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

GT account name: syan62; GT ID: 903292896

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)
```

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: None

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

```
# TODO: Remove all rows from df that do not correspond to movies
df <- df[(df$Type == "movie"),]
```

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.

```
# TODO: Remove rows with missing Gross value
df <- df[!is.na(df$Gross),]
```

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 <- df[df$Year >= 2000,]
```

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
# Before remove mismatch, there were 3332 rows in df.

mismatch = vector("numeric", length = 100) # Create a vector to store the mismatchs
counter = 0
# loop through the rows of df, split the strings in df$Released and exact the released year from it. Th
for (i in seq(1, nrow(df), by = 1)) {
  release_split = strsplit(as.character(df$Released[i]), split = "-", fixed = T)
  releasedYear = as.numeric(release_split[[1]][1])
  # Rows with released year as NA won't be removed. The Released year is smaller than the value in $Yea
  if (!is.na(df$Released[i]) && (releasedYear - df$Year[i] > 1 || releasedYear - df$Year[i] < 0)) {
    counter = counter + 1
    mismatch[counter] = i
  }
}
# Remove the mismatched rows
df <- df[-mismatch,]
# 3257 rows left after remove the mismatch between 'Year' and 'Released'

mismatch2 =  vector("numeric", length = 100) # Create a vector to store the mismatchs
counter2 = 0
for (i in seq(1, nrow(df), by = 1)) {
  if (!is.na(df$Date[i]) && (abs(df$Date[i] - df$Year[i]) > 1)) {
    counter2 = counter2 + 1
    mismatch2[counter2] = i
  }
}
df <- df[-mismatch2,]
```

4. Comments:

Before remove mismatch, there were 3332 rows in df. 3179 rows left after remove the mismatches between 'Year' and 'Date' in the dataframe with mismatched 'Year' and 'Released' removed. About 4.6% of row was removed.

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 <- df[,colnames(df)!= "Domestic_Gross"]
```

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
# Create an empty vector to store the processed runtime values
dfRows = nrow(df)
num_Runtime = vector(length = dfRows, mode = "numeric")

# Convert the runtime to numeric value in minutes.
for (i in seq(1, dfRows, by =1)) {
  # Convert character "N/A" to NA
  if (grepl("N/A", df$Runtime[i], ignore.case = T)) {
    num_Runtime[i] = NA
  }
  # Convert the runtime originally in hour+ minutes expression to correct values with minutes as unit
  if (grepl("[0-9] h", df$Runtime[i], ignore.case = T)) {
    if (grepl("[0-9] h [0-9]+ min", df$Runtime[i])) {
      runtimeSplit <- strsplit(x = df$Runtime[i], split = " ", fixed = T)
      num_Runtime[i] = as.numeric(runtimeSplit[[1]][1]) * 60 + as.numeric(runtimeSplit[[1]][3])
    }
    else if (grepl("[0-9] h", df$Runtime[i])) {
      runtimeSplit <- strsplit(x = df$Runtime[i], split = " ", fixed = T)
      num_Runtime[i] = as.numeric(runtimeSplit[[1]][1]) * 60
    }
  }
  # Convert the runtime originally in minutes to numeric values
  if (grepl("^[0-9]+ min", df$Runtime[i], ignore.case = T)) {
     runtimeSplit <- strsplit(x = df$Runtime[i], split = " ", fixed = T)
     num_Runtime[i] = as.numeric(runtimeSplit[[1]][1])
  }
}

# replace the column df$Runtime with the new numeric column
df$Runtime = num_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
# Remove the column date, because it has almost the same information as 'Year' and 'Release' columns
#
df <- df[,colnames(df)!= "Date"]
```

***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
print(paste("The final preprocessed dataset has",dim(df)[1],"rows and", dim(df)[2], "columns"))
```

```
## [1] "The final preprocessed dataset has 3179 rows and 37 columns"
```

```r
print(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] "Gross"
```

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 (%).

1. Numeric variables

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

```r
# The helper function that applies the data dividing, modeling and evaluation procedures.
# @Parameter: formula is the model formula used for linear regression function lm();
# df is the dataframe where training set and test set come from
# @Return a table summarizes the average RMSDE of training set and test set at each training set size
evaluation <- function(formula, df) {
  rmse_size_sum <- data.frame(size = seq(5,95, by =5), train_rmse = rep(0, 19), test_rmse = rep(0,19), 

for (n in seq(0.05, 0.95, by =0.05)) {   # Loop through different training set size
  # Create vectors to store training and test accuracy of individual experiment
  rmse_train20 <- vector(mode = "numeric", 20)
```

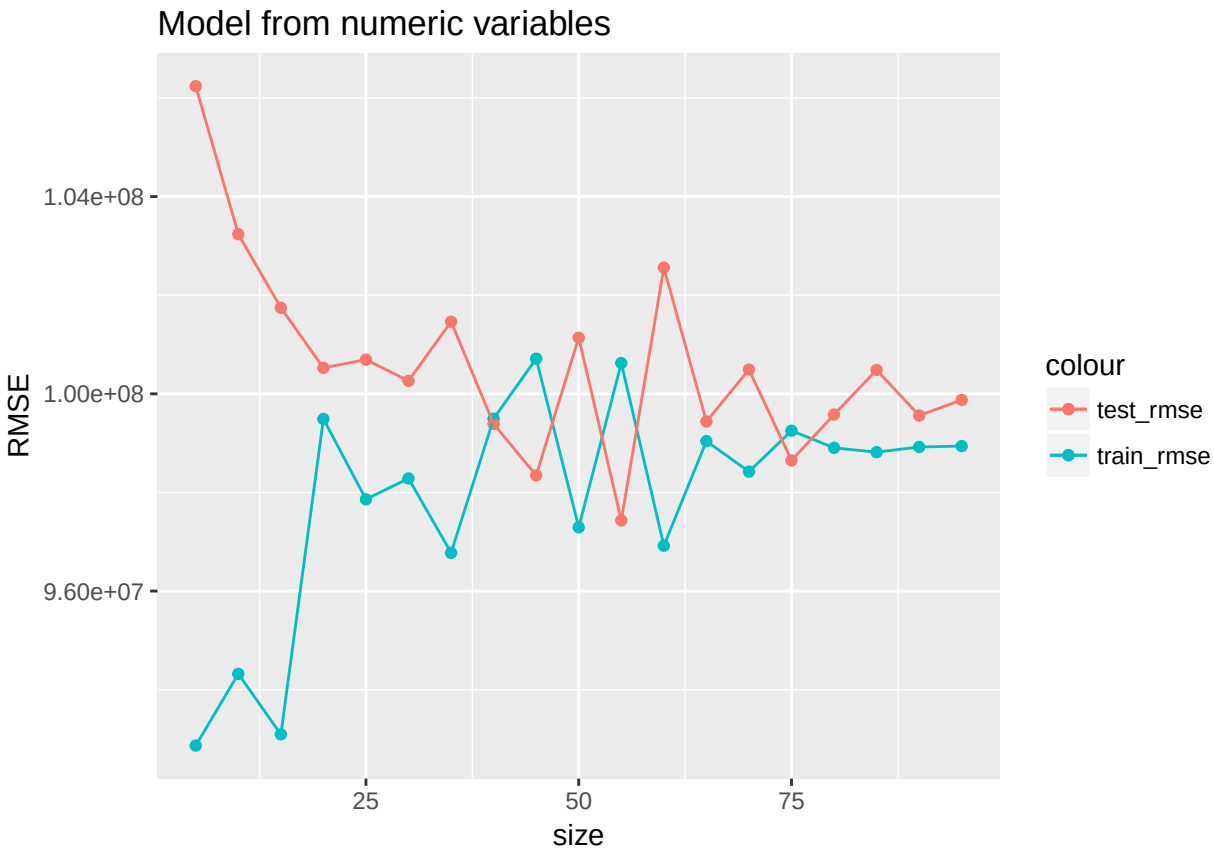
```
  rmse_test20 <- vector(mode = "numeric", 20)
  r2_20 <- vector(mode = "numeric", 20)
  for (i in seq(1, 20, by = 1)) {
    # Generate random subsets columns of the df
    df_train_id <- sample(1:nrow(df), nrow(df) * n)
    df_train <- df[df_train_id,]
    df_test <- df[-df_train_id,]
    # Train the model on each subset
    M1 <- lm(formula, df_train, na.action = na.exclude)
    theta_M1 = coef(M1)
    train_pred <- predict(M1, df_train)
    test_pred <- predict(M1, df_test)
    # Compute the training accuracy
    rmse_train = sqrt(sum((train_pred - df_train$Gross)^2)/length(train_pred))
    rmse_test = sqrt(sum((test_pred - df_test$Gross)^2)/length(test_pred))
    rmse_train20[i] = rmse_train
    rmse_test20[i] = rmse_test
    r2_20[i] = summary(M1)$r.squared
  }
  # Calculate the average RMSE of 20 experiments of training set and test set and average r-square
  ave_rmse_train <- round(mean(rmse_train20), digits = 2)
  ave_rmse_test <- round(mean(rmse_test20), digits = 2)
  ave_r2 <- round(mean(r2_20), digits = 2)
  # Store the average RMSE of training set and test set and average r-square to a dataframe summarizing
  rmse_size_sum$train_rmse[round(n/0.05, digits = 0)] = ave_rmse_train
  rmse_size_sum$test_rmse[round(n/0.05, digits = 0)] = ave_rmse_test
  rmse_size_sum$r_Square[round(n/0.05, digits = 0)] = ave_r2
}

return(rmse_size_sum)
}
```

```
# TODO: Build & evaluate model 1 (numeric variables only)

# Get the numeric variables
varType <- sapply(df, class)
numericVar <- which(varType == "numeric")
print(numericVar)
```

```
##             Year          Runtime       imdbRating        imdbVotes
##                2                5               16               17
##     tomatoRating  tomatoUserMeter tomatoUserRating           Budget
##               22               27               28               36
##            Gross
##               37
```

```
# Create a new dataframe constructed by the numberic columns of the original df. Avoid to
df_numeric <- df[,numericVar]
df_numeric <- na.omit(df_numeric)
df_numeric <- df_numeric[df_numeric$Gross != 0,]

# Define the formula for the regression experiments
regFormula <- Gross~ Year + Runtime + imdbRating + imdbVotes + tomatoRating + tomatoUserMeter + tomatoUs
# Run the evaluation function to get the table of average RMSEs at different training size
```

```
rmse_size_sum_onlyNum = evaluation(regFormula, df_numeric)

# Plot the RMSE against training size
library(ggplot2)
ggplot(rmse_size_sum_onlyNum, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + geo
```



Q: List all the numeric variables you used.

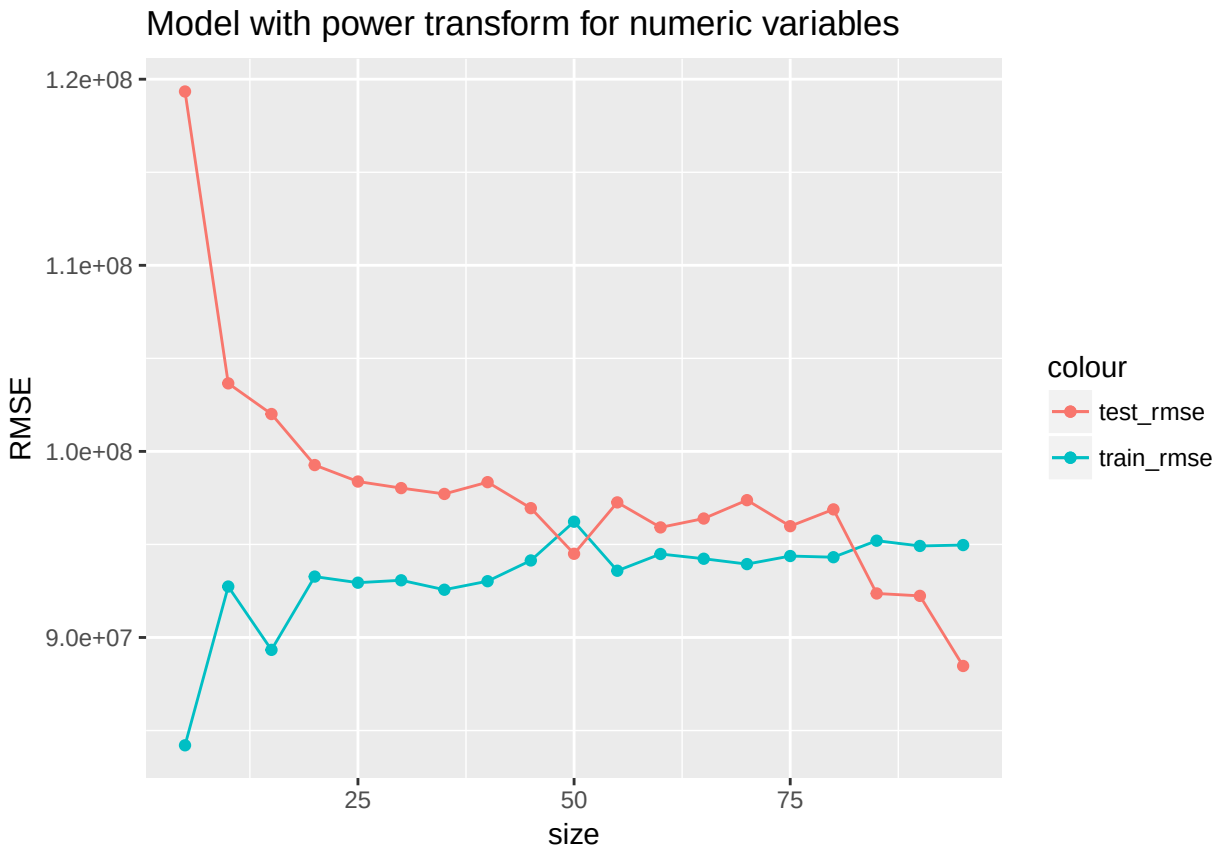
A: I have used variables: Year, Runtime, imdbRating, imdbVotes, tomatoRating, tomatoUserMeter, tomatoUserRating and Budget. I didn't use all of the numeric variables, because the unused variables (Metascore, tomatoMeter, tomatoReviews, tomatoFresh, tomatoRotten, tomatoUserReviews) are correlated with the used parameters. Even imdbRating, imdbVotes, tomatoRating, tomatoUserMeter, tomatoUserRating are positively correlated, but I found to keep them in the regression does help to get a smaller test RMSE.

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)

# Model with power transformation only
library(ggplot2)

# Add a square term for Runtime, imdbRating, imdbVotes and Budget
regFormula <- Gross ~Year + Runtime+ I(Runtime^2) +imdbRating + I(imdbRating^2) + imdbVotes + I(imdbVot
```

```
# Use the evaluation function to calculate the rmse with different training size.
rmse_size_sum_ptr = evaluation(regFormula, df_numeric)

library(ggplot2)
ggplot(rmse_size_sum_ptr, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + geom_poi
```



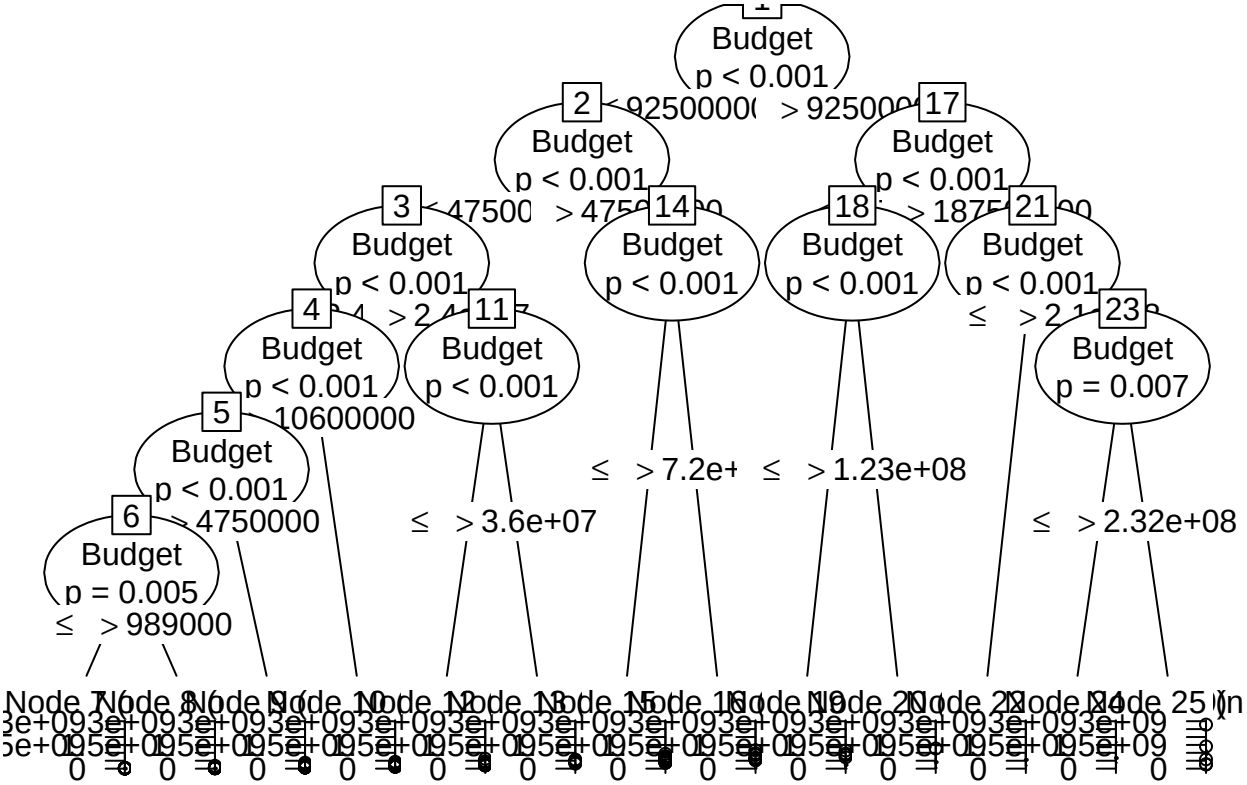

```
# Use the decision tree to find the cut points of variable binning
library(party)
```

```
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: sandwich
```

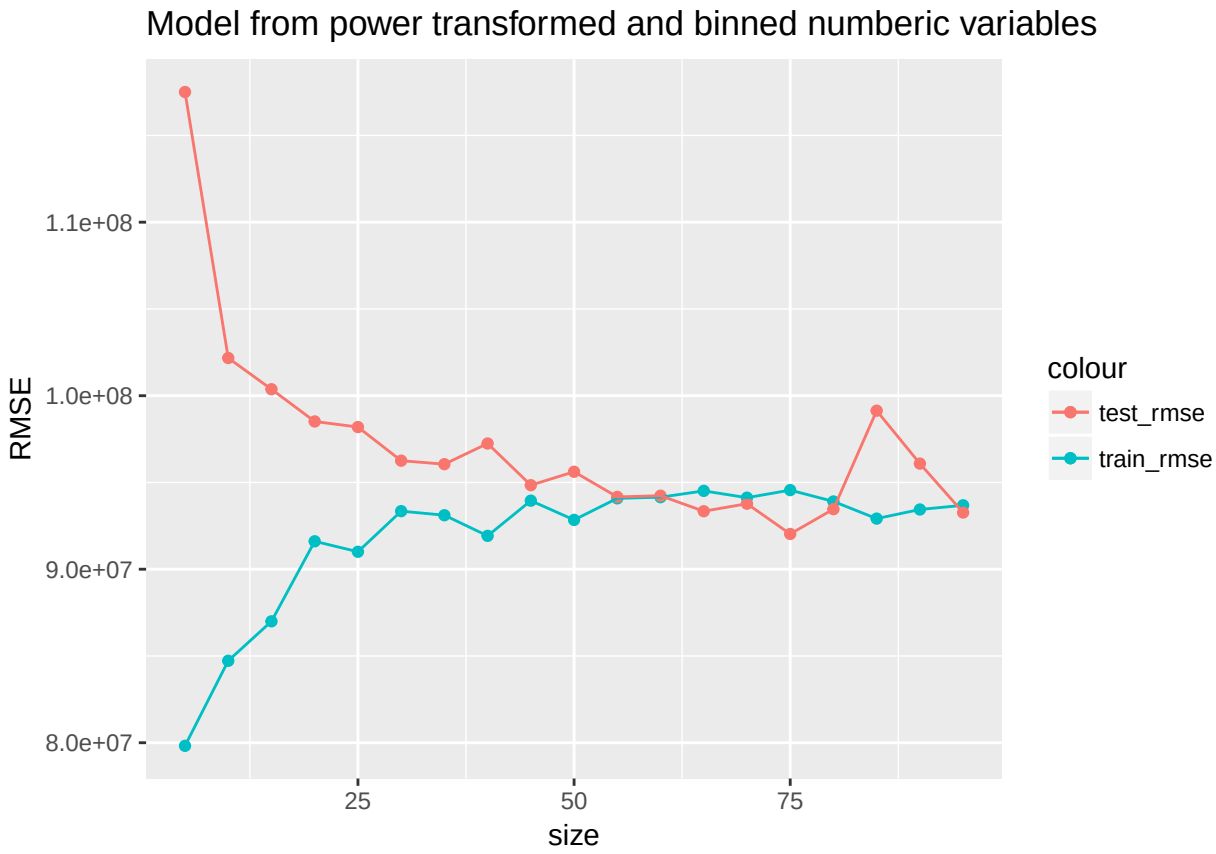
```r
df_ctree <- ctree(Gross ~ Budget, data=df_numeric)
print(df_ctree)
```

```
##
##   Conditional inference tree with 13 terminal nodes
##
## Response:  Gross
## Input:  Budget
## Number of observations:  2774
##
## 1) Budget <= 92500000; criterion = 1, statistic = 1665.71
##   2) Budget <= 47500000; criterion = 1, statistic = 804.025
##     3) Budget <= 2.4e+07; criterion = 1, statistic = 419.19
##       4) Budget <= 10600000; criterion = 1, statistic = 149.469
##         5) Budget <= 4750000; criterion = 1, statistic = 35.725
##           6) Budget <= 989000; criterion = 0.995, statistic = 7.786
##             7)*  weights = 112
##           6) Budget > 989000
##             8)*  weights = 292
##         5) Budget > 4750000
##           9)*  weights = 384
##       4) Budget > 10600000
##         10)*  weights = 617
##     3) Budget > 2.4e+07
##       11) Budget <= 3.6e+07; criterion = 0.999, statistic = 11.689
##         12)*  weights = 395
##       11) Budget > 3.6e+07
##         13)*  weights = 209
##   2) Budget > 47500000
##     14) Budget <= 7.2e+07; criterion = 1, statistic = 25.194
##       15)*  weights = 312
##     14) Budget > 7.2e+07
##       16)*  weights = 166
## 1) Budget > 92500000
##   17) Budget <= 187500000; criterion = 1, statistic = 95.324
##     18) Budget <= 1.23e+08; criterion = 1, statistic = 19.059
##       19)*  weights = 90
##     18) Budget > 1.23e+08
##       20)*  weights = 144
##   17) Budget > 187500000
##     21) Budget <= 2.1e+08; criterion = 1, statistic = 13.948
##       22)*  weights = 28
##     21) Budget > 2.1e+08
##       23) Budget <= 2.32e+08; criterion = 0.993, statistic = 7.352
##         24)*  weights = 9
##       23) Budget > 2.32e+08
##         25)*  weights = 16
```

```r
plot(df_ctree)
```




```
# Bin the Budget column using the result from the decision tree.
df_numeric$Budget[which(df_numeric$Budget < 989000)] = 494500
df_numeric$Budget[which(df_numeric$Budget >= 989000 & df_numeric$Budget < 4.75e+06)] = 2.8695e+06
df_numeric$Budget[which(df_numeric$Budget >= 4.75e+06 & df_numeric$Budget < 1.06e+07)] = 7.675e+06
df_numeric$Budget[which(df_numeric$Budget >= 1.06e+07 & df_numeric$Budget < 2.4e+07)] = 1.73e+07
df_numeric$Budget[which(df_numeric$Budget >= 2.4e+07 & df_numeric$Budget < 3.6e+07)] = 3.0e+07
df_numeric$Budget[which(df_numeric$Budget >= 3.6e+07 & df_numeric$Budget < 4.75e+07)] = 4.175e+07
df_numeric$Budget[which(df_numeric$Budget >= 4.75e+07 & df_numeric$Budget < 7.2e+07)] = 5.975e+07
df_numeric$Budget[which(df_numeric$Budget >= 7.2e+07 & df_numeric$Budget < 9.25e+07)] = 8.225e+07
df_numeric$Budget[which(df_numeric$Budget >= 9.25e+07 & df_numeric$Budget < 1.23e+08)] = 1.0775e+08
df_numeric$Budget[which(df_numeric$Budget >= 1.23e+08 & df_numeric$Budget < 1.875e+08)] = 1.5525e+08
df_numeric$Budget[which(df_numeric$Budget >= 1.875e+08 & df_numeric$Budget < 2.1e+08)] = 1.9875e+08
df_numeric$Budget[which(df_numeric$Budget >= 2.1e+08 & df_numeric$Budget < 2.32e+08)] = 2.21e+08

# Use the binned Budget in the same formula
regFormula <- Gross ~Year + Runtime+ I(Runtime^2) +imdbRating + I(imdbRating^2) + imdbVotes + I(imdbVot

rmse_size_sum_ptr_bin = evaluation(regFormula, df_numeric)

library(ggplot2)
ggplot(rmse_size_sum_ptr_bin, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + geo
```



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

A: I chose power transformations for Runtime, imdbRating, imdbVotes and Budget. I chose them because when I plot them against the Gross individually, I found the scatter plots of Runtime and imdbRating with Gross do show some sort of quadratic distribution. Adding the I(Runtime ^2) and I(imdbRating ^2) does improve the RMSE a little. For imdbVotes and Budget, I found their scatter plots with Gross must contain linear term but not pure linear. With the I(imdbVotes ^2) and I(Budget ^2) term, I do see improvements with them. After the power transformation, I tried to bin budget, because I think Budget is the most critical variable affect the Gross. I have used decision tree to find out the binning ranges. I replace the budget value for budget in each range with the middle value of that range. However, after binning of Budget, the improvement of RMSE is not significant compared with the power transformation alone, but the r-square value increases, indicating better correlation.

Using the power transformation and binning transformation, I can improve the RMSE by 5% - 10% compared with the RMSE in task1 for different training size. The r-square also improved from 0.71~0.73 in task1 to 0.75~0.79.

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.

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

# Convert the categorical variables 'Genre', 'Actors' and 'Directors' to binary columns
```

```r
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```r
# Remove rows with zeros in 'Gross' column
df<- df[df$Gross != 0,]

# Use 'tm' package to analyze name frequency of director and actors.
# Prepare the 'Director column' and 'Actor column' to a format easier for freqency analysis.
director_noComma1 = vector("character", length = nrow(df))
director_noComma2 = vector("character", length = nrow(df))

actor_noComma1 = vector("character", length = nrow(df))
actor_noComma2 = vector("character", length = nrow(df))

# Remove commas, combine firstname and lastname of every director and actor as one word.
for (i in seq(1, nrow(df), by = 1)) {
  director_noComma1[i] = gsub(", ", ",", df$Director[i])
  text = director_noComma1[i]
  director_noComma1[i] = gsub(" ", "_", text)

  actor_noComma1[i] = gsub(", ", ",", df$Actors[i])
  text = actor_noComma1[i]
  actor_noComma1[i] = gsub(" ", "_", text)
}
director_noComma2 = sapply(director_noComma1, function(x) strsplit(x, split = ",", fixed = T))
actor_noComma2 = sapply(actor_noComma1, function(x) strsplit(x, split = ",", fixed = T))

genre_noComma = sapply(df$Genre, function(x) strsplit(x, split = ",", fixed = T))

# Use the text analysis functions in 'tm' package to obtain the name frequency (number of movies) of ea
myCorpusGenre = VCorpus(VectorSource(genre_noComma))
myDTMGenre = DocumentTermMatrix(myCorpusGenre, control = list(minWordLength = 1))
genre_dict = findFreqTerms(myDTMGenre, 1)

genre_dict = genre_dict[genre_dict != "n/a"]  # remove the frequency of n/a.

myCorpusDirector = VCorpus(VectorSource(director_noComma2))
myDTM_Director = DocumentTermMatrix(myCorpusDirector, control = list(minWordLength = 1))
freq = sort(colSums(as.matrix(myDTM_Director)), decreasing = T)
director_nameFreq = data.frame(name = names(freq), MoviesNum = freq)

myCorpusActor = VCorpus(VectorSource(actor_noComma2))
myDTM_Actor = DocumentTermMatrix(myCorpusActor, control = list(minWordLength = 1))
freq = sort(colSums(as.matrix(myDTM_Actor)), decreasing = T)
actor_nameFreq = data.frame(name = names(freq), MoviesNum = freq)

director_nameFreq$name = sapply(director_nameFreq$name, function(x) gsub("_", " ", x))
```

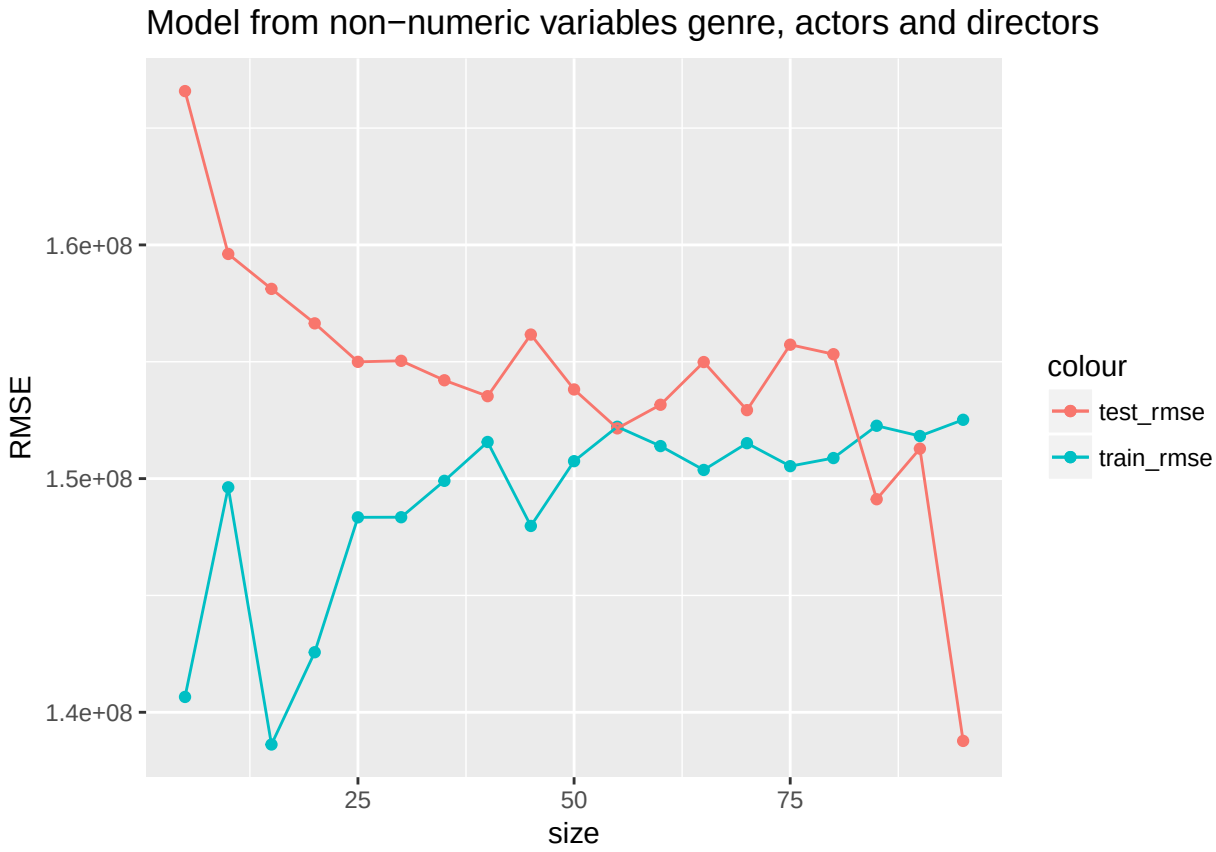
```r
actor_nameFreq$name = sapply(actor_nameFreq$name, function(x) gsub("_", " ", x))

# Remove the frequency of "n/a" in Director and Actor frequency counts
director_nameFreq = director_nameFreq[director_nameFreq$name != "n/a",]
actor_nameFreq = actor_nameFreq[actor_nameFreq$name != "n/a",]
```

```r
# Add new binary columns for separated Genre and popularity of actors and directors

variableNames = c(names(df), genre_dict, "popDirectors", "otherDirectors", "popActors", "otherActors")

for (i in seq(1, length(genre_dict) + 4, by=1)) {
  df[,i+37] <- 0
}

names(df) = variableNames

popDirector = director_nameFreq[director_nameFreq$MoviesNum >5,] # Directors have more than 6 movies in
popActor = actor_nameFreq[actor_nameFreq$MoviesNum >15,] # Actors have more than 15 movies in the dataf
```

```r
# Assign binary values for separated Genre columns and actor/director popularity columns

for (i in seq(1, nrow(df), by =1)) { # Loop through every row of df
  # Check whether genre matches, if yes, set 1 to that Genre column, otherwise set 0.
  for (j in seq(1, length(genre_dict), by=1)) {
    if (grepl(names(df)[j+37], df$Genre[i], ignore.case = TRUE)) {
      df[i,j+37] = 1
    }
    else {
      df[i,j+37] = 0
    }
  }

  # Check if one movie is directed by a popular director
  for (j in seq(1, nrow(popDirector), by =1)) {
    if (grepl(popDirector$name[j], df$Director[i], ignore.case = T)) {
      df$popDirectors[i] = 1
      df$otherDirectors[i] = 0
      break
    }
  }

  # Check if one movie has a popular actor
  for (j in seq(1, nrow(popActor), by = 1)) {
    if (grepl(popActor$name[j], df$Actors[i], ignore.case = T)) {
      df$popActors[i] = 1
      df$otherActors[i] = 0
      break
    }
  }
}

#
colnames(df)[56] <- "scifi"
# The value in non-popular director and actor column are opposite to the value in popDirector and popAc
```

```r
df$otherDirectors[which(df$popDirectors==0)] = 1
df$otherDirectors[which(df$popDirectors==1)] = 0

df$otherActors[which(df$popActors==0)] = 1
df$otherActors[which(df$popActors==1)] = 0
```

```r
library(ggplot2)
# Use the binary columns converted from catagorical variables (Genre, Actor and Director columns) for t
regFormula <- Gross ~ action + adult + adventure + animation + biography + comedy + crime + documentary

rmse_size_sum_binary = evaluation(regFormula, df)
ggplot(rmse_size_sum_binary, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + geom_
```




```r
# Convert Awards to 2 numeric columns: wins and nominations (Same code from Project 1)

df$Wins <- NA
df$Nominations <- NA
# Counter for movies without any wins or nominations
nonAward_count = 0

# Loop through every row in df
for (i in seq(1, nrow(df), by = 1)) {
  # Convert movies do not have any awards
  if (df$Awards[i] == "N/A") {
    df$Wins[i] = NA
    df$Nominations[i] = NA
    nonAward_count = nonAward_count + 1
```

```
  }
  # Convert df$Awards contents start with "Won [0-9]+".
  if (grepl("Won [0-9]+", df$Awards[i], ignore.case = T)) {
    won.regexpr = regexpr(pattern= "Won [0-9]+", text= df$Awards[i])
    won = regmatches(m=won.regexpr, x = df$Awards[i])
    won_split = unlist(strsplit(won, " "))
    won_num = as.numeric(won_split[2])
    # Convert movies df$Awards contents matching the regular expression "Won [0-9]+.*Another"
    if (grepl("Won [0-9]+.*Another", df$Awards[i], ignore.case = T)) {
      if (grepl("Won [0-9]+.*Another [0-9]+ win.*& [0-9]+ nomination", df$Awards[i], ignore.case = T)) {
        win.regexpr = regexpr(pattern= "[0-9]+ win", text= df$Awards[i])
        win = regmatches(m = win.regexpr, x = df$Awards[i])
        win_split = unlist(strsplit(win, " "))
        win_num = as.numeric(win_split[1])

        nomination.regexpr = regexpr(pattern= "[0-9]+ nomination", text= df$Awards[i])
        nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
        nomination_split = unlist(strsplit(nomination, " "))
        nomination_num = as.numeric(nomination_split[1])

        df$Wins[i] = won_num + win_num
        df$Nominations[i] = nomination_num
      }

      else if (grepl("Won [0-9]+.*Another [0-9]+ win", df$Awards[i], ignore.case = T)) {
        win.regexpr = regexpr(pattern= "[0-9]+ win", text= df$Awards[i])
        win = regmatches(m = win.regexpr, x = df$Awards[i])
        win_split = unlist(strsplit(win, " "))
        win_num = as.numeric(win_split[1])

        df$Wins[i] = won_num + win_num
      }
      # Convert movies df$Awards contents matching the regular expression "Won [0-9]+.*Another [0-9]+ n
      else {
        nomination.regexpr = regexpr(pattern= "[0-9]+ nomination", text= df$Awards[i])
        nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
        nomination_split = unlist(strsplit(nomination, " "))
        nomination_num = as.numeric(nomination_split[1])

        df$Wins[i] = won_num
        df$Nominations[i] = nomination_num
      }
    }
    # Convert df$Awards contents start with "Won [0-9]+" and no other wins and nominations.
    else {
      df$Wins[i] = won_num
    }
  }
  # Convert df$Awards contents start with "Nominated for [0-9]+".
  if (grepl("Nominated for [0-9]+", df$Awards[i], ignore.case = T)) {
    nominated.regexpr = regexpr(pattern= "Nominated for [0-9]+", text= df$Awards[i])
    nominated = regmatches(m = nominated.regexpr, x = df$Awards[i])
    nominated_split = unlist(strsplit(nominated, " "))
```

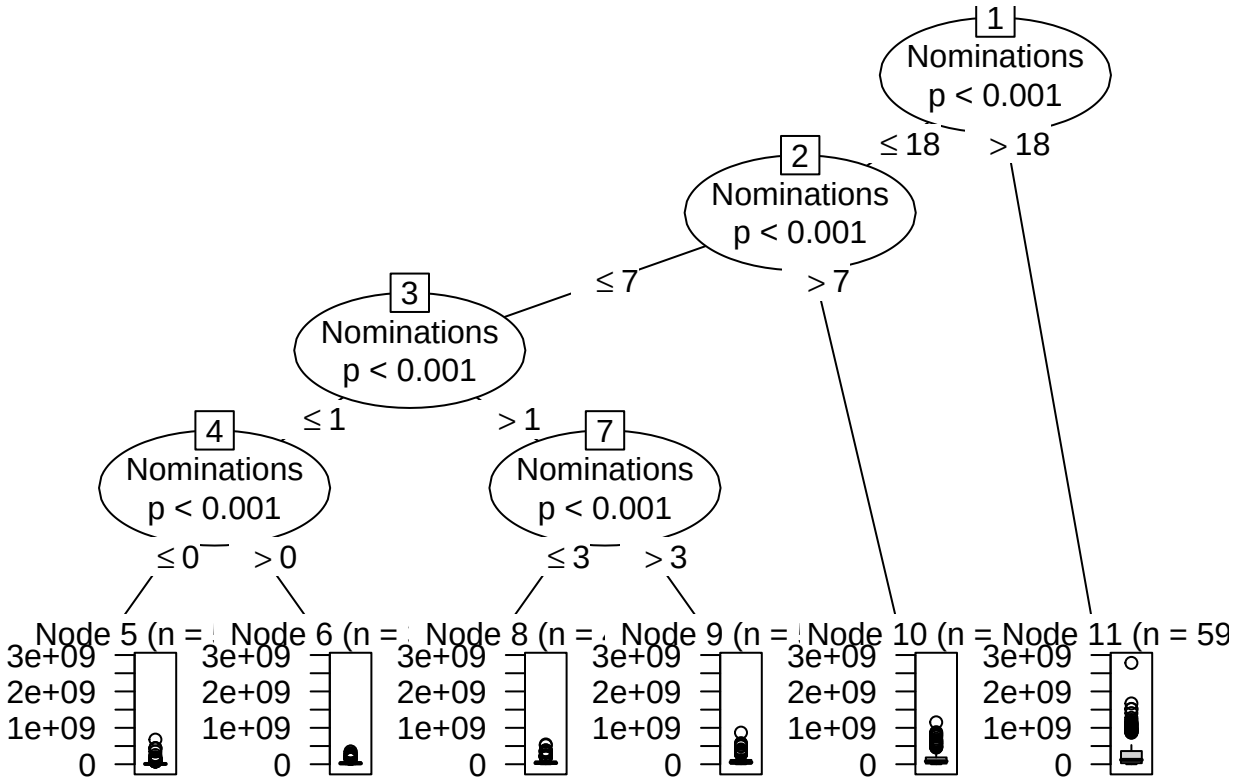
```
  nominated_num = as.numeric(nominated_split[3])

  # Convert movies df$Awards contents matching the regular expression "Nominated for     [0-9]+.*Anot
  if (grepl("Nominated for [0-9]+.*Another [0-9]+ win.*& [0-9]+ nomination", df$Awards[i], ignore.cas
    win.regexpr = regexpr(pattern= "[0-9]+ win", text= df$Awards[i])
    win = regmatches(m = win.regexpr, x = df$Awards[i])
    win_split = unlist(strsplit(win, " "))
    win_num = as.numeric(win_split[1])

    nomination.regexpr = regexpr(pattern= "[0-9]+ nomination", text= df$Awards[i])
    nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
    nomination_split = unlist(strsplit(nomination, " "))
    nomination_num = as.numeric(nomination_split[1])

    df$Wins[i] = win_num
    df$Nominations[i] = nomination_num + nominated_num
  }
  # Convert movies df$Awards contents matching the regular expression "Nominated for     [0-9]+.*Anot
  else if (grepl("Nominated for [0-9]+.*Another [0-9]+ win", df$Awards[i], ignore.case = T)) {
    win.regexpr = regexpr(pattern= "[0-9]+ win", text= df$Awards[i])
    win = regmatches(m = win.regexpr, x = df$Awards[i])
    win_split = unlist(strsplit(win, " "))
    win_num = as.numeric(win_split[1])

    df$Wins[i] = win_num
    df$Nominations[i] = nominated_num
  }
  # Convert movies df$Awards contents matching the regular expression "Nominated for     [0-9]+.*Anot
  else if (grepl("Nominated for [0-9]+.*Another [0-9]+ nomination", df$Awards[i], ignore.case = T)) {
    nomination.regexpr = regexpr(pattern= "[0-9]+ nomination", text= df$Awards[i])
    nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
    nomination_split = unlist(strsplit(nomination, " "))
    nomination_num = as.numeric(nomination_split[1])

    df$Nominations[i] = nomination_num + nominated_num
  }
  # Convert df$Awards contents have "Nominated for [0-9]+" only without other awards.
  else {
    df$Nominations[i] = nominated_num
  }
}

# Convert df$Awards contents start with "^[0-9]+ win".
if (grepl("^[0-9]+ win", df$Awards[i], ignore.case = T)) {
  win.regexpr = regexpr(pattern= "[0-9]+ win", text= df$Awards[i])
  win = regmatches(m = win.regexpr, x = df$Awards[i])
  win_split = unlist(strsplit(win, " "))
  win_num = as.numeric(win_split[1])

  if (grepl("^[0-9]+ win.*& [0-9]+ nomination", df$Awards[i], ignore.case = T)) {
    nomination.regexpr = regexpr(pattern= "[0-9]+ nomination", text= df$Awards[i])
    nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
    nomination_split = unlist(strsplit(nomination, " "))
```

```
      nomination_num = as.numeric(nomination_split[1])

      df$Wins[i] = win_num
      df$Nominations[i] = nomination_num
    }
    else {
      df$Wins[i] = win_num
    }
  }

  # Convert df$Awards contents start with "^[0-9]+ nomination".
  if (grepl("^[0-9]+ nomination", df$Awards[i], ignore.case = T)) {
    nomination.regexpr = regexpr(pattern= "^[0-9]+ nomination", text= df$Awards[i])
    nomination = regmatches(m = nomination.regexpr, x = df$Awards[i])
    nomination_split = unlist(strsplit(nomination, " "))
    nomination_num = as.numeric(nomination_split[1])
    df$Nominations[i] = nomination_num
  }
}

df$Nominations[which(is.na(df$Nominations))] = 0
```

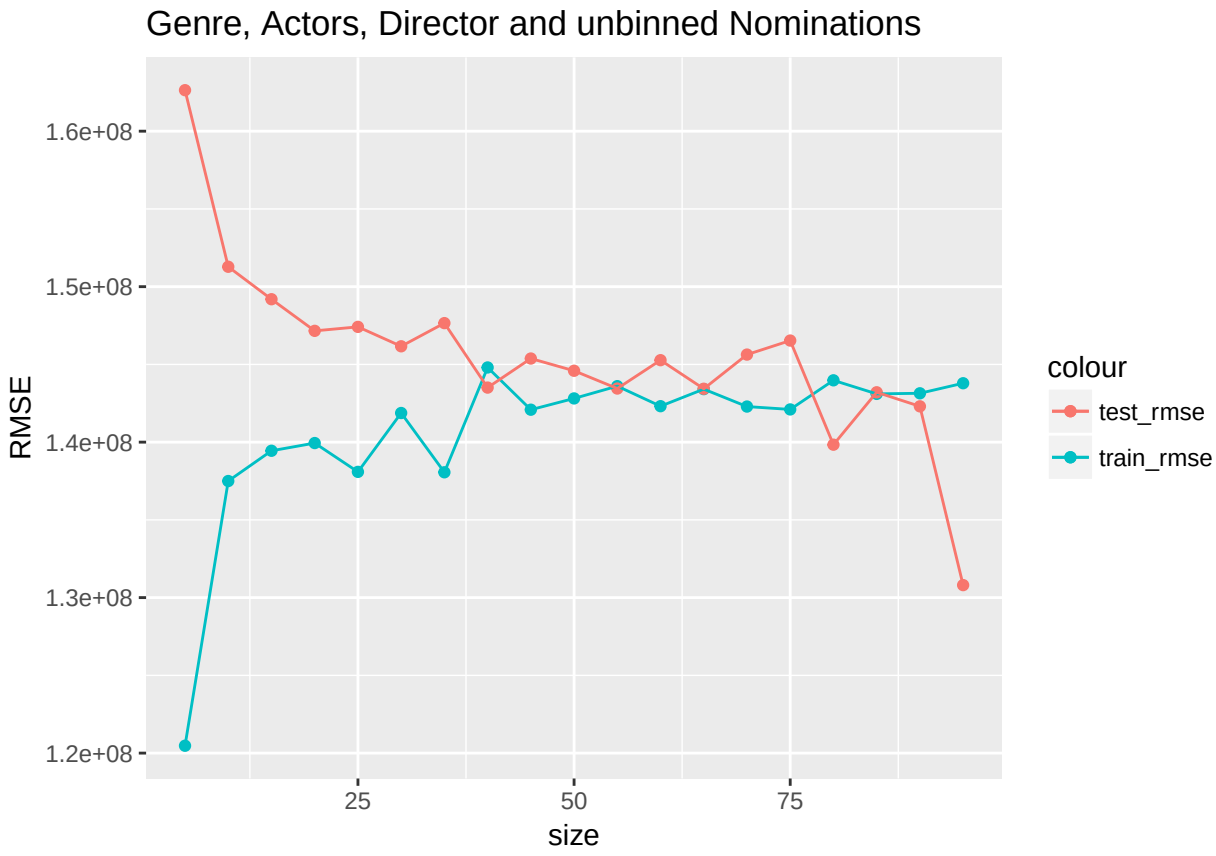
```
# Use decision tree to determine the binning range of number of nominations

library(party)

df_ctree <- ctree(Gross ~ Nominations, data=df)
print(df_ctree)
```

```
##
##   Conditional inference tree with 6 terminal nodes
##
## Response:  Gross
## Input:  Nominations
## Number of observations:  2950
##
## 1) Nominations <= 18; criterion = 1, statistic = 340.578
##   2) Nominations <= 7; criterion = 1, statistic = 266.543
##     3) Nominations <= 1; criterion = 1, statistic = 126.68
##       4) Nominations <= 0; criterion = 1, statistic = 18.645
##         5)*  weights = 508
##       4) Nominations > 0
##         6)*  weights = 337
##     3) Nominations > 1
##       7) Nominations <= 3; criterion = 1, statistic = 13.222
##         8)*  weights = 449
##       7) Nominations > 3
##         9)*  weights = 566
##   2) Nominations > 7
##     10)*  weights = 493
## 1) Nominations > 18
##   11)*  weights = 597
```

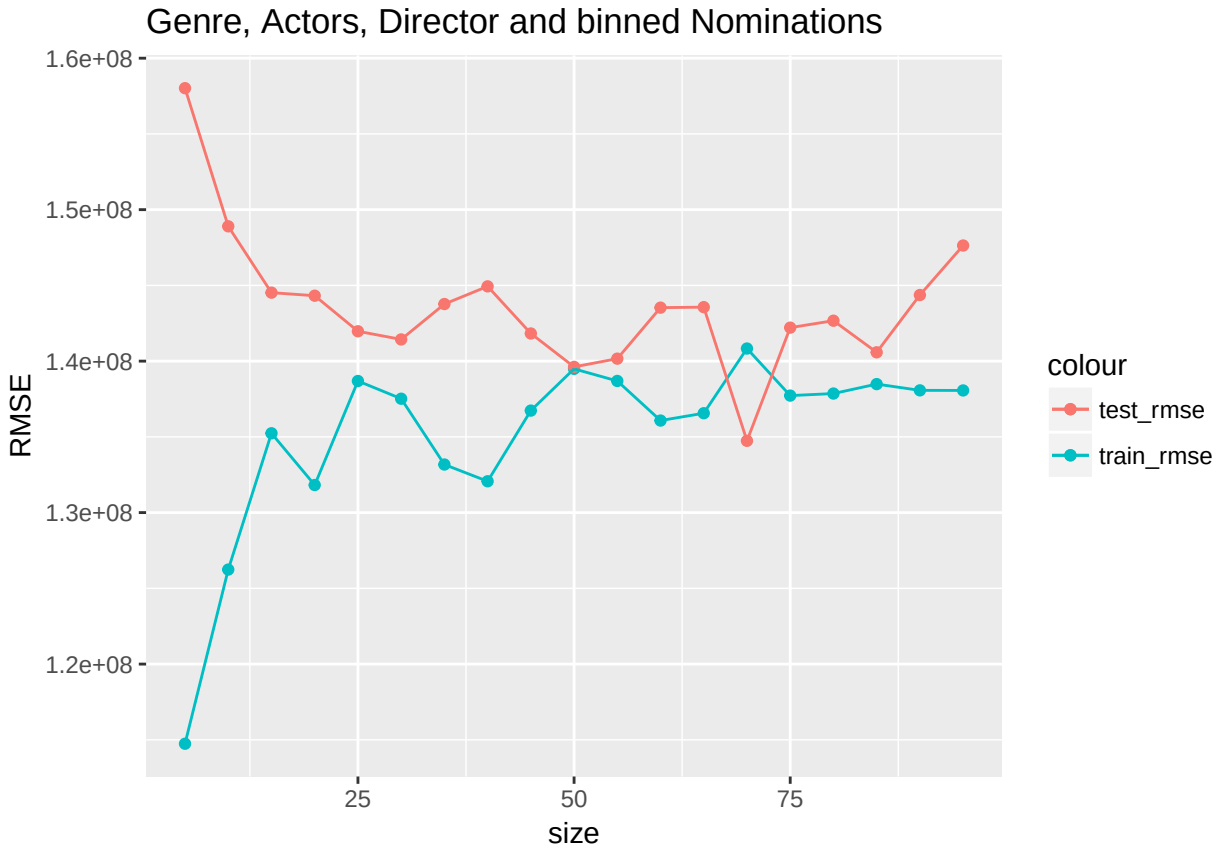
```
plot(df_ctree)
```




```
library(ggplot2)

# Bin nominations according to the decision tree. Create separate binary columns for each binning range
df$Nominations0_1 = 0
df$Nominations0_1[which(df$Nominations <= 1)] = 1

df$Nominations1_3 = 0
df$Nominations1_3[which(df$Nominations > 1 & df$Nominations <= 3)] = 1

df$Nominations3_7 = 0
df$Nominations3_7[which(df$Nominations > 3 & df$Nominations <= 7)] = 1

df$Nominations7_18 = 0
df$Nominations7_18[which(df$Nominations > 7 & df$Wins <= 18)] = 1

df$Nominations18above = 0
df$Nominations18above[which(df$Nominations > 18)] = 1

# Regression with the non-binned Nominations
regFormula <- Gross ~ action + adult + adventure + animation + biography + comedy + crime + documentary

library(ggplot2)
rmse_size_sum_binary_unBinNominations = evaluation(regFormula, df)
ggplot(rmse_size_sum_binary_unBinNominations, aes(size)) + geom_point(aes(y = train_rmse, colour = "tra
```




```
# Regression with the binary columns with the binned Nominations
regFormula <- Gross ~ action + adult + adventure + animation + biography + comedy + crime + documentary

rmse_size_sum_binaryall = evaluation(regFormula, df)
ggplot(rmse_size_sum_binaryall, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + g
```



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

A: Firstly, I transform categorical variables Genre, Actors and Directors into binary features. For the categorical variable transformation, I used the 'tm' package to create dictionaries for all genres, all actor names and all director names. For the Genres, I used the same processing method as I did in project 1. I created individual columns for each Genre in the df. My code parse each text string in the Genre column into a binary vector with "1" representing the presence of a genre and '0' representing the absence, and add each individual genre to the dataframe as additional columns. For the Actors, I created two new columns named 'popActors' and 'otherActors' in df. I firstly connectted the actors' first name and last name by "_", to ensure the 'tm' functions can treat one actor's full name as a whole word. Then, I created the dictionary of actor names and got the frequency of each name appearance (The frequency is the number of movies one actor participated). I sortted the frequency in a decreasing order, and picked out the actors whose name frequency is higher than 15 times (perform more 15 movies in this dataframe). These actors are classified as popular actors. Then, my code check whether these popular actors' name appears in the 'Actors' column of each movie in df. If yes, assign '1' to the 'popActors' column on that row. If not, assign '0' to the 'popActors' column and assign '1' to the 'otherActors' column. In this way, I converted the popularity of actors into binary features. For the Directors, I got their name frequency and convert their popularity into features in binary columns 'popDirectors' and 'otherDirectors' in df. The processing procedure are exactly the same as what I did for the actors' names. The only difference is that I define the directors' name appears more than 5 times as popular directors.

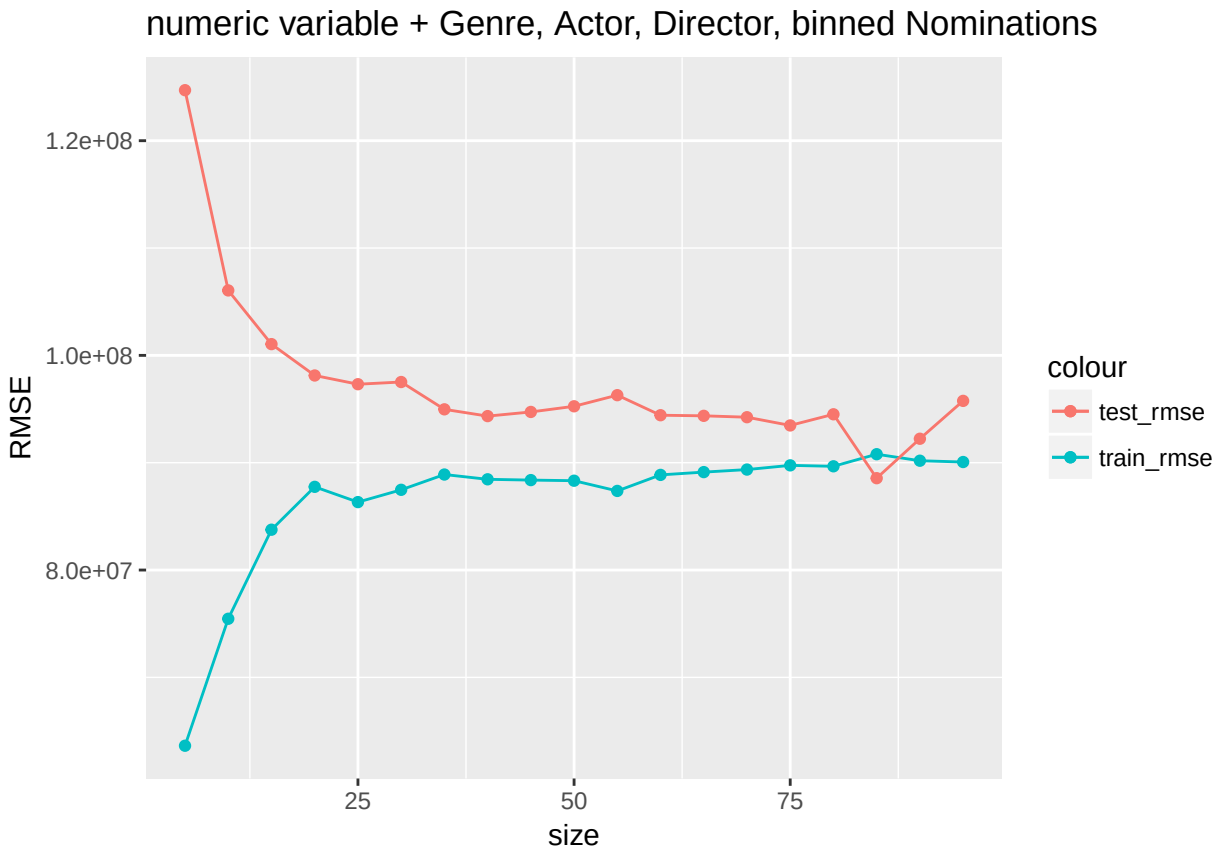
Secondly, I extract Nominations from Awards column and bin them. Since the number of wins and the number of nominations are highly correlated, I only used Nominations as a new feature to avoid feature redundancy. I extracted the Nominations from 'Awards' column and store them in a new 'Nominations' column, by using the same code in Project 1. Then, I used decision tree to find out the proper binning ranges of number of nominations. According the number of nominations of each movie, '1' or '0' was assigned to binary columns 'Nominations0_1', 'Nominations1_3', 'Nominations3_7', 'Nominations7_18', 'Nominations18above'.

The 'Nominations' does help to improve the quality of regression. With the coverted categorical variables (binary genres, popActors, otherActors, popDirectors and otherDirectors), the RMSE of test set is around 1.5e+08 or higher. When incorporating non-binned nominations, the test set RMSE fluctuates around 1.45e+08. Using the binned nomination columns, the test set RMSE fluctuates around 1.40e+08 and even drops to 1.35e+08 for bigger training size.

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**.

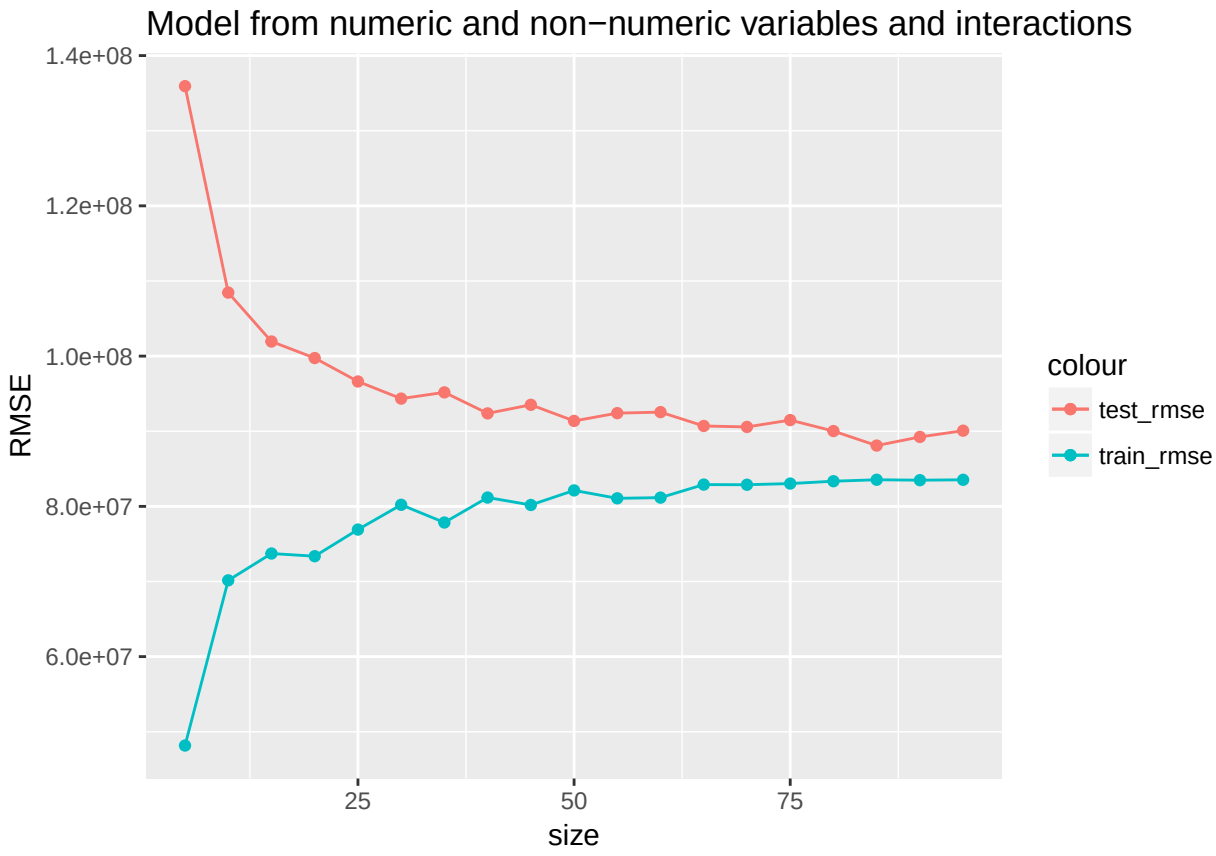
```
# TODO: Build & evaluate model 4 (numeric & converted non-numeric variables)

# Create a clean dataframe with only the numeric and converted non-numeric variables.
drops = c("Title", "Rated", "Released", "Genre","Director","Writer","Actors","Plot","Language", "Country

df_clean <- df[,!(names(df) %in% drops)]
df_clean1 <- na.omit(df_clean)

# Bin 'Budget' as what I have done in Task2.
df_clean1$Budget[which(df_clean1$Budget < 989000)] = 494500
df_clean1$Budget[which(df_clean1$Budget >= 989000 & df_clean1$Budget < 4.75e+06)] = 2.8695e+06
df_clean1$Budget[which(df_clean1$Budget >= 4.75e+06 & df_clean1$Budget < 1.06e+07)] = 7.675e+06
df_clean1$Budget[which(df_clean1$Budget >= 1.06e+07 & df_clean1$Budget < 2.4e+07)] = 1.73e+07
df_clean1$Budget[which(df_clean1$Budget >= 2.4e+07 & df_clean1$Budget < 3.6e+07)] = 3.0e+07
df_clean1$Budget[which(df_clean1$Budget >= 3.6e+07 & df_clean1$Budget < 4.75e+07)] = 4.175e+07
df_clean1$Budget[which(df_clean1$Budget >= 4.75e+07 & df_clean1$Budget < 7.2e+07)] = 5.975e+07
df_clean1$Budget[which(df_clean1$Budget >= 7.2e+07 & df_clean1$Budget < 9.25e+07)] = 8.225e+07
df_clean1$Budget[which(df_clean1$Budget >= 9.25e+07 & df_clean1$Budget < 1.23e+08)] = 1.0775e+08
df_clean1$Budget[which(df_clean1$Budget >= 1.23e+08 & df_clean1$Budget < 1.875e+08)] = 1.5525e+08
df_clean1$Budget[which(df_clean1$Budget >= 1.875e+08 & df_clean1$Budget < 2.1e+08)] = 1.9875e+08
df_clean1$Budget[which(df_numeric$Budget >= 2.1e+08 & df_clean1$Budget < 2.32e+08)] = 2.21e+08

# Combine numeric & converted non-numeric variables for the regression (the useful variables determined
regFormula <- Gross ~Year + Runtime+ I(Runtime^2) +imdbRating + I(imdbRating^2) + imdbVotes + I(imdbVot

library(ggplot2)
rmse_size_sum_all = evaluation(regFormula, df_clean1)
ggplot(rmse_size_sum_all, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + geom_po
```



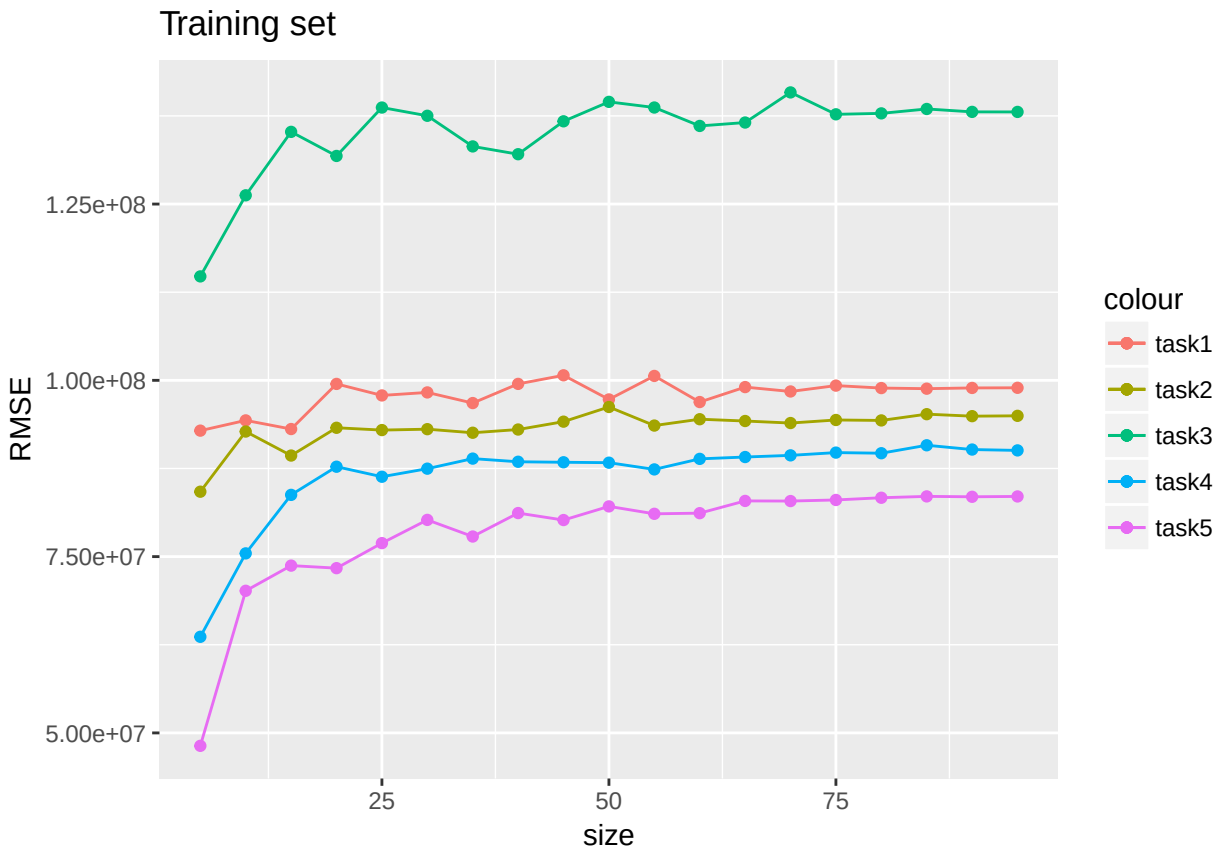
5. Additional features

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`).

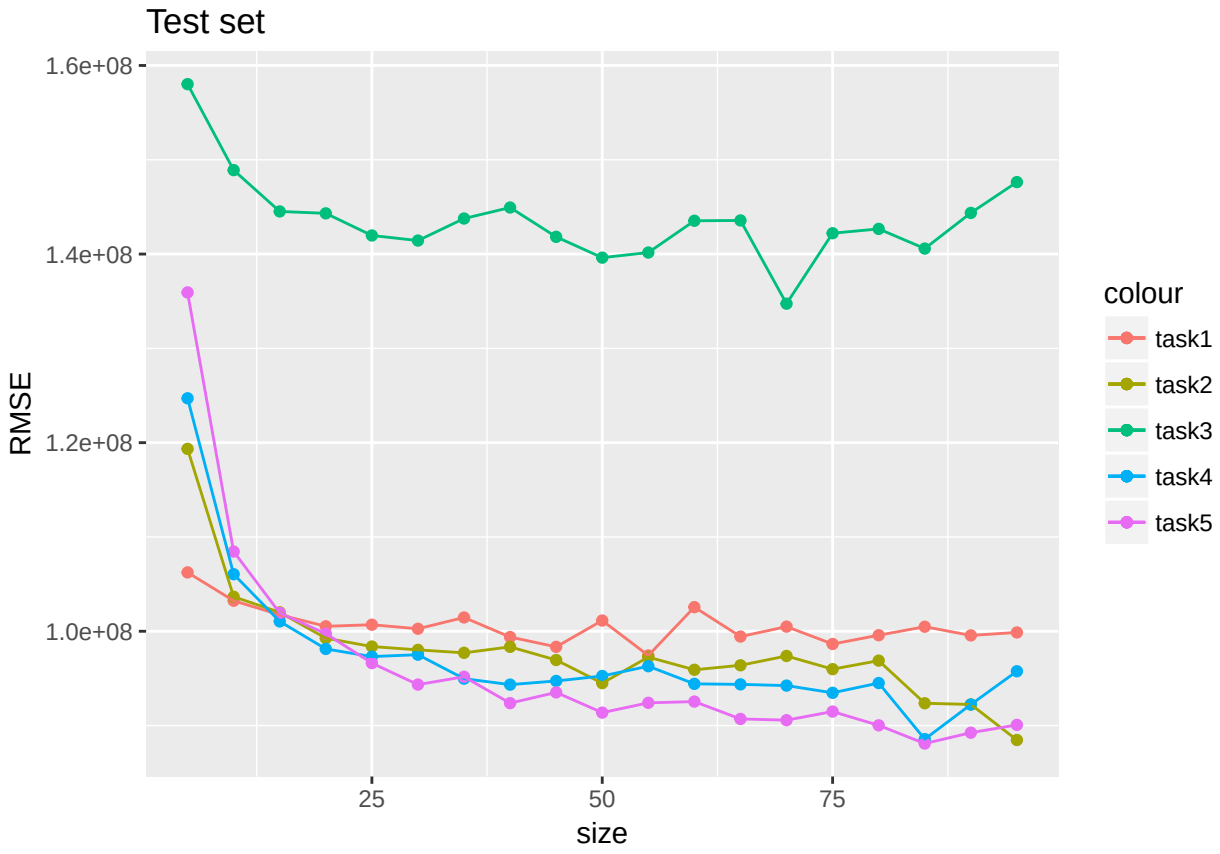
```
# TODO: Build & evaluate model 5 (numeric, non-numeric and additional features)

# Use all the features in task4 plus interaction terms of Budget with other variables
regFormula <- Gross ~Year + Runtime+ I(Runtime^2) +imdbRating + I(imdbRating^2) + imdbVotes + I(imdbVot

library(ggplot2)
rmse_size_sum_all_inter = evaluation(regFormula, df_clean1)
ggplot(rmse_size_sum_all_inter, aes(size)) + geom_point(aes(y = train_rmse, colour = "train_rmse")) + g
```




```
# Create summary tables to store the RMSE results at each training size from Task1, Task2, Task3, Task4
summary_train_RMSE = data.frame(size = seq(5,95, by =5), task1 = rmse_size_sum_onlyNum$train_rmse, task

summary_test_RMSE = data.frame(size = seq(5,95, by =5), task1 = rmse_size_sum_onlyNum$test_rmse, task2 =

library(ggplot2)
# Plot the RMSE of training set of task1 to task5
ggplot(summary_train_RMSE, aes(size)) + geom_point(aes(y = task1, colour = "task1")) + geom_point(aes(y
```




```
# Plot the RMSE of test set of task1 to task5
ggplot(summary_test_RMSE, aes(size)) + geom_point(aes(y = task1, colour = "task1")) + geom_point(aes(y =
```



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

A: I added 18 interactions terms with 'Budget'. They are Budget:Year + Budget:Runtime + Budget:Nominations0_1 + Budget:Nominations1_3 + Budget:Nominations3_7 + Budget:Nominations7_18 + Budget:Nominations18above + Budget:animation + Budget:adventure + Budget:documentary + Budget:thriller + Budget:crime + Budget:action + Budget:romance + Budget:short + Budget:comedy + Budget:drama + Budget:imdbVotes. By revisiting the analysis I have done in Project 1 and experimenting of useful features for the first several questions in Project 2, I found that Budget has the strongest relation with Gross, nominations, the top 10 Genres (drama, comedy, short, romance, action, crime,thriller, documentary,adventure and animation), imdbVotes, Runtime and production Year all show some correlation with Gross. Meanwhile, Budget also correlates with these features as what I have discussed in Question 9 of Project 1 and by common sense. So I think adding the interactions between Budget and those features will improve the regression performance.

I compared the plot of RMSE-traing size for task1 to task5 in one graph. For both training set and test set, task3 has highest RMSE, because it only uses the categorical variables and binned nominations. Besides Task3, from task1 to task5 the overall RMSE of both training set and test set decreases step by step, indicating the improvement of regression accuracy by adding new features. I note that all the RMSE-size plots of training set show a roughly monotonic increasing trend. The RMSE of training set with 5% and 10% rows are much smaller than the RMSE of other training set size. Not surprisingly, all the RMSE-size plots of test set show a roughly monotonic decreasing trend, and the test set RMSE using the model from training set with 5% and 10% rows are much higher than the RMSE of other training set size. The explanation for this is when the training set is small (5% of total rows is about 150 rows) the number of features I used for regression is comparable with the training set size, so the regression model suffers from overfitting problem. As a result, we can observe low RMSE of training set but high RMSE of test set. When the training set size is larger than 15% (more than 400 rows), the number of features is much smaller than training size, so the model becomes more and more accurate and stable.