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

Name: Fan Zhou

GTAccount: fzhou32

Data

We will use the same dataset as Project 1: `movies_merged` (https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6242/project/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 (http://rmarkdown.rstudio.com) 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", fil
e="")
```

```
## 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 = subset(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 = subset(df, is.na(Gross) == 0)
```

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
release_year = rep(0, times=nrow(df))
for (i in 1:nrow(df)) {
  if (is.na(df$Released[i])==0) {
    temp = unlist(strsplit(as.character(df$Released[i]),split='-'))[1]
    release_year[i] = as.numeric(temp) #this step is optional.
  }
}
df$release_year = release_year
df = subset(df, release_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
df = subset(df, release_year == Date)
```

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 = subset(df, select = -Domestic_Gross)
```

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minu
tes
names(df)[names(df)=='Runtime']='Old_Runtime'

convert_to_min = function(x) {
  if (x=='N/A') time = NA
  else {
    temp = unlist(strsplit(x, split=' '))
    if (length(temp)==4 & temp[2]=='h' & temp[4]=='min') {
      time=as.numeric(temp[1])*60+as.numeric(temp[3])
    } else if (temp[2]=='h' & length(temp)==2) {
      time=as.numeric(temp[1])*60
    } else {time=as.numeric(temp[1])
    }
  }
  return (time)
}


for (i in 1:nrow(df)) {
   df$Runtime[i] = convert_to_min(df$Old_Runtime[i])
}

df = subset(df, select = -Old_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
# Delete unnecessary columns
df = subset(df, select = -c(Country, Poster, tomatoConsensus, tomatoURL, BoxOff
ice, Website, imdbID, DVD, Response))
```

***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(dim(df))
```

```
## [1] 2682   30
```

```
print(names(df))
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Genre"             "Director"
##  [7] "Writer"            "Actors"            "Plot"
## [10] "Language"          "Awards"            "Metascore"
## [13] "imdbRating"        "imdbVotes"         "Type"
## [16] "tomatoMeter"       "tomatoImage"       "tomatoRating"
## [19] "tomatoReviews"     "tomatoFresh"       "tomatoRotten"
## [22] "tomatoUserMeter"   "tomatoUserRating"  "tomatoUserReviews"
## [25] "Production"        "Budget"            "Gross"
## [28] "Date"              "release_year"      "Runtime"
```

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

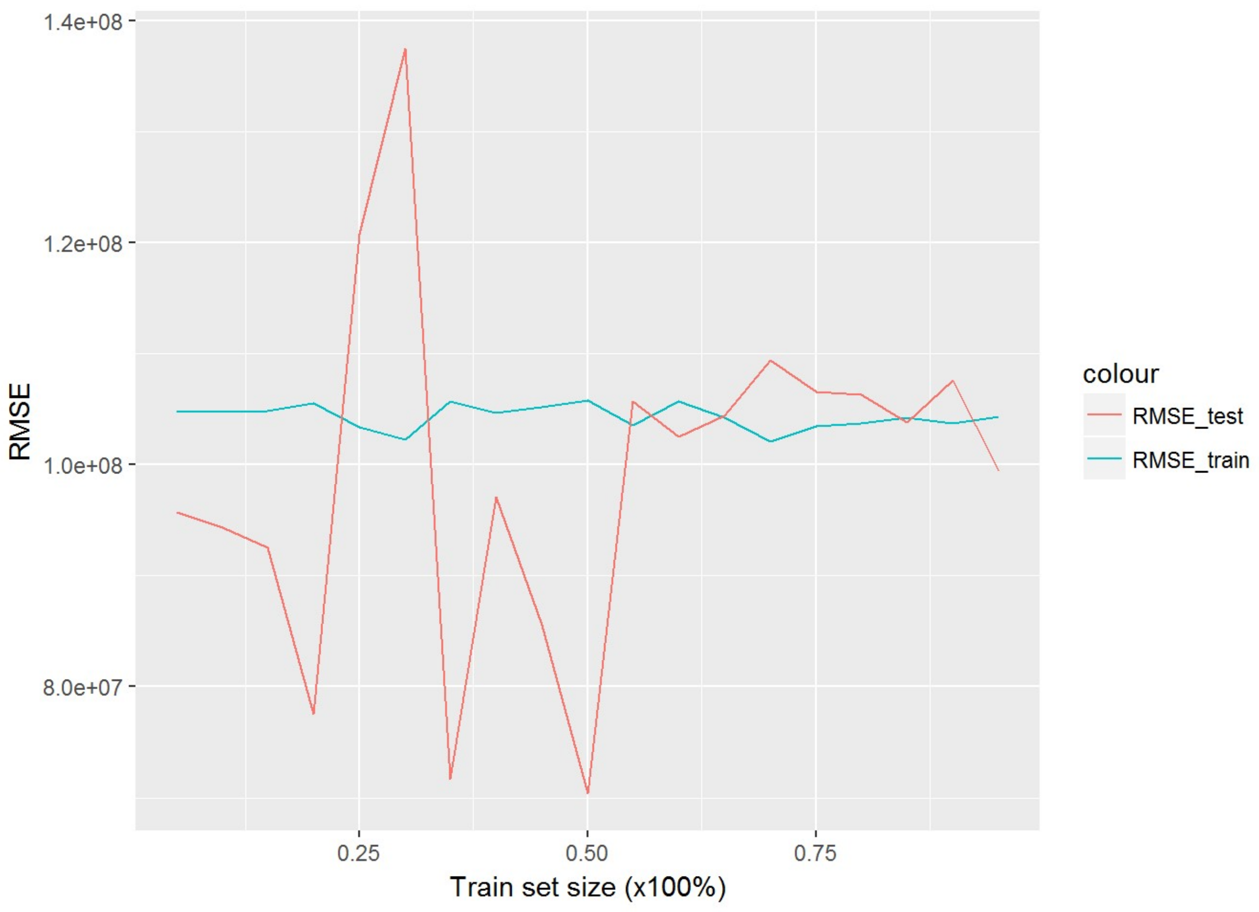
Each of the following tasks is worth 20 points. Remember to build each model as specified, evaluate it using the strategy outlined above, and plot the training and test errors by training set size (%).

1. Numeric variables

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

```
random_sample = function(df, percentage) {
  sample_number = sample(dim(df)[1], percentage*(dim(df)[1]))
  train_sample = df[sample_number, ]
  test_sample = df[-sample_number, ]
  return (list(train_sample, test_sample))
}
```

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

# Delelte rows whose Gross is 0
df = subset(df, Gross!=0)

# Delete rows whose imdb or tomatoUserReviews is NA
df = subset(df, is.na(imdbVotes)==0)

# Delete rows whose tomatoUserReviews is NA
df = subset(df, is.na(tomatoUserReviews)==0)

# Delete rows whose Runtime is NA
df = subset(df, is.na(Runtime)==0)

RMSE_train =c()
RMSE_test = c()
for (i in 1:19) {
  for (time in (1:10)) {
    data = random_sample(df,i*0.05)
    train = data[[1]]
    test = data[[2]]
    M1 = lm(Gross~imdbVotes+tomatoUserReviews+Budget+imdbRating+Runtime+Year, t
rain)
    theta=coef(M1)
    #X1 = 1
    #X2 = test$imdbVotes
    #X3 = test$tomatoUserReviews
    #X4 = test$Budget
    Y = test$Gross
    #Y_test = theta[1]*X1+theta[2]*X2+theta[3]*X3+theta[4]*X4+theta[5]*X5
    Y_test = predict(M1,test)
    RMSE_train[time] = sqrt(mean(residuals(M1)^2))
    RMSE_test[time] = sqrt(sum((Y-Y_test)^2)/length(Y))
  }
  RMSE_train[i] = mean(RMSE_train)
  RMSE_test[i] = mean(RMSE_test)
}

vis1 = data.frame('train_size'=seq(0.05, 0.95, 0.05),'RMSE_train'=RMSE_train,
'RMSE_test'=RMSE_test)

ggplot(vis1, aes(train_size)) +
  geom_line(aes(y = RMSE_train, colour = "RMSE_train")) +
  geom_line(aes(y = RMSE_test, colour = "RMSE_test")) +
  xlab('Train set size (x100%)') + ylab('RMSE')
```



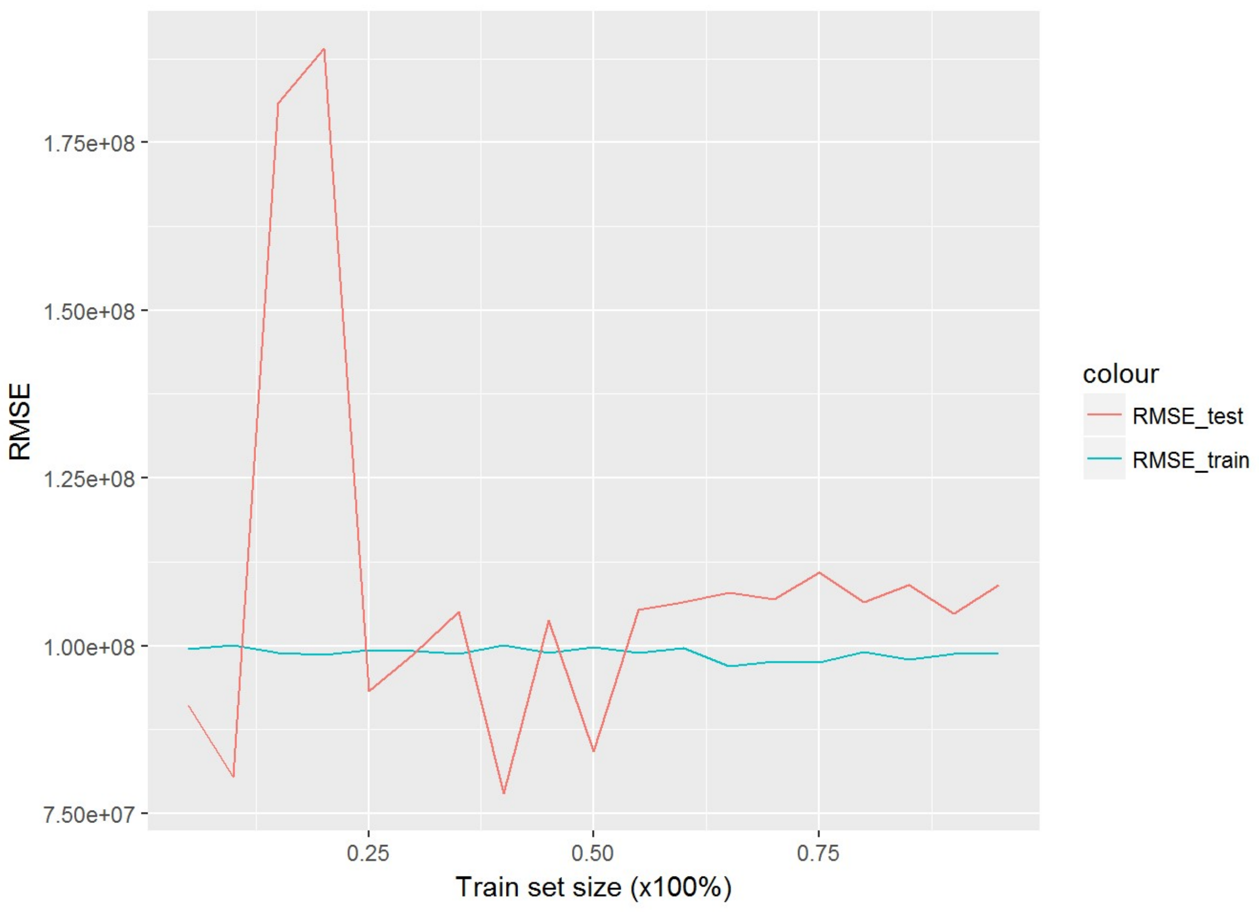
Q: List all the numeric variables you used.

A: I selected “imdbVotes”, “tomatoUserReviews”, “Budget”, and “Runtime”. Practically, their relations are not linear, so both RMSE_train and RMSE_test are high. When training set becomes larger and larger, the RMSE_test converges to the RMSE_train.

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)


for (i in 1:dim(df)[1]) {
  if (df$Budget[i] >= 3000000) {
    df$budget_greater_than_3M[i] = 1
  }
  else df$budget_greater_than_3M[i] = 0
}
#df$budget_greater_than_3M =factor(df$budget_greater_than_3M, levels=c("No", "Y
es"), ordered=TRUE)

RMSE_train =c()
RMSE_test = c()
for (i in 1:19) {
  for (time in (1:10)) {
    data = random_sample(df,i*0.05)
    train = data[[1]]
    test = data[[2]]
    M2 = lm(Gross~sqrt(Budget^5)+I(Budget^2)+sqrt(Budget^(3))+Budget+sqrt(Budge
t)+I(imdbVotes^2)+imdbVotes+I(Runtime^3)+I(Runtime^2)+Runtime+budget_greater_th
an_3M+I(imdbRating^2)+imdbRating+I(Year^2)+Year, train)
    theta=coef(M2)
    Y = test$Gross
    Y_train = predict(M2, train)
    Y_test = predict(M2, test)
    RMSE_train[time] = sqrt(sum((train$Gross-Y_train)^2)/length(Y_train))
    RMSE_test[time] = sqrt(sum((test$Gross-Y_test)^2)/length(Y_test))
    }
  RMSE_train[i] = mean(RMSE_train)
  RMSE_test[i] = mean(RMSE_test)
}

vis = data.frame('train_size'=seq(0.05, 0.95, 0.05),'RMSE_train'=RMSE_train, 'R
MSE_test'=RMSE_test)

ggplot(vis, aes(train_size)) +
  geom_line(aes(y = RMSE_train, colour = "RMSE_train")) +
  geom_line(aes(y = RMSE_test, colour = "RMSE_test")) +
  xlab('Train set size (x100%)') + ylab('RMSE')
```



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

A: I applied power transformation for the “Budget”, “imdbVotes”, “Runtime”, “imdbRating”, and “Year”. The reason is the gross is more or less related to these variables. By taking the power transformation to these varibales, the RMSE_train can obviously reduce. But if the power level is too high, the RMSE_test will increase due to overfit. In addition, I add an category variable “budget_greater_than_3M”.

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
# Split each genre to several items
genre=c()
n=1
for (i in 1:nrow(df)) {
  temp = unlist(strsplit(df$Genre[i],', '))
  for (j in 1:length(temp)){
    genre[n]=temp[j]
    n=n+1
  }
}
index=duplicated(genre) # remove repeated types
genre=genre[!index]
genre=genre[genre!='N/A'] # remove 'N/A' type

library(hash)
```

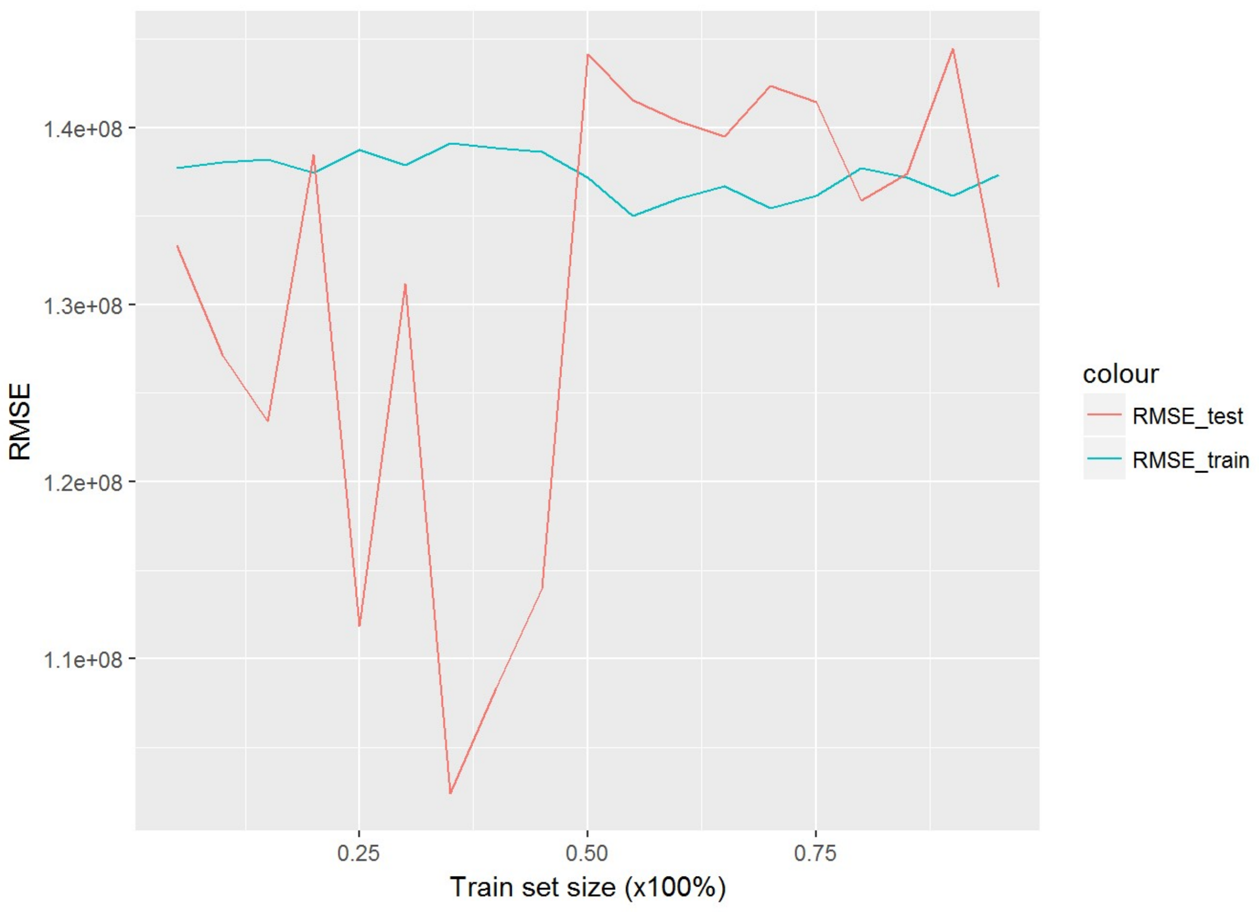
```
## hash-2.2.6 provided by Decision Patterns
```

```r
genre_hash = hash(keys=genre, values=1:length(genre)) # a dictionary for all ty
pes

for (i in 1:nrow(df)) {
  temp_vector = vector(length=length(genre), mode='numeric') # vector [0,0,
0...]
  if (df$Genre[i]!='N/A') {
    temp = unlist(strsplit(df$Genre[i],', '))
    for (j in 1:length(temp)) {
        temp_vector[values(genre_hash,temp[j])] = 1
    }
  }
  df$genre_vector[i]=paste(temp_vector,collapse='')
}
df$genre_vector = unlist(df$genre_vector)

# Switch genre indicator format from '0000...' to c(0,0,0,0...), and output gen
re
reverse_genre_hash = hash(keys=values(genre_hash), values=keys(genre_hash))

vector_to_genre = function(str) {
  genre = c()
  temp = as.numeric(unlist(strsplit(str,split='')))
  for (i in 1:length(temp)) {
    if (temp[i]==1) genre = c(genre,values(reverse_genre_hash, as.character
(i)))
  }
  genre = paste(genre,collapse=', ')
  return (genre)
}

standard_genre=c()
for (i in 1:nrow(df)) {
  standard_genre[i] = vector_to_genre(df$genre_vector[i])
}
df$standard_genre = standard_genre

for (i in 1:nrow(df)) {
  if (df$genre_vector[i] =='00000000000000000000000') df$standard_genre[i]='N/
A'
}

# Create multiple columns for each genre
for (i in 1:length(keys(genre_hash))) {
  df[,keys(genre_hash)[i]]=0
}

for (i in 1:nrow(df)) {
  temp = unlist(strsplit(df$standard_genre[i],split=', '))
  for (j in 1:length(temp)) {
    df[i,temp[j]]=1
  }
}
```

```
# TODO: Convert Awards to 2 numeric columns: wins and nominations
wins = c()
nominations = c()
for (i in 1:nrow(df)) {
  temp_wins=0
  temp_nominations=0
  if (df$Awards[i] !='N/A') {
    temp = unlist(strsplit(df$Awards[i],' '))
    j=1
    for (j in 1:length(temp)) {
      if (temp[j] == 'wins' | temp[j] == 'win' | temp[j] =='wins.') temp_wins
= temp_wins+as.numeric(temp[j-1])
      if (temp[j] == 'Won') temp_wins = temp_wins+as.numeric(temp[j+1])
      if (temp[j] == 'nominations.' | temp[j] == 'nomination.') temp_nomination
s = temp_nominations+as.numeric(temp[j-1])
      if (temp[j] == 'for') temp_nominations = temp_nominations+as.numeric(temp
[j+1])
    }
  }
  wins = c(wins,temp_wins)
  nominations = c(nominations, temp_nominations)
}
df$wins = wins
df$nominations = nominations
```

```r
# Find director names in top 100 gross movies
gross_top = df[order(df$Gross, decreasing = 1),][1:100,] # Sort movies based o
n the gross and find top 200
top_director=c()
n=1
for (i in 1:nrow(gross_top)) {
  temp = unlist(strsplit(gross_top$Director[i],', '))
  for (j in 1:length(temp)){
    top_director[n]=temp[j]
    n=n+1
  }
}
index=duplicated(top_director) # remove repeated types
top_director=top_director[!index]

# Check all movies if each one has director name in the top_director array and
add such feature.
# Add director name features and initialize them as 0
for (each in top_director) {
  df[, each] = 0
}

for (i in 1:nrow(df)) {
  temp = unlist(strsplit(df$Director[i],', '))
  for (j in 1:length(temp)) {
    if (temp[j] %in% top_director) df[i,temp[j]]=1
  }
}

# Find actor names from top 100 gross movies
top_actor=c()
n=1
for (i in 1:nrow(gross_top)) {
  temp = unlist(strsplit(gross_top$Actors[i],', '))
  for (j in 1:2){
    top_actor[n]=temp[j] # find first j actors in eaach movie, here I only fin
d the first two (j=2)
    n=n+1
  }
}
index=duplicated(top_actor) # remove repeated types
top_actor=top_actor[!index]

for (each in top_actor) {
  df[, each] = 0
}

for (i in 1:nrow(df)) {
  temp = unlist(strsplit(df$Actors[i],', '))
  for (j in 1:length(temp)) {
    if (temp[j] %in% top_actor) df[i,temp[j]]=1
  }
}

# Check if it comes from a famours top 30 director
df$famours_director = 0
for (i in 59:89) {# director column starts from 59
  df$famours_director = df$famours_director|df[i]
}

# Check it if comes from a famours top 50 actor
df$famours_actor = 0
for (i in 141:191) {# actor column starts from 191
  df$famours_actor = df$famours_actor|df[i]
}
```

```
# TODO: Build & evaluate model 3 (converted non-numeric variables only)
# Regression
RMSE_train =c()
RMSE_test = c()
for (i in 1:19) {
  for (time in (1:10)) {
    data = random_sample(df,i*0.05)
    train = data[[1]]
    test = data[[2]]
    modelData = data.frame(train['Gross'],train['wins'], train['nominations'],
train['budget_greater_than_3M'], train[,34:35], train[,38:39], train[,41:43], t
rain[45], train[50], train['famours_director'], train['famours_actor']) # 34Act
ion, 35Advanture, 38Comedy, 39Crime, 41Drama, 42Family, 43Fantasy, 45Horror, 50
Romance
    M3=lm(Gross ~ ., data=modelData)
    testData = data.frame(test['Gross'],test['wins'], test['nominations'],test
['budget_greater_than_3M'], test[,34:35], test[,38:39], test[,41:43], test[4
5], test[50], test['famours_director'], test['famours_actor'])
    Y_test = predict(M3,testData)
    RMSE_train[time] = sqrt(mean(residuals(M3)^2))
    RMSE_test[time] = sqrt(sum((test$Gross-Y_test)^2)/length(Y_test))
  }
  RMSE_train[i] = mean(RMSE_train)
  RMSE_test[i] = mean(RMSE_test)
}

vis3 = data.frame('train_size'=seq(0.05, 0.95, 0.05),'RMSE_train'=RMSE_train,
'RMSE_test'=RMSE_test)

ggplot(vis3, aes(train_size)) +
  geom_line(aes(y = RMSE_train, colour = "RMSE_train")) +
  geom_line(aes(y = RMSE_test, colour = "RMSE_test")) +
  xlab('Train set size (x100%)') + ylab('RMSE')
```



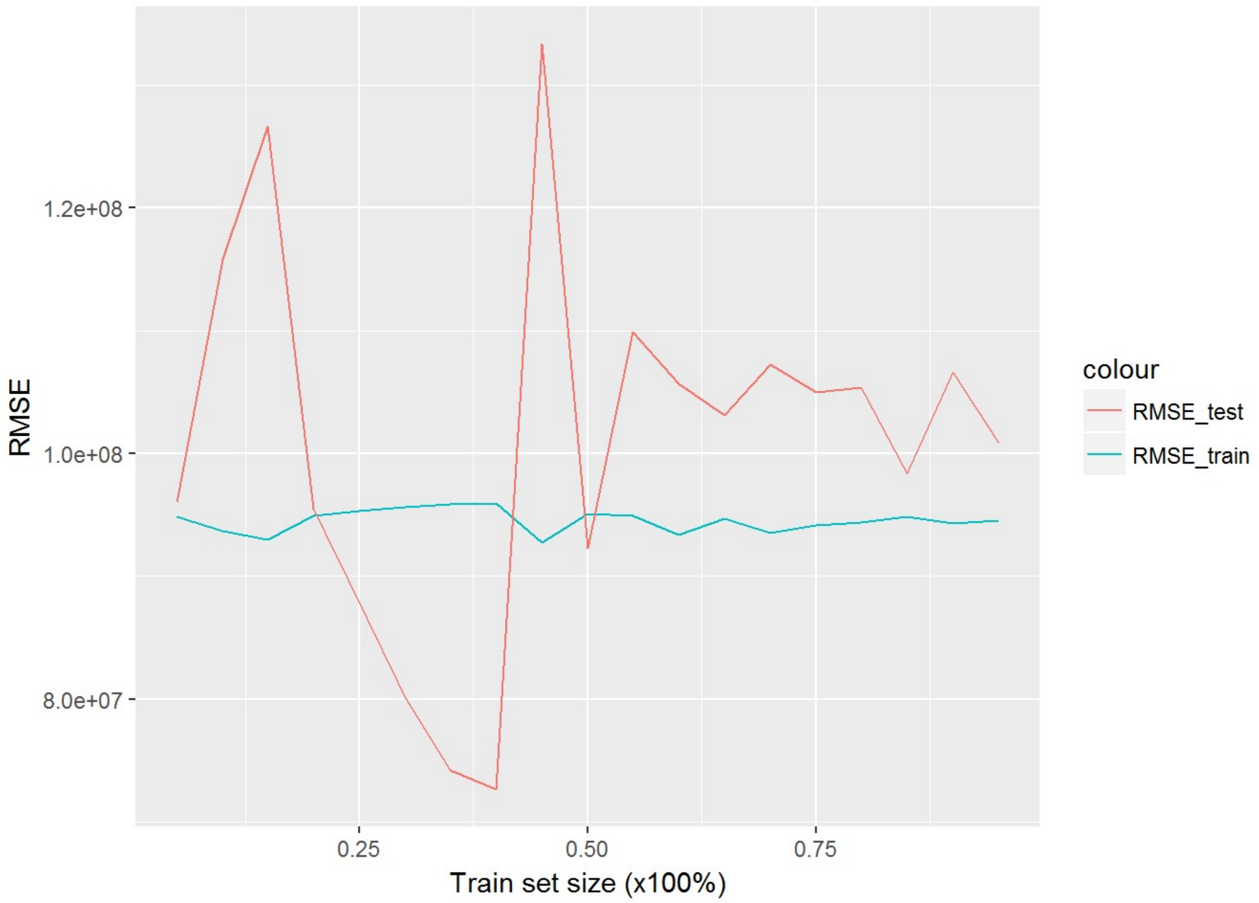
Q: Explain which categorical variables you used, and how you encoded them into features.

A: The categorical variables I use includes “budget_greater_than_3M”, “Action”, “Advanture”, “Comedy”, “Crime”, “Drama”, “Family”“,”Fantasy“,”Horror“,”Romance“,”famours_director“,”famours_actor“. To get the detailed genre information, I extracted all genre information from each movie and delete the duplicated genre to get the non-repeated genre array. By settting a dictionary of genre and comparing each movies’ genre to this dictionary, I got the genre_vector for each movie, but these vectors are in a single column. I decomposed the vector to fill the binary value to single genre column. To get the famours diectors binary value, I sort the df based on gross first and pickup top 100 high gross movies and find their directors. Some movies may have 2 directors. I removed duplicated ones and consider the top 30 directors (this number can be changed) in this list. The reason I select top 100 gross movie is because I have wider range to the tune the number of top director, otherwise I can only select top ~30 movies. By comparing the director from each movie in df with the top 30 directors and run some logistic operations, I can find if a movie is directed by the one in top 30 director list. Same thing can be applied to the famours actors binaray value.

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**.

```r
# TODO: Build & evaluate model 4 (numeric & converted non-numeric variables)
#df_ = subset(df, wins!=0 & nominations!=0)

RMSE_train =c()
RMSE_test = c()
for (i in 1:19) {
  for (time in (1:10)) {
    data = random_sample(df,i*0.05)
    train = data[[1]]
    test = data[[2]]
    modelData = data.frame(train['Gross'],
                           sqrt(train['Budget']^5), train['Budget']^2, sqrt(tra
in['Budget']^3), train['Budget'], sqrt(train['Budget']),
                           train['imdbVotes']^2, train['imdbVotes'],
                           train['tomatoUserReviews']^2, train['tomatoUserRevie
ws'],
                           train['budget_greater_than_3M'],
                           train[,34:35], train[,38:39], train[,41:43], train[4
5], train[50],
                           train['famours_director'], train['famours_actor'])
    testData = data.frame(test['Gross'],
                          sqrt(test['Budget']^5), test['Budget']^2, sqrt(test
['Budget']^3), test['Budget'], sqrt(test['Budget']),
                          test['imdbVotes']^2, test['imdbVotes'],
                          test['tomatoUserReviews']^2, test['tomatoUserReview
s'],
                          test['budget_greater_than_3M'],
                          test[,34:35], test[,38:39], test[,41:43], test[45], t
est[50],
                          test['famours_director'], test['famours_actor'])
    M4=lm(Gross ~ ., data=modelData)
    Y_train = predict(M4, modelData)
    Y_test = predict(M4,testData)
    RMSE_train[time] = sqrt(sum((train$Gross-Y_train)^2)/length(Y_train))
    RMSE_test[time] = sqrt(sum((test$Gross-Y_test)^2)/length(Y_test))
  }
  RMSE_train[i] = mean(RMSE_train)
  RMSE_test[i] = mean(RMSE_test)
}

vis4 = data.frame('train_size'=seq(0.05, 0.95, 0.05),'RMSE_train'=RMSE_train,
'RMSE_test'=RMSE_test)

ggplot(vis4, aes(train_size)) +
  geom_line(aes(y = RMSE_train, colour = "RMSE_train")) +
  geom_line(aes(y = RMSE_test, colour = "RMSE_test")) +
  xlab('Train set size (x100%)') + ylab('RMSE')
```

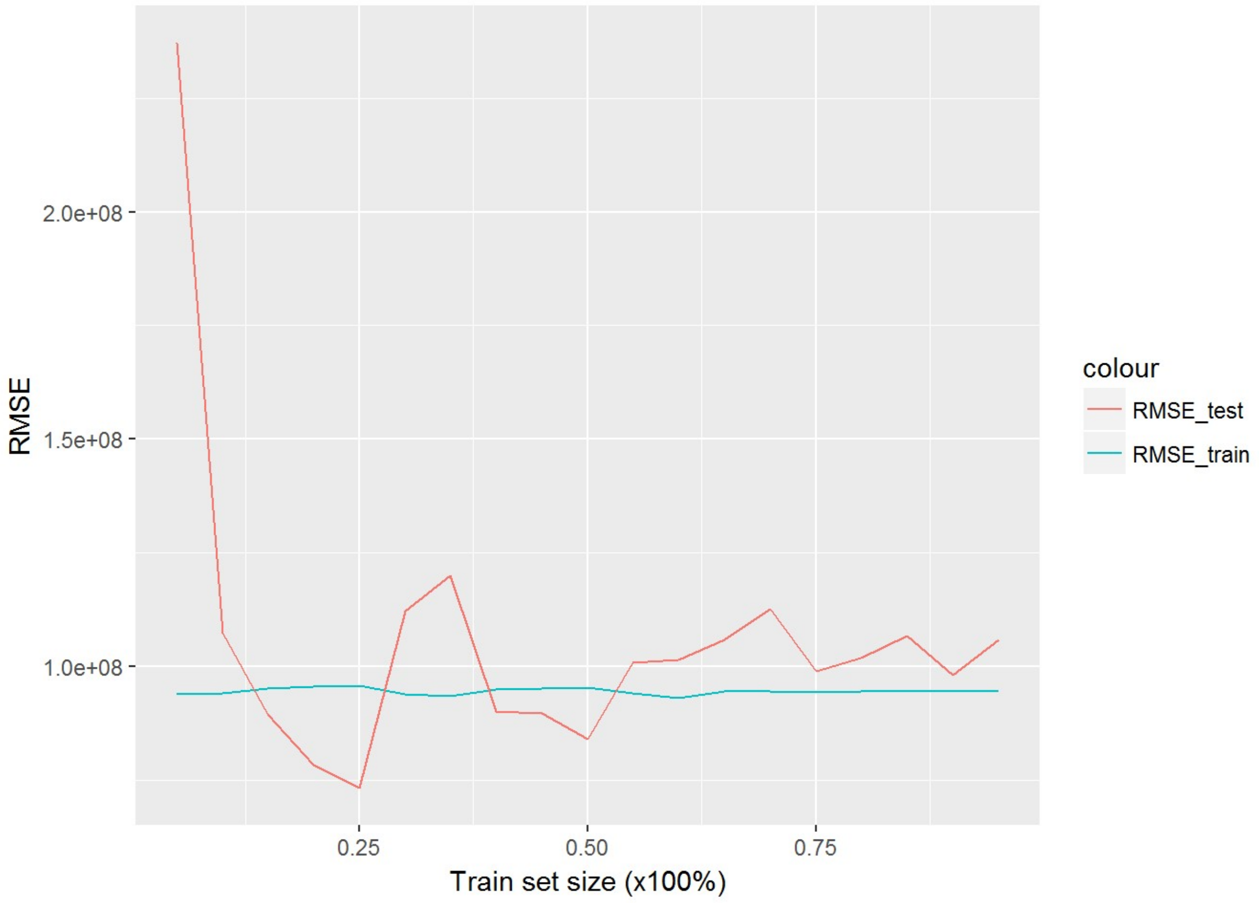


5. Additional features

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`).

```
# Find Release Month
df$famours_month = FALSE
for (i in 1:nrow(df)) {
  release_month = unlist(strsplit(as.character(df$Released[i]),split='-'))[2]
  if (as.numeric(release_month)==6 | as.numeric(release_month)==7 | as.numeric
(release_month)==12) {
    df$famours_month[i] = TRUE
  }
}
```

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

RMSE_train =c()
RMSE_test = c()
for (i in 1:19) {
  for (time in (1:10)) {
    data = random_sample(df,i*0.05)
    train = data[[1]]
    test = data[[2]]
    modelData = data.frame(train['Gross'],
                           sqrt(train['Budget']^5), train['Budget']^2, sqrt(tra
in['Budget']^3), train['Budget'], sqrt(train['Budget']),
                           train['imdbVotes']^2, train['imdbVotes'],
                           train['tomatoUserReviews']^2, train['tomatoUserRevie
ws'],
                           #train['budget_greater_than_3M'],
                           train[,34:35], train[,38:39], train[,41:43], train[4
5], train[50],
                           train[38]*train['budget_greater_than_3M'],
                           train['famours_director'], train['famours_actor'],
                           train['famours_month'])
    testData = data.frame(test['Gross'],
                          sqrt(test['Budget']^5), test['Budget']^2, sqrt(test
['Budget']^3), test['Budget'], sqrt(test['Budget']),
                          test['imdbVotes']^2, test['imdbVotes'],
                          test['tomatoUserReviews']^2, test['tomatoUserReview
s'],
                          #test['budget_greater_than_3M'],
                          test[,34:35], test[,38:39], test[,41:43], test[45], t
est[50],
                          test[38]*test['budget_greater_than_3M'],
                          test['famours_director'], test['famours_actor'],
                          test['famours_month'])
    M5=lm(Gross ~ ., data=modelData)
    Y_train = predict(M5, modelData)
    Y_test = predict(M5,testData)
    RMSE_train[time] = sqrt(sum((train$Gross-Y_train)^2)/length(Y_train))
    RMSE_test[time] = sqrt(sum((test$Gross-Y_test)^2)/length(Y_test))
  }
  RMSE_train[i] = mean(RMSE_train)
  RMSE_test[i] = mean(RMSE_test)
}

vis5 = data.frame('train_size'=seq(0.05, 0.95, 0.05),'RMSE_train'=RMSE_train,
'RMSE_test'=RMSE_test)

ggplot(vis5, aes(train_size)) +
  geom_line(aes(y = RMSE_train, colour = "RMSE_train")) +
  geom_line(aes(y = RMSE_test, colour = "RMSE_test")) +
  xlab('Train set size (x100%)') + ylab('RMSE')
```



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

A: I add the “famours_month” feature, which is 1 when a movie is released during summer (June, July) or Christmas (December). These two periods will affect gross obviously and high-gross movies tend to release during these periods. Another feature is [‘Comedy’]*[‘budget_greater_than_3M’] since these two varible have more obvious interaction.