Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics - Fall 2017

Due: Sunday, November 26, 2017 at 11:59 PM UTC-12:00 on T-Square

Yin Yuan (yyuan321)

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, and a PDF export of it (as **pr2.pdf**) which should include the outputs and plots as well.

*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(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(stringr)
```

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

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_movie_gross = df_movie[!is.na(df_movie$Gross), ]
df_movie_gross = filter(df_movie, !is.na(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_movie_gross_release = df_movie_gross[df_movie_gross$Year >= 2000, ]
df_movie_gross_release = filter(df_movie_gross, 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_movie_gross_release_match = df_movie_gross_release[abs(as.numeric(substring(df_movie_gross_release
df_movie_gross_release_match = filter(df_movie_gross_release, abs(as.numeric(substring(df_movie_gross_r
```

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_gross_release_match$Domestic_Gross <- NULL
#df_movie_gross_release_match = within(df_movie_gross_release_match, rm("Domestic_Gross"))
#df_movie = df_movie[, !(colnames(df_movie) %in% c("Domestic_Gross"))]
```

6. Process `Runtime` column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
df_movie_gross_release_match$Runtime =
  sapply(df_movie_gross_release_match$Runtime, function(x) {
    #grep return the index value of each matched pattern
    #grepl returns a logical output for each indices in the original vector
    if (grepl('h', x)){
      as.numeric(strsplit(x, ' ')[[1]][1]) * 60 + as.numeric(strsplit(x, ' ')[[1]][3])
    }
    if (grepl('min', x)){
      as.numeric(strsplit(x, ' ')[[1]][1])
    }
    else 0
    })
```

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
dim(df_movie_gross_release_match)
```

```
## [1] 3216   38
```

```
colnames(df_movie_gross_release_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, you will compute the training and test Root Mean Squared Error (RMSE) at different training set sizes.

First, randomly sample 10-20% of the preprocessed dataset and keep that aside as the **test set**. Do not use these rows for training! The remainder of the preprocessed dataset is your **training data**.

Now use the following evaluation procedure for each model:

- Choose a suitable sequence of training set sizes, e.g. 10%, 20%, 30%, ..., 100% (10-20 different sizes should suffice). For each size, sample that many inputs from the training data, train your model, and compute the resulting training and test RMSE.
- Repeat your training and evaluation at least 10 times at each training set size, and average the RMSE results for stability.
- Generate a graph of the averaged train and test RMSE values as a function of the train set size (%), with optional error bars.

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

Tasks

Each of the following tasks is worth 20 points, for a total of 100 points for this project. 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 available *numeric* variables. You can choose to include all or a subset of them.

```
# TODO: Build & evaluate model 1 (numeric variables only)
set.seed(42)
# function to do linear regression and compute rmse
estimate <- function(training, testing) {
  # numeric_training_data = Filter(is.numeric, training)
  # numeric_testing_data = Filter(is.numeric, testing)

  model = lm(Gross~., data=training, na.action = na.omit)
  # training RMSE
  predicted_train = predict(model, training)
  predicted_train_resid = (training$Gross - predicted_train) ** 2
  predicted_train_rmse = sqrt(sum(predicted_train_resid) / dim(training)[1])

  # testing RMSE
  predicted_test = predict(model, testing)
  predicted_test_resid = (testing$Gross - predicted_test) ** 2
  predicted_test_rmse = sqrt(sum(predicted_test_resid) / dim(testing)[1])

  return(list(predicted_train_rmse, predicted_test_rmse))
}

selected_features = c("Gross", "Runtime", "imdbRating", "imdbVotes", "tomatoRating", "tomatoReviews", "
df_selected = subset(df_movie_gross_release_match, select = selected_features)
df_selected = df_selected[sample(nrow(df_selected)), ]

testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

sample_size = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

training_rmses = matrix(0, nrow=10, ncol=10)
testing_rmses = matrix(0, nrow=10, ncol=10)

# start evaluation
for (i in 1:10) {
  # shuffle training
  training1 = training1[sample(nrow(training1)), ]
  for (j in 1:10) {
    training_cur = training1[1:round(sample_size[j] * nrow(training1)), ]
    #print(dim(training_cur))
    #print(dim(testing1))
    res = estimate(training_cur, testing1)
    training_rmses[i, j] = res[[1]]
    testing_rmses[i, j] = res[[2]]
```

```
  }
}

# plot results
training_rmses = colMeans(training_rmses)
testing_rmses = colMeans(testing_rmses)

results = data.frame(cbind(sample_size, training_rmses, testing_rmses))
colnames(results) <- c("sample_size", "training_RMSEs", "testing_RMSEs")
print(min(testing_rmses))
```

```
## [1] 100769762
```

```
print(which.min(testing_rmses))
```

```
## [1] 9
```

```
ggplot(results, aes(sample_size)) +
  geom_line(aes(y=training_RMSEs, x=sample_size, colour="training_RMSEs")) +
  geom_line(aes(y=testing_RMSEs, x=sample_size, colour="testing_RMSEs")) +
  xlab("sample size") +
  ylab("RMSE") + labs(title="Task1: Training RMSEs VS Testing RMSEs")
```

Q: List the numeric variables you used.

A: "Gross", "Runtime", "imdbRating", "imdbVotes", "tomatoRating", "tomatoReviews", "tomatoFresh", "Budget"

Q: What is the best mean test RMSE value you observed, and at what training set size?

A: The best mean test RMSE 100769762, at 90% training set size

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)
set.seed(42)
selected_features = c("Gross", "Runtime", "imdbRating", "imdbVotes", "tomatoRating", "tomatoReviews", "
df_selected = subset(df_movie_gross_release_match, select = selected_features)
df_selected = df_selected[sample(nrow(df_selected)), ]

testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

# feature transformation
# log transform on budget
training1$LogBudget = log(training1$Budget)
testing1$LogBudget = log(testing1$Budget)
training1$LogimdbVotes = log(training1$imdbVotes)
testing1$LogimdbVotes = log(testing1$imdbVotes)

# binning on
training1$BintomatoRating = cut(training1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4
testing1$BintomatoRating = cut(testing1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4, 5
# training1$tomatoFresh = cut(training1$tomatoFresh, 10, include.lowest = TRUE, labels = c(1, 2, 3, 4,
# testing1$tomatoFresh = cut(testing1$tomatoFresh, 10, include.lowest = TRUE, labels = c(1, 2, 3, 4, 5,
# training1$tomatoFresh = cut(training1$tomatoFresh, 10, include.lowest = TRUE, labels = c(1, 2, 3, 4,
# testing1$tomatoFresh = cut(testing1$tomatoFresh, 10, include.lowest = TRUE, labels = c(1, 2, 3, 4, 5,

sample_size = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

training_rmses = matrix(0, nrow=10, ncol=10)
testing_rmses = matrix(0, nrow=10, ncol=10)

# start evaluation
for (i in 1:10) {
  # shuffle training
  training1 = training1[sample(nrow(training1)), ]
  for (j in 1:10) {
    training_cur = training1[1:round(sample_size[j] * nrow(training1)), ]
    # print(summary(training_cur$tomatoFresh))
    # print(summary(testing1$tomatoFresh))

    res = estimate(training_cur, testing1)
    training_rmses[i, j] = res[[1]]
    testing_rmses[i, j] = res[[2]]
  }
}
```

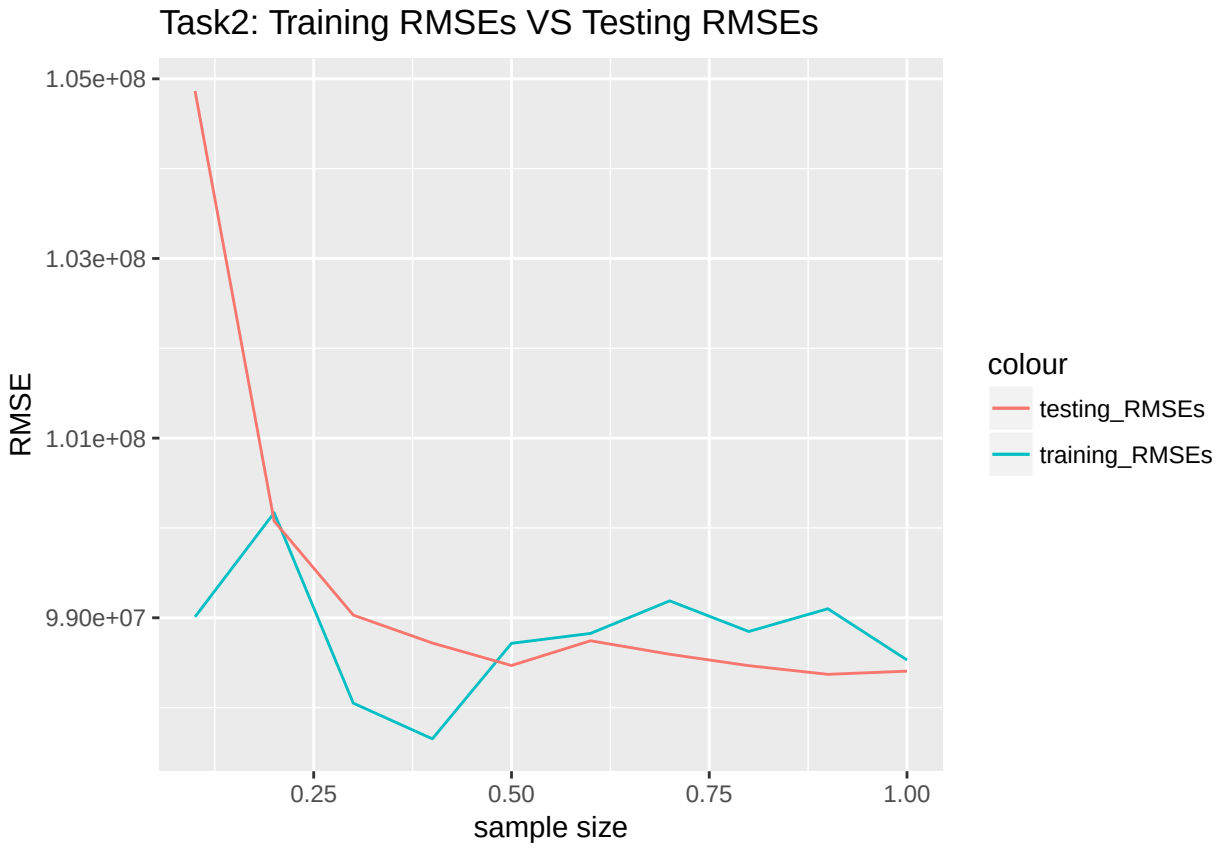
```r
# plot results
training_rmses = colMeans(training_rmses)
testing_rmses = colMeans(testing_rmses)

results = data.frame(cbind(sample_size, training_rmses, testing_rmses))
colnames(results) <- c("sample_size", "training_RMSEs", "testing_RMSEs")
print(min(testing_rmses))
```

```
## [1] 98370192
```

```r
print(which.min(testing_rmses))
```

```
## [1] 9
```

```r
ggplot(results, aes(sample_size)) +
  geom_line(aes(y=training_RMSEs, x=sample_size, colour="training_RMSEs")) +
  geom_line(aes(y=testing_RMSEs, x=sample_size, colour="testing_RMSEs")) +
  xlab("sample size") +
  ylab("RMSE") + labs(title="Task2: Training RMSEs VS Testing RMSEs")
```



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

A: I used log tranform for budget and imdbVotes and binning for tomatoRating. The scales of budget and imdbVotes are large compare with other variables, log transform can make the scales of budget and imdbvotes comparable with other variables. The binning on tomatoRating seems make the variable more discriminative, bining on other variables does not improve the performance.

Q: How did the RMSE change compared to Task 1?

A: The testing RMSE dropped from 100769762 to 98370192.

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)

set.seed(42)
selected_features = c("Gross", "Genre", "Awards")
df_selected = subset(df_movie_gross_release_match, select = selected_features)
df_selected = df_selected[sample(nrow(df_selected)), ]

# transform binary genre variable
genre_df = data.frame(df_selected$Genre)
all_genre = apply(genre_df, 2, paste, collapse = ', ')
genre_dic_temp = unique(unlist(strsplit(all_genre, ', ')))
genre_dic = genre_dic_temp[genre_dic_temp!='N/A']
binary_genre = data.frame(matrix(0, nrow = length(df_selected$Genre), ncol = length(genre_dic)))
colnames(binary_genre) = genre_dic

for (i in 1:length(df_selected$Genre)){
  gener_i = df_selected$Genre[i]
  if(i%%10000 == 0)  print(gener_i)

  for (j in 1:length(genre_dic)){
    if(grepl(genre_dic[j], gener_i) == TRUE){
      binary_genre[i,j] = 1
    }
  }
}

df_selected = cbind(df_selected, binary_genre)
df_selected$Genre = NULL

# transform binary director variable
# director_df = data.frame(df_selected$Director)
# all_director = apply(director_df, 2, paste, collapse = ', ')
# director_dic_temp = unique(unlist(strsplit(all_director, ', ')))
# director_dic = director_dic_temp[director_dic_temp!='N/A']
# binary_director = data.frame(matrix(0, nrow = length(df_selected$Director), ncol = length(director_di
# colnames(binary_director) = director_dic
#
# for (i in 1:length(df_selected$Director)){
#   director_i = df_selected$Director[i]
#   if(i%%10000 == 0)  print(director_i)
#
#   for (j in 1:length(director_dic)){
#     if(grepl(director_dic[j], director_i) == TRUE){
#       binary_director[i,j] = 1
```

```
#     }
#   }
# }
#
# df_selected = cbind(df_selected, binary_director)
# df_selected$Director = NULL

# process Awards
df_selected$wins = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\swin)"))
df_selected$wins[is.na(df_selected$wins)] = 0
# "Won {number}"
won = as.numeric(str_extract(string = df_selected$Awards, "(?<=Won\\s)\\d+"))
won[is.na(won)] = 0
df_selected$wins = df_selected$wins + won

# "{number} nominations"
df_selected$nominations = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\snomination)
df_selected$nominations[is.na(df_selected$nominations)] = 0
# "Nominated for {number}"
nominated = as.numeric(str_extract(string = df_selected$Awards, "(?<=Nominated for\\s)\\d+"))
nominated[is.na(nominated)] = 0
df_selected$nominations = df_selected$nominations + nominated
df_selected$Awards = NULL

testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

sample_size = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

training_rmses = matrix(0, nrow=10, ncol=10)
testing_rmses = matrix(0, nrow=10, ncol=10)

# start evaluation
for (i in 1:10) {
  # shuffle training
  training1 = training1[sample(nrow(training1)), ]
  for (j in 1:10) {
    # print((i-1)*10 + j)
    training_cur = training1[1:round(sample_size[j] * nrow(training1)), ]
    # print(summary(training_cur$tomatoFresh))
    # print(summary(testing1$tomatoFresh))

    res = estimate(training_cur, testing1)
    training_rmses[i, j] = res[[1]]
    testing_rmses[i, j] = res[[2]]
  }
}

# plot results
training_rmses = colMeans(training_rmses)
testing_rmses = colMeans(testing_rmses)
```

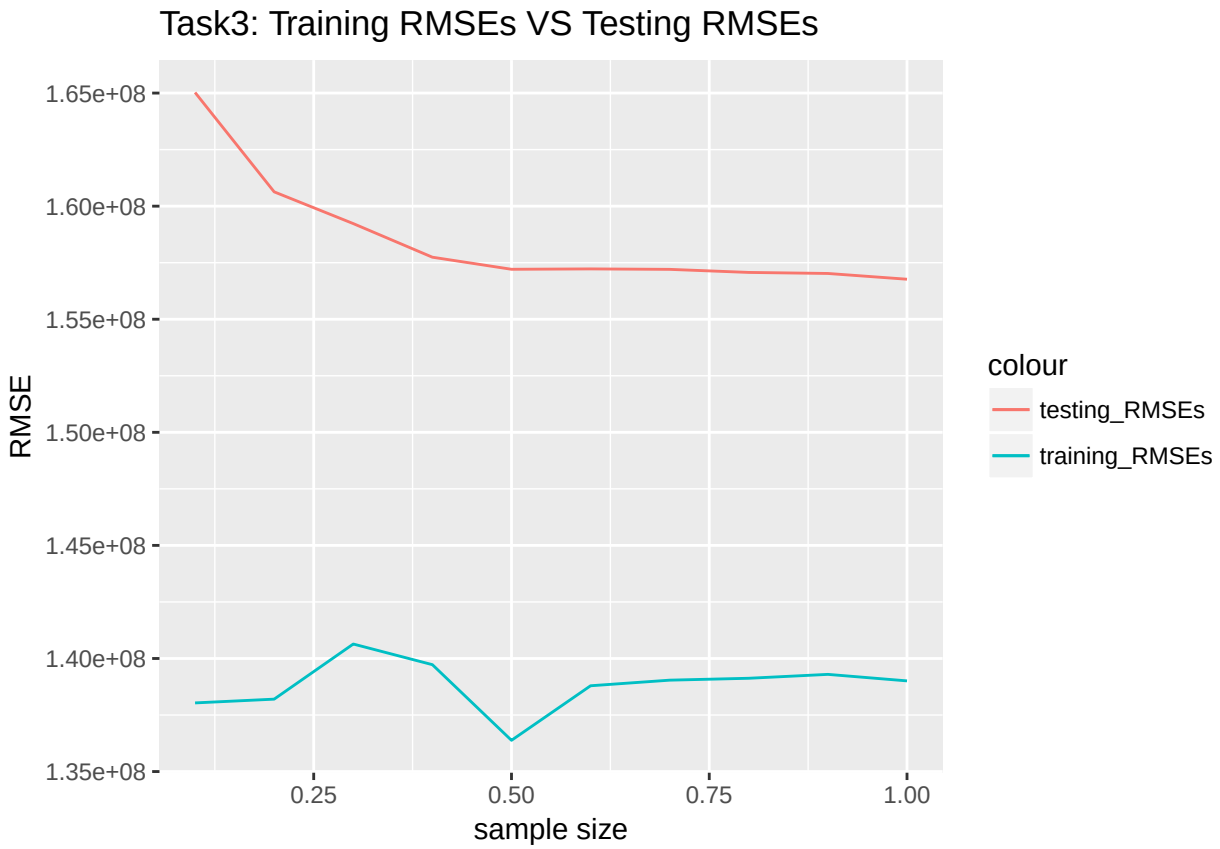
```
results = data.frame(cbind(sample_size, training_rmses, testing_rmses))
colnames(results) <- c("sample_size", "training_RMSEs", "testing_RMSEs")
print(min(testing_rmses))
```

```
## [1] 156769385
```

```
print(which.min(testing_rmses))
```

```
## [1] 10
```

```
ggplot(results, aes(sample_size)) +
  geom_line(aes(y=training_RMSEs, x=sample_size, colour="training_RMSEs")) +
  geom_line(aes(y=testing_RMSEs, x=sample_size, colour="testing_RMSEs")) +
  xlab("sample size") +
  ylab("RMSE") + labs(title="Task3: Training RMSEs VS Testing RMSEs")
```



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

A: I used Genre and Awards. I encoded Genre to binary number following the method in project 1. I converted Awards to wins and nominations.

Q: What is the best mean test RMSE value you observed, and at what training set size? How does this compare with Task 2?

A: The test RMSE increated from 98370192 to 156769385.

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)
set.seed(42)
selected_features = c("Gross", "Runtime", "imdbRating", "imdbVotes", "tomatoRating", "tomatoReviews", "
df_selected = subset(df_movie_gross_release_match, select = selected_features)
df_selected = df_selected[sample(nrow(df_selected)), ]

# convert non-numeric variables
# transform binary genre variable
genre_df = data.frame(df_selected$Genre)
all_genre = apply(genre_df, 2, paste, collapse = ', ')
genre_dic_temp = unique(unlist(strsplit(all_genre, ', ')))
genre_dic = genre_dic_temp[genre_dic_temp!='N/A']
binary_genre = data.frame(matrix(0, nrow = length(df_selected$Genre), ncol = length(genre_dic)))
colnames(binary_genre) = genre_dic

for (i in 1:length(df_selected$Genre)){
  gener_i = df_selected$Genre[i]
  if(i%%10000 == 0)  print(gener_i)

  for (j in 1:length(genre_dic)){
    if(grepl(genre_dic[j], gener_i) == TRUE){
      binary_genre[i,j] = 1
    }
  }
}

df_selected = cbind(df_selected, binary_genre)
df_selected$Genre = NULL

# transform binary director variable
# director_df = data.frame(df_selected$Director)
# all_director = apply(director_df, 2, paste, collapse = ', ')
# director_dic_temp = unique(unlist(strsplit(all_director, ', ')))
# director_dic = director_dic_temp[director_dic_temp!='N/A']
# binary_director = data.frame(matrix(0, nrow = length(df_selected$Director), ncol = length(director_di
# colnames(binary_director) = director_dic
#
# for (i in 1:length(df_selected$Director)){
#   director_i = df_selected$Director[i]
#   if(i%%10000 == 0)  print(director_i)
#
#   for (j in 1:length(director_dic)){
#     if(grepl(director_dic[j], director_i) == TRUE){
#       binary_director[i,j] = 1
#     }
#   }
# }
#
# df_selected = cbind(df_selected, binary_director)
```

```r
# df_selected$Director = NULL

# process Awards
# df_selected$wins = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\swin)"))
# df_selected$wins[is.na(df_selected$wins)] = 0
# # "Won {number}"
# won = as.numeric(str_extract(string = df_selected$Awards, "(?<=Won\\s)\\d+"))
# won[is.na(won)] = 0
# df_selected$wins = df_selected$wins + won
#
# # "{number} nominations"
# df_selected$nominations = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\snominatio
# df_selected$nominations[is.na(df_selected$nominations)] = 0
# # "Nominated for {number}"
# nominated = as.numeric(str_extract(string = df_selected$Awards, "(?<=Nominated for\\s)\\d+"))
# nominated[is.na(nominated)] = 0
# df_selected$nominations = df_selected$nominations + nominated
# df_selected$Awards = NULL

# numeric feature transform
testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

# feature transformation
# log transform on budget
training1$LogBudget = log(training1$Budget)
testing1$LogBudget = log(testing1$Budget)
training1$LogimdbVotes = log(training1$imdbVotes)
testing1$LogimdbVotes = log(testing1$imdbVotes)

# binning on
training1$BintomatoRating = cut(training1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4
testing1$BintomatoRating = cut(testing1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4, 5

# evaluation
testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

sample_size = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

training_rmses = matrix(0, nrow=10, ncol=10)
testing_rmses = matrix(0, nrow=10, ncol=10)

# start evaluation
for (i in 1:10) {
  # shuffle training
  training1 = training1[sample(nrow(training1)), ]
  for (j in 1:10) {
    training_cur = training1[1:round(sample_size[j] * nrow(training1)), ]
```

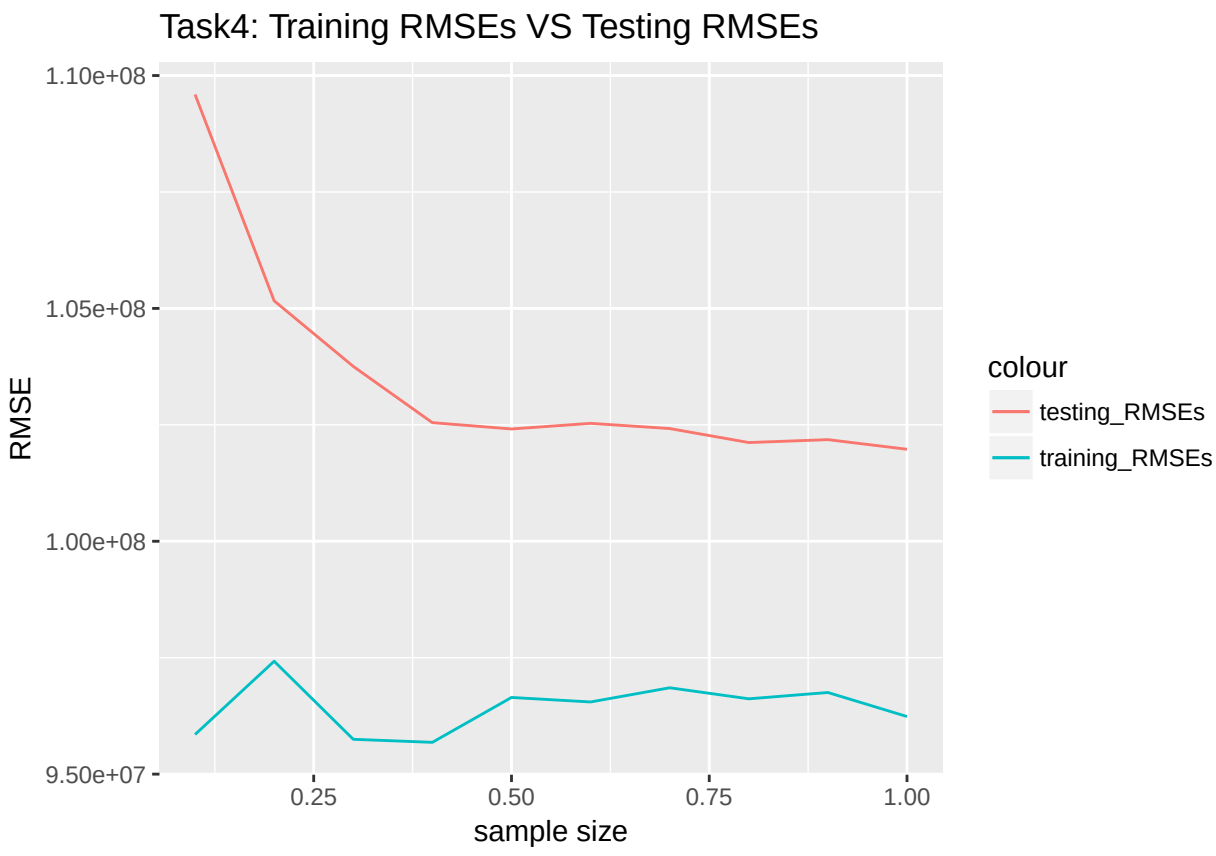
```
    # print(summary(training_cur$tomatoFresh))
    # print(summary(testing1$tomatoFresh))

    res = estimate(training_cur, testing1)
    training_rmses[i, j] = res[[1]]
    testing_rmses[i, j] = res[[2]]
  }
}

# plot results
training_rmses = colMeans(training_rmses)
testing_rmses = colMeans(testing_rmses)

results = data.frame(cbind(sample_size, training_rmses, testing_rmses))
colnames(results) <- c("sample_size", "training_RMSEs", "testing_RMSEs")
print(min(testing_rmses))
```

```
## [1] 101975483
```

```
print(which.min(testing_rmses))
```

```
## [1] 10
```

```
ggplot(results, aes(sample_size)) +
  geom_line(aes(y=training_RMSEs, x=sample_size, colour="training_RMSEs")) +
  geom_line(aes(y=testing_RMSEs, x=sample_size, colour="testing_RMSEs")) +
  xlab("sample size") +
  ylab("RMSE") + labs(title="Task4: Training RMSEs VS Testing RMSEs")
```



Q: Compare the observed RMSE with Tasks 2 & 3.

A: The test RMSE changed from 98370192 and 156769385 to 101975483.

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)
set.seed(42)
selected_features = c("Gross", "Runtime", "imdbRating", "imdbVotes", "tomatoRating", "tomatoReviews", "
df_selected = subset(df_movie_gross_release_match, select = selected_features)
df_selected = df_selected[sample(nrow(df_selected)), ]

# convert non-numeric variables
# transform binary genre variable
genre_df = data.frame(df_selected$Genre)
all_genre = apply(genre_df, 2, paste, collapse = ', ')
genre_dic_temp = unique(unlist(strsplit(all_genre, ', ')))
genre_dic = genre_dic_temp[genre_dic_temp!='N/A']
binary_genre = data.frame(matrix(0, nrow = length(df_selected$Genre), ncol = length(genre_dic)))
colnames(binary_genre) = genre_dic

for (i in 1:length(df_selected$Genre)){
  gener_i = df_selected$Genre[i]
  if(i%%10000 == 0)  print(gener_i)

  for (j in 1:length(genre_dic)){
    if(grepl(genre_dic[j], gener_i) == TRUE){
      binary_genre[i,j] = 1
    }
  }
}

df_selected = cbind(df_selected, binary_genre)
df_selected$Genre = NULL

# transform binary director variable
# director_df = data.frame(df_selected$Director)
# all_director = apply(director_df, 2, paste, collapse = ', ')
# director_dic_temp = unique(unlist(strsplit(all_director, ', ')))
# director_dic = director_dic_temp[director_dic_temp!='N/A']
# binary_director = data.frame(matrix(0, nrow = length(df_selected$Director), ncol = length(director_di
# colnames(binary_director) = director_dic
#
# for (i in 1:length(df_selected$Director)){
#   director_i = df_selected$Director[i]
#   if(i%%10000 == 0)  print(director_i)
#
#   for (j in 1:length(director_dic)){
#     if(grepl(director_dic[j], director_i) == TRUE){
#       binary_director[i,j] = 1
#     }
```

```
#   }
# }
#
# df_selected = cbind(df_selected, binary_director)
# df_selected$Director = NULL


# process Awards
df_selected$wins = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\swin)"))
df_selected$wins[is.na(df_selected$wins)] = 0
# "Won {number}"
won = as.numeric(str_extract(string = df_selected$Awards, "(?<=Won\\s)\\d+"))
won[is.na(won)] = 0
df_selected$wins = df_selected$wins + won

# "{number} nominations"
df_selected$nominations = as.numeric(str_extract(string = df_selected$Awards, "\\d+\\b(?=\\snomination)
df_selected$nominations[is.na(df_selected$nominations)] = 0
# "Nominated for {number}"
nominated = as.numeric(str_extract(string = df_selected$Awards, "(?<=Nominated for\\s)\\d+"))
nominated[is.na(nominated)] = 0
df_selected$nominations = df_selected$nominations + nominated
df_selected$Awards = NULL

# numeric feature transform
testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)

# feature transformation
# log transform on budget
training1$LogBudget = log(training1$Budget)
testing1$LogBudget = log(testing1$Budget)
training1$LogimdbVotes = log(training1$imdbVotes)
testing1$LogimdbVotes = log(testing1$imdbVotes)

# binning on
training1$BintomatoRating = cut(training1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4
testing1$BintomatoRating = cut(testing1$tomatoRating, 5, include.lowest = TRUE, labels = c(1, 2, 3, 4, 5

# add additional features
training1$is_budget_greater_than_3M <- ifelse(training1$Budget > 3000000, 1, 0)
testing1$is_budget_greater_than_3M <- ifelse(testing1$Budget > 3000000, 1, 0)
training1$is_imdbRating <- ifelse(training1$imdbRating > 6.5, 1, 0)
testing1$is_imdbRating <- ifelse(testing1$imdbRating > 6.5, 1, 0)

# evaluation
testing = df_selected[1 : (dim(df_selected)[1] * 0.2), ]
training = df_selected[(dim(df_selected)[1] * 0.2) : dim(df_selected)[1], ]
testing1 = na.omit(testing)
training1 = na.omit(training)
```

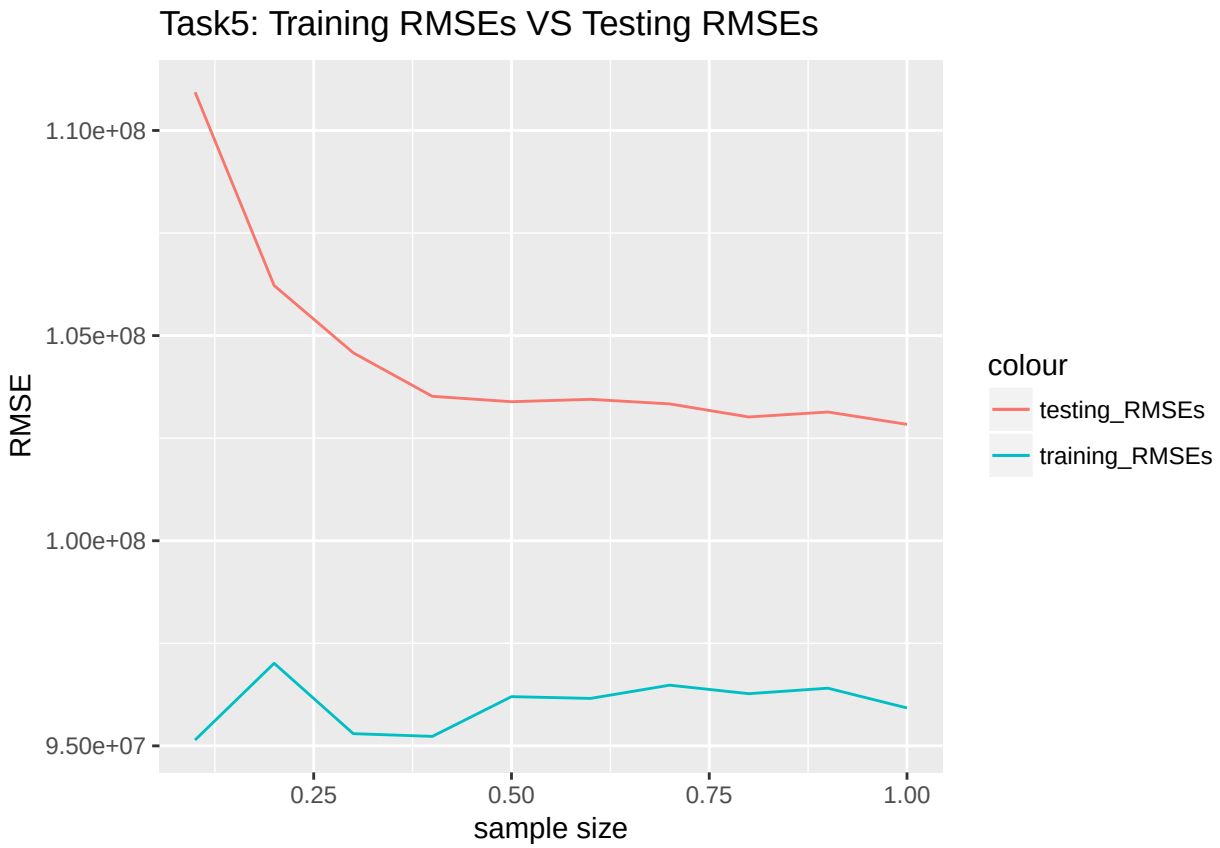
```r
sample_size = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

training_rmses = matrix(0, nrow=10, ncol=10)
testing_rmses = matrix(0, nrow=10, ncol=10)

# start evaluation
for (i in 1:10) {
  # shuffle training
  training1 = training1[sample(nrow(training1)), ]
  for (j in 1:10) {
    training_cur = training1[1:round(sample_size[j] * nrow(training1)), ]
    # print(summary(training_cur$tomatoFresh))
    # print(summary(testing1$tomatoFresh))

    res = estimate(training_cur, testing1)
    training_rmses[i, j] = res[[1]]
    testing_rmses[i, j] = res[[2]]
  }
}

# plot results
training_rmses = colMeans(training_rmses)
testing_rmses = colMeans(testing_rmses)

results = data.frame(cbind(sample_size, training_rmses, testing_rmses))
colnames(results) <- c("sample_size", "training_RMSEs", "testing_RMSEs")
print(min(testing_rmses))
```

```
## [1] 102836144
```

```r
print(which.min(testing_rmses))
```

```
## [1] 10
```

```r
ggplot(results, aes(sample_size)) +
  geom_line(aes(y=training_RMSEs, x=sample_size, colour="training_RMSEs")) +
  geom_line(aes(y=testing_RMSEs, x=sample_size, colour="testing_RMSEs")) +
  xlab("sample size") +
  ylab("RMSE") + labs(title="Task5: Training RMSEs VS Testing RMSEs")
```



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

A: I created is_genre_comedy and is_budget_greater_than_3M features. If a movie's budget is greater than 3M and imdbRating is larger than 6.5, it is more likely to be a good movie.

Q: Comment on the final RMSE values you obtained, and what you learned through the course of this project.

A: The final RMSE is 102836144. What I learned through the course of this project is that data processing, feature engineering and training samples are very important. Choosing the best features to fit the model is a nontrival task.