Movie Review Sentiment Analysis

——classify the sentiment of sentences from the Rotten Tomatoes dataset

陈敬贤 15338013 20%
陈鸿旭 15338012 20%
莫昊宇 15338149 20%
谢瑞鸿 15338198 20%
陈恩城 15338010 20%

1. Background description

The Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis. In the work on sentiment treebanks used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus. This competition presents a chance to benchmark your sentiment-analysis ideas on the Rotten Tomatoes dataset. You are asked to label phrases on a scale of five values: negative(0), somewhat negative(1), neutral(2), somewhat positive(3), positive(4). Obstacles like sentence negation, sarcasm, terseness, language ambiguity, and many others make this task very challenging. We need to develop an effective automatic sentiment rating model for different movie reviews and give a value from 0 to 4 of each movie reviews.

2. Main problems

(1) What kinds of words have important effects of the sentiment direction and the values of movie reviews?

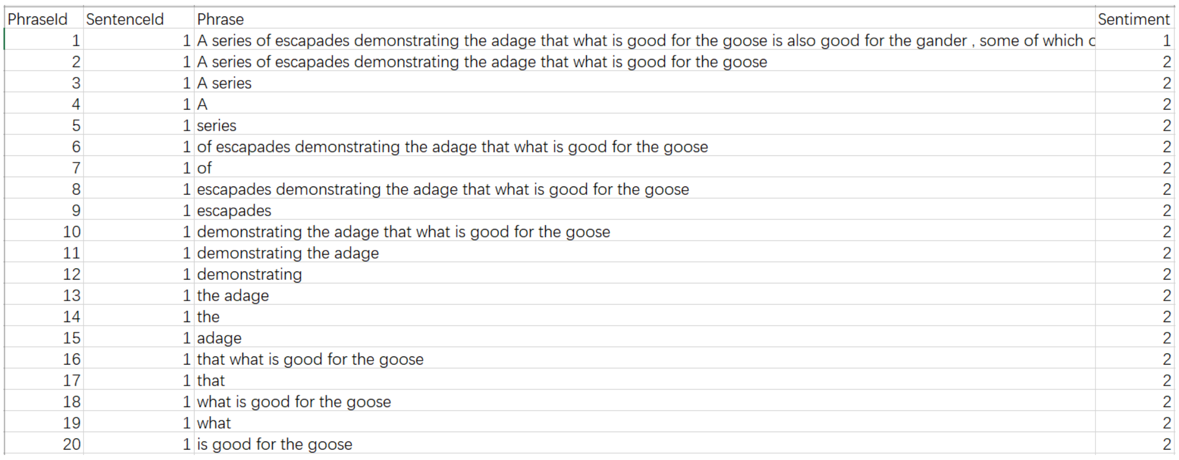
(2) Are punctuation and stop words have direct effects on the values of movie reviews?

(3) Why some tiny differences like some neutral words would change the values of movie reviews?

3. Data description

train.tsv：The training set (size: 156061×4）
test.tsv：The testing set (size: 66292×4)
Phrase：Containing whole sentences, part of sentence and some individual words. Average count of phrases per sentence in train is 18.
Sentiment：Values from 0 to 4; negative(0), somewhat negative(1), neutral(2), somewhat positive(3), positive(4).

Independent variable: Phrase
Dependent variable: Sentiment



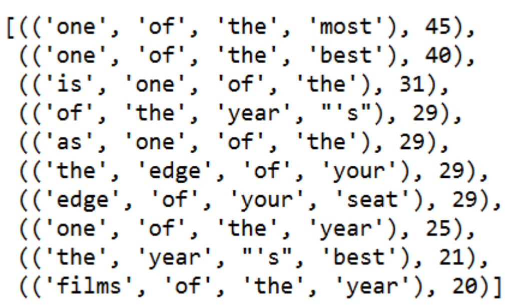
PhraseId	SentenceId	Phrase	Sentiment
1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which c	1
2	1	A series of escapades demonstrating the adage that what is good for the goose	2
3	1	A series	2
4	1	A	2
5	1	series	2
6	1	of escapades demonstrating the adage that what is good for the goose	2
7	1	of	2
8	1	escapades demonstrating the adage that what is good for the goose	2
9	1	escapades	2
10	1	demonstrating the adage that what is good for the goose	2
11	1	demonstrating the adage	2
12	1	demonstrating	2
13	1	the adage	2
14	1	the	2
15	1	adage	2
16	1	that what is good for the goose	2
17	1	that	2
18	1	what is good for the goose	2
19	1	what	2
20	1	is good for the goose	2

4. Data preprocessing

(1) Remove the punctuation and keep stop words

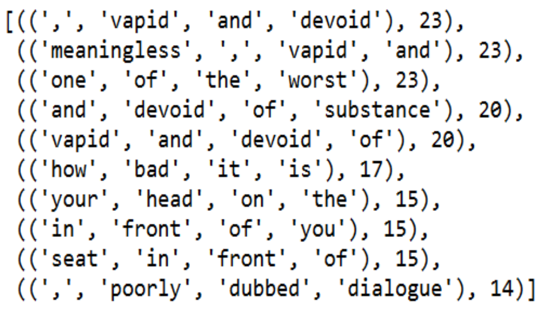
We select the most 10 common tetrads (4 words appear in one phrase) for positive phrases and negative phrases and list it as follows:

Positive



```
[(('one', 'of', 'the', 'most'), 45),
 (('one', 'of', 'the', 'best'), 40),
 (('is', 'one', 'of', 'the'), 31),
 (('of', 'the', 'year', "'s"), 29),
 (('as', 'one', 'of', 'the'), 29),
 (('the', 'edge', 'of', 'your'), 29),
 (('edge', 'of', 'your', 'seat'), 29),
 (('one', 'of', 'the', 'year'), 25),
 (('the', 'year', "'s", 'best'), 21),
 (('films', 'of', 'the', 'year'), 20)]
```

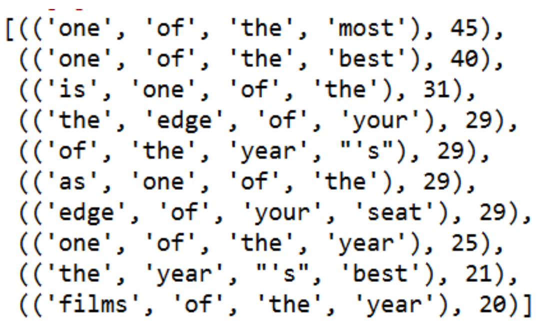
Negative



```
[((',', 'vapid', 'and', 'devoid'), 23),
 (('meaningless', ',', 'vapid', 'and'), 23),
 (('one', 'of', 'the', 'worst'), 23),
 (('and', 'devoid', 'of', 'substance'), 20),
 (('vapid', 'and', 'devoid', 'of'), 20),
 (('how', 'bad', 'it', 'is'), 17),
 (('your', 'head', 'on', 'the'), 15),
 (('in', 'front', 'of', 'you'), 15),
 (('seat', 'in', 'front', 'of'), 15),
 ((',', 'poorly', 'dubbed', 'dialogue'), 14)]
```

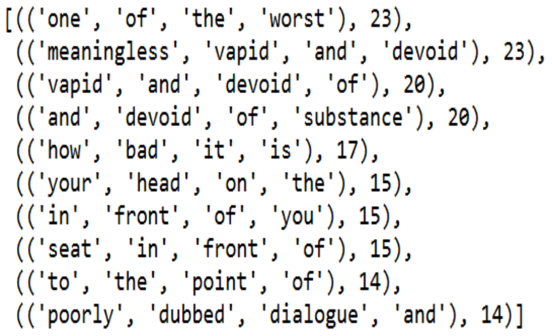
We find that stop words play an important role in each tetrad and punctuation is useless and redundant (because tetrads in negative phrases have so many punctuation to embody its sentiment effectively), so we decide to remove the punctuation and keep stop words to see the results.

Positive



```
[(('one', 'of', 'the', 'most'), 45),
 (('one', 'of', 'the', 'best'), 40),
 (('is', 'one', 'of', 'the'), 31),
 (('the', 'edge', 'of', 'your'), 29),
 (('of', 'the', 'year', "'s"), 29),
 (('as', 'one', 'of', 'the'), 29),
 (('edge', 'of', 'your', 'seat'), 29),
 (('one', 'of', 'the', 'year'), 25),
 (('the', 'year', "'s", 'best'), 21),
 (('films', 'of', 'the', 'year'), 20)]
```

Negative

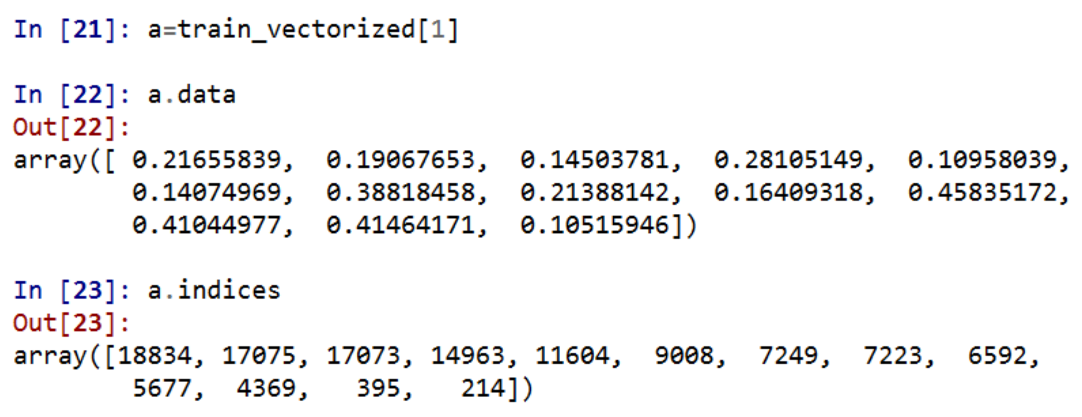


```
[(('one', 'of', 'the', 'worst'), 23),
 (('meaningless', 'vapid', 'and', 'devoid'), 23),
 (('vapid', 'and', 'devoid', 'of'), 20),
 (('and', 'devoid', 'of', 'substance'), 20),
 (('how', 'bad', 'it', 'is'), 17),
 (('your', 'head', 'on', 'the'), 15),
 (('in', 'front', 'of', 'you'), 15),
 (('seat', 'in', 'front', 'of'), 15),
 (('to', 'the', 'point', 'of'), 14),
 (('poorly', 'dubbed', 'dialogue', 'and'), 14)]
```

The result is much more effective and each tetrad in positive phrases and negative phrases have a good embodiment to its sentiment. Thus, we decide to keep stop words and remove punctuation.

(2)Change uppercase letters to lowercase letters
We do not distinguish uppercase letters from lowercase letters in our model.

(3) Word vectorization (TFIDF)
In information retrieval TFIDF, short for term frequency inverse document frequency, is the product of two statistics, term frequency and inverse document frequency that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The TFIDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. Variations of the TFIDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. TFIDF can be successfully used for stop-words filtering in various subject fields, including text summarization and classification. One of the simplest ranking functions is computed by summing the TFIDF for each query term; many more sophisticated ranking functions are variants of this simple model.
Example:



```
In [21]: a=train_vectorized[1]

In [22]: a.data
Out[22]:
array([ 0.21655839,  0.19067653,  0.14503781,  0.28105149,  0.10958039,
        0.14074969,  0.38818458,  0.21388142,  0.16409318,  0.45835172,
        0.41044977,  0.41464171,  0.10515946])

In [23]: a.indices
Out[23]:
array([18834, 17075, 17073, 14963, 11604,  9008,  7249,  7223,  6592,
        5677,  4369,   395,   214])
```

5. Data visualization

We got a train data (156061*4) with 4 features: PhraseID, SentenceID, Phrase, Sentiment (0-4) and a test data (66292*4) without Sentiment.

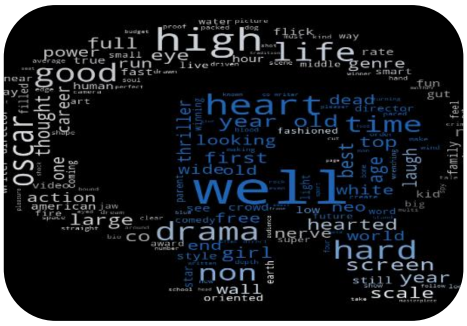
(1) The first 5 phrases of raw train data:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1
1	2	1	A series of escapades demonstrating the adage that what is good for the goose	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

(2) The first 5 phrases of train data after cleaning process:

	PhraseId	SentenceId	Phrase	Sentiment	clean_review
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1	a series of escapade demonstrating the adage that what is good for the goose is also good for the gander some of which occasionally amuses but none of which amount to much of a story
1	2	1	A series of escapades demonstrating the adage that what is good for the goose	2	a series of escapade demonstrating the adage that what is good for the goose
2	3	1	A series	2	a series
3	4	1	A	2	a
4	5	1	series	2	series

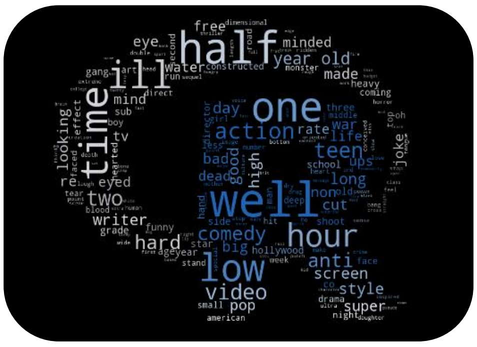
(3) Most frequent words of each sentiment dictionary:



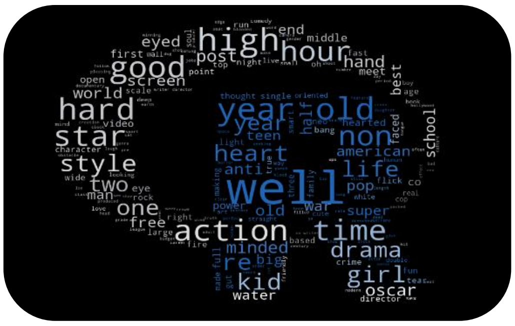
Sentiment 0

half
screen
one
mind
made
two
bad
pseudo
hour
day
anti
action
well
high
long
teen
point
low
ill
rate
looking
age
time
video
death
number
comedy
good
water
second
night
control
year
old

Sentiment 1



Sentiment 2



Sentiment 3

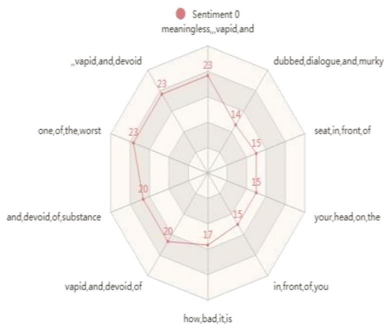
We find that whether it is a good or bad sentiment dictionary, the most frequent word, which is the largest word in each picture, is “well”. Obviously, it does not conform to the actual situation. Therefore, we cannot depend on every single word’s appearance to judge the sentiment of a phrase.

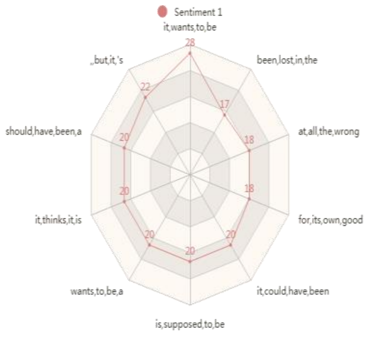
Solutions:

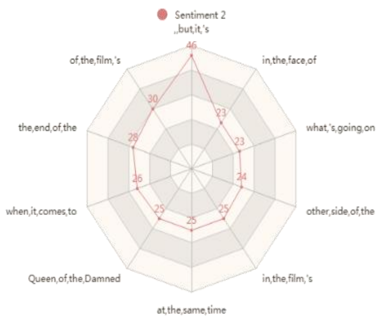
Instead, we figure out two possible methods. The first one is we consider the frequency of 4-word phrase of each sentiment’s dictionary. The second is we develop a new rating system for each word. For a specific word, we find out all the phrases that contain it and calculate the average sentiment grade as its score.

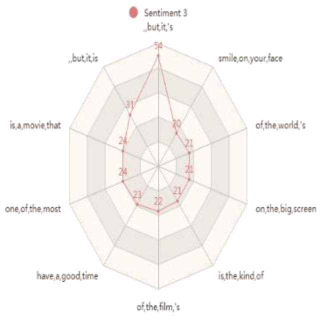
(4) Method I:

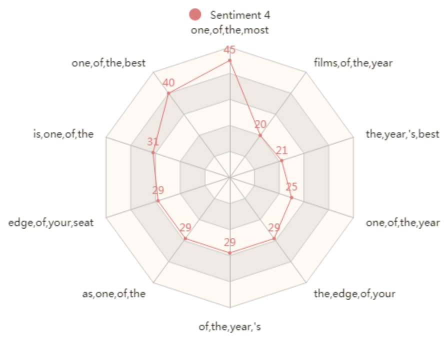
Most frequent 4-term phrases of each sentiment:







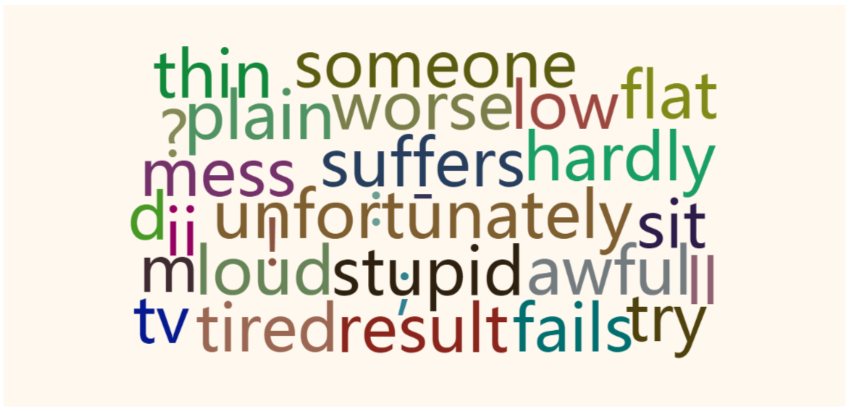




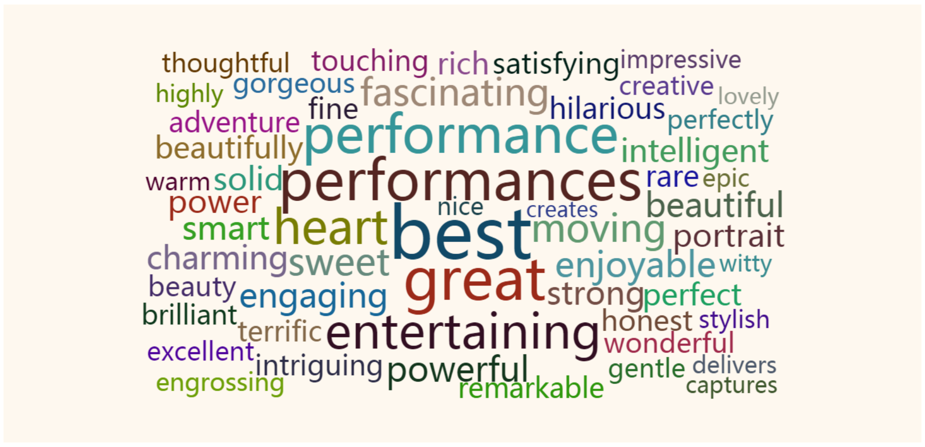
From the pictures above, we can clearly see that in the sentiment 0 dictionary, “one of the worst” is the most frequent phrase and “one of the most/best” is the most frequent one in the sentiment 4 dictionary.
In this way, we can conclude that the 4-term phrase frequency can partly reflect the sentiment level of the whole phrase.

(5) Method II:
Develop a new rating system for each word.

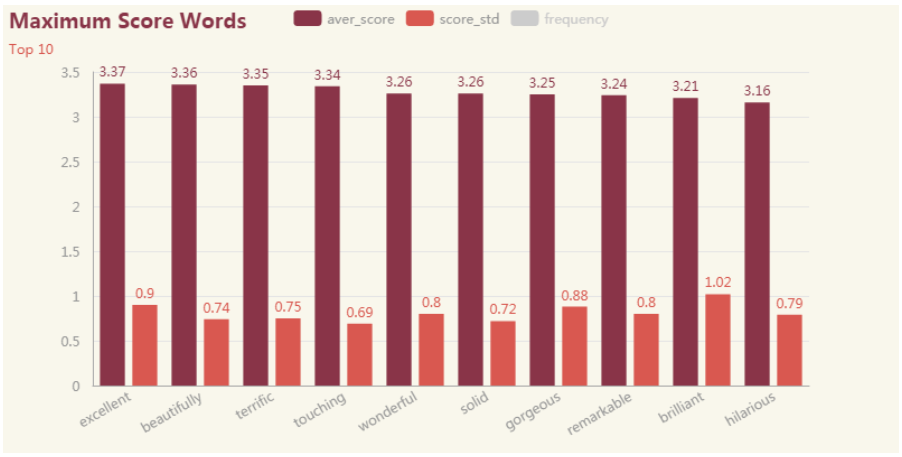
Words score below 2.7 point:



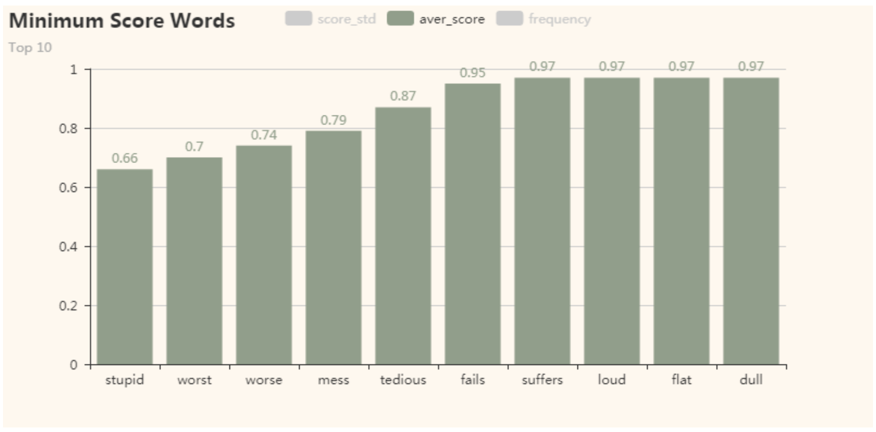
Words score higher than 2.7 point:



Top 10 maximum score words:



Top 10 minimum score words:



From the charts above, we can see that words like “stupid”, “worst” will get a low score whereas “excellent”, “beautifully” will get a high score. Therefore, we can depend on this rating system to judge a phrase’s sentiment.

6. Models description

The alternative models:

Logistic Regression: One vs Rest Classifier

In our case, we need to predict ‘Sentiment’ variable that has 5 levels.

On their own, logistic regressions are only binary classifiers, meaning they cannot handle target vectors with more than two classes. However, there are clever extensions to logistic regression to do just that. In one-vs-rest logistic regression (OVR) a separate model is trained for each class predicted whether an observation is that class or not (thus making it a binary classification problem). It assumes that each classification problem (e.g. class 0 or not) is independent.

Ridge Classifier
To use ridge regression for multi-label classification.

Linear SVC Classifier:
To use linear SVM method for multi-label classification.

Naïve Bayes Classifier:
In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Random Forest Classifier:
Random forests or random decision forests are an ensemble learningmethod for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Decision Trees Classifier:
A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Committee Machine:
A committee machine is a type of artificial neural network using a divide and conquer strategy in which the responses of multiple neural networks (experts) are combined into a single response. The combined response of the committee machine is supposed to be superior to those of its constituent experts. Compare with ensembles of classifiers.

The chosen model:

RNN and LSTM
Traditionally, RNN model is adopted to find the key factors in text sentiment analysis. Before the introduction of RNN model, CNN model, which rely on assigning particular weights onto each words to minimize the loss function through iterative training, is widely used. However, one fatal drawback is that CNN neglects the natural distinction of language, that is the connection to its context. Words can represent different sentiments under different context. From this perspective, CNN fails because words in a sentence are treated into individual cells.

In response to this defect, RNN model is introduced to specify in such kind of series data.

Compared with CNN, RNN has an advantageous cell series structure. Information is transmitted through the series structure and renewed in each cell, which can be seen as individual learning layers. This structure allows words to be studied in a rather complete context. As is widely known, the basic neural network consists of three fundamental elements, input layer, hidden layer and output layer. CNN builds connections between layers, but RNN strengthen the learning process by building connections within layers as well. This "within-layer-connection" provides basis to explore the sentiment of a particular word in a complete sentence through repeated learning.

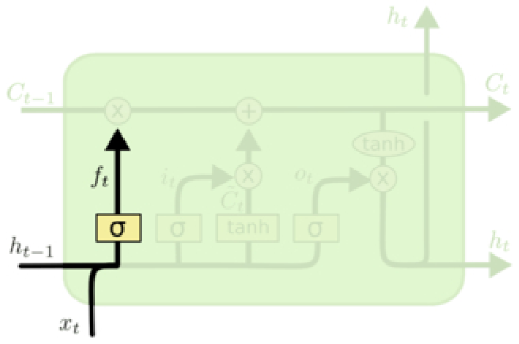
Unfortunately, RNN, through making great progress to CNN, still has its limits. RNN specify in coping with series data. However, in practice, chances are high that not every sentences in the training set is tightly related. Unrelated sentences can lead to the gradient explosion to the model, sapping the reliability. Therefore, based on RNN model, LSTM is developed to overall the drawbacks mentioned above.

LSTM is the abbreviation of Long-Short Term Memory. It simulates the distinction how human's brains work when learning and memorizing. The structure of a simple LSTM model is below:

As is illustrated, LSTM shares a same cell series structure with RNN model basically, but what makes the difference is that LSTM allows for the transition of not only the newly learned result of words' possible sentiment, but also which of them should be forgotten in the next cell so as not to cause the gradient explosion.

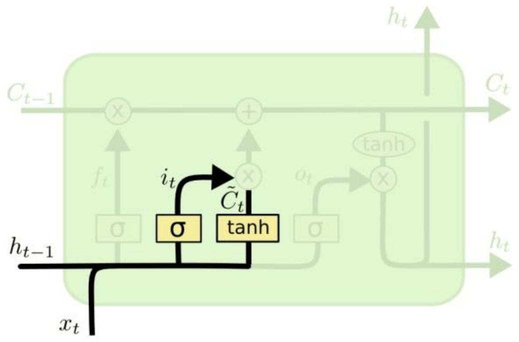
The Specific process of LSTM

(1) New training data and the pre-learning information from the last cell are input to the current cell. Pre-learning information is defined as the information extracted from the last cell specifies on what should be forgotten in next cell’s learning process.



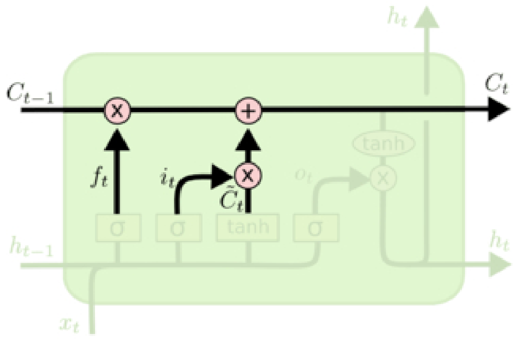
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

(4) Particular part of the new training data is eliminated from the learning process due to the pre-learning information.



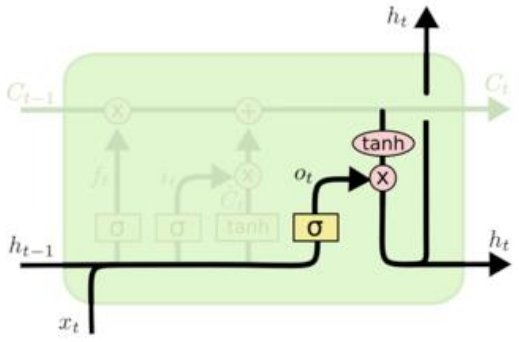
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(3) Knowledge from the last cell is transmitted and renewed by the new processed learning data.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(5) Knowledge is renewed and pre-learning information is generated to guide the learning process in the next cell.



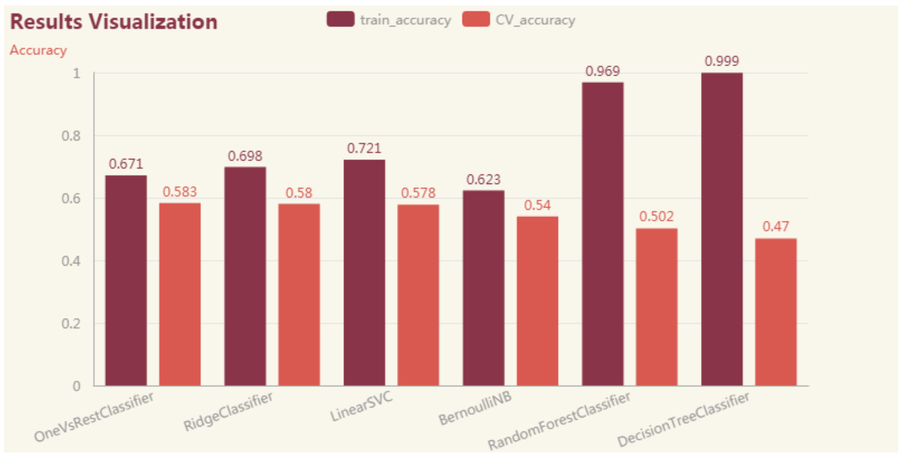
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

(6) Iterate until all training data are fully used

7. Results and conclusions

The alternative models:

Model	OneVsRest	Ridge	LinearSVC	BernoulliNB	RandomForest	DecisionTree	CommitteeMachine
CV accuracy	58.31%	57.91%	57.81%	53.98%	50.15%	47.03%	41.05%



LSTM

Epoch	1	2	3
CV accuracy	66.85%	66.91%	67.00%

After we analyze the data and build up the best fitted model, we can predict the phrases in test data with a 67% accuracy. This whole coursework gave us a valuable experience dealing with data science problem about natural language processing. We also learnt about some deep learning knowledge like LSTM network by ourselves, which enhanced our interest in developing our statistical method and coding skills. Although we still get some flaws when solving this problem, for applying knowledge out of our range and time restriction, we all are satisfied with this coursework.

8. Deficiencies and improvements

1. The parameters used in LSTM are so complex that we actually understand some parameters' meaning like dropout but do not understand other parameters and give the best values of them.
2. We find some papers to learn deep learning models and apply it to solve our problem. It works and the result is better than all of our models, including LSTM. But we cannot explain the principle and algorithm of it so we do not list it in our report. We need to find more information to learn the process and explain it.

9. Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 11 19:46:25 2018

@author: Chenhx
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from nltk.tokenize import TweetTokenizer
import datetime
import lightgbm as lgb
from scipy import stats
from scipy.sparse import hstack, csr_matrix
from sklearn.model_selection import train_test_split, cross_val_score
from wordcloud import WordCloud
from collections import Counter
from nltk.corpus import stopwords
from nltk.util import ngrams
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.neighbors import NearestCentroid
pd.set_option('max_colwidth',400)

#data input
train = pd.read_csv('c:/Users/Administrator/Desktop/kw/train.csv',sep=',')
test = pd.read_csv('C:/Users/Administrator/Desktop/kw/test.csv',sep=',')
sub = pd.read_csv('C:/Users/Administrator/Desktop/kw/sampleSubmission.csv', sep=",")

#data analysis
train.head(10)
print('Average number of phrases per sentence in train is
{0:.0f}.'.format(train.groupby('SentenceId')['Phrase'].count().mean()))
print('Average number of phrases per sentence in test is
{0:.0f}.'.format(test.groupby('SentenceId')['Phrase'].count().mean()))

print('Number of phrases in train: {}. Number of sentences in train:
{}.'.format(train.shape[0], len(train.SentenceId.unique())))
print('Number of phrases in test: {}. Number of sentences in test: {}.'.format(test.shape[0],
len(test.SentenceId.unique())))
```

```
print('Average word length of phrases in train is
{0:.0f}.'.format(np.mean(train['Phrase'].apply(lambda x: len(x.split())))))
print('Average word length of phrases in test is
{0:.0f}.'.format(np.mean(test['Phrase'].apply(lambda x: len(x.split())))))

text4 = ' '.join(train.loc[train.Sentiment == 4, 'Phrase'].values)
text4_trigrams = [i for i in ngrams(text4.split(), 4)]
Counter(text4_trigrams).most_common(10)

text0 = ' '.join(train.loc[train.Sentiment == 0, 'Phrase'].values)
text0_trigrams = [i for i in ngrams(text0.split(), 4)]
Counter(text0_trigrams).most_common(10)

text4 = ' '.join(train.loc[train.Sentiment == 4, 'Phrase'].values)
text4_nostopword = [i for i in text4.split() if i not in stopwords.words('english')]
text4_nostopword_trigrams = [i for i in ngrams(text4_nostopword, 4)]
Counter(text4_nostopword_trigrams).most_common(10)

english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '*', '@', '#', '$', '%']
text4_nopunctuation = [i for i in text4.split() if i not in english_punctuations]
text4_nopunctuation_trigrams = [i for i in ngrams(text4_nopunctuation, 4)]
Counter(text4_nopunctuation_trigrams).most_common(10)

english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '*', '@', '#', '$', '%']
text0_nopunctuation = [i for i in text0.split() if i not in english_punctuations]
text0_nopunctuation_trigrams = [i for i in ngrams(text0_nopunctuation, 4)]
Counter(text0_nopunctuation_trigrams).most_common(10)
#seperate
tokenizer = TweetTokenizer()
#vectorizer = TfidfVectorizer(ngram_range=(1, 2), tokenizer=tokenizer.tokenize)
vectorizer = TfidfVectorizer(tokenizer=tokenizer.tokenize)
full_text = list(train['Phrase'].values) + list(test['Phrase'].values)
vectorizer.fit(full_text)
train_vectorized = vectorizer.transform(train['Phrase'])
test_vectorized = vectorizer.transform(test['Phrase'])
aaa=vectorizer.get_feature_names()
aaa.index('of')

sentiment = train['Sentiment']

#1
logreg = LogisticRegression()
ovr = OneVsRestClassifier(logreg)
of = ovr.fit(train_vectorized, sentiment);
```

```
of.score(train_vectorized,sentiment)
scores_o = cross_val_score(ovr, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_o) *
100, np.std(scores_o) * 100))
#Cross-validation mean accuracy 58.31%, std 0.07. fit.score: 0.67055619633474306

#2
svc = LinearSVC(dual=False)
sf = svc.fit(train_vectorized,sentiment);
sf.score(train_vectorized,sentiment)
scores_s = cross_val_score(svc, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_s) *
100, np.std(scores_s) * 100))
#Cross-validation mean accuracy 57.81%, std 0.54. fit.score: 0.72063949762911705

#3
dec = DecisionTreeClassifier()
df = dec.fit(train_vectorized, sentiment);
df.score(train_vectorized,sentiment)
scores_d = cross_val_score(dec, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_d) *
100, np.std(scores_d) * 100))
#Cross-validation mean accuracy 47.03%, std 3.38. fit.score: 0.99911572472126109

#4
ber = BernoulliNB()
bf = ber.fit(train_vectorized, sentiment);
bf.score(train_vectorized,sentiment)
scores_b = cross_val_score(ber, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_b) *
100, np.std(scores_b) * 100))
#Cross-validation mean accuracy 53.98%, std 0.62. fit.score: 0.62381135460720238

#5
ran = RandomForestClassifier()
rf = ran.fit(train_vectorized, sentiment);
rf.score(train_vectorized,sentiment)
scores_r = cross_val_score(ran, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_r) *
100, np.std(scores_r) * 100))
#Cross-validation mean accuracy 50.15%, std 2.98. fit.score: 0.96962706651287967

#7
rid = RidgeClassifier()
rf = rid.fit(train_vectorized, sentiment);
rf.score(train_vectorized,sentiment)
scores_r = cross_val_score(rid, train_vectorized, sentiment, scoring='accuracy',
n_jobs=-1, cv=3)
print('Cross-validation mean accuracy {0:.2f}%, std {1:.2f}.'.format(np.mean(scores_r) *
100, np.std(scores_r) * 100))
#Cross-validation mean accuracy 57.91%, std 0.45 . fit.score:   0.69803280789439959

#8

#lstm
import numpy as np
import pandas as pd
import nltk
import os
import gc
from keras.preprocessing import sequence,text
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import
Dense,Dropout,Embedding,LSTM,Conv1D,GlobalMaxPooling1D,Flatten,MaxPooling1D,
GRU,SpatialDropout1D,Bidirectional
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report,f1_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

train = pd.read_csv('c:/Users/Administrator/Desktop/kw/train.csv',sep=',')
print(train.shape)
train.head()
test = pd.read_csv('C:/Users/Administrator/Desktop/kw/test.csv',sep=',')
sub = pd.read_csv('C:/Users/Administrator/Desktop/kw/sampleSubmission.csv', sep=",")
test['Sentiment']=-999
```

```
df=pd.concat([train,test],ignore_index=True)
print(df.shape)
del train,test
gc.collect()

from nltk.tokenize import word_tokenize
from nltk import FreqDist
from nltk.stem import SnowballStemmer,WordNetLemmatizer
stemmer=SnowballStemmer('english')
lemma=WordNetLemmatizer()
from string import punctuation
import re

def clean_review(review_col):
    review_corpus=[]
    for i in range(0,len(review_col)):
        review=str(review_col[i])
        review=re.sub('[^a-zA-Z]',' ',review)
        #review=[stemmer.stem(w) for w in word_tokenize(str(review).lower())]
        review=[lemma.lemmatize(w) for w in word_tokenize(str(review).lower())]
        review=' '.join(review)
        review_corpus.append(review)
    return review_corpus

df['clean_review']=clean_review(df.Phrase.values)
df.head()
df_train=df[df.Sentiment!=-999]
df_test=df[df.Sentiment==-999]
df_test.drop('Sentiment',axis=1,inplace=True)
del df
gc.collect()

train_text=df_train.clean_review.values
test_text=df_test.clean_review.values
target=df_train.Sentiment.values
y=to_categorical(target)
print(train_text.shape,target.shape,y.shape)

#cv
X_train_text,X_val_text,y_train,y_val=train_test_split(train_text,y,test_size=0.2,stratify=y,r
andom_state=123)

all_words=' '.join(X_train_text)
all_words=word_tokenize(all_words)
```

```
dist=FreqDist(all_words)
num_unique_word=len(dist)

r_len=[]
for text in X_train_text:
    word=word_tokenize(text)
    l=len(word)
    r_len.append(l)

MAX_REVIEW_LEN=np.max(r_len)
MAX_REVIEW_LEN

max_features = num_unique_word
max_words = MAX_REVIEW_LEN
batch_size = 128
epochs = 3
num_classes=5

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(list(X_train_text))
X_train = tokenizer.texts_to_sequences(X_train_text)
X_val = tokenizer.texts_to_sequences(X_val_text)
X_test = tokenizer.texts_to_sequences(test_text)

#sequence padding?
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_val = sequence.pad_sequences(X_val, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)

model1=Sequential()
model1.add(Embedding(max_features,100,mask_zero=True))
model1.add(LSTM(64,dropout=0.4, recurrent_dropout=0.4,return_sequences=True))
model1.add(LSTM(32,dropout=0.5, recurrent_dropout=0.5,return_sequences=False))
model1.add(Dense(num_classes,activation='softmax'))
model1.compile(loss='categorical_crossentropy',optimizer=Adam(lr=0.001),metrics=['acc
uracy'])
model1.summary()

history1=model1.fit(X_train, y_train, validation_data=(X_val, y_val),epochs=3,
batch_size=batch_size, verbose=1)
#Cross-validation mean accuracy 67%


# encoding: utf-8
```

```
@author: Chenjx

import numpy as np
import pandas as pd
import sys
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
import os
from gensim import corpora, models, similarities
import logging
from collections import defaultdict

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.INFO)
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import FreqDist
from nltk.stem.lancaster import LancasterStemmer

st = LancasterStemmer()

train_data = pd.read_table("D:/multivariate/kaggle2/all/train.tsv")

test_data = pd.read_table("D:/multivariate/kaggle2/all/test.tsv")

article = list(train_data.Phrase)
article1 = list(train_data.Phrase[train_data.Sentiment == 1])
article2 = list(train_data.Phrase[train_data.Sentiment == 2])
article3 = list(train_data.Phrase[train_data.Sentiment == 3])
article4 = list(train_data.Phrase[train_data.Sentiment == 4])
article5 = list(train_data.Phrase[train_data.Sentiment == 0])

def vocu_construct(article):
    texts_tokenized = [[word.lower() for word in word_tokenize(document)] for document
in article]
    ## 去停用词
    stoplist = stopwords.words('english')
    useful_words = [[word for word in document if not word in stoplist] for document in
texts_tokenized]
    ## 去标点
    english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '*', '@', '#', '$', '%']
    useful_words_nopunc = [[word for word in document if not word in
english_punctuations] for document in useful_words]
    #useful_words_nopunc1 = [[word for word in document if not word in
english_punctuations] for document in texts_tokenized]
    ## 构建词库
    #dictionary = corpora.Dictionary(useful_words_nopunc1)
    dictionary = corpora.Dictionary(useful_words_nopunc)
    text = [i for i in dictionary.values()]
    text = ' '.join(text)
    return text

dd1 = vocu_construct(article)
d0 = vocu_construct(article5)
d1 = vocu_construct(article1)
d2 = vocu_construct(article2)
d3 = vocu_construct(article3)
d4 = vocu_construct(article4)

# 求每个评分的特征词
d=set(d4)
dd=set(d0+d2+d3+d1)
word4= list(d.difference(dd))



#计算文本中词频
def wordcount(strl_ist):
    count_dict = {}
    # 如果字典里有该单词则加 1，否则添加入字典
    for i in range(len(strl_ist)):
        for str in strl_ist[i]:
            if str in count_dict.keys():
                count_dict[str] = count_dict[str] + 1
            else:
                count_dict[str] = 1
    # 按照词频从高到低排列
    count_list = sorted(count_dict.items(), key=lambda x: x[1], reverse=True)
    return count_list

count_list = wordcount(useful_words_nopunc)

def get_freq(word):
    index=[]
    value=[]
    for i in range(len(count_list)):
        index.append(count_list[i][0])
        value.append(count_list[i][1])
    a=[index,value]
    ii=[]
    for i in range(len(word)):
        ii.append(a[0].index(word[i]))
    #频率
    freq = [a[1][i] for i in ii]
    return(freq)

freq0 = get_freq(word0)
freq1 = get_freq(word1)
freq2 = get_freq(word2)
freq3 = get_freq(word3)
freq4 = get_freq(word4)


#corpus = [dictionary.doc2bow(text) for text in useful_words_nopunc]
#tfidf1 = models.TfidfModel(corpus)
#corpus_tfidf1 = tfidf1[corpus1]

text = vocu_construct(article)
text0 = vocu_construct(article5)
text1 = vocu_construct(article1)
text2 = vocu_construct(article2)
text3 = vocu_construct(article3)
text4 = vocu_construct(article4)
#from sklearn.cross_validation import train_test_split
#from gensim.models.word2vec import Word2Vec

###########################################################

## 做标签云
from PIL import   Image
from wordcloud import WordCloud, STOPWORDS,ImageColorGenerator

def wc_english(text):
    background_Image =
np.array(Image.open('C:/Users/Administrator/Desktop/Rlogo.jpg'))
    # 提取背景图片颜色
    #img_colors = ImageColorGenerator(background_Image)
    stopwords = set(STOPWORDS)
    stopwords.add('film')
    stopwords.add('movie')
    stopwords.add('minute')
    stopwords.add('self')
    wc = WordCloud(
        margin=2,
        mask=background_Image,
        scale=2,
        max_words=200,
        min_font_size=4,
        stopwords=stopwords,
        random_state=42,
        background_color='#EFFFC1',
        max_font_size=150
    )
    # 生成词云
    wc.generate_from_text(text)
    #wc.recolor(color_func=img_colors)
    plt.imshow(wc,interpolation='bilinear')
    plt.axis('off')
    plt.tight_layout()
    plt.title('Sentiment 0')
    plt.show()
    wc.to_file('C:/Users/Administrator/Desktop/sentiment 0.png')
    return wc

wc = wc_english(text0)

## 从标签云中获取词频最高的前 n 个词
def get_most_freq(wc,text,n):
    process_word = WordCloud.process_text(wc,text)
    sort = sorted(process_word.items(),key= lambda e:e[1],reverse=True)
    print(sort[:n])
    index=[]
    value=[]
    for i in range(n):
        index.append(sort[i][0])
        value.append(sort[i][1])
    a=[index,value]
    return a


index = get_most_freq(wc,text0,5)[0]
value = get_most_freq(wc,text0,5)[1]

## 柱型图展示高频词
from pyecharts import configure
configure(global_theme='essos')
```

```
from pyecharts import Bar,Grid,Line,Overlap

index=['stupid',
'worst',
'worse',
'mess',
'tedious',
'fails',
'suffers',
'loud',
'flat',
'dull',
]
aver_score=[0.66,0.7,0.74,0.79,0.87,0.95,0.97,0.97,0.97,0.97]
count=[159,
310,
149,
183,
128,
200,
104,
143,
223,
375
]
aver_std=[0.72,
0.77,
0.71,
0.81,
0.81,
0.93,
0.71,
0.83,
0.69,
0.97
]

def bar(index,value):
    bar = Bar("Minimum Score Words", 'Top 10')
    bar.add("score_std", index, aver_std, is_more_utils=True, is_label_show=True)
    bar.add("aver_score", index, aver_score, is_more_utils=True, is_label_show=True)
    bar.add("frequency",index,count,is_more_utils=True, is_label_show=True)
    scatter = Scatter("散点图示例")
    scatter.add("A", v1, v2)
    scatter.add(
        "B",
        v1[::-1],
        v2,
        is_visualmap=True,
        visual_type="size",
        visual_range_size=[20, 80],
    )
    scatter.render('C:/Users/Administrator/Desktop/bar.html')

bar(index,value)

radar(inde4,value4)

### 展示每个评分的特有词及其频率
def plot(word,freq):
    attr = word
    v1 = freq
    bar = Bar("Sentiment 0 特有词及频率")
    bar.add(
        "",
        attr,
        v1,
        is_datazoom_show=True,
        datazoom_type="both",
        datazoom_range=[10, 25],
        xaxis_interval=0, xaxis_rotate=20, yaxis_rotate=30
    )
    bar.render('C:/Users/Administrator/Desktop/plot4.html')


plot(word4,freq4)

##特征词云
from pyecharts import WordCloud
from pyecharts import configure

# 将这行代码置于首部
configure(global_theme='vintage')

name = word4
value = freq4

wordcloud = WordCloud(width=1300, height=620)
```

```
wordcloud.add("Sentiment4", name, value, word_size_range=[30, 100],
                    shape='diamond')
wordcloud.render('C:/Users/Administrator/Desktop/wc4.html')

#### 画高频 4 个单词的词组
from collections import Counter
from nltk.util import ngrams

text = ' '.join(train_data.loc[train_data.Sentiment == 4, 'Phrase'].values)
text_trigrams1 = [i for i in ngrams(text.split(), 4)]
b = Counter(text_trigrams1).most_common(10)
inde4=[]
value4=[]
for i in range(10):
    inde4.append(b[i][0])
    value4.append(b[i][1])

## 雷达图展示高频词
def radar(index,value):
    schema=[]
    for i in range(10):
        schema.append((index[i],50))

    v1 = [value]
    radar = Radar()
    radar.config(schema)
    radar.add("Sentiment 4", v1, is_splitline=True,
is_axisline_show=True,is_label_show=True)
#radar.add("实际开销", v2, label_color=["#4e79a7"], is_area_show=False,
#              legend_selectedmode='single')
    radar.render('C:/Users/Administrator/Desktop/radar.html')


radar(inde4,value4)

### 结果可视化
from pyecharts import configure

# 将这行代码置于首部
configure(global_theme='vintage')

index=['stupid',
'worst',
'worse',
'mess',
'tedious',
'fails',
'suffers',
'loud',
'flat',
'dull',
]
aver_score=[0.66,0.7,0.74,0.79,0.87,0.95,0.97,0.97,0.97,0.97]
count=[159,
310,
149,
183,
128,
200,
104,
143,
223,
375
]
aver_std=[0.72,
0.77,
0.71,
0.81,
0.81,
0.93,
0.71,
0.83,
0.69,
0.97
]
bar = Bar("Results Visualization", 'Accuracy')
bar.add("train_accuracy", name, train_accuracy,is_label_show=True,xaxis_rotate=20)
bar.add("CV_accuracy", name, CV_accuracy,is_label_show=True,xaxis_rotate=20)

from pyecharts import Line, Bar, Overlap,Scatter

index=['stupid',
'worst',
'worse',
'mess',
'tedious',
'fails',
'suffers',
'loud',
'flat',
'dull',
]
aver_score=[0.66,0.7,0.74,0.79,0.87,0.95,0.97,0.97,0.97,0.97]
count=[159,
310,
149,
183,
128,
200,
104,
143,
223,
375
]
aver_std=[0.72,
0.77,
0.71,
0.81,
0.81,
0.93,
0.71,
0.83,
0.69,
0.97
]
bar = Bar("Results Visualization", 'Accuracy')
bar.add("aver_score", index, aver_score, is_more_utils=True,
is_label_show=True,xaxis_rotate=20,
        yaxis_interval=0.2, yaxis_max=3)
scatter=Scatter()
scatter.add(
    "frequency",
    index,
    count,
   is_visualmap=True,
    visual_type="size",
    visual_range_size=[100, 400],
   is_label_show=True,yaxis_interval=300,yaxis_max=400)
#line = Line()
#line.add("frequency",index,count,is_more_utils=True,
is_label_show=True,yaxis_interval=300,yaxis_max=400)
overlap = Overlap(width=1200, height=600)
```

```
# 默认不新增 x y 轴，并且 x y 轴的索引都为 0
overlap.add(bar)
# 新增一个 y 轴，此时 y 轴的数量为 2，第二个 y 轴的索引为 1（索引从 0 开始），
所以设置 yaxis_index = 1
# 由于使用的是同一个 x 轴，所以 x 轴部分不用做出改变
overlap.add(scatter, yaxis_index=1, is_add_yaxis=True)
overlap.render('C:/Users/Administrator/Desktop/ww.html')
```