Classify user's rating based on IMDb data

STA208, University of California, Davis

Huiyu Bi (hybi@ucdavis.edu)
Miao Wang (mgwang@ucdavis.edu)
Yuan Tian (fiytian@ucdavis.edu)

June 2016

Abstract

In this project, we used machine learning methods to classify a user's rating for a movie into negative and positive sentiment based on his/her comments for this movie. And we tried to combine the movie information such as released year, runtime, director, actors, etc. with comments to see whether adding this information can be an improvement for the classification accuracy. The intuition behind combining user's comments and movie information together to predict user's rating is that a user may have a positive or negative attitude towards a movie not only based on his/her own subjective opinion, but also on some objective facts about the movie. We conducted 4 machine learning methods: Logistic Regression, Random Forest, XGBoost and RBF SVM to fit the classification model, and developed a voting classifier based on three well-performed classification methods.

Keyword Sentiment Analysis, Logistic Regression, Random Forest, XGBoost, RBF SVM, IMDb

1. Introduction

The Internet Movie Database (abbreviated IMDb) is an online database of information related to films, television programs and video games, including cast, production crew, fictional characters, biographies, plot summaries, trivia and reviews. In this project, we tried to use machine learning methods to conduct sentimental analysis of a user's polar rating for a particular movie based on his/her comment for this movie. We also innovatively combined the movie feature variables to predict the user's attitude, and it is proved to be a successful supplement for the sentimental analysis. In order to get a better classifier, we conducted attempts at two layers: to find the best predictor variables combination which has the best prediction performance and to find the best classification model to make the most accurate prediction.

2. Data Source

The data that we used for our analysis came from three parts: (1). IMDb Large Movie Review Dataset; (2). OMDb API (Open Movie dataset) (3). Website Box Office Mojo.

IMDb Large Movie Review Dataset contains users' 25000 comments with their corresponding rating (either ≥ 7 or ≤ 4) and IMDb movie ID in both training set and test set. Training set has 25000 comments for 4347 movies and test set has 25000 comments for 4367 movies. We also have each comment's word counts. There are in total 89527 words appeared in all the comments. Our classification goal is to get a user's attitude towards a movie: positive (rating ≥ 7) or negative (rating ≤ 4).

OMDb API (Open Movie dataset) is a free web service to obtain movie information. We got 7036 movies' information which are rated by the 50000 comments, such as directors, actors, released year and so on.

Website Box Office Mojo provides top 852 directors and top 787 actors ordered by their total gross box office. We used web scraping to get these information, and put them in the dataset.

3. Feature and processing

There are two potential groups of features that we considered might be useful: comments data and the movie information.

3.1 Comments Data

3.1.1 Feature Selection

In the comments data, we have count 89537 different words' counts in each comment. But 89527 words is too much. Only "important" words are informative in the classification. So we did the following processes to filter the words.

(1). Remove stop words: Stop words are words with no information value but appear too common in a language, such as "I", "again", "most", "when", … In computing, stop words are usually filtered out before analysis of language data. In our data set, we found 161 stop words and eliminated them.

(2). Remove rare words: Some words appeared rarely, so they were not very influential in the classification. We defined rare words whose proportion in the total number of words is < 0.00001 and eliminated them.

(3). Remove moderate sentiment words: Some words that appeared almost equally in positive comments and negative comments are also not important in distinguishing binary classes. We kept the "intense sentiment words", that is the ratio of this word's total number in positive (negative) comments to its total number in negative (positive) comments is > 2.

After above processes, we had only 3139 words left. So the counts of these 3139 words composed our comments feature variables.

3.1.2 Feature Engineering

Features are important in machine learning problems. So besides the 3139 words counts features that we had, we wanted to extract other useful information hiding in them. These features might improve the prediction performance of our model.
We considered creating three kinds of features that might be of interest.

(1). The length of comment: It happens that when the comment is very long, people always had intense emotion, so it is highly possible that rating is extreme.

(2). Total number of transitional words in a comment: Transitional words such as but, though, nevertheless, can influence the classification in opposite direction. So their existences may be influential in the classification.

(3). Total number of "pure positive/negative words" from a comment: People tend to use more positive/negative words with positive/negative attitudes towards the movie. So in a comment, the counts of negative words and positive words can be a decisive value. We counted 200 most frequently appeared words in both the positive and negative comments, and eliminated 80 overlapped words, then we got 120 "pure positive word" such as fantastic, superb, perfectly, powerful, incredible, sweet, awesome… and 120 "pure negative word": such as awful, waste, horrible, crap, ridiculous, dull, lame, poorly, badly…

Figure 3.1
Pure Positive Words WordCloud

Figure 3.2
Pure Negative Words WordCloud

3.2 Movie Information

3.2.1 Introduction

The movie information dataset contains unique 7036 movies rated by 50000 comments with their information. There are two kinds of variables, numeric and categorical. Numerical variables are Released year, Runtime, Awards, imdbRating, imdbVotes. And categorical variables include Type, MPAA rate, Language, Genre, Director and Actors. In order to make better use of these information in our model, we should do some transformations.

3.2.2 Variable Transformation

(1). “Award” variable:
It contains different types of awards and nominations, which may have different effects on classifying user’s rating for a movie. For example, if a movie has won Oscar or been nominated for Oscar, the user is more likely to give it a high rating than that has won other rewards or been nominated for others. So we transformed “Award” to 4 variables depending on whether it is famous or not (i.e. “famous_wins”, “other_wins”, “famous_nominations”, “other_nominations”).

(2). “MPAA rate” variable:
There are 18 different kinds of MPAA rate in the dataset. Referring to the history of American rating system, some rates can be merged into one. For example, the rates “Approved” and “Passed” are reasonable to be regarded as “G”. The rate “Unrated” can be thought of as “X” and “NOT RATED” as “N/A”. Moreover, “M” and “GP” were merged into “PG” since they have the same meaning but were used in different decades. Finally, we only had 12 kinds of MPAA rate and then transformed them into dummy variables.

(3). “Language” variable:
Firstly, we transformed it into dummy variables. Then we wanted to find more information from this variable. Some movies contain more than one languages. It is possible that a movie containing more than one languages is more popular than others because it may be showed in different countries and thus people tend to give it a high rating. So we created a new variable called “number of languages”. Moreover, another new variable “main language” was created, which will also be of interest in our model.

(4). “Director” and “Actors” variables:
They include people’s names, so we want to transform them into numerical variables. The basic idea is to rank them according to their gross box office. But it is difficult to obtain everyone’s box office since there are so many people in our dataset. To make things more feasible, we looked into our dataset to find whether the actor or the director is among the top directors or the top actors. The information about top directors and actors were obtained from a website named Box Office Mojo, which summarized top 852 directors and top 787 actors ordered by their total gross box office. The criteria for rating a director or an actor in our dataset is showed in table 3.1:

Box Office rank	1-50	50-200	100-200	200-400	400-800	Not in the list
Our rating	10	9	7	4	1	0

Table 3.1
Rating criterion for directors and actors

(5). “Type” and “Genre” variables:
We simply transformed them into dummy variables.

3.2.3 Data Cleaning

After data transformation, we found there are some NA's in the numeric variables "runtime", "imdbVotes", "imdbRating", "famous_wins", "famous_nominations", "other_wins", "other_nominations".

(1). "Runtime" and "ImdbVotes" variables:
we replaced NA by their median since the median can avoid the effect of extreme values.
(2). "imdbRating" variable:
Since there are only two NA's in it and it is not reasonable to replace it with the mean or the median, we delete the whole rows.
(3). "famous_wins", "famous_nominations", "other_nominations" and "other_wins" variables:
We replaced NA's by zero since it is appropriate to believe that the movie has no rewards or nominations when there is no information about it.

4. Model

4.1 Features evaluated

We wanted to build models with following candidate datasets to know which combination of predictors has the best prediction performance.

(1). Original dataset (3139 predictors):
It contains all the word counts variables.
(2). Plus1 dataset (3143 predictors):
It contains all the word counts variables (3139) and comment feature variables (4).
(3). Plus2 dataset (3279 predictors):
It contains all the word counts variables (3139) and movie feature variables (140).
(4). PlusPlus dataset (3283 predictors):
It contains all the word counts variables (3139), comment feature variables (4) and movie feature variables (140).
(5). PlusPlus dataset with tf-idf transformation (3283predictors):
It contains tf-idf transformed word counts variables (3139), comment feature variables (4), and movie feature variables (140).

4.2 Models evaluated

4.2.1 Logistic Regression

It can be used to do binary classification. Unlike SVM, which directly give us the result that a user's rating for a movie is positive or negative, logistic regression tells us the probability of a positive rating. When the predicted value is larger than 0.5, it is regarded as positive rating. And when it is less than 0.5, it is regarded as negative rating. After tuning the parameter, we found the best cost is 0.0001.

4.2.2 Random Forest

The basic idea of random forest is bagging and fully-grown CART (Classification And Regression Tree). We firstly build many CART with bootstrapping and then average their results. This algorithm is parallel and efficient to learn since all the trees are independent and CART itself is efficient. Moreover, it inherits performance advantages from CART, like it can process multiclass problems and process categorical variables. Also through bagging, it can eliminate disadvantages from fully-grown tree which easily leads to overfitting. We found the optimal number of trees for different candidate datasets are also different. They are 200(Original), 160(Plus1), 180(Plus2), 190(PlusPlus) and 190(PPw/tf-idf).

4.2.3 XGBoost (Extreme Gradient Boosting)

XGBoost improves on gradient boosting. The basic idea of gradient boosting is to build each tree using gradient descent, which means based on the trees generated, it takes an appropriate step to a direction in which optimizes the objective function. Under reasonable parameters, it usually needs a lot of trees to obtain a satisfactory accuracy rate. So it may take thousands of iterations when the dataset is large and thus it takes much time to process. XGBoost solves this problem. It runs very fast and also increases the precision by improving the algorithm. After tuning the parameters, we found that the test error is minimized when max.depth is 6, eta is 1, nthread is 3 and nrounds is 14.
Moreover, the optimal parameters are the same for different candidate datasets.

4.2.4 Radial Basis Function SVM

Since hard-margin SVM is too strict and easily leads to overfitting, we tend to use the soft-margin SVM. It introduces a parameter c, which represents the trade-off of large margin and noise tolerance. The larger the c, the less misclassification we want to make. Although linear SVM is easy to understand, it is simple and we want to make our boundary more sophisticated and thus our model more powerful. Therefore, radial basis function SVM is used instead of linear SVM.

4.3 Feature and model selection

After tuning the parameters, we obtained the test errors for each combination of classification method and candidate dataset, so there are 20 of them. The test errors are shown in table 4.1 and figure 4.1.

Test Error	Logistic	Random Forest	XGBoost	RBF SVM
Original	0.1489	0.1672	0.1875	0.1861
Plus1	0.1492	0.1590	0.1825	0.1855
Plus2	0.1494	0.1639	0.1930	0.1824
PlusPlus	0.1486	0.1543	0.1741	0.1833
PPw/tf-idf	0.1575	0.1545	0.1826	0.1835

Table 4.1
Test Error for feature-method combinations

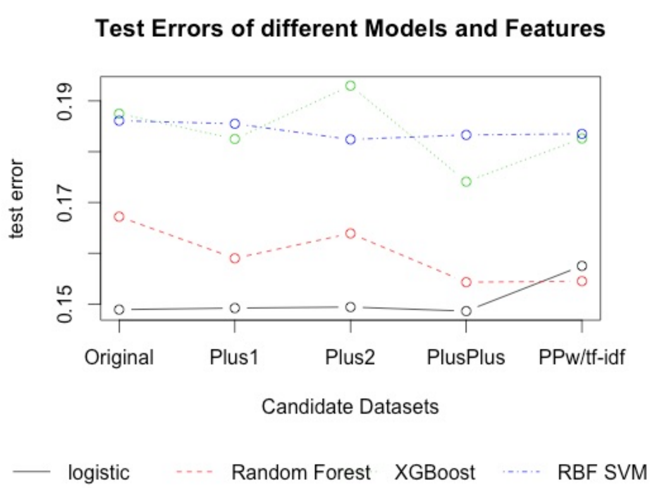

Figure 4.1
Test Error for feature-method combinations

We found that dataset "PlusPlus" has the best prediction performance, since the smallest test error of each method all appeared in the dataset "PlusPlus" except RBF SVM. Also, tf-idf transformation did not improve the performance.
From graph 4.1, it is clearly shown that Logistic Regression and Random Forest performed better than the other 2 methods in our case. For the dataset "PlusPlus", XGBoost performed better than RBF SVM.
So we decided to further investigate the performance of Logistic Regression, Random Forest and XGBoost in the dataset "PlusPlus". We also drew the ROC curve (figure 4.2) for them.

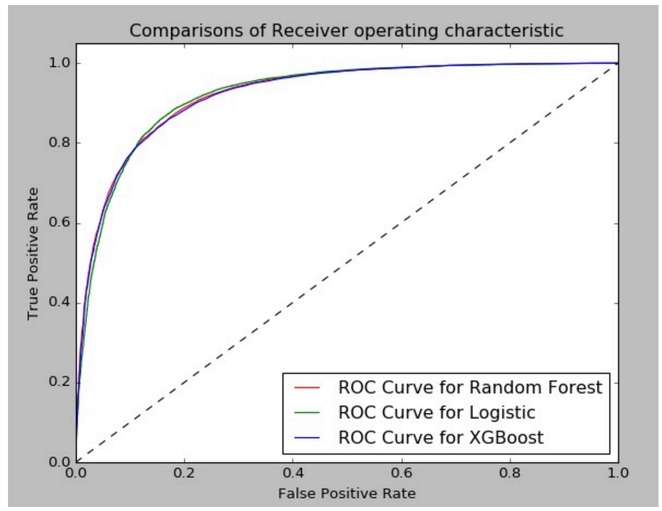

Figure 4.2
ROC curves

ROC Curves are used to see how well the classifier can separate positive and negative examples and to identify the best threshold for separating them. The diagonal line in this graph is the base line, that can be obtained with a random classifier. The further our ROC curve from this line, the better. The ROC Curves for Logistic Regression, Random Forest and XGBoost were reasonable and seemed to show respective strength.

4.4 Voting Classifier

It is natural that different classifier has their individual strength and weakness. So if we can combine them together, we might balance out their weaknesses and get better results. There are two different kinds of voting procedures. One is majority voting. The predicted class label is the class label that represents the majority of the class labels predicted by each individual classifier.

Another one is soft voting. It returns the class label with the highest sum of predicted weighted probabilities. Specific weights should be assigned to each classifier at first. Soft voting is more flexible and comprehensive in considering all the classifiers by their weights.
We choose the best three models under the application with “PlusPlus” dataset: Logistic Regression, Random Forest and XGBoost to vote.
The majority voting returns 0.1473 test error, while soft voting returns 0.1432 test error. It is clear that voting classifications improved the performance of the model. Finally, we chose soft voting as our model.

5. Conclusion and discussion

5.1 Feature Importance

We thought it is crucial to find several most important features to make classification for each method. The top 10 important features for Logistic Regression, Random Forest and XGBoost were shown in figure 5.1, 5.2 and 5.3.

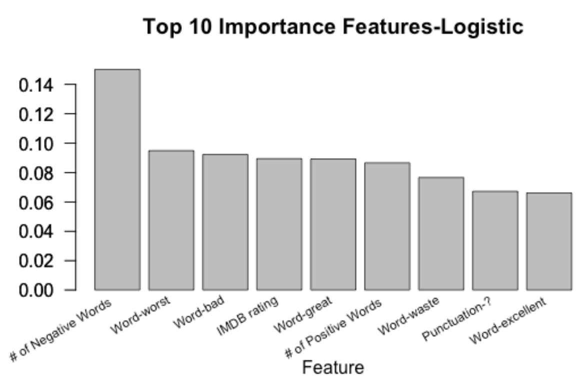

Figure 5.1
Top 10 Important features – Logistic Regression

We found that the 3 methods all selected “Number of pure negative words” as the most important feature. The features they all selected included “Number of pure positive words”, “imdbRating”, “Word-bad”, “Word-worst” and “Word-great”. Furthermore, “imdbVotes”, “punctuation-?” and “Word-excellent” were selected by two methods and “Length of comment”, “Runtime” and “Word-waste” were also of importance for being selected by one method.

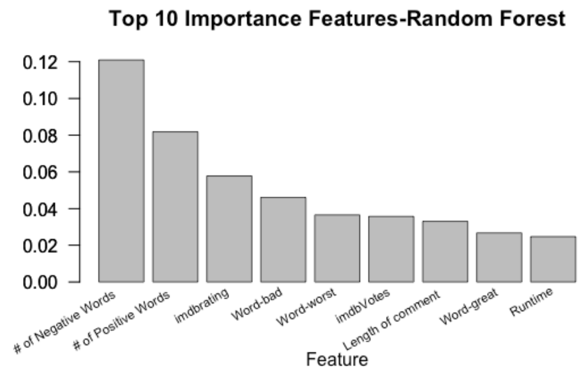

Figure 5.2
Top 10 Important features – Random Forest

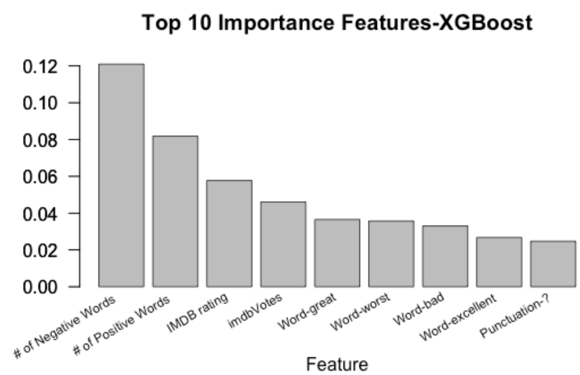

Figure 5.3
Top 10 Important features – XGBoost

5. 2 Limitation of tf-idf transformation

In our project, we considered using tf-idf transformed dataset to train our models. But the previous test error result showed that it didn’t help improve the performance of the models. And we think because we have pre-processed our original word counts matrix, that is deleting the rare words, only keeping the important words. Some comments can be classified by certain tf-idf transformed words, but these words have been deleted already. The remain words and their counts are not representative after tf-idf transformation, which means it was inappropriate to use tf-idf transformation here.

5.3 Improve Features

In machine learning problem, features are important for the performance of the model. They are not only representing the attributes, but also helping us understanding the context of problem. If the feature is useful, it usually improves the model. On the one hand, we can obtain more features by exploring other features, such as the movie's box office and the production country, which are related to the movie rating. One the other hand, we can do feature engineering to create features. For example, some directors are good at certain movie genre, so the combination of director and genre could influence the movie quality and movie rating. This information is under the relations between the exist features. Although we have done this part, the features are endless and our work is not the end.

References

[1] https://en.wikipedia.org/wiki/IMDb

[2] http://ai.stanford.edu/~amaas//data/sentiment/

[3] http://www.boxofficemojo.com/people/?view=Actor&pagenu=1&sort=sumgross&order=DESC&&p=.htm

[4] http://www.omdbapi.com

[5] S.B. Kotsiantis (2007). Supervised Machine Learning: A review of Classification Techiniques. *Informatica* 31: 249-268

[6] Nithin VR, Pranav M, Sarath Babu PB, Lijiya A(2014). Predicting Movie Success Based on IMDb Data. *International Journal of Data Mining Techniques and Applications* Volume 03: 365-368

[7] Jeffrey Ericson, Jesse Grodman. A Predictor for Movie Success. *Stanford CS229 Project*

[8] Dan Cocuzzo, Stephen Wu. Hit or Flop: Box Office Prediction for Feature Films. *Stanford CS229 Project*

Appendix: R and Python Code

#STA208 Final Report
#Classify user's rating based on IMDB data
#Huiyu Bi, Miao Wang, Yuan Tian

```
#R code
library(tm)
library(wordcloud)
library(XML)
library(RCurl)
library(textir)
library(xgboost)
##Comments part
#read data
bow_full= read.table("aclImdb/imdb.vocab.TXT")
bow_full = sapply(bow_full, as.character)
train_word = readLines("aclImdb/train/labeledBow.feat.txt")
test_word = readLines("aclImdb/test/labeledBow.feat.txt")
train_word_pro=train_word[1:12500]
train_word_neg=train_word[12501:25000]
test_word_pro=test_word[1:12500]
test_word_neg=test_word[12501:25000]
#remove stop wprds
stop_words = stopwords("en")
index = bow_full%in%stop_words
stop_words_index = which(index == TRUE)
bow_nostop = bow_full[-stop_words_index]
#read one comment fuction
insert_num = function(comment)
 {
 train_pos_fre = rep(0,times = 89527)
 len = length(comment)
 for (i in 1:len)
   {
   df1 = as.numeric(comment[[i]][1])+1
   df2 = as.numeric(comment[[i]][2])
   train_pos_fre[df1] = df2}
 train_pos_fre = train_pos_fre[-stop_words_index]
 return (train_pos_fre)
 }
read_onecom = function(com)
{
 ori = strsplit(com," ")[[1]][-1]
 len = length(ori)
 splitcom = sapply(1:len,function(x) strsplit(ori[x],":"))
 insert_num(splitcom)
}
sep_fre = lapply(train_word_pro, read_onecom)
pos_fre = do.call(rbind, sep_fre)
sep_fre1 = lapply(train_word_neg, read_onecom)
neg_fre = do.call(rbind, sep_fre1)
seq_fre2 = lapply(test_word_pro, read_onecom)
pos_fre_test = do.call(rbind, seq_fre2)
seq_fre3 = lapply(test_word_neg, read_onecom)
```

```
neg_fre_test = do.call(rbind, seq_fre3)
#count total number of words in one comment
count_total_num =
 function(comment)
 {
   len = length(comment)
   df2 = rep(0,times=len)
   for (i in 1:len)
   {
   df2[i] = as.numeric(comment[[i]][2])
   }
   total = sum(df2)
   return (total)
 }
count_total_onecom =
 function(com)
   {
    ori = strsplit(com," ")[[1]][-1]
    len = length(ori)
    splitcom = sapply(1:len,function(x) strsplit(ori[x],":"))
    count_total_num(splitcom)
    }
#count total number of transitional words in one comment
Transitional = c("but", "yet", "though", "although", "however", "wheras", "nevertheless", "despite",
               "regardless", "nonetheless", "notwithstanding", "rather", "conversely", "opposed",
               "contrast", "contrary","unlike","instead", "otherwise", "while")
trans_index = which(bow_full%in%Transitional == TRUE)
count_total_trans =
 function(comment)
 {
   len = length(comment)
   l = length(trans_index)
   trans = rep(0,len)
   j = 1
   for (i in 1:len)
   {
    df1 = as.numeric(comment[[i]][1])+1
    if(df1 == trans_index[j])
     {
      trans[j] = df2 = as.numeric(comment[[i]][2])
      j = j+1
     }
   }
   total = sum(trans)
   return (total)
 }
count_trans_onecom =
 function(com)
 {
   ori = strsplit(com," ")[[1]][-1]
   len = length(ori)
   splitcom = sapply(1:len,function(x) strsplit(ori[x],":"))
   count_total_trans(splitcom)
 }
seq_fre4 = lapply(train_word, count_trans_onecom)
train_trans_total = do.call(rbind, seq_fre4)
```

```
seq_fre5 = lapply(test_word, count_trans_onecom)
test_trans_total = do.call(rbind, seq_fre5)
#eliminating rare words
pos_total = apply(pos_fre, 2, sum)
neg_total = apply(neg_fre, 2, sum)
pos_proportion = pos_total/sum(pos_total)
neg_proportion = neg_total/sum(neg_total)
rare = pos_proportion < 0.00001
rare_index = which(rare == TRUE)
bow_norare = bow_nostop[-rare_index]
pos_fre_high = pos_fre[ ,pos_proportion > 0.00001]
neg_fre_high = neg_fre[ ,pos_proportion > 0.00001]
#Eliminating features with low information
pos_high_total = apply(pos_fre_high, 2, sum)
neg_high_total = apply(neg_fre_high, 2, sum)
p_to_n = pos_high_total/neg_high_total
n_to_p = neg_high_total/pos_high_total
intense = (p_to_n > 2) | (n_to_p > 2)
intense_index = which(intense == TRUE)
pos_fre_intense = pos_fre_high[ , (p_to_n > 2) | (n_to_p > 2) ]
neg_fre_intense = neg_fre_high[ , (p_to_n > 2) | (n_to_p > 2) ]
bow_final = bow_norare[intense_index]
final = bow_nostop%in%bow_final
final_index = which(final == TRUE)
pos_test_final = pos_fre_test[ , final_index]
neg_test_final = neg_fre_test[ , final_index]
colnames(pos_fre_intense) = bow_final
colnames(neg_fre_intense) = bow_final
colnames(pos_test_final) = bow_final
colnames(neg_test_final) = bow_final
bow = read.csv("final_true.csv")[ ,-1]
bow = as.character(bow)
#training and test data
train = rbind(pos_fre_intense, neg_fre_intense)
test = rbind(pos_test_final, neg_test_final)
y = c(rep(1, 12500), rep(0,12500))
train = cbind(y, train)
test = cbind(y, test)
#pure positive/negative words
pos_comment = read.csv("pos_train.csv", stringsAsFactors = FALSE)[ ,-1]
pos_total = apply(pos_comment, 2, sum)
pos_rank = 3140 - rank(pos_total, ties.method = "random")
poswords = c()
for (i in 1:200) {
 poswords[i] = bow[pos_rank == i]
}
neg_comment = read.csv("neg_train.csv", stringsAsFactors = FALSE)[ ,-1]
neg_total = apply(neg_comment, 2, sum)
neg_rank = 3140 - rank(neg_total, ties.method = "random")
negwords = c()
for (i in 1:200) {
 negwords[i] = bow[neg_rank == i]
}
pure_pos = poswords[-which(poswords%in%negwords)]
pure_neg = negwords[-which(negwords%in%poswords)]
train = read.csv("train.csv", stringsAsFactors = FALSE)[ ,-1]
```

```
test = read.csv("test.csv", stringsAsFactors = FALSE)[ ,-1]
num_pure_pos_train = apply(train[ , which(colnames(train) %in% pure_pos)], 1, sum)
num_pure_neg_train = apply(train[ , which(colnames(train) %in% pure_neg)], 1, sum)
num_pure_pos_test = apply(test[ , which(colnames(test) %in% pure_pos)], 1, sum)
num_pure_neg_test = apply(test[ , which(colnames(test) %in% pure_neg)], 1, sum)
train_plus = cbind(train, train_num, num_pure_pos_train, num_pure_neg_train, train_trans_total)
test_plus = cbind(test, test_num, num_pure_pos_test, num_pure_neg_test, test_trans_total)
#wordclouds
pos_total_df = as.data.frame(pos_total)
neg_total_df = as.data.frame(neg_total)
pos_word_fre = pos_total_df[row.names(pos_total_df)%in% pure_pos, ]
neg_word_fre = neg_total_df[row.names(neg_total_df)%in% pure_neg, ]
pos_df = as.data.frame(cbind(pure_pos, pos_word_fre))
neg_df = as.data.frame(cbind(pure_neg, neg_word_fre))
pos_df$pure_pos = as.character(pos_df$pure_pos)
pos_df$pos_word_fre = as.numeric(pos_df$pos_word_fre)
neg_df$pure_neg = as.character(neg_df$pure_neg)
neg_df$neg_word_fre = as.numeric(neg_df$neg_word_fre)
wordcloud(words = pos_df$pure_pos, freq = pos_df$pos_word_fre, min.freq = 1, scale = c(1.5, 0.2),
        max.words=120, random.order=FALSE, rot.per=0.3,
        colors=brewer.pal(8, "Dark2"))
wordcloud(words = neg_df$pure_neg, freq = neg_df$neg_word_fre, min.freq = 1, scale = c(1.5, 0.2),
        max.words=120, random.order=FALSE, rot.per=0.3,
        colors=brewer.pal(8, "Dark2"))
```

##Movie features part

```
#obtain title id
title_no = function(path){
 urls = read.table(path, stringsAsFactors = FALSE)
 title_no = c()
 for(i in 1:nrow(urls)){
   title_no[i] = gsub('.*/(tt[0-9]+)/.*', '\\1', urls[i,])
 }
 title_no
}
train_pos_title = title_no("train/urls_pos.txt")
train_neg_title = title_no("train/urls_neg.txt")
test_pos_title = title_no("test/urls_pos.txt")
test_neg_title = title_no("test/urls_neg.txt")
#obtain all contents from the websites
data_content = function(title){
 plain_text = c()
 content = c()
 for(i in 1:length(title)){
   url0 = paste0("http://www.omdbapi.com/?i=", title[i], collapse = "")
   url = paste0(url0, "&plot=short&r=json", collapse = "")
   doc_page = getURLContent(url)
   html_page = htmlParse(doc_page, asText = TRUE)
   text = getNodeSet(html_page, "//p")[[1]]
   plain_text = xmlValue(text)
   content[i] = gsub("\"","", plain_text)
 }
 content
}
#now use unique id, after all, expand to original
uniq_train_pos_title = unique(train_pos_title)
```

```
uniq_train_neg_title = unique(train_neg_title)
uniq_test_pos_title = unique(test_pos_title)
uniq_test_neg_title = unique(test_neg_title)
content_train_pos = data_content(uniq_train_pos_title)
content_train_neg = data_content(uniq_train_neg_title)
content_test_pos = data_content(uniq_test_pos_title)
content_test_neg = data_content(uniq_test_neg_title)
#obtain potential variables
information = function(texts){
 inf = c()
 for(i in 1:length(texts)){
   text = texts[i]
   title = gsub(".*Title:(.*),Year.*", "\\1", text)
   year = gsub(".*Year:([0-9]+).*", "\\1", text)
   rated = gsub(".*Rated:(.*),Released.*", "\\1", text)
   runtime = gsub(".*Runtime:(N/A|[0-9]+).*", "\\1", text)
   genre = gsub(".*Genre:(.*),Director.*", "\\1", text)
   director = gsub(".*Director:(.*),Writer.*", "\\1", text)
   actors = gsub(".*Actors:(.*),Plot.*", "\\1", text)
   language = gsub(".*Language:(.*),Country.*", "\\1", text)
   awards = gsub(".*Awards:(.*),Poster.*", "\\1", text)
   metascore = gsub(".*Metascore:(.*),imdbRating.*", "\\1", text)
   imdbrating = gsub(".*imdbRating:(.*),imdbVotes.*", "\\1", text)
   imdbVotes = gsub(".*imdbVotes:(.*),imdbID.*", "\\1", text)
   imdbid = gsub(".*imdbID:(.*),(seriesID|Type).*", "\\1", text)
   type = gsub(".*Type:(.*),Response.*", "\\1", text)
   all = cbind(title, year, rated, runtime, genre, director, actors, language,
            awards, metascore, imdbrating, imdbVotes, imdbid, type)
   inf = rbind(inf, all)
 }
 inf
}
data_inf_train_pos = information(content_train_pos)
data_inf_train_neg = information(content_train_neg)
data_inf_test_pos = information(content_test_pos)
data_inf_test_neg = information(content_test_neg)
#variable transformation
#rate
merge_rate = function(data){
 data$rated[data$rated == "M"] = "PG"
 data$rated[data$rated == "GP"] = "PG"
 data$rated[data$rated == "Approved"] = "G"
 data$rated[data$rated == "APPROVED"] = "G"
 data$rated[data$rated == "PASSED"] = "G"
 data$rated[data$rated == "UNRATED"] = "X"
 data$rated[data$rated == "Unrated"] = "X"
 data$rated[data$rated == "NOT RATED"] = "N/A"
 data
}
merge_train_pos = merge_rate(data_inf_train_pos)
merge_train_neg = merge_rate(data_inf_train_neg)
merge_test_pos = merge_rate(data_inf_test_pos)
merge_test_neg = merge_rate(data_inf_test_neg)
#genre
genre = strsplit(merge_train_pos$genre, split = ", ")
genre_length = sapply(genre, length)
```

```
uniq_genre = unique(unlist(genre))
count = function(data, inf){
 index = which(inf == colnames(data))
 split_inf = strsplit(data[, index], split = ", ")
 uniq_inf = unique(unlist(split_inf))
 all_count = c()
 for(j in 1:length(split_inf)){
   one_count = c()
   for(i in 1:length(uniq_inf)){
     one_count[i] = sum(as.numeric(split_inf[[j]] == uniq_inf[i]))
   }
   all_count = rbind(all_count, one_count)
 }
 rownames(all_count) = seq(1, length(split_inf), 1)
 colnames(all_count) = uniq_inf
 all_count
}
genre_train_pos = cbind(merge_train_pos, count(merge_train_pos, "genre"))
genre_train_neg = cbind(merge_train_neg, count(merge_train_neg, "genre"))
genre_test_pos = cbind(merge_test_pos, count(merge_test_pos, "genre"))
genre_test_neg = cbind(merge_test_neg, count(merge_test_neg, "genre"))
#language
other_inf_languages = function(data){
 language = strsplit(data$language, split = ", ")
 ##how many languages in a movie
 language_length = sapply(language, length)
 main_language = sapply(language, '[', 1)
 cbind(No_language = language_length, main_language)
}
lan_genre_train_pos = cbind(genre_train_pos, other_inf_languages(merge_train_pos), count(merge_train_pos, "language"))
lan_genre_train_neg = cbind(genre_train_neg, other_inf_languages(merge_train_neg), count(merge_train_neg, "language"))
lan_genre_test_pos = cbind(genre_test_pos, other_inf_languages(merge_test_pos), count(merge_test_pos, "language"))
lan_genre_test_neg = cbind(genre_test_neg, other_inf_languages(merge_test_neg), count(merge_test_neg, "language"))
#awards
wins_nominations = function(data){
 award = data$awards
 famous_wins = regmatches(award, gregexpr("^Won [0-9]+", award))
 famous_wins = sapply(famous_wins, "[", 1)
 famous_wins = sapply(famous_wins, function(x){
   regmatches(x, gregexpr("[0-9]+", x))
 }
 )
 famous_wins = as.numeric(sapply(famous_wins, "[", 1))
 famous_notations = regmatches(award, gregexpr("^Nominated for [0-9]+", award))
 famous_notations = sapply(famous_notations, "[", 1)
 famous_notations = sapply(famous_notations, function(x){
   regmatches(x, gregexpr("[0-9]+", x))
 }
 )
 famous_notations = as.numeric(sapply(famous_notations, "[", 1))
 other_wins = regmatches(award, gregexpr("[0-9]+ win(s)?", award))
 other_wins = sapply(other_wins, "[", 1)
```

```
 other_wins = sapply(other_wins, function(x){
   regmatches(x, gregexpr("[0-9]+", x))
 }
 )
 other_wins = as.numeric(sapply(other_wins, "[", 1))
 other_nomins = regmatches(award, gregexpr("[0-9]+ nomination(s)?", award))
 other_nomins = sapply(other_nomins, "[", 1)
 other_nomins = sapply(other_nomins, function(x){
   regmatches(x, gregexpr("[0-9]+", x))
 }
 )
 other_nomins = as.numeric(sapply(other_nomins, "[", 1))
 all = cbind(famous_wins, other_wins, famous_notations, other_nomins)
 all[is.na(all)] = "N/A"
 all
}
lan_genre_train_pos = cbind(lan_genre_train_pos, wins_nominations(merge_train_pos))
lan_genre_train_neg = cbind(lan_genre_train_neg, wins_nominations(merge_train_neg))
lan_genre_test_pos = cbind(lan_genre_test_pos, wins_nominations(merge_test_pos))
lan_genre_test_neg = cbind(lan_genre_test_neg, wins_nominations(merge_test_neg))
#directors and actors
#top directors and actors
directors = read.csv("Dir.csv", stringsAsFactors = FALSE)[, -1]
actors = read.csv("Act.csv", stringsAsFactors = FALSE)[, -1]
directors_actors = function(data, inf){
 index = which(inf == colnames(data))
 value = strsplit(data[, index], split = ", ")
 value1 = sapply(1:length(value), function(i) gsub("[^A-Za-z]", "", value[[i]]))
 value2 = sapply(1:length(value), function(i) gsub("^[0-9]", "", tolower(value1[[i]])))
 final_value = do.call(rbind, value2)
 final_value
}
famous_directors = gsub("[^A-Za-z]", "", tolower(directors$name))
famous_actors = gsub("[^A-Za-z]", "", tolower(actors$name))
ranks = function(data, inf, famous){
 dire_act = directors_actors(data, inf)
 ind_rank = list()
 for(j in 1:ncol(dire_act)){
   indexes = which(dire_act[, j] %in% famous)
   rank = c()
   for(i in 1:length(indexes)){
     ind = indexes[i]
     rank[i] = which(famous %in% dire_act[ind, j])
   }
   ind_rank[[j]] = cbind(indexes, rank)
   dire_act[ind_rank[[j]][, 1], j] = ind_rank[[j]][, 2]
   dire_act[-ind_rank[[j]][, 1], j] = 0
 }
 apply(dire_act, 1, function(x){
   min(x[x != 0])
 })
}
ranks_directors_train_pos = ranks(lan_genre_train_pos, "director", famous_directors)
ranks_actors_train_pos = ranks(lan_genre_train_pos, "actors", famous_actors)
ranks_directors_train_neg = ranks(lan_genre_train_neg, "director", famous_directors)
ranks_actors_train_neg = ranks(lan_genre_train_neg, "actors", famous_actors)
```

```
ranks_directors_test_pos = ranks(lan_genre_test_pos, "director", famous_directors)
ranks_actors_test_pos = ranks(lan_genre_test_pos, "actors", famous_actors)
ranks_directors_test_neg = ranks(lan_genre_test_neg, "director", famous_directors)
ranks_actors_test_neg = ranks(lan_genre_test_neg, "actors", famous_actors)
defined_ranks = function(data){
 ranks_data = as.numeric(data)
 ranks_data[is.na(ranks_data)] = 0
 ranks_data[ranks_data < 50 & ranks_data > 0] = 10
 ranks_data[ranks_data < 100 & ranks_data >= 50] = 9
 ranks_data[ranks_data < 200 & ranks_data >= 100] = 7
 ranks_data[ranks_data < 400 & ranks_data >= 200] = 4
 ranks_data[ranks_data >= 400] = 1
 ranks_data
}
final_directors_train_pos = defined_ranks(ranks_directors)
final_actors_train_pos = defined_ranks(ranks_actors)
final_directors_train_neg = defined_ranks(ranks_directors_train_neg)
final_actors_train_neg = defined_ranks(ranks_actors_train_neg)
final_directors_test_pos = defined_ranks(ranks_directors_test_pos)
final_actors_test_pos = defined_ranks(ranks_actors_test_pos)
final_directors_test_neg = defined_ranks(ranks_directors_test_neg)
final_actors_test_neg = defined_ranks(ranks_actors_test_neg)
all_final_train_pos = cbind(lan_genre_train_pos, rank_directors = final_directors_train_pos, rank_actors = final_actors_train_pos)
all_final_train_neg = cbind(lan_genre_train_neg, rank_directors = final_directors_train_neg, rank_actors = final_actors_train_neg)
all_final_test_pos = cbind(lan_genre_test_pos, rank_directors = final_directors_test_pos, rank_actors = final_actors_test_pos)
all_final_test_neg = cbind(lan_genre_test_neg, rank_directors = final_directors_test_neg, rank_actors = final_actors_test_neg)
#delete useless variables
all_final_train_pos = all_final_train_pos[, -c(1, 5, 6, 7, 8, 9, 10, 13)]
all_final_train_neg = all_final_train_neg[, -c(1, 5, 6, 7, 8, 9, 10, 13)]
all_final_test_pos = all_final_test_pos[, -c(1, 5, 6, 7, 8, 9, 10, 13)]
all_final_test_neg = all_final_test_neg[, -c(1, 5, 6, 7, 8, 9, 10, 13)]
#make four datasets have the same columns
common_col = Reduce(intersect, list(colnames(all_final_train_pos), colnames(all_final_train_neg), colnames(all_final_test_pos), colnames(all_final_test_neg)))
final_train_pos = all_final_train_pos[, common_col]
final_train_neg = all_final_train_neg[, common_col]
final_test_pos = all_final_test_pos[, common_col]
final_test_neg = all_final_test_neg[, common_col]
#transform type "factor" into "charactor"
final_train_pos[, "main_language"] = as.character(final_train_pos[, "main_language"])
final_train_neg[, "main_language"] = as.character(final_train_neg[, "main_language"])
final_test_pos[, "main_language"] = as.character(final_test_pos[, "main_language"])
final_test_neg[, "main_language"] = as.character(final_test_neg[, "main_language"])
final_train_pos[, "No_language"] = as.character(final_train_pos[, "No_language"])
final_train_neg[, "No_language"] = as.character(final_train_neg[, "No_language"])
final_test_pos[, "No_language"] = as.character(final_test_pos[, "No_language"])
final_test_neg[, "No_language"] = as.character(final_test_neg[, "No_language"])
final_train_pos[, "other_wins"] = as.character(final_train_pos[, "other_wins"])
final_train_neg[, "other_wins"] = as.character(final_train_neg[, "other_wins"])
final_test_pos[, "other_wins"] = as.character(final_test_pos[, "other_wins"])
final_test_neg[, "other_wins"] = as.character(final_test_neg[, "other_wins"])
final_train_pos[, "other_nomins"] = as.character(final_train_pos[, "other_nomins"])
```

```
final_train_neg[, "other_nomins"] = as.character(final_train_neg[, "other_nomins"])
final_test_pos[, "other_nomins"] = as.character(final_test_pos[, "other_nomins"])
final_test_neg[, "other_nomins"] = as.character(final_test_neg[, "other_nomins"])
final_train_pos[, "famous_notations"] = as.character(final_train_pos[, "famous_notations"])
final_train_neg[, "famous_notations"] = as.character(final_train_neg[, "famous_notations"])
final_test_pos[, "famous_notations"] = as.character(final_test_pos[, "famous_notations"])
final_test_neg[, "famous_notations"] = as.character(final_test_neg[, "famous_notations"])
#expand to original dataset
seq_train_pos = as.numeric(table(train_pos_title))
seq_train_neg = as.numeric(table(train_neg_title))
seq_test_pos = as.numeric(table(test_pos_title))
seq_test_neg = as.numeric(table(test_neg_title))
expand = function(n, seq_data, data){
 sub_expand = as.data.frame(matrix(rep(0, seq_data[n]*ncol(data)), ncol= ncol(data)))
 rep = seq_data[n]
 for (i in (1:rep))
 {
   sub_expand [i,] = data[n,]
 }
 return(sub_expand)
}
sep_train_pos = lapply(1:nrow(final_train_pos), function(i) expand(i, seq_train_pos, final_train_pos))
Movie_info_train_pos = do.call(rbind, sep_train_pos)
sep_train_neg = lapply(1:nrow(final_train_neg), function(i) expand(i, seq_train_neg, final_train_neg))
Movie_info_train_neg = do.call(rbind, sep_train_neg)
sep_test_pos = lapply(1:nrow(final_test_pos), function(i) expand(i, seq_test_pos, final_test_pos))
Movie_info_test_pos = do.call(rbind, sep_test_pos)
sep_test_neg = lapply(1:nrow(final_test_neg), function(i) expand(i, seq_test_neg, final_test_neg))
Movie_info_test_neg = do.call(rbind, sep_test_neg)
Movie_info_train = rbind(Movie_info_train_pos, Movie_info_train_neg)
Movie_info_test = rbind(Movie_info_test_pos, Movie_info_test_neg)
colnames(Movie_info_train) = common_col
colnames(Movie_info_test) = common_col
#creat dummy variables
creat_dummy = function(data, inf){
 index = which(inf == colnames(data))
 inf = data[, index]
 uniq_inf = unique(inf)

 all_count = c()
 for(i in 1:length(uniq_inf)){
   one_count = as.numeric(inf == uniq_inf[i])
   all_count = cbind(all_count, one_count)
 }
 colnames(all_count) = uniq_inf
 all_count
}
dummy_rate_train = creat_dummy(Movie_info_train, "rated")
dummy_type_train = creat_dummy(Movie_info_train, "type")
dummy_main_train = creat_dummy(Movie_info_train, "main_language")
colnames(dummy_main_train) = paste("main:", colnames(dummy_main_train))
dummy_rate_test = creat_dummy(Movie_info_test, "rated")
dummy_rate_test = dummy_rate_test[, common_rate]
dummy_type_test = creat_dummy(Movie_info_test, "type")
dummy_type_test = dummy_type_test[, common_type]
dummy_main_test = creat_dummy(Movie_info_test, "main_language")
```

```
colnames(dummy_main_test) = paste("main:", colnames(dummy_main_test))
common_main_lan = Reduce(intersect, list(colnames(dummy_main_train),colnames(dummy_main_test)))
common_type = Reduce(intersect, list(colnames(dummy_type_train),colnames(dummy_type_test)))
common_rate = Reduce(intersect, list(colnames(dummy_rate_train),colnames(dummy_rate_test)))
dummy_main_train_com = dummy_main_train[, common_main_lan]
dummy_main_test_com = dummy_main_test[, common_main_lan]
change_na_zero = function(data, inf){
 index = which(inf == colnames(data))
 inf = data[, index]
 for(i in 1:length(inf)){
   if(inf[i] == "N/A"){
     inf[i] = 0
   }else{
     inf[i] = as.numeric(inf[i])
   }
 }
 as.numeric(inf)
}
train_win_nomins = cbind(famous_wins = change_na_zero(Movie_info_train, "famous_wins"),
                         other_wins = change_na_zero(Movie_info_train, "other_wins"),
                         famous_notations = change_na_zero(Movie_info_train, "famous_notations"),
                         other_nomins = change_na_zero(Movie_info_train, "other_nomins"))
test_win_nomins = cbind(famous_wins = change_na_zero(Movie_info_test, "famous_wins"),
                        other_wins = change_na_zero(Movie_info_test, "other_wins"),
                        famous_notations = change_na_zero(Movie_info_test, "famous_notations"),
                        other_nomins = change_na_zero(Movie_info_test, "other_nomins"))
which(colnames(Movie_info_train) %in%c("rated", "type", "main_language", "famous_wins", "other_wins",
"famous_notations", "other_nomins"))
Movie_info_train = Movie_info_train[, -c(2, 6, 36, 81, 82, 83, 84)]
Movie_info_test = Movie_info_test[, -c(2, 6, 36, 81, 82, 83, 84)]
trans_movie_inf_train = cbind(Movie_info_train, dummy_rate_train, dummy_type_train, dummy_main_train_com,
train_win_nomins)
trans_movie_inf_test = cbind(Movie_info_test, dummy_rate_test, dummy_type_test, dummy_main_test_com,
test_win_nomins)
num_movie_inf_train = apply(trans_movie_inf_train, 2, as.numeric)
num_movie_inf_test = apply(trans_movie_inf_test, 2, as.numeric)
table(is.na(num_movie_inf_train[, "imdbVotes"]))
table(is.na(num_movie_inf_train[, "runtime"]))
summary(num_movie_inf_train[, "imdbVotes"])
#   Min. 1st Qu.  Median    Mean   3rd Qu.    Max.      NA's
#   12     296     815     10610    3259    1212000      18
summary(num_movie_inf_train[, "runtime"])
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#     1      83      93      93      105     883     799
summary(num_movie_inf_test[, "imdbVotes"])
#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#    5     274     752    8177    2940 1212000      59
summary(num_movie_inf_test[, "runtime"])
#   Min. 1st Qu.  Median   Mean   3rd Qu.    Max.     NA's
#  1.00   83.00   93.00   93.57   104.00    729.00     932
num_movie_inf_train[which(is.na(num_movie_inf_train[,"imdbVotes"])), "imdbrating"]
#remove NA
num_movie_inf_train[, "imdbVotes"][is.na(num_movie_inf_train[, "imdbVotes"])] = 815
num_movie_inf_train[, "runtime"][is.na(num_movie_inf_train[, "runtime"])] = 93
num_movie_inf_test[, "imdbVotes"][is.na(num_movie_inf_test[, "imdbVotes"])] = 752
num_movie_inf_test[, "runtime"][is.na(num_movie_inf_test[, "runtime"])] = 93
```

```
colnames(num_movie_inf_test)
a = sapply(1:140, function(i){
 table(is.na(num_movie_inf_test[, i]))
})
a[[1]] #year
a[[3]] #imdbrating
colnames(train_only_trans)
a = sapply(1:3140, function(i){
 table(is.na(train_only_trans[, i]))
})
unlist(a)
na_indexes = which(is.na(num_movie_inf_test[, "year"]))
num_movie_inf_test = num_movie_inf_test[-na_indexes, ]
dictionary_test2 = dictionary_test[-na_indexes, ]
trans_movie_inf_train$imdbVotes = gsub(",", "", trans_movie_inf_train$imdbVotes)
trans_movie_inf_test$imdbVotes = gsub(",", "", trans_movie_inf_test$imdbVotes)
#plus plus
train_plus_plus = cbind(dictionary_train, num_movie_inf_train)
test_plus_plus = cbind(dictionary_test2, num_movie_inf_test )
#original
train_only = read.csv("train.csv", stringsAsFactors = FALSE)[, -1]
test_only = read.csv("test.csv", stringsAsFactors = FALSE)[, -1]
test_only = test_only[-na_indexes,]
#plus minus
train_plus_minus = cbind(train_only, num_movie_inf_train)
test_plus_minus = cbind(test_only, num_movie_inf_test)
#tf_idf transformation
train_only_trans = cbind(train_only[, 1], tfidf(train_only[, -1], normalize = TRUE))
test_only_trans = cbind(test_only[, 1], tfidf(test_only[, -1], normalize = TRUE))
train_only_trans = train_only_trans[-tf_na_index,]
test_only_trans = test_only_trans[-tf_na_index_test,]
tf_train_plus = cbind(train_only_trans, dictionary_train[, 3141:3144])
tf_test_plus = cbind(test_only_trans, dictionary_test2[, 3141:3144])
tf_train_plus = tf_train_plus[-tf_na_index, ]
tf_test_plus = tf_test_plus[-tf_na_index_test, ]
tf_train_plus_minus = cbind(train_only_trans, num_movie_inf_train)
tf_test_plus_minus = cbind(test_only_trans, num_movie_inf_test)
tf_train_plus_minus = tf_train_plus_minus[-tf_na_index, ]
tf_test_plus_minus = tf_test_plus_minus[-tf_na_index_test, ]
tf_train_plus_plus = cbind(train_only_trans, dictionary_train[, 3141:3144], num_movie_inf_train)
tf_test_plus_plus = cbind(test_only_trans, dictionary_test2[, 3141:3144], num_movie_inf_test)
tf_train_plus_plus = tf_train_plus_plus[-tf_na_index,]
tf_test_plus_plus = tf_test_plus_plus[-tf_na_index_test,]
#Build models
#gradient boosting
#tune parameters
 tune_xgboost = function(train_data, test_data, md, et, nth, nr){
   param = list(max.depth = md, eta = et, nthread = nth, objective = "binary:logistic")
   bst = xgboost(params = param, data = as.matrix(train_data)[, -1], label = as.matrix(train_data)[, 1], nrounds = nr)
   y_pred = predict(bst, as.matrix(test_data)[, -1])
   y_pred[y_pred >= 0.5] = 1
   y_pred[y_pred < 0.5] = 0
   1 - classAgreement(table(y_pred, as.matrix(test_data)[, 1]))$diag
 }
#plus_plus, tune max_depth
sapply(6:10, function(i){
```

```
 tune_xgboost(train_plus_plus, test_plus_plus, i, 1, 3, 5)
})
#1:10   choose 6
#0.2341949 0.2101472 0.2037852 0.1915813 0.1872599 0.1844990 0.1869798 0.1865397 0.1967430 0.1899808
#plus_plus, eta
sapply(seq(0, 1, 0.1), function(i){
 tune_xgboost(train_plus_plus, test_plus_plus, 6, i, 3, 5)
})
#0, 0.1, ..., 1   choose 1
#0.4998800 0.2070663 0.2059859 0.2017446 0.1933819 0.1926617 0.1890605 0.1871799
#0.1852593 0.1899008 0.1844990
##plus plus, nth
sapply(seq(1, 10, 1), function(i){
 tune_xgboost(train_plus_plus, test_plus_plus, 6, 1, i, 14)
})
##the same
#0.184499
##plus plus, nround
sapply(seq(11, 15, 1), function(i){
 tune_xgboost(train_plus_plus, test_plus_plus, 6, 1, 3, i)
})
#2:15   choose 14
#0.2139885 0.1966629 0.1906610 0.1844990 0.1832186 0.1811780 0.1798175 0.1772567 0.1767766
#0.1767366 0.1748560 0.1741757 0.1741357 0.1750560
##plus plus
tune_xgboost(train_plus_plus, test_plus_plus, 6, 1, 3, 14)
#0.1741357
#tf plus plus
tune_xgboost(tf_train_plus_plus, tf_test_plus_plus, 6, 1, 3, 14)
#0.1825836
#orignal
tune_xgboost(train_only, test_only, 6, 1, 3, 14)
# 0.1875
#plus
tune_xgboost(dictionary_train, dictionary_test2, 6, 1, 3, 14)
#0.1724552
#plus_minus
tune_xgboost(train_plus_minus, test_plus_minus, 6, 1, 3, 14)
#0.1929818
```

##python code:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import os
from sklearn import svm,preprocessing, cross_validation,neighbors
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
```

```
Trans = pd.read_csv("train_plus_plus.csv")
Test= pd.read_csv("test_plus_plus.csv")
Trans_lable =Trans.ix[:,1]
Trans_data = pd.DataFrame(preprocessing.scale(Trans.ix[:,2:]))
Test_lable = Test.ix[:,1]
Test_data = pd.DataFrame(preprocessing.scale(Test.ix[:,2:]))
def test_error(yhat,y):
    n = float(len(y))
    return np.sum((y - yhat)**2)/n
#RBF SVM
rbf_testerror=[]
gamma_para = range(1,11,1)
for i in gamma_para:
gamma = gamma_para[i]
rbf_svc = svm.SVC(kernel='rbf', gamma = gamma)
rbf_svc.fit(Trans_data, Trans_lable)
yhat = rbf_svc.predict(Test_data)
rbf_testerror.append(test_error(yhat,Test_data))
#Logistic regression
log = LogisticRegression(C = 0.0001)
log.fit(Trans_data, Trans_lable)
tlog =log.predict(Test_data)
test_error(tlog,Test_lable)
#0.15336
#p-0.1922
#Random forest
#tune
error=[]
C_para = (1,10,50,100)
for i in C_para:
c =i
rfc = RandomForestClassifier(n_estimators=c)
rfc.fit(Trans_data, Trans_lable)
trfc = rfc.predict(Test_data)
error.append(test_error(trfc,Test_lable))
rfc = RandomForestClassifier(n_estimators=190)
rfc.fit(Trans_data, Trans_lable)
trfc = rfc.predict(Test_data)
test_error(trfc,Test_lable)
#Voting Classifier
#Hard
eclf = VotingClassifier(estimators=[('rbf', rbf_svc), ('log', log), ('rfc', rfc)], voting='hard')
eclf.fit(Trans_data, Trans_lable)
teclf = eclf.predict(Test_data)
test_error(teclf,Test_lable)

#Soft
eclf1 = VotingClassifier(estimators=[('rbf', rbf_svc), ('log', log), ('rfc', rfc)], voting='soft', weights=[1,2,2])
eclf1.fit(Trans_data, Trans_lable)
teclf1 = eclf1.predict(Test_data)
test_error(teclf1,Test_lable)
#define a function that can give the plot of ROC, and score
def rocplot(methods, X_train, Y_train, X_test, Y_test,i):
methods.fit(X_train, Y_train)
Y_score = methods.predict_proba(X_test)
fpr,tpr,_ = roc_curve(Y_test, Y_score[:,1],pos_label=1)
```

```
plt.figure()
plt.plot(fpr, tpr,i)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic curve')
predictions = rfc.predict_proba(Test_data)
false_positive_rate, true_positive_rate, thresholds = roc_curve(Test_lable, predictions[:, 1])
predictions1 = log.predict_proba(Test_data)
false_positive_rate1, true_positive_rate1, thresholds = roc_curve(Test_lable, predictions1[:, 1])
predictions3 = clas.predict_proba(Test_data1)
false_positive_rate3, true_positive_rate3, thresholds = roc_curve(Test_lable1, predictions3[:, 1])
plt.plot(false_positive_rate, true_positive_rate,"r",label = "ROC Curve for Random Forest")
plt.plot(false_positive_rate1, true_positive_rate1,"g",label = "ROC Curve for Logistic")
plt.plot(false_positive_rate3, true_positive_rate3,"b",label = "ROC Curve for XGBoost")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparisons of Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```