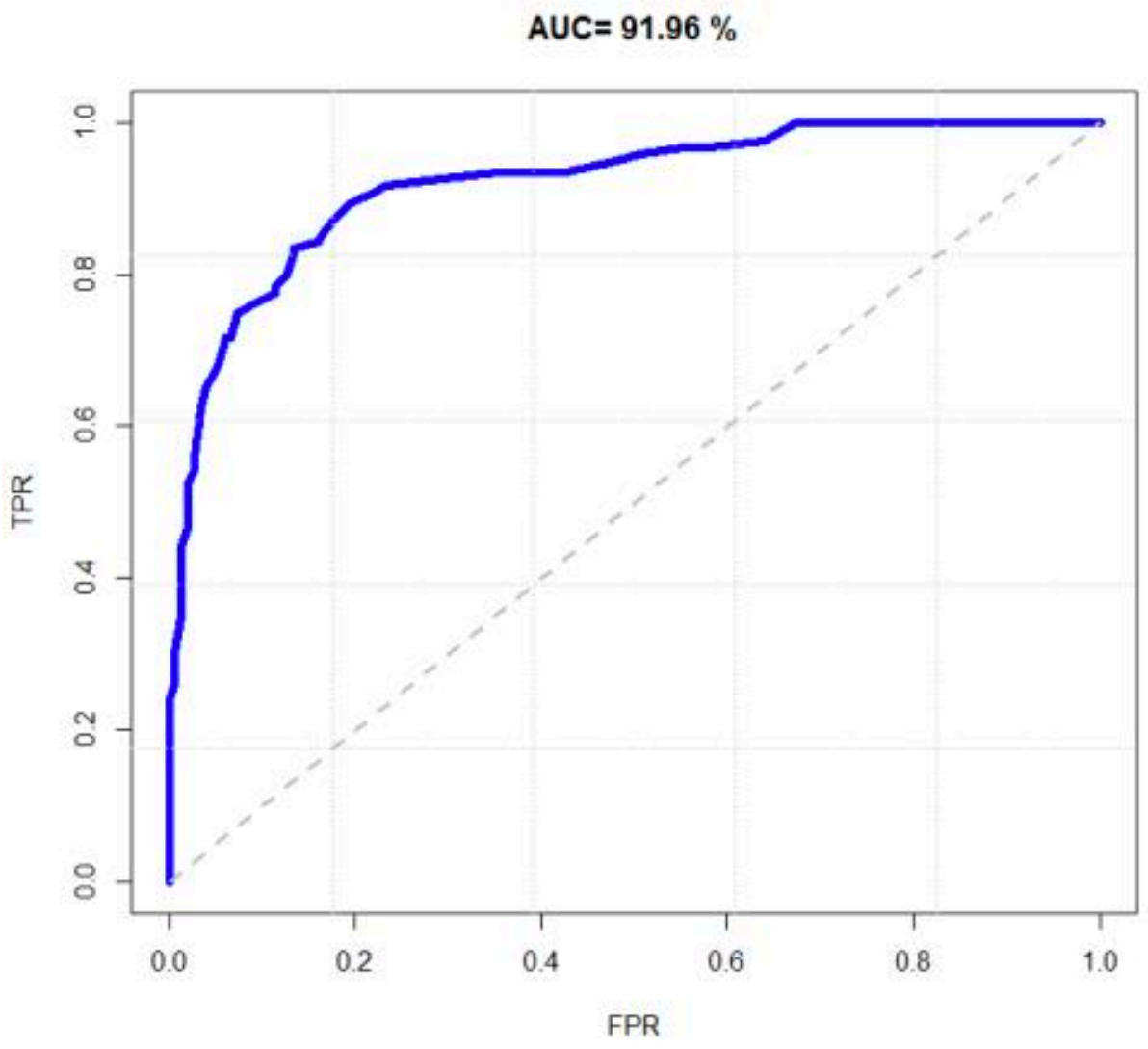
and output the final result after sigmoid activation.

```
# dropout
with tf.name_scope("dropout"):
    dropout = tf.nn.dropout(flattend_pool, KEEP_PROB)

# output
with tf.name_scope("output"):
    W = tf.get_variable("W", shape=(total_filters, 1), initializer=tf.contrib.layers.xavier_initializer())
    b = tf.Variable(tf.zeros(1), name="b")

    logits = tf.add(tf.matmul(dropout, W), b)
    predictions = tf.nn.sigmoid(logits, name="predictions")
```

performance metrics



3. LSTM

We did the following when manually optimizing the hyper-parameters of LSTM (RNN):

(1) Parameter initialization

We use Gauss initialization:W = Np.random. Randn (n)/SQRT (n),n is the number of parameters.

(2) Data preprocessing

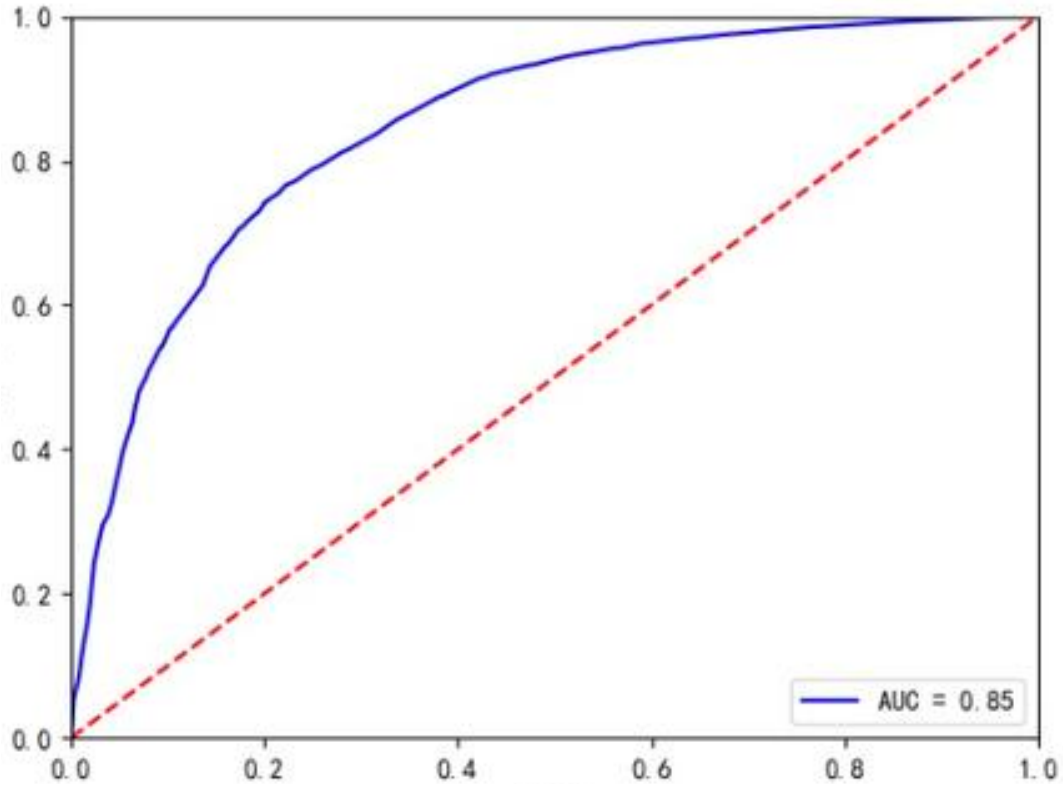
X -= np.mean(X, axis = 0) # zero-center
X /= np.std(X, axis = 0) # normalize

(3) Clip C (gradient clipping):

Limit the maximum gradient, which is value = SQRT(w1^2+w2^2...), if the value exceeds the threshold value, consider a attenuation coefficient and make the value equal to the threshold value: 5,10,15.

(4) Instead of using Sigmoid, we use activation functions like tanh or relu, except for things like gate, where you limit the output to 0-1.

performance metrics



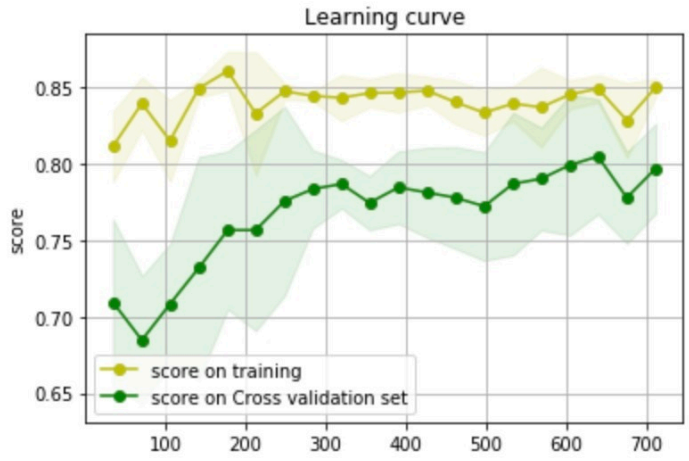
Experimental Results

1. KNN

We set k from 3 to 20 and when k=10 we have the best result.

```
plot_learning_curve(knn,X_train, Y_train)
```

```
score: 71.14
cross_val_score: [0.6        0.79775281 0.71910112 0.84269663 0.83146067 0.82022472
 0.82022472 0.83146067 0.84269663 0.79775281]
mean cross_val_score: 79.03370786516855
```



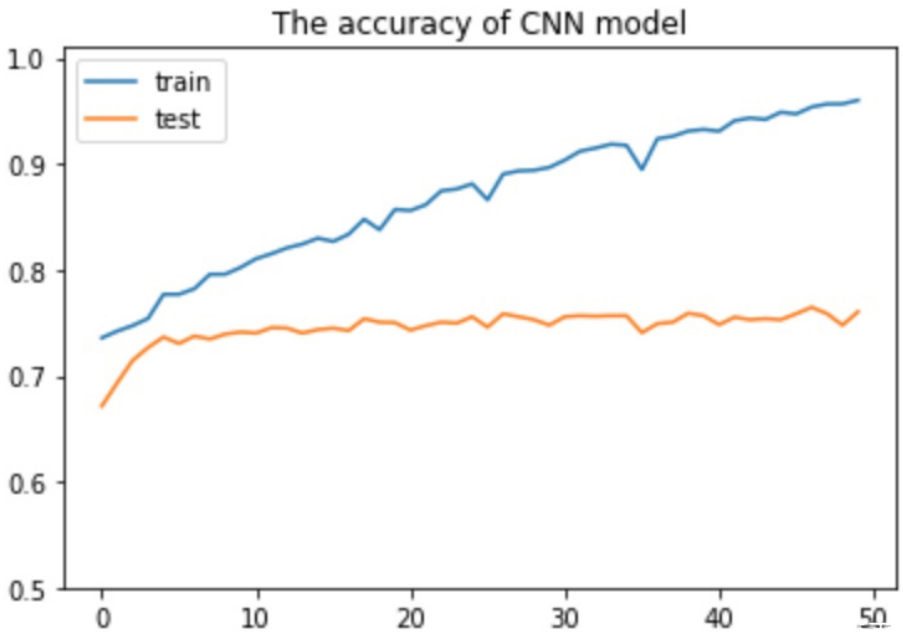
2. TextCNN

```
# Accuracy on test set
with tf.Session() as sess:
    saver.restore(sess, "checkpoints/cnn")

    total_correct = sess.run(accuracy,
                             feed_dict={inputs: x_test, targets: y_test})

    print("The CNN model accuracy on test set: {:.2f}%".format(100 * total_correct / x_test.shape[0]))
```

```
INFO:tensorflow:Restoring parameters from checkpoints/cnn
The CNN model accuracy on test set: 75.25%
```



3. LSTM

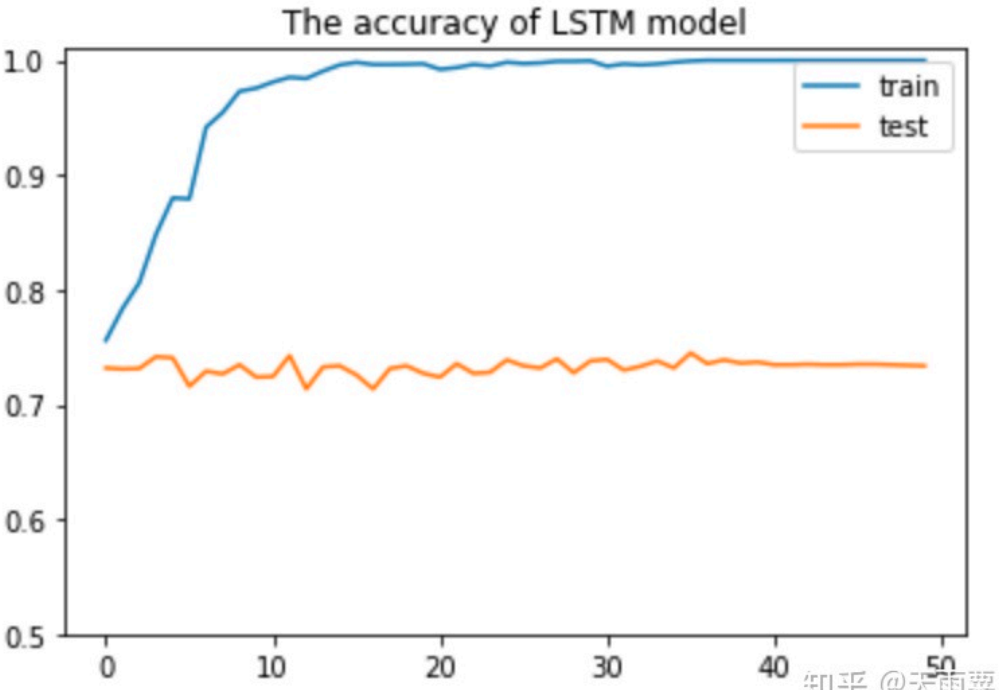
Here I use a single layer of 512 nodes LSTM.

```
with tf.Session() as sess:
    saver.restore(sess, "checkpoints/rnn")

    total_correct = sess.run(accuracy,
                             feed_dict={inputs: x_test, targets: y_test})

    print("The LSTM model accuracy on test set: {:.2f}%".format(100* total_cor
```

```
INFO:tensorflow:Restoring parameters from checkpoints/rnn
The LSTM model accuracy on test set: 73.37%
```



Discussion and Conclusion

So far, we have completed the task of handling sentence classification with KNN, LSTM and CNN respectively. Among them, the accuracy of KNN is not satisfactory but we believe if the parameter K is well selected, it can be more accurate. LSTM in the test is acceptable, while the accuracy of CNN in the test is 1% ~ 2% higher than LSTM.

Our models are relatively simple, but on the whole, these models have achieved good accuracy except KNN, which is largely due to the pre-trained word embedding, which shows the importance of word embedding in NLP model, while the accuracy of multi channels CNN is improved by adding task specific information.