Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics

Due: Friday, April 21, 2017 at 11:59 PM UTC-12:00 on T-Square

Data

We will use the same dataset as Project 1: `movies_merged`.

Objective

Your goal in this project is to build a linear regression model that can predict the `Gross` revenue earned by a movie based on other variables. You may use R packages to fit and evaluate a regression model (no need to implement regression yourself). Please stick to linear regression, however.

Instructions

You should be familiar with using an RMarkdown Notebook by now. Remember that you have to open it in RStudio, and you can run code chunks by pressing *Cmd+Shift+Enter*.

Please complete the tasks below and submit this R Markdown file (as **pr2.Rmd**) containing all completed code chunks and written responses, as well as a PDF export of it (as **pr2.pdf**) which should include all of that plus output, plots and written responses for each task.

*Note that **Setup** and **Data Preprocessing** steps do not carry any points, however, they need to be completed as instructed in order to get meaningful results.*

Setup

Same as Project 1, load the dataset into memory:

```
load('movies_merged')
```

This creates an object of the same name (`movies_merged`). For convenience, you can copy it to `df` and start using it:

```
df = movies_merged
cat("Dataset has", dim(df)[1], "rows and", dim(df)[2], "columns", end="\n", file="")
```

```
## Dataset has 40789 rows and 39 columns
```

```
colnames(df)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
```

```
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Domestic_Gross"    "Gross"             "Date"
```

Load R packages

Load any R packages that you will need to use. You can come back to this chunk, edit it and re-run to load any additional packages later.

```
library(ggplot2)
library(GGally)
library(stringr)
library(reshape)
library("tokenizers")
library(discretization)
```

If you are using any non-standard packages (ones that have not been discussed in class or explicitly allowed for this project), please mention them below. Include any special instructions if they cannot be installed using the regular `install.packages('<pkg name>')` command.

Non-standard packages used: GGally tokenizers reshape discretization

Data Preprocessing

Before we start building models, we should clean up the dataset and perform any preprocessing steps that may be necessary. Some of these steps can be copied in from your Project 1 solution. It may be helpful to print the dimensions of the resulting dataframe at each step.

1. Remove non-movie rows

```
## [1] 40000
```

2. Drop rows with missing `Gross` value

Since our goal is to model `Gross` revenue against other variables, rows that have missing `Gross` values are not useful to us.

```
## [1] 4558
```

3. Exclude movies released prior to 2000

Inflation and other global financial factors may affect the revenue earned by movies during certain periods of time. Taking that into account is out of scope for this project, so let's exclude all movies that were released prior to the year 2000 (you may use `Released`, `Date` or `Year` for this purpose).

```
# TODO: Exclude movies released prior to 2000
df_movie_2000=df_movie_gross[df_movie_gross$Year>=2000,]
nrow(df_movie_2000)
```

```
## [1] 3332
```

4. Eliminate mismatched rows

Note: You may compare the ***`Released`*** *column (string representation of release date) with either* ***`Year`*** *or* ***`Date`*** *(numeric representation of the year) to find mismatches. The goal is to avoid removing more than 10% of the rows.*

```
# TODO: Remove mismatched rows
test=df_movie_2000[setdiff(rownames(df_movie_2000), rownames(df_movie_2000[is.na(df_movie_2000$Released)
test_match=test[abs((test$Year-as.numeric(substr(test$Released, 1, 4))))<=1,]
test_dismatch=df_movie_2000[setdiff(rownames(df_movie_2000), rownames(df_movie_2000[rownames(test_match)
df_movie_match=test_match
nrow(df_movie_match)
```

```
## [1] 3216
```

5. Drop `Domestic_Gross` column

`Domestic_Gross` is basically the amount of revenue a movie earned within the US. Understandably, it is very highly correlated with `Gross` and is in fact equal to it for movies that were not released globally. Hence, it should be removed for modeling purposes.

```
# TODO: Exclude the `Domestic_Gross` column
df_movie_match <- df_movie_match[,!(colnames(df_movie_match) %in% c("Domestic_Gross"))]
```

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
# Check the text feature of the Runtime column
unique(str_extract_all(df_movie_match$Runtime, "[A-z]+"))
```

```
## [[1]]
## [1] "min"
##
## [[2]]
## [1] "N" "A"
```

```
#Replace df$Runtime with a numeric column containing the runtime in minutes
for(i in c(1:nrow(df_movie_match))){
  if(is.na(str_extract_all(df_movie_match[i,"Runtime"], "[A-z]+"))){
    df_movie_match[i,"Runtime"]=as.numeric(str_extract_all(df_movie_match[i,"Runtime"], "[[:digit:]]+"))
  }else if(length(str_extract_all(df_movie_match[i,"Runtime"], "[A-z]+")[[1]])==2){
    df_movie_match[i,"Runtime"]=(as.numeric(str_extract_all(df_movie_match[i,"Runtime"], "[[:digit:]]+"
  }else if(str_extract_all(df_movie_match[i,"Runtime"],"[A-z]+")=="h"){
    df_movie_match[i,"Runtime"]=(as.numeric(str_extract_all(df_movie_match[i,"Runtime"], "[[:digit:]]+"
  }else{
    df_movie_match[i,"Runtime"]=as.numeric(str_extract_all(df_movie_match[i,"Runtime"], "[[:digit:]]+"))
  }
}
```

```
df_movie_match$Runtime=as.numeric(df_movie_match[,"Runtime"])
```

Perform any additional preprocessing steps that you find necessary, such as dealing with missing values or highly correlated columns (feel free to add more code chunks, markdown blocks and plots here as necessary).

```
# TODO(optional): Additional preprocessing
```

***Note**: Do NOT convert categorical variables (like `Genre`) into binary columns yet. You will do that later as part of a model improvement task.*

Final preprocessed dataset

Report the dimensions of the preprocessed dataset you will be using for modeling and evaluation, and print all the final column names. (Again, `Domestic_Gross` should not be in this list!)

```
# TODO: Print the dimensions of the final preprocessed dataset and column names
cat("Dataset has", dim(df_movie_match)[1], "rows and", dim(df_movie_match)[2], "columns", end="\n", fil
```

```
## Dataset has 3216 rows and 38 columns
```

```
colnames(df_movie_match)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Gross"             "Date"
```

Evaluation Strategy

In each of the tasks described in the next section, you will build a regression model. In order to compare their performance, use the following evaluation procedure every time:

1. Randomly divide the rows into two sets of sizes 5% and 95%.
2. Use the first set for training and the second for testing.
3. Compute the Root Mean Squared Error (RMSE) on the train and test sets.
4. Repeat the above data partition and model training and evaluation 10 times and average the RMSE results so the results stabilize.
5. Repeat the above steps for different proportions of train and test sizes: 10%-90%, 15%-85%, ..., 95%-5% (total 19 splits including the initial 5%-95%).
6. Generate a graph of the averaged train and test RMSE as a function of the train set size (%).

You can define a helper function that applies this procedure to a given model and reuse it.

Tasks

Each of the following tasks is worth 20 points. Remember to build each model as specified, evaluate it using the strategy outlined above, and plot the training and test errors by training set size (%).

```
#Build a function to split data based on percentage criteria
sample_and_shuffle=function(data, sample_perc){

smp_size <- floor(sample_perc * nrow(data))
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]

new_data=list(train, test)
  return(new_data)
}
```

```
#Build a function to compute RMSE
rmse=function(lmmodel){
  rss=c(crossprod(lmmodel$residuals))
  mse=rss/length(lmmodel$residuals)
  result_rmse=sqrt(mse)
  return(result_rmse)
}
```

```
#Build a function to compute RMSE for test data
test_rmse=function(resid)
{
  result_rmse=(mean( (resid)^2,  na.rm =TRUE ))^0.5
  return(result_rmse)
}
```

```
# Build a function to generate columns string for modeling
model_criteria=function(col_list){
columns="Gross"
i=1
for(column in col_list)
{
  if(column!="Gross"){
  if(i==1)
  {
  columns=paste (columns,column,sep = "~", collapse = NULL)}
  else{
  columns=paste (columns,column,sep = "+", collapse = NULL)
  }

  i=i+1}
}
return(columns)
}
```

```
#Build a function to train based on training data and test data split criteria
train_by_split=function(split, col_list,dataset){
data=sample_and_shuffle(dataset,split)
training_data=data[[1]]
```

```
test_data=data[[2]]
fit=lm(eval(parse(text=model_criteria(col_list))), training_data)
residual=test_data$Gross-predict(fit, newdata=test_data)
training_rmse=rmse(fit)
testing_rmse=test_rmse(residual)
return(c(training_rmse,testing_rmse))
}
```

```
#Build a function to compute average RMSE
train_avg=function(split,col_list,dataset,repeat_times=10){
  result= data.frame(matrix(ncol = 2, nrow = 0))
  colnames(result)=c("RMSE_train", "RMSE_test")
  for(i in c(1:repeat_times)){

    result=rbind(result, as.data.frame(matrix(train_by_split(split,column_list,dataset), ncol = 2, nrow

  }
  colnames(result)=c("RMSE_train", "RMSE_test")

  return(colMeans(result))
}
```

```
#Build a function to compute RMSE over different training sizes
train_over_size=function(size_step, column_list,dataset){
  size=0.05
  result= data.frame(matrix(ncol = 3, nrow = 0))
  colnames(result)=c("Train_Perc", "RMSE_train", "RMSE_test")
  while(size<1){
  result=rbind(result,as.data.frame(matrix(unlist(list(size,train_avg(size,column_list,dataset))),1,3))
  size=size+size_step
  }
    colnames(result)=c("Train_Perc", "RMSE_train", "RMSE_test")
  return(result)
}
```

1. Numeric variables

Use linear regression to predict `Gross` based on all available *numeric* variables.

```
# TODO: Build & evaluate model 1 (numeric variables only)
# 1.Select the numeric columns (In this case, integer are also considered numeric)
column_list=colnames(df_movie_match[,sapply(df_movie_match, class) %in% c("numeric","integer")])
print(column_list)
```

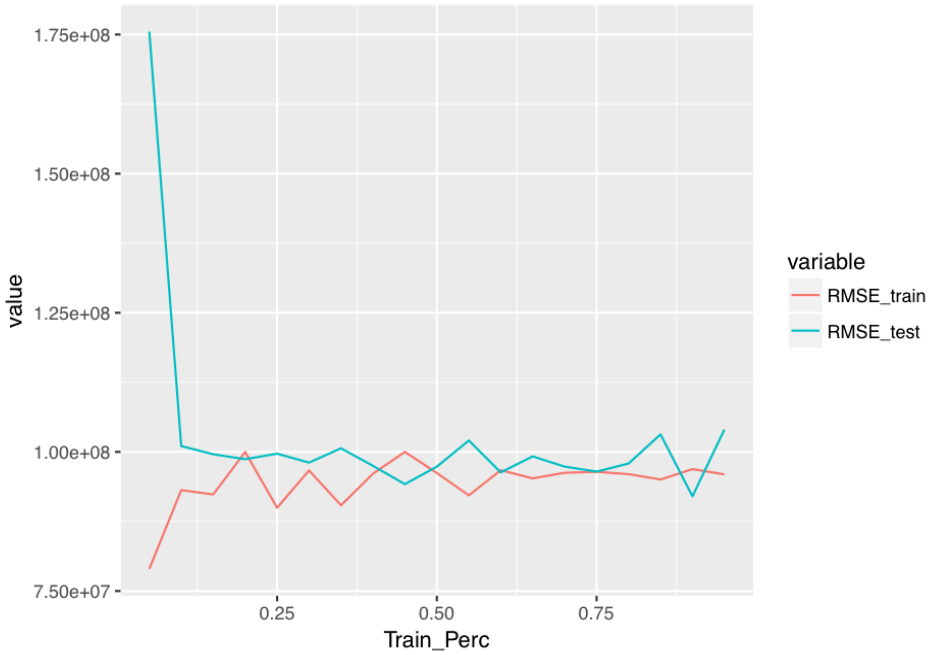
```
##  [1] "Year"              "Runtime"           "imdbRating"
##  [4] "imdbVotes"         "tomatoMeter"       "tomatoRating"
##  [7] "tomatoReviews"     "tomatoFresh"       "tomatoRotten"
## [10] "tomatoUserMeter"   "tomatoUserRating"  "tomatoUserReviews"
## [13] "Budget"            "Gross"             "Date"
```

```
# Train over different sizes and get the result
train_over_size(0.05,column_list,df_movie_match)
```

```
##    Train_Perc RMSE_train RMSE_test
## 1        0.05   86848289 187053654
```

```
## 2         0.10   90582741 101053773
## 3         0.15   90829742 100113842
## 4         0.20   87799862 100739799
## 5         0.25   96704259  98441734
## 6         0.30   94110659  99102675
## 7         0.35   94799087  98444757
## 8         0.40   94597854  98501725
## 9         0.45   97428932  96276472
## 10        0.50   94957870  98438510
## 11        0.55   94696183  99057357
## 12        0.60   96317942  97048676
## 13        0.65   95226964  99235620
## 14        0.70   96571155  96337273
## 15        0.75   96366409  96895972
## 16        0.80   95864210  98717741
## 17        0.85   96474729  96549472
## 18        0.90   95698802 100212738
## 19        0.95   96281007  96416599
```

```r
#Plot the result
data <- melt(train_over_size(0.05,column_list,df_movie_match), id.vars = "Train_Perc")
ggplot(data, aes(x = Train_Perc, y = value, colour = variable)) + geom_line()
```



Q: List all the numeric variables you used.

A: "Year" ,"Runtime" ,"imdbRating" ,"imdbVotes" ,"tomatoMeter", "tomatoRating", "tomatoReviews", "tomatoFresh","tomatoRotten", "tomatoUserMeter", "tomatoUserRating","tomatoUserReviews", "Budget", "Gross", "Date"

2. Feature transformations

Try to improve the prediction quality from **Task 1** as much as possible by adding feature transformations of the numeric variables. Explore both numeric transformations such as power transforms and non-numeric transformations of the numeric variables like binning (e.g. `is_budget_greater_than_3M`).

```
# TODO: Build & evaluate model 2 (transformed numeric variables only)
# 1. Check the data distribution of the numeric values
summary(df_movie_match[,colnames(df_movie_match[,sapply(df_movie_match, class) %in% c("numeric","intege
```

```
##       Year         Runtime        imdbRating      imdbVotes
##  Min.   :2000   Min.   :  1.0   Min.   :1.600   Min.   :      5
##  1st Qu.:2004   1st Qu.: 93.0   1st Qu.:5.700   1st Qu.:   9892
##  Median :2008   Median :102.0   Median :6.400   Median :  38771
##  Mean   :2008   Mean   :105.1   Mean   :6.291   Mean   :  87118
##  3rd Qu.:2012   3rd Qu.:115.0   3rd Qu.:7.100   3rd Qu.: 103797
##  Max.   :2017   Max.   :219.0   Max.   :9.000   Max.   :1670736
##                 NA's   :24      NA's   :26      NA's   :26
##   tomatoMeter      tomatoRating   tomatoReviews    tomatoFresh
##  Min.   :  0.00   Min.   :1.000   Min.   :  5.0   Min.   :  0.0
##  1st Qu.: 28.00   1st Qu.:4.500   1st Qu.: 83.0   1st Qu.: 21.0
##  Median : 52.00   Median :5.600   Median :127.0   Median : 57.0
##  Mean   : 51.67   Mean   :5.594   Mean   :127.3   Mean   : 72.8
##  3rd Qu.: 76.00   3rd Qu.:6.700   3rd Qu.:170.0   3rd Qu.:110.0
##  Max.   :100.00   Max.   :9.200   Max.   :355.0   Max.   :343.0
##  NA's   :340      NA's   :340     NA's   :340     NA's   :340
##   tomatoRotten    tomatoUserMeter  tomatoUserRating tomatoUserReviews
##  Min.   :  0.00   Min.   :  0.00   Min.   :0.700    Min.   :       0
##  1st Qu.: 25.00   1st Qu.: 44.00   1st Qu.:3.000    1st Qu.:   11054
##  Median : 50.00   Median : 60.00   Median :3.300    Median :   53540
##  Mean   : 54.51   Mean   : 58.39   Mean   :3.294    Mean   :  492970
##  3rd Qu.: 79.00   3rd Qu.: 74.00   3rd Qu.:3.600    3rd Qu.:  198387
##  Max.   :249.00   Max.   :100.00   Max.   :5.000    Max.   :34675430
##  NA's   :340      NA's   :157      NA's   :156      NA's   :64
##      Budget              Gross                Date
##  Min.   :     1100   Min.   :0.000e+00   Min.   :2000
##  1st Qu.:  6000000   1st Qu.:3.886e+06   1st Qu.:2004
##  Median : 20000000   Median :3.302e+07   Median :2008
##  Mean   : 35606137   Mean   :9.742e+07   Mean   :2008
##  3rd Qu.: 46000000   3rd Qu.:1.045e+08   3rd Qu.:2012
##  Max.   :425000000   Max.   :2.784e+09   Max.   :2017
##
```

```
# 2. Logarithmize and discretize some numeric columns
# make a new data
df_movie_match_new=df_movie_match
df_movie_match_new$tomatoUserReviews= cut_number(df_movie_match[,"tomatoUserReviews"], n = 6)
df_movie_match_new$tomatoRotten=cut_number(df_movie_match[,"tomatoRotten"], n = 6)
df_movie_match_new$tomatoFresh=cut_number(df_movie_match[,"tomatoFresh"], n = 6)
df_movie_match_new$tomatoReviews=cut_number(df_movie_match[,"tomatoReviews"], n = 6)
df_movie_match_new$imdbVotes=log(df_movie_match[,"imdbVotes"])
```

```
# 3. Check again
summary(df_movie_match_new[,colnames(df_movie_match[,sapply(df_movie_match, class) %in% c("numeric","in
```

```
##       Year         Runtime        imdbRating      imdbVotes
```

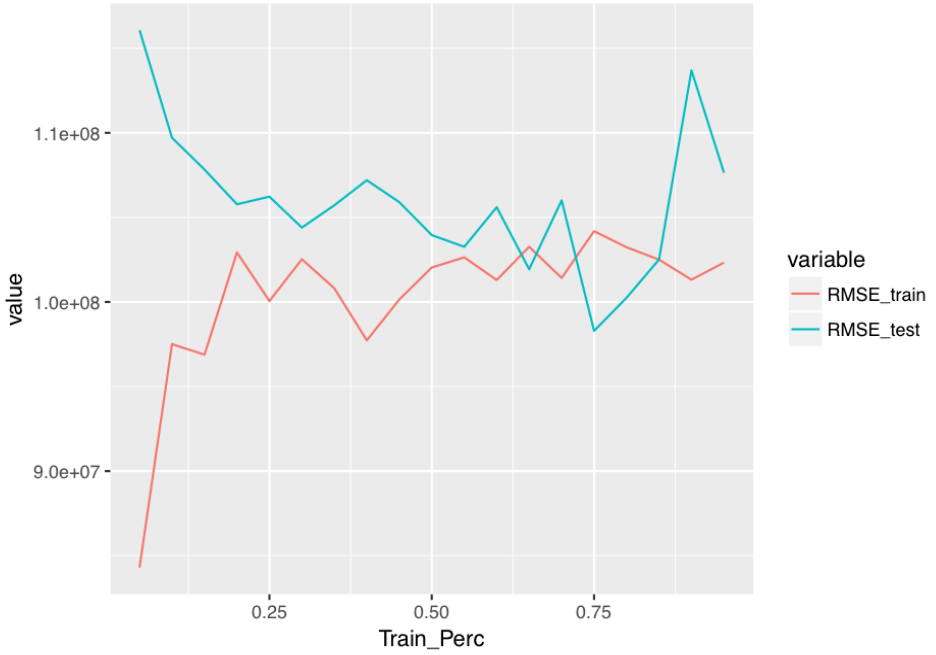
```
##  Min.   :2000   Min.   :  1.0   Min.   :1.600   Min.   : 1.609
##  1st Qu.:2004   1st Qu.: 93.0   1st Qu.:5.700   1st Qu.: 9.200
##  Median :2008   Median :102.0   Median :6.400   Median :10.565
##  Mean   :2008   Mean   :105.1   Mean   :6.291   Mean   :10.171
##  3rd Qu.:2012   3rd Qu.:115.0   3rd Qu.:7.100   3rd Qu.:11.550
##  Max.   :2017   Max.   :219.0   Max.   :9.000   Max.   :14.329
##                 NA's   :24      NA's   :26      NA's   :26
##   tomatoMeter      tomatoRating     tomatoReviews    tomatoFresh
##  Min.   :  0.00   Min.   :1.000   [5,60]   :481   [0,13]   :506
##  1st Qu.: 28.00   1st Qu.:4.500   (60,98]  :481   (13,30]  :458
##  Median : 52.00   Median :5.600   (98,127] :486   (30,57]  :494
##  Mean   : 51.67   Mean   :5.594   (127,154]:480   (57,89]  :464
##  3rd Qu.: 76.00   3rd Qu.:6.700   (154,192]:468   (89,136] :476
##  Max.   :100.00   Max.   :9.200   (192,355]:480   (136,343]:478
##  NA's   :340      NA's   :340     NA's     :340   NA's     :340
##    tomatoRotten tomatoUserMeter  tomatoUserRating
##  [0,17]  :503   Min.   :  0.00   Min.   :0.700
##  (17,33] :483   1st Qu.: 44.00   1st Qu.:3.000
##  (33,50] :460   Median : 60.00   Median :3.300
##  (50,68] :489   Mean   : 58.39   Mean   :3.294
##  (68,91] :464   3rd Qu.: 74.00   3rd Qu.:3.600
##  (91,249]:477   Max.   :100.00   Max.   :5.000
##  NA's    :340   NA's   :157      NA's   :156
##            tomatoUserReviews     Budget              Gross
##  [0,4.03e+03]       :526     Min.   :     1100   Min.   :0.000e+00
##  (4.03e+03,2.15e+04]:525     1st Qu.:  6000000   1st Qu.:3.886e+06
##  (2.15e+04,5.35e+04]:525     Median : 20000000   Median :3.302e+07
##  (5.35e+04,1.19e+05]:525     Mean   : 35606137   Mean   :9.742e+07
##  (1.19e+05,3.14e+05]:525     3rd Qu.: 46000000   3rd Qu.:1.045e+08
##  (3.14e+05,3.47e+07]:526     Max.   :425000000   Max.   :2.784e+09
##  NA's               : 64
##       Date
##  Min.   :2000
##  1st Qu.:2004
##  Median :2008
##  Mean   :2008
##  3rd Qu.:2012
##  Max.   :2017
##
```

```
#Plot the result
result=train_over_size(0.05,column_list,df_movie_match_new)
print(result)
```

```
##    Train_Perc RMSE_train RMSE_test
## 1        0.05   84302470 116063710
## 2        0.10   97508709 109705644
## 3        0.15   96882321 107830934
## 4        0.20  102927582 105776962
## 5        0.25  100057529 106225900
## 6        0.30  102526962 104395936
## 7        0.35  100806633 105718807
## 8        0.40   97729728 107198930
## 9        0.45  100142933 105901549
## 10       0.50  102038116 103956679
```

```
## 11       0.55  102637311 103260513
## 12       0.60  101299162 105600750
## 13       0.65  103263986 101932010
## 14       0.70  101420755 106007261
## 15       0.75  104187262  98290693
## 16       0.80  103235456 100250257
## 17       0.85  102503944 102508413
## 18       0.90  101316038 113687415
## 19       0.95  102326621 107641948
```

```
data <- melt(result, id.vars = "Train_Perc")
ggplot(data, aes(x = Train_Perc, y = value, colour = variable)) + geom_line()
```



Q: Explain which transformations you used and why you chose them.

A: I categorized columns: 1. tomatoUserReviews 2. tomatoRotten 3. tomatoFresh 4. tomatoReviews and logarithmized column: 5. imdbVotes

```
Because reviews can be categorize into different categories and the imdbVotes is very skewed, Logari
```

3. Non-numeric variables

Write code that converts genre, actors, directors, and other categorical variables to columns that can be used for regression (e.g. binary columns as you did in Project 1). Also process variables such as awards into more useful columns (again, like you did in Project 1). Now use these converted columns only to build your next model.

```r
# TODO: Build & evaluate model 3 (converted non-numeric variables only)

# Convert Awards to 2 numeric columns: wins and nominations
# Process the "Awards" column and fill in the "wins" and "nominations" columns
award_convert=function(df){
for (i in c(1:nrow(df))){
    win=0
    nomination=0
  if(grepl("Won",df[i, "Awards"])){
    win=as.numeric(str_extract(str_extract(df[i,"Awards"], "Won ([\\d]+)"), "[[:digit:]]+"))
    win=win+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) win"), "[[:digit:]]+"))
    nomination=nomination+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) nomination"), "[
  }else if(grepl("Nominated",df[i, "Awards"])){
    nomination=as.numeric(str_extract(str_extract(df[i,"Awards"], "Nominated for ([\\d]+)"), "[[:digit:]
    win=win+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) win"), "[[:digit:]]+"))
    nomination=nomination+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) nomination"), "[
  }else{
    win=win+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) win"), "[[:digit:]]+"))
    nomination=nomination+as.numeric(str_extract(str_extract(df[i,"Awards"], "([\\d]+) nomination"), "[
  }
    df[i,"wins"]=win
    df[i, "nominations"]=nomination
    if(is.na(df[i,"wins"])){df[i,"wins"]=0}
    if(is.na(df[i,"nominations"])){df[i,"nominations"]=0}
}
return(df)
}


# Build a function to replace Genre with a collection of binary columns
# We list all of the genres
genre_vector=function(df){
unique(unlist(tokenize_regex(df$Genre,pattern=", ")))
#Combine the genre with the original dataset
column_genre_names=unique(unlist(tokenize_regex(df$Genre,pattern=", ")))
genre=data.frame(matrix(nrow=nrow(df), ncol=length(column_genre_names)))
colnames(genre)=column_genre_names
genre[]=0
for(val in column_genre_names){
  genre[grep(val, df[, "Genre"]), val]=1
}
genre_vectored=cbind(df, genre)
return(genre_vectored)
}
```

```r
column_list_new=colnames(df_movie_match[,sapply(df_movie_match, class) %in% c("character","Date")])
print(column_list_new)
```

```
##  [1] "Title"           "Rated"           "Released"
##  [4] "Genre"           "Director"        "Writer"
##  [7] "Actors"          "Plot"            "Language"
## [10] "Country"         "Awards"          "Poster"
## [13] "Metascore"       "imdbID"          "Type"
## [16] "tomatoImage"     "tomatoConsensus" "tomatoURL"
```

```
## [19] "DVD"             "BoxOffice"       "Production"
## [22] "Website"         "Response"
```

```
df_movie_match_new=df_movie_match
df_movie_match_new=award_convert(df_movie_match_new)
df_movie_match_new=genre_vector(df_movie_match_new)
```

```
colnames(df_movie_match_new)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Gross"             "Date"              "wins"
## [40] "nominations"       "Horror"            "Sci-Fi"
## [43] "Adventure"         "Comedy"            "Family"
## [46] "Crime"             "Music"             "Drama"
## [49] "Mystery"           "Thriller"          "Romance"
## [52] "Sport"             "Fantasy"           "Action"
## [55] "Documentary"       "Biography"         "History"
## [58] "Animation"         "Musical"           "Western"
## [61] "War"               "Short"             "N/A"
## [64] "News"
```

```
df_movie_match_new_temp=df_movie_match_new[,c(37,39:64)]
```

```
sapply(df_movie_match_new_temp, class)
```

```
##       Gross        wins nominations      Horror      Sci-Fi   Adventure
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##      Comedy      Family       Crime       Music       Drama     Mystery
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##    Thriller     Romance       Sport     Fantasy      Action Documentary
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##   Biography     History   Animation     Musical     Western         War
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##       Short         N/A        News
##   "numeric"   "numeric"   "numeric"
```

```
column_list=colnames(df_movie_match_new_temp[,sapply(df_movie_match_new_temp, class) %in% c("numeric","
df_movie_match_new_temp=df_movie_match_new_temp[,sapply(df_movie_match_new_temp, class) %in% c("numeric
column_list[5]="SciFi"
colnames(df_movie_match_new_temp)=column_list
column_list=colnames(df_movie_match_new_temp[,sapply(df_movie_match_new_temp, class) %in% c("numeric","
column_list=column_list[-26]
print(column_list)
```

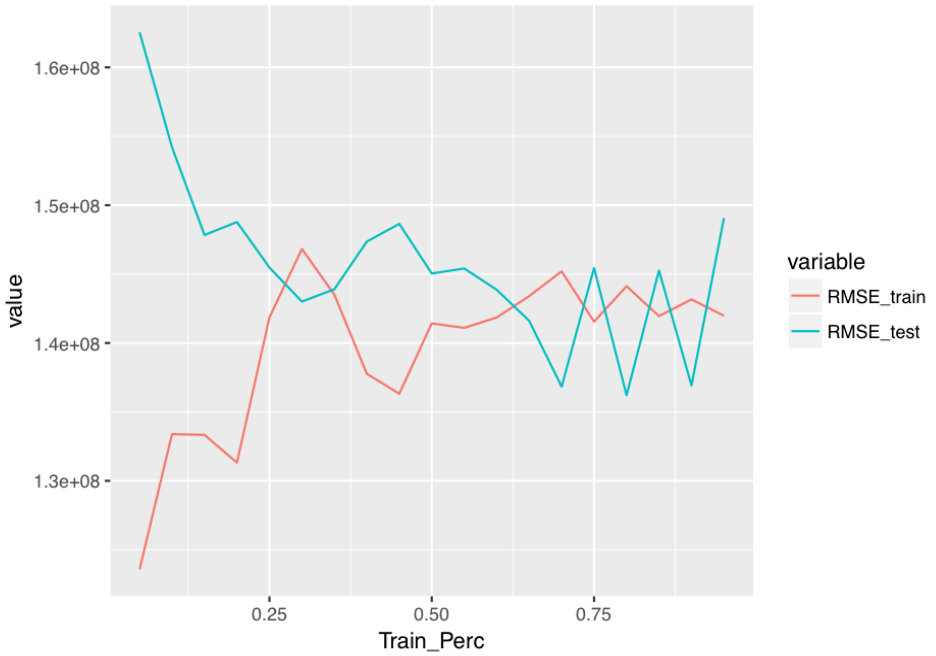
```
##  [1] "Gross"       "wins"        "nominations" "Horror"      "SciFi"
##  [6] "Adventure"   "Comedy"      "Family"      "Crime"       "Music"
```

```
## [11] "Drama"       "Mystery"     "Thriller"    "Romance"     "Sport"
## [16] "Fantasy"     "Action"      "Documentary" "Biography"   "History"
## [21] "Animation"   "Musical"     "Western"     "War"         "Short"
## [26] "News"
```

```
#Plot the result
result=train_over_size(0.05,column_list,df_movie_match_new_temp)
print(result)
```

```
##    Train_Perc RMSE_train RMSE_test
## 1        0.05  123573302 162552276
## 2        0.10  133386243 154198908
## 3        0.15  133336871 147839598
## 4        0.20  131320151 148778521
## 5        0.25  141856329 145478802
## 6        0.30  146821861 143006567
## 7        0.35  143513070 143902129
## 8        0.40  137768173 147361119
## 9        0.45  136319755 148644710
## 10       0.50  141421041 145044747
## 11       0.55  141100015 145407143
## 12       0.60  141851627 143865482
## 13       0.65  143404010 141611390
## 14       0.70  145196597 136837696
## 15       0.75  141550311 145439155
## 16       0.80  144118112 136212850
## 17       0.85  141953036 145257455
## 18       0.90  143165155 136920518
## 19       0.95  141974506 149064506
```

```
data <- melt(result, id.vars = "Train_Perc")
ggplot(data, aes(x = Train_Perc, y = value, colour = variable)) + geom_line()
```



Q: Explain which categorical variables you used, and how you encoded them into features.

A: I choose the categrical variable: 1. Awards 2. Genre Because it contains many different types and can be converted to binary columns. The column `Genre` represents a list of genres associated with the movie in a string format. Write code to parse each text string into a binary vector with 1s representing the presence of a genre and 0s the absence, and add it to the dataframe as additional columns. Then remove the original `Genre` column.

For example, if there are a total of 3 genres: Drama, Comedy, and Action, a movie that is both Action and Comedy should be represented by a binary vector <0, 1, 1>. Note that you need to first compile a dictionary of all possible genres and then figure out which movie has which genres (you can use the R `tm` package to create the dictionary).

```
The variable `Awards` describes nominations and awards in text format. Convert it to 2 numeric columns
```

Note that the format of the `Awards` column is not standard; you may have to use regular expressions to find the relevant values. Try your best to process them, and you may leave the ones that don't have enough information as NAs or set them to 0s.

4. Numeric and categorical variables

Try to improve the prediction quality as much as possible by using both numeric and non-numeric variables from **Tasks 2 & 3**.

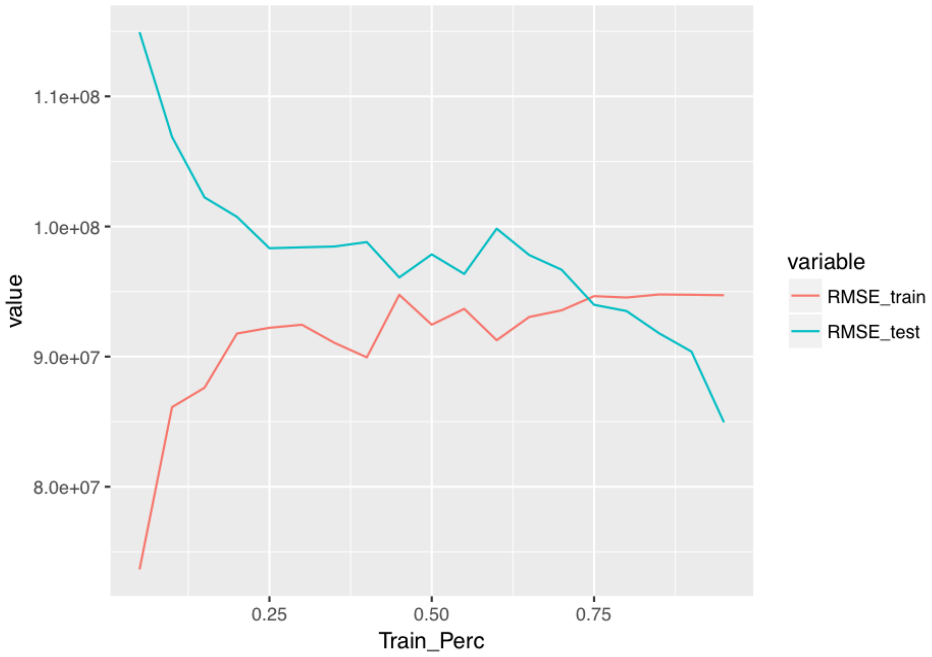
```
# TODO: Build & evaluate model 4 (numeric & converted non-numeric variables)

df_movie_match_model4=cbind(df_movie_match[,sapply(df_movie_match, class) %in% c("numeric","integer")],
column_list=colnames(df_movie_match_model4)
print(column_list)
```

```
##  [1] "Year"              "Runtime"           "imdbRating"
##  [4] "imdbVotes"         "tomatoMeter"       "tomatoRating"
##  [7] "tomatoReviews"     "tomatoFresh"       "tomatoRotten"
## [10] "tomatoUserMeter"   "tomatoUserRating"  "tomatoUserReviews"
## [13] "Budget"            "Gross"             "Date"
## [16] "wins"              "nominations"       "Horror"
## [19] "SciFi"             "Adventure"         "Comedy"
## [22] "Family"            "Crime"             "Music"
## [25] "Drama"             "Mystery"           "Thriller"
## [28] "Romance"           "Sport"             "Fantasy"
## [31] "Action"            "Documentary"       "Biography"
## [34] "History"           "Animation"         "Musical"
## [37] "Western"           "War"               "Short"
```

```
#Plot the result
result=train_over_size(0.05,column_list,df_movie_match_model4)
print(result)
```

```
##    Train_Perc RMSE_train RMSE_test
## 1        0.05   73650544 114927767
## 2        0.10   86117631 106881322
## 3        0.15   87608725 102236083
## 4        0.20   91772804 100739768
## 5        0.25   92216126  98329201
## 6        0.30   92452001  98399348
## 7        0.35   91065399  98462278
## 8        0.40   89943674  98800653
## 9        0.45   94751189  96086114
## 10       0.50   92453068  97856901
## 11       0.55   93671895  96353604
## 12       0.60   91260387  99831988
## 13       0.65   93045789  97808348
## 14       0.70   93558748  96675230
## 15       0.75   94647691  93983122
## 16       0.80   94541424  93503402
## 17       0.85   94775195  91799592
## 18       0.90   94751619  90386825
## 19       0.95   94719213  84952095
```

```
data <- melt(result, id.vars = "Train_Perc")
ggplot(data, aes(x = Train_Perc, y = value, colour = variable)) + geom_line()
```



5. Additional features

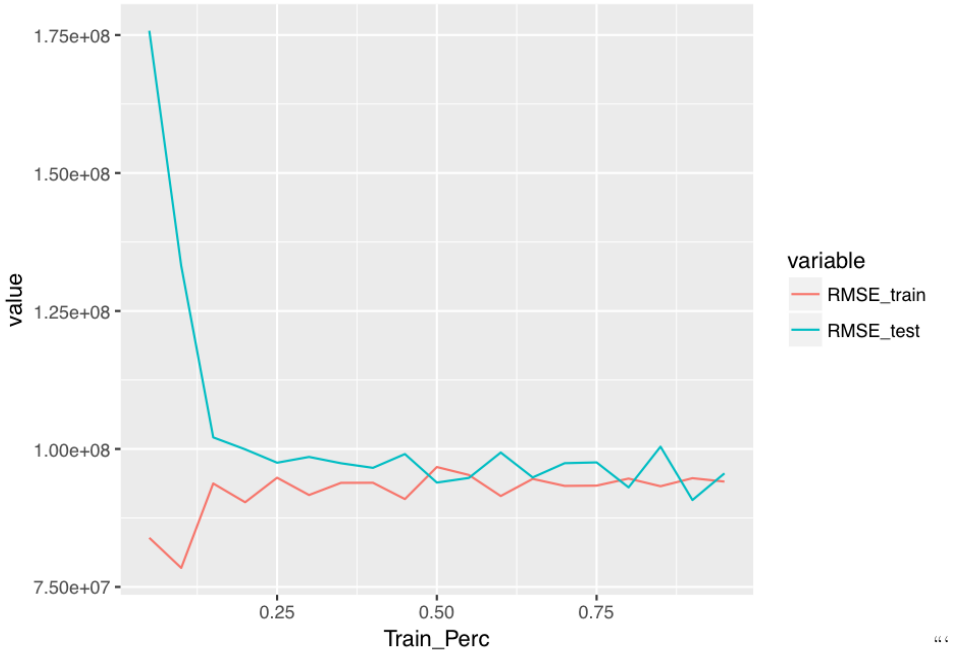
Now try creating additional features such as interactions (e.g. `is_genre_comedy` x `is_budget_greater_than_3M`) or deeper analysis of complex variables (e.g. text analysis of full-text columns like `Plot`).

```
# TODO: Build & evaluate model 5 (numeric, non-numeric and additional features)
# In the dataset, column "Comedy" indicates if the genre is Comedy or not. it counts as an additional f
#Plot the result
result=train_over_size(0.05,column_list,df_movie_match_model4)
print(result)
```

```
##    Train_Perc RMSE_train RMSE_test
## 1        0.05   83898235 175780341
## 2        0.10   78441996 133148526
## 3        0.15   93743426 102099384
## 4        0.20   90340782  99930954
## 5        0.25   94789626  97509033
## 6        0.30   91652164  98559005
## 7        0.35   93858890  97409507
## 8        0.40   93873435  96577883
## 9        0.45   90919548  99076518
## 10       0.50   96729293  93909692
## 11       0.55   95291481  94748852
## 12       0.60   91471258  99352782
## 13       0.65   94577415  94843948
## 14       0.70   93308807  97422381
```

```
## 15        0.75   93356567  97562738
## 16        0.80   94635174  93037740
## 17        0.85   93244310 100411115
## 18        0.90   94703213  90742121
## 19        0.95   94092404  95572992
```

```
data <- melt(result, id.vars = "Train_Perc")
ggplot(data, aes(x = Train_Perc, y = value, colour = variable)) + geom_line()
```



‘‘

Q: Explain what new features you designed and why you chose them.

A: I choose if the genre is comedy. Since it's an interaction feature.