Classification of Movie Quality

Wenshu Yang

Part A: Introduction

A data set containing information of about 5000 movies, including features such as gross and budget, actors' facebook likes and the number of reviews, is used in this classification project.

The research goal is to classify the movies into four different classes of quality. The response variable is a categorical variable which comes from the IMDB scores of the movies. Movies with 0-4 IMDB scores are categorized as "poor"; those with 4-6 scores are categorized as "fair"; those with 6-8 are "good"; and those with 8-10 are "excellent".

The feature space considers all the variables in the data set except imdb_score and the variables with text such as the name of the actors which cannot be transformed into binary ones. As for the classification algorithms, three methods including Naive Bayes, support vector machine, and neural network are applied.

The data source is the IMDB 5000 Movie Dataset [IMDB 5000 Movie Dataset]https://www.mozilla.org.

Part B: Data Exploration and Preprocessing

1. Simple data cleaning

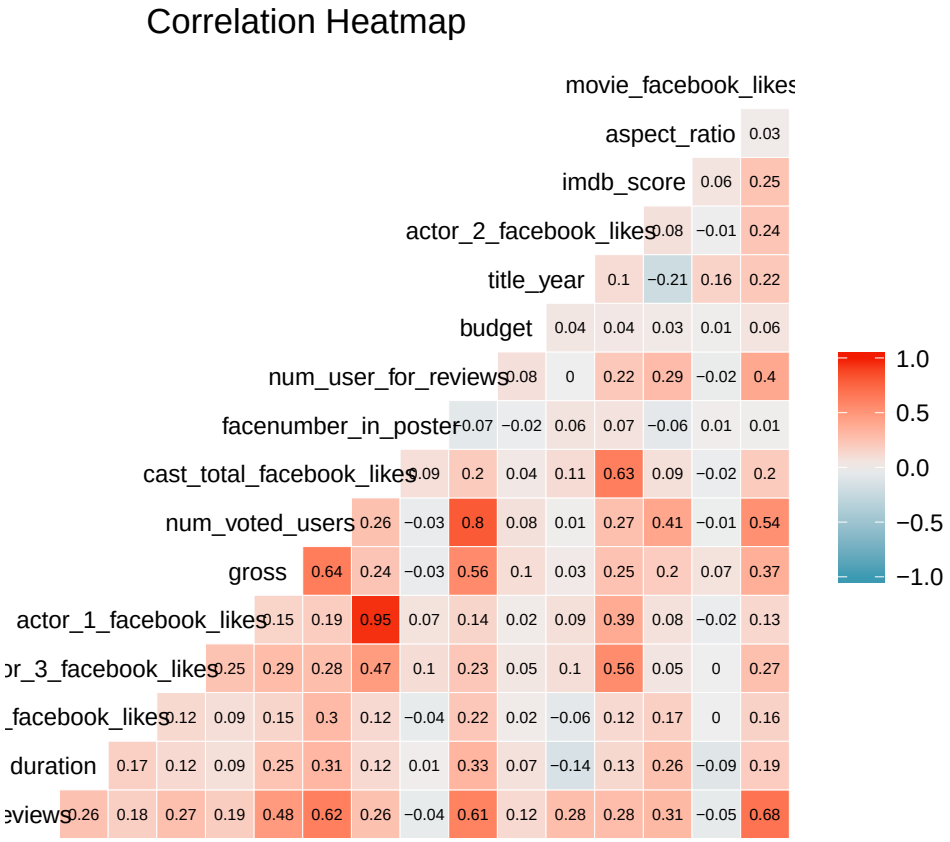
```
# Get the data
movie <- read.csv("D:/NYU/COURSE/Materials/2019 Spring/2011 Supervised and Unsupervised Machine Learning

# Remove duplicates
movie <- movie[!duplicated(movie), ]

# Remove text variables such as names and titles
movie <- movie %>% dplyr::select(-c(director_name, actor_2_name, actor_1_name, movie_title, actor_3_name

nrow(movie) # 4998 obs in the data set after removing duplications
```

```
## [1] 4998
```

```
# Convert title year into the number of years relative to the baseline earliest year
movie$title_year <- movie$title_year - min(movie$title_year, na.rm = TRUE)
```

2. Correlation between features

```
# Correlation heat map
ggcorr(movie, label = TRUE, label_round = 2, label_size = 2, size = 3, hjust = 0.85) +
  ggtitle("Correlation Heatmap") +
  theme(plot.title = element_text(hjust = 0.3))
```

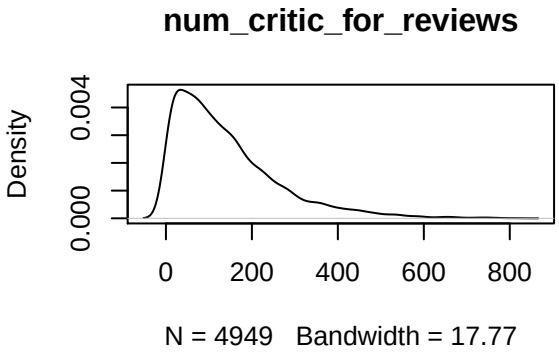


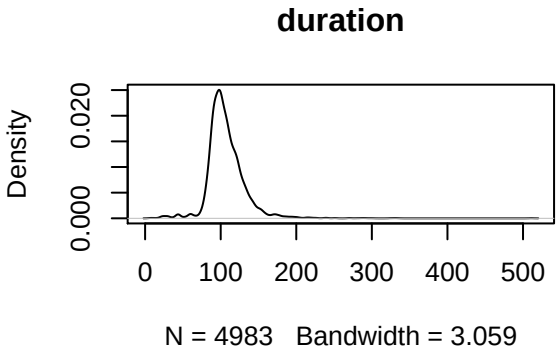
The correlation heat map shows that most of the features are not highly correlated, except "actor_1_facebook_likes" and "cast_total_facebook_likes". These two features have a correlation up to 0.95. Another two features that are quite correlated are the number of user for reviews and the number of voted users.

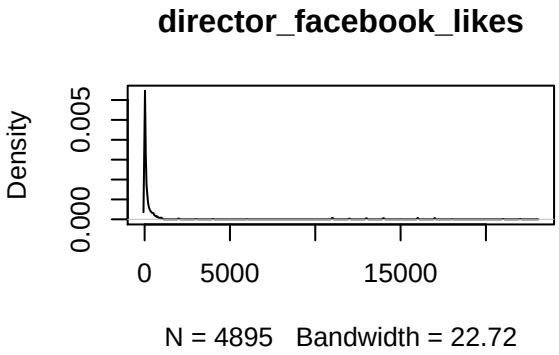
3. Exploration of continuous variables

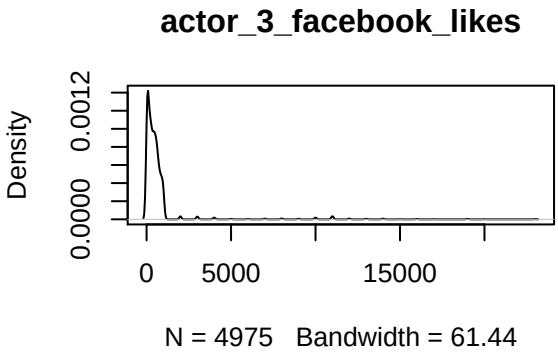
Density plots

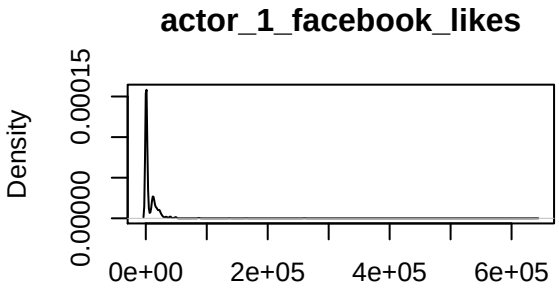
```
movie.continous <- dplyr::select(movie, -c(color, language, country, content_rating, imdb_score, facenu

# Density plots of continuous variables
par(mfrow = c(2, 2))
for (i in 1:ncol(movie.continous)) {
  plot(density(movie.continous[, i], na.rm = TRUE), main = colnames(movie.continous)[i])
} # All of them are highly right skewed
```

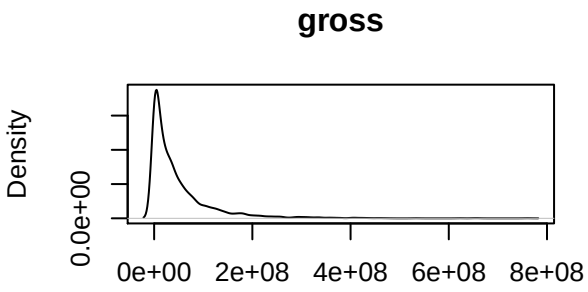

num_critic_for_reviews
Density
0.000
0.004
0
200
400
600
800
N = 4949 Bandwidth = 17.77

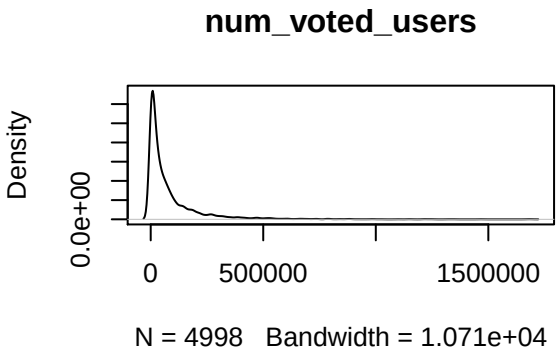

duration
Density
0.000
0.020
0
100
200
300
400
500
N = 4983 Bandwidth = 3.059

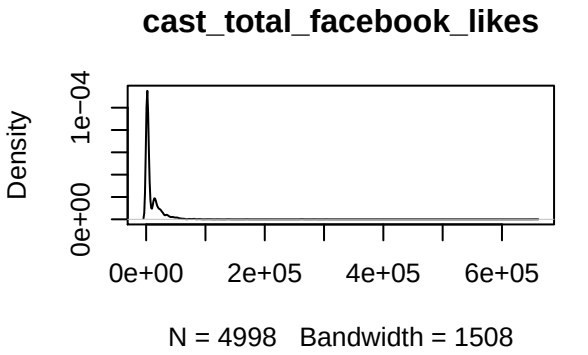

director_facebook_likes
Density
0.000
0.005
0
5000
15000
N = 4895 Bandwidth = 22.72

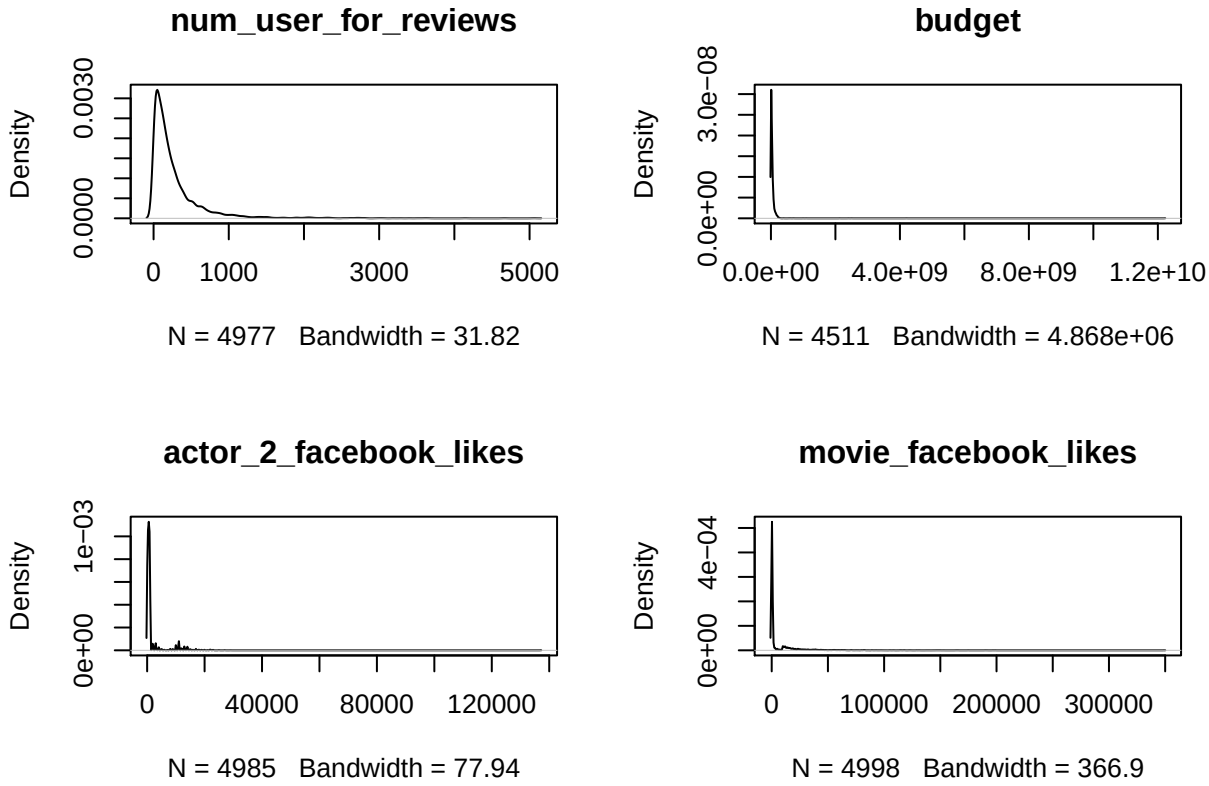

actor_3_facebook_likes
Density
0.0000
0.0012
0
5000
15000
N = 4975 Bandwidth = 61.44


actor_1_facebook_likes
Density
0.00000
0.00015
0e+00
2e+05
4e+05
6e+05
N = 4991 Bandwidth = 1271


gross
Density
0.0e+00
0e+00
2e+08
4e+08
6e+08
8e+08
N = 4124 Bandwidth = 7.245e+06


num_voted_users
Density
0.0e+00
0
500000
1500000
N = 4998 Bandwidth = 1.071e+04


cast_total_facebook_likes
Density
0e+00
1e−04
0e+00
2e+05
4e+05
6e+05
N = 4998 Bandwidth = 1508




```
par(mfrow = c(1, 1))

# Take log of all the above continuous variables except facebook likes
movie$num_critic_for_reviews <- log(movie$num_critic_for_reviews)
movie$num_voted_users <- log(movie$num_voted_users)
movie$num_user_for_reviews <- log(movie$num_user_for_reviews)
movie$gross <- log(movie$gross)
movie$budget <- log(movie$budget)
movie$duration <- log(movie$duration)
```

The univariate density plots of the continuous variables show that all of them are highly right skewed. Therefore, these variables should be transformed by taking log of them.

0s in facebook likes

```
## 0s in some variables (facebook likes) look like missing values
apply(movie.continous, 2, function(x) sum(as.numeric(x)==0, na.rm = TRUE))
```

```
##    num_critic_for_reviews                  duration
##                         0                         0
##   director_facebook_likes    actor_3_facebook_likes
##                       897                        89
##    actor_1_facebook_likes                     gross
##                        26                         0
##          num_voted_users cast_total_facebook_likes
##                         0                        33
```

```
##      num_user_for_reviews                    budget
##                         0                         0
##    actor_2_facebook_likes      movie_facebook_likes
##                        55                      2162
```

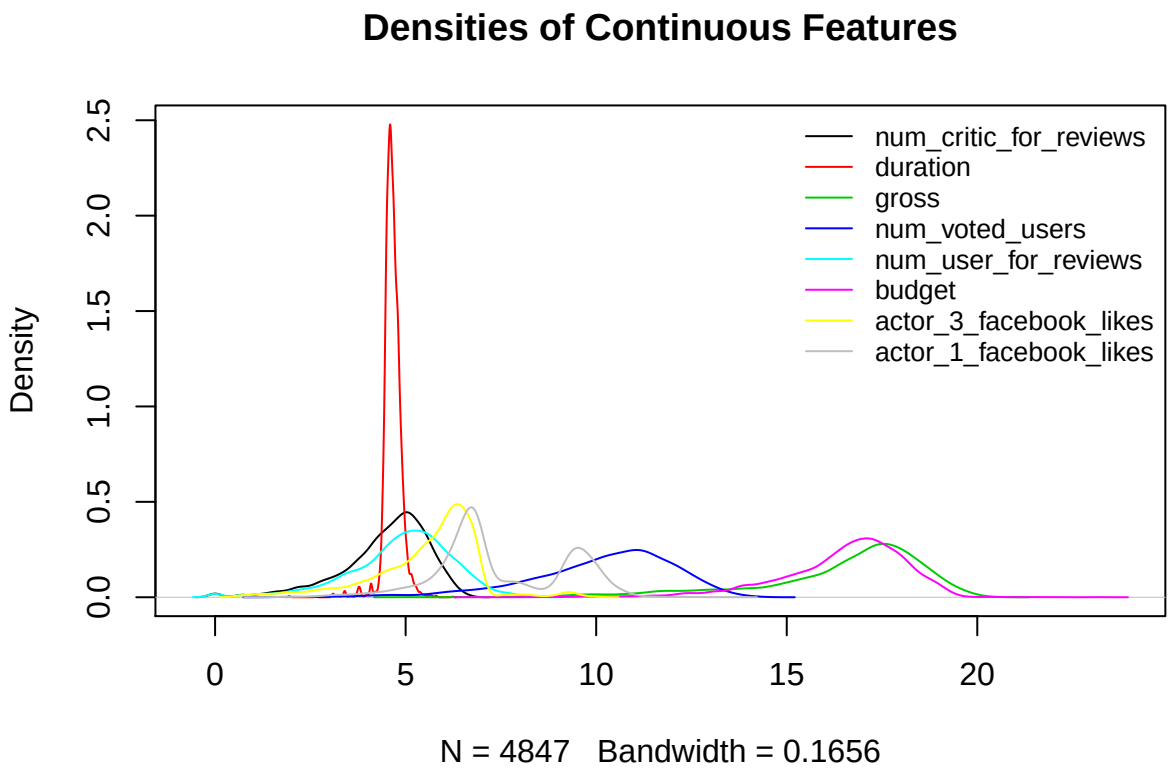
```
  # movie_facebook_likes and director_facebook_likes have lots of 0s (2162, 897)

# delete director_facebook_likes and movie_facebook_likes
movie <- dplyr::select(movie, -c(director_facebook_likes, movie_facebook_likes))

# Remove 0s in other facebook likes, and take log of them
fb <- dplyr::select(movie, ends_with("facebook_likes"))
fb0.row <- apply(fb, 1, function(x) any(x==0))

fb <- log(fb[!fb0.row, ])
movie <- movie[!fb0.row, ]

movie <- movie %>% dplyr::select(-ends_with("facebook_likes"))
movie <- cbind(movie, fb)
```

All of the continuous variables except those measuring facebook likes have no 0 values, so taking log of them will not generate negative infinite values. As for the 6 variables of different kinds of facebook likes, the 0 values seem to be missing values actually. For example, some famous directors such as James Cameron have 0 facelikes in the data set, which is not the case in the truth. These 0s should be treated as missing values and removed before taking log of the variables.

What's more, the variables of director facebook likes and movie facebook likes have a large number of 0 values, which is respectively 2162 and 897. The number still remains large after removing NAs for all the other variables. As a result, keeping them in the feature space after removing their 0 values would make the remaining data set become just a small subset of the whole data set and damage the predictive power. Therefore, these two variables are removed from the feature space.

Density plots after taking log

```
# Density plots after data cleaning and transformation
movie.continous <- dplyr::select(movie, -c(color, language, country, content_rating, imdb_score, facenu
d.list <- list()
d.x <- vector()
d.y <- vector()
for (i in 1:8) {
  d.list[[i]] <- density(movie.continous[, i], na.rm = TRUE)
  d.x <- c(d.x, d.list[[i]]$x)
  d.y <- c(d.y, d.list[[i]]$y)
}
plot(d.list[[1]], xlim = range(d.x), ylim = range(d.y), main = "Densities of Continuous Features")
for (i in 2:8) {
  lines(d.list[[i]], col = i)
}
legend('topright', legend = colnames(movie.continous)[1:8], col = 1:8, lty = 1, cex = 0.8, bty = "n")
```



The plot above shows the densities of the continuous variables after data cleaning and transformation. The distributions of the variables become more symmetric after taking log. According to the plot, the variances of these variables differ a lot from each other, and they have very different scales and units of measurement. Therefore, it would be better to standardize these variables.

4. Exploration of categorical variables

```
# tables of categorical variables
table(movie$color) # most are colored
```

```
##
##                   Black and White             Color
##                16               201              4669
```

```
table(movie$color)["Color"]/sum(table(movie$color)) # 95.6% are colored
```

```
##     Color
## 0.9555874
```

```
table(movie$language) # most are in English
```

```
##
##            Aboriginal      Arabic     Aramaic     Bosnian  Cantonese
##             9           2           3           1           1          11
##     Chinese      Czech     Danish       Dari      Dutch   Dzongkha
##             3           1           4           1           4           0
##     English    Filipino     French     German      Greek     Hebrew
##        4584           1          67          19           1           2
```

```
##      Hindi  Hungarian  Icelandic Indonesian    Italian   Japanese
##         27          0          2          1         11         16
##    Kannada     Kazakh     Korean   Mandarin       Maya  Mongolian
##          0          1          7         23          1          1
##       None  Norwegian    Panjabi    Persian     Polish Portuguese
##          1          4          1          3          4          8
##   Romanian    Russian  Slovenian    Spanish    Swahili    Swedish
##          1          9          0         36          1          5
##      Tamil     Telugu       Thai       Urdu Vietnamese       Zulu
##          1          1          3          1          1          2
```

```
table(movie$language)["English"]/sum(table(movie$language)) # 93.8% are in English
```

```
##   English
## 0.9381907
```

```
table(movie$country) # most are in USA
```

```
##
##                          Afghanistan            Argentina
##                    3                0                    3
##                Aruba          Australia              Bahamas
##                    1               53                    1
##              Belgium           Brazil             Bulgaria
##                    4                8                    1
##             Cambodia           Cameroon               Canada
##                    1                0                  122
##                Chile            China             Colombia
##                    1               27                    1
##       Czech Republic          Denmark   Dominican Republic
##                    3                9                    1
##                Egypt          Finland               France
##                    1                1                  145
##              Georgia          Germany               Greece
##                    1               95                    2
##            Hong Kong          Hungary              Iceland
##                   17                1                    3
##                India        Indonesia                 Iran
##                   31                1                    3
##              Ireland           Israel                Italy
##                   11                1                   22
##                Japan            Kenya            Kyrgyzstan
##                   21                1                    0
##                Libya           Mexico          Netherlands
##                    1               17                    5
##             New Line      New Zealand              Nigeria
##                    1               15                    1
##               Norway   Official site             Pakistan
##                    7                1                    1
##               Panama             Peru          Philippines
##                    1                1                    0
##               Poland          Romania               Russia
##                    5                3                   10
##             Slovakia         Slovenia         South Africa
##                    1                0                    8
```

```
##          South Korea         Soviet Union                Spain
##                   13                    1                   31
##               Sweden          Switzerland               Taiwan
##                    5                    3                    2
##             Thailand               Turkey                   UK
##                    5                    1                  436
## United Arab Emirates                  USA         West Germany
##                    0                 3712                    3
```

```
table(movie$country)["USA"]/sum(table(movie$country)) # 76.0% are in USA
```

```
##       USA
## 0.7597217
```

```
table(movie$content_rating) # some of the ratings have the same meaning
```

```
##
##            Approved         G        GP         M     NC-17 Not Rated
##       259        55       106         6         5         7       101
##    Passed        PG     PG-13         R     TV-14      TV-G     TV-MA
##         9       692      1428      2083        30        10        17
##     TV-PG      TV-Y     TV-Y7   Unrated         X
##        13         1         1        51        12
```

```
# Convert categorical variables to binary ones coded -1/+1
## country
movie$country <- ifelse(movie$country=="USA", 1, -1)

## rating
rating.list <- list(rating.g = c("G", "TV-G"), rating.pg = c("PG", "GP", "M", "TV-PG"), rating.pg13 = c
ratings <- sapply(rating.list, function(x) ifelse(movie$content_rating %in% x, 1, -1))
movie <- cbind(movie, ratings)
### the number of movies with different ratings
kable(colSums(ratings==1), col.names = "the number of movies", caption = "the number of movies with dif
```

Table 1: the number of movies with different ratings

	the number of movies
rating.g	116
rating.pg	716
rating.pg13	1458
rating.r	2083
rating.nc17	36
rating.nr	216
rating.y7	1
rating.y	1

```
### Remove some rating binary variales with little variation
movie <- dplyr::select(movie, -c(color, language, content_rating, rating.g, rating.nc17, rating.nr, rati
```

The categorical variables are converted into binary ones. They are recoded as -1/+1 instead of 0/1 because adding some "space" between them would make them more similar to the standardized continuous variables which have values on both sides of 0.

There are 4 categorical variables in the data set, including the color of the movie, the language, the country

and the rating. The tables of the number of different categories in each variable shows that more than 90% of the movies are colored and in English. Therefore, the variables indicating color and language have little variation and should be removed. As for the country of the movie, most movies are from the US, which make up 76% of the sample. This variables is recoded as a binary one, with 1 indicating the US and -1 indicating other countries.

The table of the ratings shows that some of the ratings in the data set actually have the same meaning. Therefore, ratings with the same definitions are categorized together, and each rating category is indicated by a newly created binary variable. The table of the number of movies with different ratings show that the binary variables of rating.g, rating.nc17, rating.nr, rating.y7 and rating.y have little variation, so they are dropped from the feature space.

5. Missing values

```
# Missing values
## the number of NAs in each variable
num.na <- colSums(is.na(movie) | movie=="") #gross has largest number of NAs (830)
kable(num.na, col.names = "the number of misssing values")
```

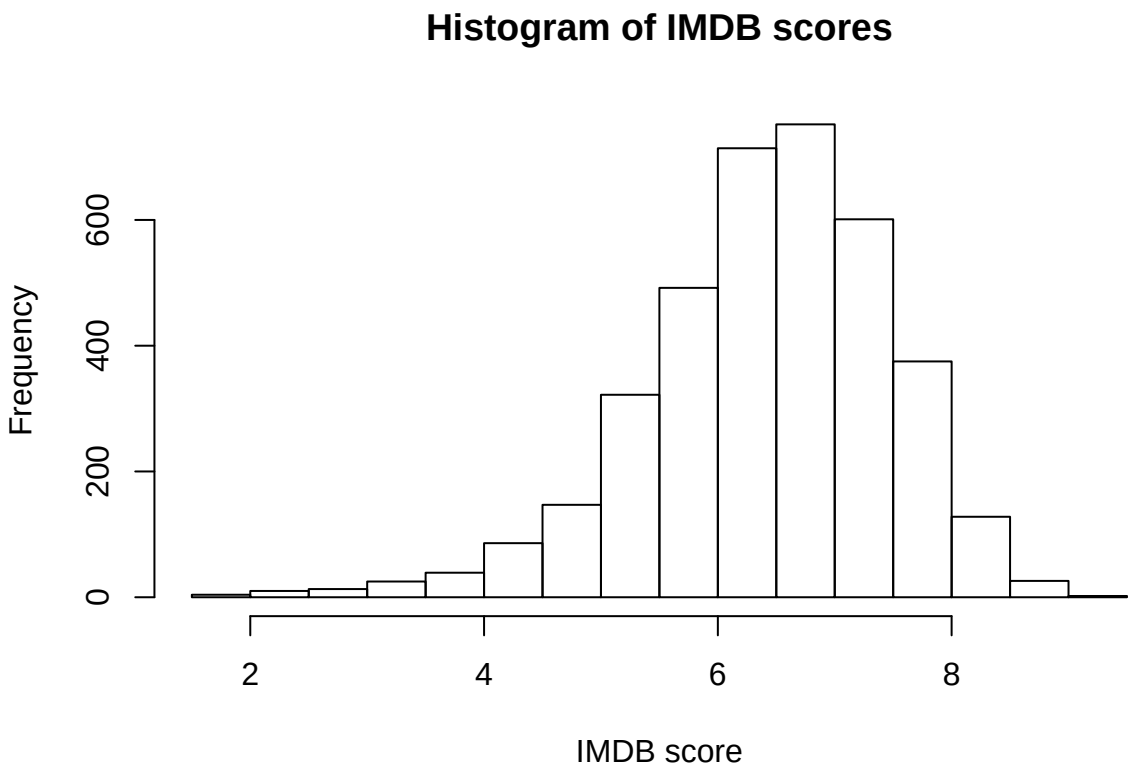
	the number of misssing values
num_critic_for_reviews	45
duration	18
gross	830
num_voted_users	6
facenumber_in_poster	19
num_user_for_reviews	23
country	6
budget	470
title_year	105
imdb_score	6
aspect_ratio	297
actor_3_facebook_likes	6
actor_1_facebook_likes	6
cast_total_facebook_likes	6
actor_2_facebook_likes	6
rating.pg	0
rating.pg13	0
rating.r	0

```
## remove all the rows with NAs
na.row <- apply(movie, 1, function(x) any(is.na(x) | x=="")) #1261 rows with NAs
movie <- movie[!na.row, ] # 3736 obs
```

The table of the number of missing values in each variable shows that the variable of gross has the largest number of missing values, which is 830. This accounts for 17% of the whole data set. Another variable which also have lots of missing values is the budget, with missing values accounting for 10% of the data. This is not a terrible case because the other variables do not have many missing values. So these variables are kept in the data set. After removing the missing values in all the variables, we have 3736 observations left.

6. Response variable

```
# Histogram of IMDB scores
hist(movie$imdb_score, xlab = "IMDB score", main = "Histogram of IMDB scores")
```




```
# Convert response variable (IMDB score) into 4 categories
movie$quality <- cut(movie$imdb_score, c(0, 4, 6, 8, 10), labels = c("poor", "fair", "good", "excellent
movie <- dplyr::select(movie, -imdb_score)
table(movie$quality)
```

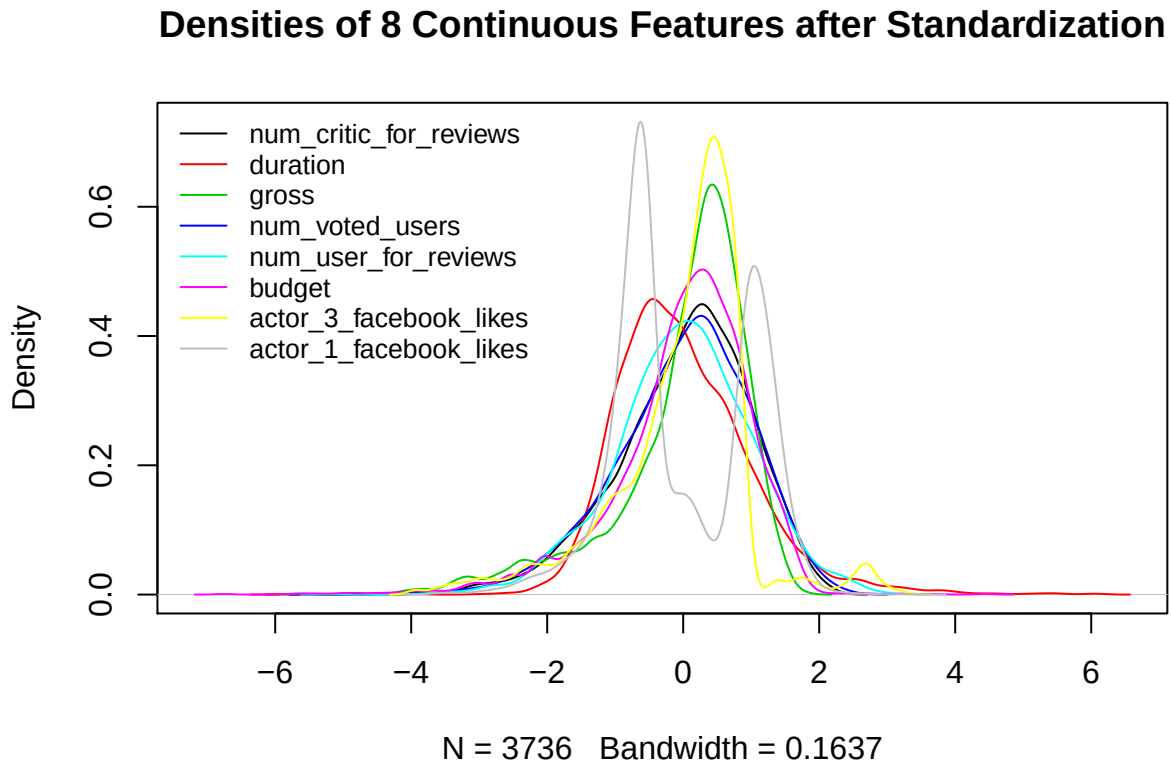
```
##
##      poor      fair      good excellent
##        91      1047      2442       156
```

The above histogram of the IMDB scores shows that the scores of most movies lie in the range of 6 to 8. There is only a small number of movies that have an IMDB score lower than 4. The response variable of classification, the quality, is created based on the IMDB score. Movies with IMDB scores between 0-4 is categorized as "poor"; those with 4-6 scores is categorized as "fair"; those with 6-8 scores are categorized as "good"; and those with 8-10 scores are categorizes with "excellent". The intervels are closed on the right and open on the left.

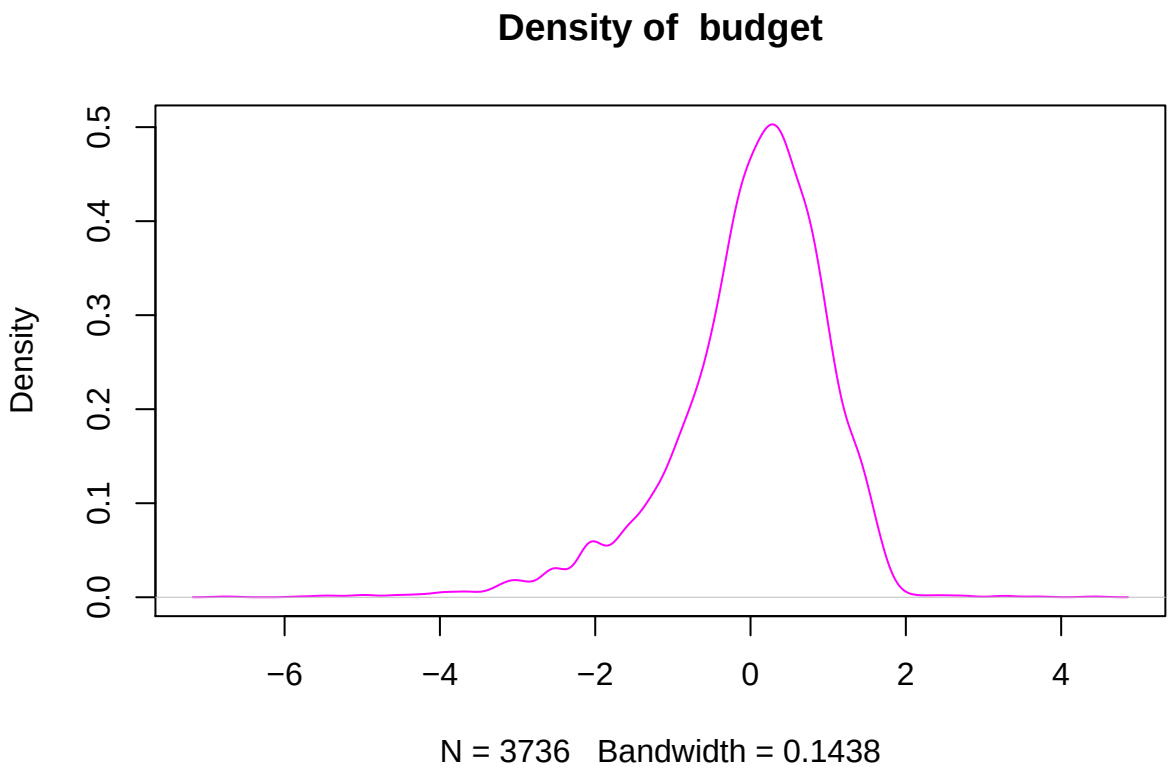
7. Standardization

```
movie.std <- movie
movie.std[, 1:(ncol(movie)-1)] <- scale(movie[, 1:(ncol(movie)-1)])
movie.std <- movie.std %>% dplyr::select(-c(quality, country, rating.pg, rating.pg13, rating.r, quality

# Density plots of standardized continuous data
movie.std.continous <- dplyr::select(movie.std, -c(quality, facenumber_in_poster, aspect_ratio, title_y

d.list <- list()
d.x <- vector()
```

```
d.y <- vector()
for (i in 1:8) {
  d.list[[i]] <- density(movie.std.continous[, i])
  d.x <- c(d.x, d.list[[i]]$x)
  d.y <- c(d.y, d.list[[i]]$y)
}
plot(d.list[[1]], xlim = range(d.x), ylim = range(d.y), main = "Densities of 8 Continuous Features afte
for (i in 2:8) {
  lines(d.list[[i]], col = i)
}
legend('topleft', legend = colnames(movie.std.continous)[1:8], col = 1:8, lty = 1, cex = 0.8, bty = "n"
```




```
# Density plot of budget
plot(d.list[[6]], col = 6, main = paste("Density of ", colnames(movie.std.continous)[6]))
```



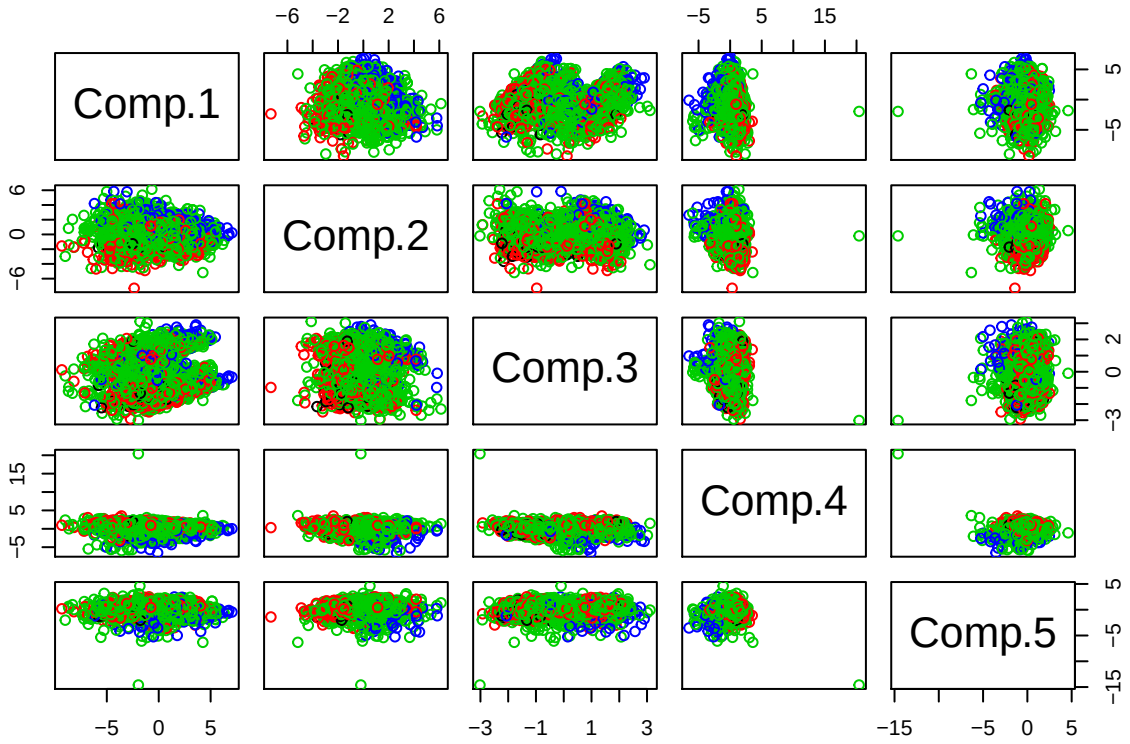
All the continuous variables are standardized after data cleaning and log transformation. Standardization is to avoid some variables with large variances dominating the distance calculation and principal components. The binary variables are coded as -1/+1 in the feature set and not standardized.

The density plot of the standardized continuous variables show that the variables are of similar scales after standardization. But there are still long tails in the densities. What's more, some of the densities are not smooth enough. For example, the density plot of budget have some "extreme values" less than -6 or greater than 4. Accordingly, there might be a lot of outliers which lie out of the range of 3 standard deviations from the 0 mean.

8. Principal Components

Bivariate plot of PCs

```
# PC of Standardized data
pc.std <- princomp(dplyr::select(movie.std, -quality))
pairs(pc.std$scores[, 1:5], col = movie$quality)
```



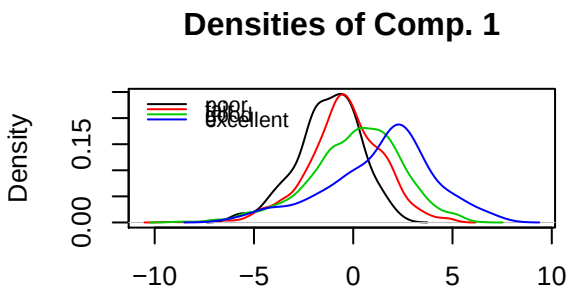

```
movie.std.pc <- as.data.frame(pc.std$scores)
movie.std.pc$quality <- movie.std$quality
```

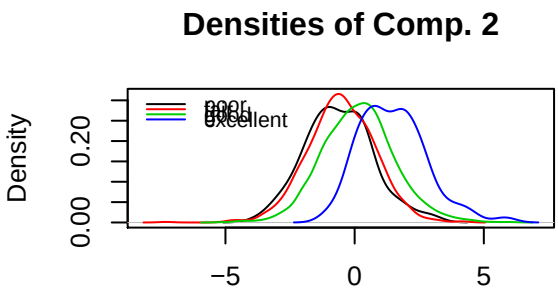
Above is the bivariate plot of the first five principal components of the standardized data, colored by different classes of the movies. The plot shows that the points with different colors are hard to separate, indicating that it may be hard to get high accuracy in the classification.

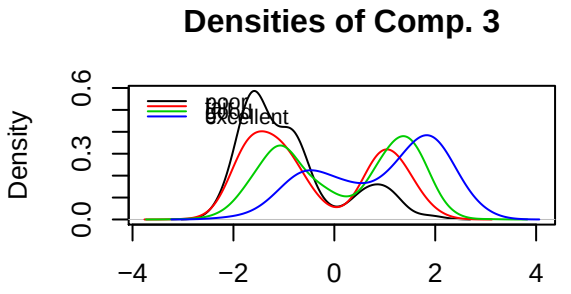
Density plot of PCs

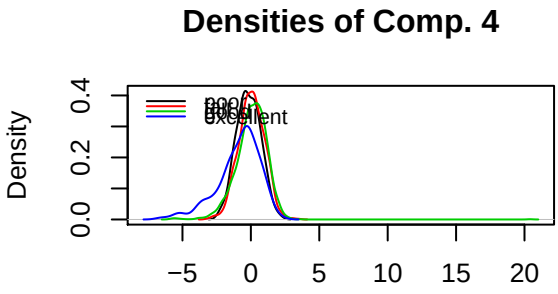
```
# Density plot of principal components 1-8 within different classes
par(mfrow = c(2, 2))

pc.d.fun <- function(n) {
  quality <- c("poor", "fair", "good", "excellent")
  d.list <- list()
  d.x <- vector()
  d.y <- vector()
  for (i in 1:4) {
    d.list[[i]] <- density(pc.std$scores[,n][movie$quality==quality[i]])
    d.x <- c(d.x, d.list[[i]]$x)
    d.y <- c(d.y, d.list[[i]]$y)
  }
  plot(d.list[[1]], xlim = range(d.x), ylim = range(d.y), main = paste("Densities of Comp.", n))
  for (i in 2:4) {
    lines(d.list[[i]], col = i)
  }
```

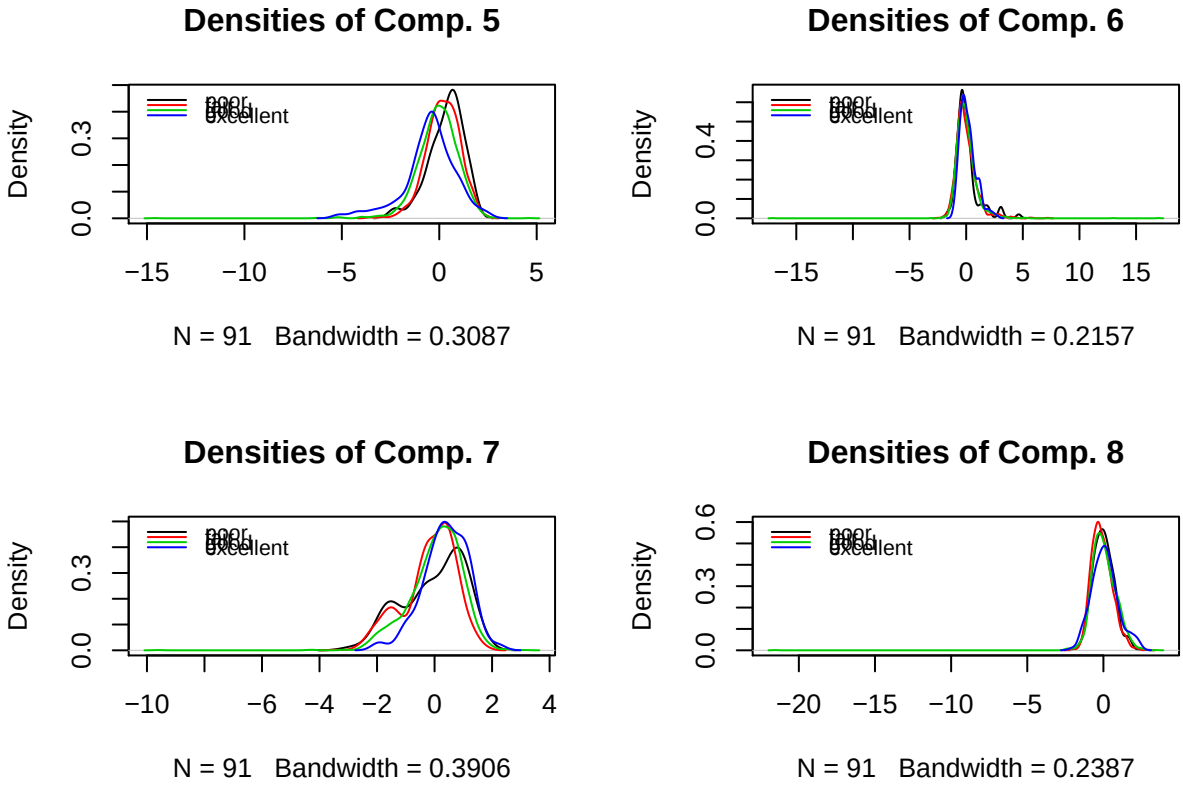
```
  legend('topleft', legend = quality, col = 1:4, lty = 1, cex = 0.8, bty = "n", y.intersp = 0.2)
}

sapply(1:8, pc.d.fun)
```












```
##      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]
## rect List,4 List,4 List,4 List,4 List,4 List,4 List,4 List,4
## text List,2 List,2 List,2 List,2 List,2 List,2 List,2 List,2
```

```
par(mfrow = c(1, 1))
```

Above is the density plots of the first 8 principal components conditional on the four different classes. It is shown that the densities of the four groups have lots of overlaps, although it seems in the plots of component 1 and component 2 that movies with higher quality may have slightly higher values in these two components. The density plots indicates that the movies are hard to classify, which is also shown in the bivariate plots.

The plots of components 4-8 also show that the densities of these components for the group of "good" have a very long tail, but the density values are nearly 0 on the tail. This may be an evidence of outliers in the group of "good" movies.

Part C: Classification Methods

Three classification methods, which are naive bayes, support vector machine and neural network, are implemented to classify the movies. In order to mitigate the problem of overfitting, 70% of the data is randomly sampled for training the model, and the remaining 30% is only used for testing. Within the training data, 10-fold cross validation is applied for choosing the "best" parameters for each model to get the largest out-of-sample accuracy.

1. Naive Bayes

The first method is Naive Bayes classifier. The cross validation within the training data set is repeated 20 times.

The assumption of this algorithm requires that the features are independent given the class. In order to satify this assumption by removing the correlation between the features, the principal components instead of the original data are used in the model. The results are shown as follows.

```
set.seed(2011)
idx <- createDataPartition(y=movie.std$quality, p=0.7, list=FALSE)
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 20)

movie.std.pc.train <- movie.std.pc[idx, ]
movie.std.pc.test <- movie.std.pc[-idx, ]

# Similar proportion of each class within training and testing set
class.prop <- round(rbind(table(movie.std.pc.train$quality)/nrow(movie.std.pc.train), table(movie.std.pc
rownames(class.prop) <- c("train", "test")
class.prop
```

```
##         poor   fair   good excellent
## train 0.0245 0.2801 0.6534    0.0420
## test  0.0241 0.2806 0.6542    0.0411
```

```
# all PCs
nb.fit.cv <- train(quality~., data = movie.std.pc.train, trControl = train.control, method = "nb")
print(nb.fit.cv, showSD = TRUE) #accuracy=0.6765672
```

```
## Naive Bayes
##
## 2617 samples
##   17 predictor
##    4 classes: 'poor', 'fair', 'good', 'excellent'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 2354, 2355, 2356, 2356, 2355, 2356, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   usekernel  Accuracy               Kappa
##   FALSE      0.6423115 (0.0314930)  0.3153264 (0.05396986)
##    TRUE      0.6765672 (0.0285742)  0.3208556 (0.05977770)
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

```
pred.nb.cv <- predict(nb.fit.cv, newdata = movie.std.pc.test)
print(cm.nb <- confusionMatrix(pred.nb.cv, movie.std.pc.test$quality)) #accuracy=0.6702
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  poor fair good excellent
##   poor         6    6    6         0
##   fair        15  146  131         0
##   good         6  162  577        25
```

```
##   excellent    0    0   18        21
##
## Overall Statistics
##
##                Accuracy : 0.6702
##                  95% CI : (0.6418, 0.6978)
##     No Information Rate : 0.6542
##     P-Value [Acc > NIR] : 0.1355
##
##                   Kappa : 0.3055
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: poor Class: fair Class: good Class: excellent
## Sensitivity             0.222222      0.4650      0.7883          0.45652
## Specificity             0.989011      0.8186      0.5013          0.98322
## Pos Pred Value          0.333333      0.5000      0.7494          0.53846
## Neg Pred Value          0.980926      0.7969      0.5559          0.97685
## Prevalence              0.024129      0.2806      0.6542          0.04111
## Detection Rate          0.005362      0.1305      0.5156          0.01877
## Detection Prevalence    0.016086      0.2609      0.6881          0.03485
## Balanced Accuracy       0.605617      0.6418      0.6448          0.71987
```

```
acc.nb.train <- 0.6765672 #save the accuracy
acc.nb.test <- 0.6702
```

The cross validation process picks the parameters of fL = 0, useKernel = TRUE, and adjust = 1. The Naive Bayes classifier does not achive high accuracy in both the traning and the testing sample. The average within training accuracy on the CV sample is 67.7%. Within the test set, the accuracy is 67%. The low accuracy seems to be consistent with the difficulty of separating the points shown in the bivariate plots.

When applying this method in R, more than 50 warnings of 0 probabilities for all classes occurred. (These warnings are set not to be shown in the final PDF report because they would take up too much space.) This may be because Naive Bayes model builds empirical densities conditional on groups. When there are lots of outliers in the data set, and the density plots are not smooth enough, 0 probabilities of all classes may occur for some observations. As the density plots show, the densities of some features and the principal components 4-8 have long tails with values close to 0. The bivariate plots of the components also show that there are outliers in components 4 and 5. This might lead to 0 probabilities for some observations to be classified into any of the four classes.

To further confirm this, the following Naive Bayes model uses only component 1-3. In this case, there are only 9 warnings when running the model codes, which is much fewer than those in the previous model with all of the components. There is still little accuracy probably because the first three components contribute only 54.5% of variance of the whole feature set. When looking closely at the confusion matrix, the prediction looks poor too, which classifies most of the movies into one category, the "good" one. The true positive rate is only high for the class "good", and is very low for the other three classes.

```
set.seed(2011)
idx <- createDataPartition(y=movie.std$quality, p=0.7, list=FALSE)
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 20)

# PC 1-3
nb.fit.cv1 <- train(quality~Comp.1+Comp.2+Comp.3, data = movie.std.pc.train, trControl = train.control,
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 254
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 250

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 252

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 254

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 251

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 247

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 251

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 257

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 252
```

```
print(nb.fit.cv1, showSD = TRUE) #accuracy=0.6757983
```

```
## Naive Bayes
##
## 2617 samples
##    3 predictor
##    4 classes: 'poor', 'fair', 'good', 'excellent'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 2354, 2355, 2356, 2356, 2355, 2356, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   usekernel  Accuracy                Kappa
##   FALSE      0.6757983 (0.01620583)  0.1521145 (0.04462876)
##    TRUE      0.6802689 (0.02179339)  0.1920906 (0.05659167)
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
```

```
pred.nb.cv1 <- predict(nb.fit.cv1, newdata = movie.std.pc.test)
print(cm.nb1 <- confusionMatrix(pred.nb.cv1, movie.std.pc.test$quality)) #accuracy=0.6711
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  poor fair good excellent
##   poor         0    0    0         0
##   fair         9   70   55         0
##   good        18  244  677        42
##   excellent    0    0    0         4
```

```
##
## Overall Statistics
##
##                Accuracy : 0.6711
##                  95% CI : (0.6427, 0.6986)
##     No Information Rate : 0.6542
##     P-Value [Acc > NIR] : 0.1222
##
##                   Kappa : 0.1627
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: poor Class: fair Class: good Class: excellent
## Sensitivity              0.00000     0.22293      0.9249         0.086957
## Specificity              1.00000     0.92050      0.2145         1.000000
## Pos Pred Value               NaN     0.52239      0.6901         1.000000
## Neg Pred Value           0.97587     0.75228      0.6014         0.962332
## Prevalence               0.02413     0.28061      0.6542         0.041108
## Detection Rate           0.00000     0.06256      0.6050         0.003575
## Detection Prevalence     0.00000     0.11975      0.8767         0.003575
## Balanced Accuracy        0.50000     0.57171      0.5697         0.543478
```

```
# Summary of the components
summary(pc.std)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     2.2397791 1.4395182 1.2936349 1.15718345 1.03820244
## Proportion of Variance 0.3123037 0.1290034 0.1041814 0.08336258 0.06710128
## Cumulative Proportion  0.3123037 0.4413070 0.5454884 0.62885101 0.69595229
##                           Comp.6     Comp.7     Comp.8     Comp.9
## Standard deviation     0.97773845 0.92535604 0.82764512 0.79827275
## Proportion of Variance 0.05951304 0.05330703 0.04264372 0.03967065
## Cumulative Proportion  0.75546533 0.80877236 0.85141607 0.89108672
##                           Comp.10    Comp.11    Comp.12    Comp.13
## Standard deviation     0.70145558 0.67874868 0.52665608 0.42021794
## Proportion of Variance 0.03063142 0.02868037 0.01726716 0.01099299
## Cumulative Proportion  0.92171814 0.95039851 0.96766567 0.97865866
##                           Comp.14     Comp.15     Comp.16      Comp.17
## Standard deviation     0.40278209 0.337277539 0.237129136 0.1029146811
## Proportion of Variance 0.01009967 0.007081766 0.003500552 0.0006593582
## Cumulative Proportion  0.98875832 0.995840090 0.999340642 1.0000000000
```

2. Support Vector Machine

Another method used for classification is support vector machine. The cross validation process is also repeated 20 times to search for the best parameters. Radial kernel is used in this SVM model. The results are shown as follows.

```
set.seed(2011)
idx <- createDataPartition(y=movie.std$quality, p=0.7, list=FALSE)
train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 20)

movie.std.train <- movie.std[idx, ]
```

```r
movie.std.test <- movie.std[-idx, ]

# linear
svm.fit.cv <- train(quality~., data = movie.std.train, trControl = train.control, method = "svmRadial")
print(svm.fit.cv, showSD = TRUE) #accuracy=0.7494268
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2617 samples
##   17 predictor
##    4 classes: 'poor', 'fair', 'good', 'excellent'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 2354, 2355, 2356, 2356, 2355, 2356, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   C     Accuracy                Kappa
##   0.25  0.7183625 (0.01963013)  0.2977441 (0.05482530)
##   0.50  0.7326739 (0.01993477)  0.3528777 (0.05133292)
##   1.00  0.7494268 (0.02138360)  0.4194016 (0.05274787)
##
## Tuning parameter 'sigma' was held constant at a value of 0.04548044
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.04548044 and C = 1.
```

```r
pred.svm.cv <- predict(svm.fit.cv, newdata = movie.std.test)
print(cm.svm <- confusionMatrix(pred.svm.cv, movie.std.test$quality)) #accuracy=0.7355
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  poor fair good excellent
##   poor         0    0    0         0
##   fair        22  154   82         0
##   good         5  160  649        26
##   excellent    0    0    1        20
##
## Overall Statistics
##
##                Accuracy : 0.7355
##                  95% CI : (0.7086, 0.7611)
##     No Information Rate : 0.6542
##     P-Value [Acc > NIR] : 3.135e-09
##
##                   Kappa : 0.4035
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: poor Class: fair Class: good Class: excellent
## Sensitivity              0.00000      0.4904      0.8866          0.43478
## Specificity              1.00000      0.8708      0.5065          0.99907
## Pos Pred Value               NaN      0.5969      0.7726          0.95238
```

```
## Neg Pred Value           0.97587      0.8142      0.7025          0.97632
## Prevalence               0.02413      0.2806      0.6542          0.04111
## Detection Rate           0.00000      0.1376      0.5800          0.01787
## Detection Prevalence     0.00000      0.2306      0.7507          0.01877
## Balanced Accuracy        0.50000      0.6806      0.6965          0.71693
```

```
acc.svm.train <- 0.7494268 #save the accuracy
acc.svm.test <- 0.7355
```

The cross validation process chooses the best parameters as gamma = 0.04548044 and C = 1. The average within training accuracy on the CV sample is 74.9%. Within the test set, the accuracy is similar, which is 73.6%.

The confusion matrix shows that the prediction of the class "poor" is very inaccurate. None of the movies in this group are classified correctly. This may be due to the small number of movies with low IMDB scores in the data set. Only approximately 2% of the movies are in the group of "poor".

3. Neural Network

For the neural network model, the parameter set is too large to use cross validation. However, cross validation can be used for choosing a few of the tuning parameters. One of the parameters is "decay", which performs a penalty to aviod overfitting. In this case, the cross validation is applied just once because repeated cross validation would be very computationally expensive, and the tuning parameters are set to be in a larger grid than the default. The parameters are searched through a grid with sizes 1-9 and weight decay between 0 and 0.1 by 0.025, as well as testing 0.01 (a total of 6 values for decay).

```
set.seed(2011)
idx <- createDataPartition(y=movie.std$quality, p=0.7, list=FALSE)
movie.std.train <- movie.std[idx, ]
movie.std.test <- movie.std[-idx, ]
trControl <- trainControl(method = "cv", number = 10, search = "grid")
tuneGrid <- expand.grid(.size = c(1:9), .decay = c(0, 0.025, 0.05, 0.075, 0.1,
0.01))

text.nn.movie.std <- capture.output(nn.fit.cv <- caret::train(quality ~ ., data = movie.std.train, metho
pred.nn.cv <- predict(nn.fit.cv, movie.std.test)
print(nn.fit.cv, showSD = TRUE) #accuracy=0.7596560
```

```
## Neural Network
##
## 2617 samples
##   17 predictor
##    4 classes: 'poor', 'fair', 'good', 'excellent'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2354, 2355, 2356, 2356, 2355, 2356, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   size  decay  Accuracy                Kappa
##   1     0.000  0.7168610 (0.02959008)  0.3871328 (0.05783338)
##   1     0.010  0.7172412 (0.03056356)  0.3880563 (0.06137323)
##   1     0.025  0.7180031 (0.03298147)  0.3888702 (0.06831372)
##   1     0.050  0.7210581 (0.03536976)  0.3929204 (0.07445922)
##   1     0.075  0.7202874 (0.03495835)  0.3895252 (0.07468952)
##   1     0.100  0.7195386 (0.03523071)  0.3871612 (0.07227507)
```

```
##   2     0.000  0.7359816 (0.03359909)  0.4334426 (0.06533198)
##   2     0.010  0.7325303 (0.03100185)  0.4221460 (0.06294360)
##   2     0.025  0.7409231 (0.02672534)  0.4378683 (0.06121324)
##   2     0.050  0.7447501 (0.02110412)  0.4458743 (0.04687284)
##   2     0.075  0.7466716 (0.01944302)  0.4494865 (0.04657667)
##   2     0.100  0.7451361 (0.02475586)  0.4448754 (0.05689254)
##   3     0.000  0.7424890 (0.03606416)  0.4458872 (0.06927758)
##   3     0.010  0.7420724 (0.02801049)  0.4401867 (0.05988752)
##   3     0.025  0.7436196 (0.02300423)  0.4441500 (0.04877290)
##   3     0.050  0.7470591 (0.02834958)  0.4497166 (0.06143116)
##   3     0.075  0.7447720 (0.02704664)  0.4479393 (0.06144952)
##   3     0.100  0.7496959 (0.02687839)  0.4577650 (0.05995426)
##   4     0.000  0.7397576 (0.02331468)  0.4423787 (0.05592526)
##   4     0.010  0.7482013 (0.03141681)  0.4522999 (0.06634141)
##   4     0.025  0.7401567 (0.02313203)  0.4404691 (0.05312120)
##   4     0.050  0.7497236 (0.02508777)  0.4596460 (0.05171426)
##   4     0.075  0.7455382 (0.02200565)  0.4508353 (0.04409863)
##   4     0.100  0.7497440 (0.02328151)  0.4572947 (0.05553383)
##   5     0.000  0.7401553 (0.02176314)  0.4472344 (0.04916363)
##   5     0.010  0.7435992 (0.02837978)  0.4522084 (0.06275981)
##   5     0.025  0.7420856 (0.02209213)  0.4495961 (0.04218185)
##   5     0.050  0.7428562 (0.01830113)  0.4479134 (0.04063417)
##   5     0.075  0.7428708 (0.02756344)  0.4480871 (0.05692829)
##   5     0.100  0.7562064 (0.02841217)  0.4742436 (0.05901936)
##   6     0.000  0.7367376 (0.02881577)  0.4387657 (0.05237291)
##   6     0.010  0.7382818 (0.02621771)  0.4462326 (0.04686328)
##   6     0.025  0.7374821 (0.02731633)  0.4381867 (0.05425515)
##   6     0.050  0.7481750 (0.02147669)  0.4603464 (0.05000019)
##   6     0.075  0.7497105 (0.02817216)  0.4610565 (0.06038579)
##   6     0.100  0.7420826 (0.01901657)  0.4439256 (0.04293280)
##   7     0.000  0.7229504 (0.01380366)  0.4101992 (0.03078456)
##   7     0.010  0.7401495 (0.02957098)  0.4441316 (0.06592097)
##   7     0.025  0.7512460 (0.01616729)  0.4669913 (0.03623270)
##   7     0.050  0.7367143 (0.01647940)  0.4387434 (0.03714827)
##   7     0.075  0.7512373 (0.01756367)  0.4671489 (0.03957880)
##   7     0.100  0.7596560 (0.02139402)  0.4859490 (0.04482107)
##   8     0.000  0.7241100 (0.01728219)  0.4173427 (0.03172136)
##   8     0.010  0.7283202 (0.01400501)  0.4307357 (0.03059387)
##   8     0.025  0.7355649 (0.02360704)  0.4379947 (0.04508188)
##   8     0.050  0.7409026 (0.02789035)  0.4487161 (0.06072739)
##   8     0.075  0.7432291 (0.02396210)  0.4491330 (0.05479950)
##   8     0.100  0.7485203 (0.02133238)  0.4640995 (0.05035095)
##   9     0.000  0.7279283 (0.01616476)  0.4269812 (0.03797969)
##   9     0.010  0.7519874 (0.02692574)  0.4702683 (0.06415202)
##   9     0.025  0.7458717 (0.02574474)  0.4622724 (0.04461984)
##   9     0.050  0.7451113 (0.01834406)  0.4591836 (0.04021470)
##   9     0.075  0.7401670 (0.01709677)  0.4454150 (0.03478321)
##   9     0.100  0.7515985 (0.02280211)  0.4757956 (0.04675628)
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 7 and decay = 0.1.
```

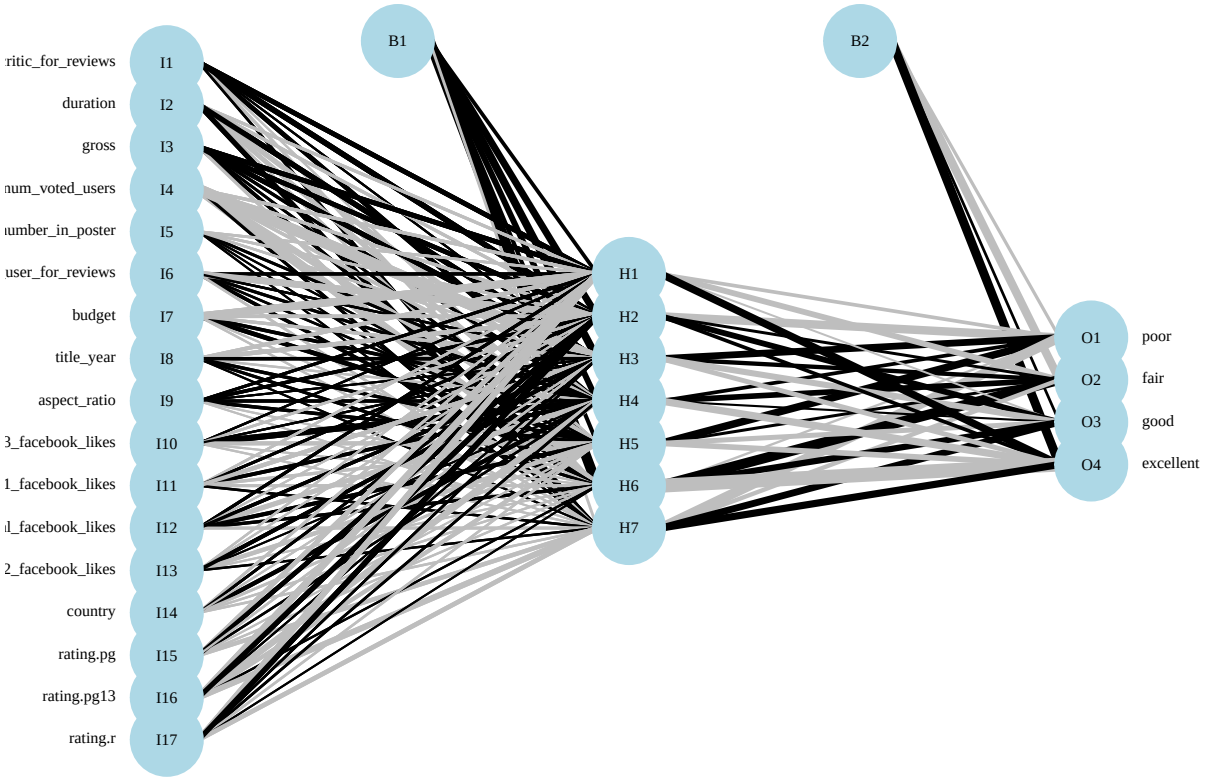
```
print(cm.nn <- confusionMatrix(pred.nn.cv, movie.std.test$quality)) #0.7462
```

```
## Confusion Matrix and Statistics
```

```
##
##            Reference
## Prediction  poor fair good excellent
##   poor         3    4    4         0
##   fair        20  172   94         0
##   good         4  138  625        11
##   excellent    0    0    9        35
##
## Overall Statistics
##
##                Accuracy : 0.7462
##                  95% CI : (0.7196, 0.7715)
##     No Information Rate : 0.6542
##     P-Value [Acc > NIR] : 2.015e-11
##
##                   Kappa : 0.4619
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: poor Class: fair Class: good Class: excellent
## Sensitivity             0.111111      0.5478      0.8538          0.76087
## Specificity             0.992674      0.8584      0.6047          0.99161
## Pos Pred Value          0.272727      0.6014      0.8033          0.79545
## Neg Pred Value          0.978339      0.8295      0.6862          0.98977
## Prevalence              0.024129      0.2806      0.6542          0.04111
## Detection Rate          0.002681      0.1537      0.5585          0.03128
## Detection Prevalence    0.009830      0.2556      0.6953          0.03932
## Balanced Accuracy       0.551893      0.7031      0.7292          0.87624
```

```
acc.nn.train <- 0.7596560 #save the accuracy
acc.nn.test <- 0.7462
```

```
# Visualization of the neural network
par(mar = numeric(4), family = "serif")
plotnet(nn.fit.cv$finalModel, cex_val = 0.5)
```



The best parameter set (decay, size) selected by cross validation is (7, 0.1). The average within training accuracy on the CV sample is 76.0%. Within the test set, the accuracy is 74.6%. The confusion matrix and the sensitivity also show that the classification does not perform well in the group "poor", which is also the case in the SVM results.

4. Comparison of the three methods

```
compare <- data.frame(NB = c(acc.nb.train, acc.nb.test), SVM = c(acc.svm.train, acc.svm.test), NN = c(a
rownames(compare) <- c("training", "testing")
compare
```

```
##                  NB       SVM       NN
## training 0.6765672 0.7494268 0.759656
## testing  0.6702000 0.7355000 0.746200
```

The classification accuracy within the training set and the testing set with the three models is shown in the table above. Generally speaking, the three models provide similar results. All of them does not achieve good accuracy. Considering the density plots and the bivariate plots, the data points themselves may be hard to classify, to some extent accounting for the low accuracy of the classification.

Compared to the Naive Bayes classifier, the support vector machine and the neural net model have slightly higher accuracy within both the CV training set and the test set. This might be due to the the large number of observations with 0 probabilities for all classes when applying the Naive Bayes model.

Neural net has similar performance with support vector machine. Although the accuracy seems a little higher for the neural net, the difference is not significant.

Part D: Conclusion

Three classification models including Naive Bayes, support vector machine and neural network are applied to classify the movies into 4 categories of quality based on IMDB scores. High accuracy is not achieved for all of the three methods. This is probably beacuse the movies themselves are hard to classify using the available features. What's more, the outliers in the data set might also affects the accuracy of classification. In addition, the sensitivity shows that all of the three methods do not predict well for the group of "poor", but perform better in classifying the group of "good".