# Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics - Spring 2018

Due: Thursday, April 26, 2018 at 11:59 PM UTC-12:00 on T-Square

*Zheng Fu / zfu66@gatech.edu*

## 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:

```r
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:

```r
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
```

```r
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.

```r
library(ggplot2)
library(plyr)
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

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

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

```r
# TODO: Remove all rows from df that do not correspond to movies
df_original <- df
df2 <- df[df$Type == "movie",]
df <- df2
nrow(df)
```

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

### 2. Drop rows with missing `Gross` value

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

```r
# TODO: Remove rows with missing Gross value
no.missing.Gross <- df[!is.na(df$Gross),]
dim(no.missing.Gross)
```

```
## [1] 4558   39
```

## 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).

```r
# TODO: Exclude movies released prior to 2000
no.missing.Gross$Year <- as.numeric(no.missing.Gross$Year)
no.missing.Gross<-no.missing.Gross[no.missing.Gross$Year>=2000,]
dim(no.missing.Gross)
```

```
## [1] 3332   39
```

## 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.*

```r
# TODO: Remove mismatched rows
oldw <- getOption("warn")
options(warn = -1)
# Check number of NAs in each column
sapply(no.missing.Gross, function(x) sum(is.na(x)))
```

```
##              Title              Year             Rated          Released 
##                  0                 0                 0                41 
##            Runtime             Genre          Director            Writer 
##                  0                 0                 0                 0 
##             Actors              Plot          Language           Country 
##                  0                 0                 0                 0 
##             Awards            Poster         Metascore        imdbRating 
##                  0                 0                 0                43 
##          imdbVotes            imdbID              Type        tomatoMeter 
##                 43                 0                 0               396 
##        tomatoImage      tomatoRating     tomatoReviews       tomatoFresh 
##                  0               396               395               395 
##       tomatoRotten   tomatoConsensus   tomatoUserMeter  tomatoUserRating 
##                395                 0               195               193 
## tomatoUserReviews         tomatoURL               DVD         BoxOffice 
##                 89                 0               276                 0 
##         Production           Website          Response            Budget 
##                  0                 0                 0                 0 
##     Domestic_Gross             Gross              Date 
##                  0                 0                 0
```

```r
# Remove mismatched rows based on release year
row1 <- nrow(no.missing.Gross)
```

```
no.missing.Gross <- no.missing.Gross[!is.na(no.missing.Gross$Released),
    ]
released.date <- as.character(no.missing.Gross$Released)

released.year <- sapply(strsplit(released.date, "-"), function(x) {
    as.numeric(x[1])
})

mismatch.judgement <- (no.missing.Gross$Date == released.year) ||
    (no.missing.Gross$Year == released.year)
no.missing.Gross <- no.missing.Gross[mismatch.judgement, ]
row2 <- nrow(no.missing.Gross)
print(paste("Percentage of rows were removed: ", 100 * (row1 -
    row2)/row1, "%"))
```

```
## [1] "Percentage of rows were removed:  1.23049219687875 %"
```

## 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
no.missing.Gross$Domestic_Gross<-NULL
dim(no.missing.Gross)
```

```
## [1] 3291   38
```

## 6. Process Runtime column

```
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
Runtime <- no.missing.Gross$Runtime
Runtime <- gsub("N/A", "0", Runtime)
Runtime <- gsub(" min", "", Runtime)
no.missing.Gross$Runtime <- as.numeric(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 Convert Metascore
# to numeric, set 'N/A' to '0'
no.missing.Gross$Metascore <- gsub("N/A", "0", no.missing.Gross$Metascore)
no.missing.Gross$Metascore <- as.numeric(no.missing.Gross$Metascore)

# For 'Rated', set 'NOT RATED' and 'N/A' to UNRATED
no.missing.Gross$Rated <- gsub("N/A", "UNRATED", no.missing.Gross$Rated)
no.missing.Gross$Rated <- gsub("NOT RATED", "UNRATED", no.missing.Gross$Rated)

# Using column median to fill NA in numeric columns
for (i in c(16, 17, 20, 22, 23, 24, 25, 27, 28, 29)) {
    no.missing.Gross[, i][is.na(no.missing.Gross[, i])] <- median(no.missing.Gross[,
        i], na.rm = T)
}

# Check NA in each column
sapply(no.missing.Gross, function(x) sum(is.na(x)))
```

```
##              Title               Year              Rated           Released
##                  0                  0                  0                  0
##            Runtime              Genre           Director             Writer
##                  0                  0                  0                  0
##             Actors               Plot           Language            Country
##                  0                  0                  0                  0
##             Awards             Poster          Metascore         imdbRating
##                  0                  0                  0                  0
##          imdbVotes             imdbID               Type        tomatoMeter
##                  0                  0                  0                  0
##        tomatoImage       tomatoRating      tomatoReviews        tomatoFresh
##                  0                  0                  0                  0
##       tomatoRotten    tomatoConsensus     tomatoUserMeter   tomatoUserRating
##                  0                  0                  0                  0
## tomatoUserReviews          tomatoURL                DVD          BoxOffice
##                  0                  0                248                  0
##         Production            Website           Response             Budget
##                  0                  0                  0                  0
##              Gross               Date
##                  0                  0
```

***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
final.dataset <- no.missing.Gross
print("The dimensions of the final preprocessed dataset:")
```

```
## [1] "The dimensions of the final preprocessed dataset:"
```

```
dim(final.dataset)
```

```
## [1] 3291   38
```

```
print("The column names of the final preprocessed dataset:")
```

```
## [1] "The column names of the final preprocessed dataset:"
```

```
colnames(final.dataset)
```

```
##  [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 (%).

### 0. Create helper functions

In this section I created few helper functions that could be reused for evaluating models with a given set of features.

```r
# Function that returns Root Mean Squared Error
rmse <- function(error) {
    sqrt(mean(error^2))
}

# Function for calulating mean and sd by groups
summarySE <- function(data = NULL, measurevar, groupvars = NULL,
    na.rm = FALSE, conf.interval = 0.95, .drop = TRUE) {
    length2 <- function(x, na.rm = FALSE) {
        if (na.rm)
            sum(!is.na(x)) else length(x)
    }
    datac <- ddply(data, groupvars, .drop = .drop, .fun = function(xx,
        col) {
        c(N = length2(xx[[col]], na.rm = na.rm), mean = mean(xx[[col]],
            na.rm = na.rm), sd = sd(xx[[col]], na.rm = na.rm))
    }, measurevar)
    datac <- rename(datac, c(mean = measurevar))
    datac$se <- datac$sd/sqrt(datac$N)
    ciMult <- qt(conf.interval/2 + 0.5, datac$N - 1)
    datac$ci <- datac$se * ciMult
    return(datac)
}

# Model evaluation
model_evaluation <- function(data, modelName) {

    oldw <- getOption("warn")
    options(warn = -1)

    # Figure title
    title <- paste("Learning curve on model:", modelName)

    ## 80% of the data as training set
    split_size <- floor(0.8 * nrow(data))

    ## set the seed to make partition reproductible
    set.seed(2018)
    train_ind <- sample(seq_len(nrow(data)), size = split_size)
    train <- data[train_ind, ]
    test <- data[-train_ind, ]

    sampleSize_train_rmse <- c()
    sampleSize_test_rmse <- c()
    trained_models <- list()
    model.index <- 1

    for (i in seq(0.1, 1, 0.1)) {
        for (j in seq(1, 10, 1)) {

            # sample data
            sampleSize_train <- train[sample(nrow(train), nrow(train) *
                i), ]
            train.results <- lm(Gross ~ ., data = sampleSize_train)
            sampleSize_train_rmse <- c(sampleSize_train_rmse,
                rmse(train.results$residuals))
            prediction <- predict(train.results, newdata = test)
            sampleSize_test_rmse <- c(sampleSize_test_rmse, rmse(test$Gross -
                prediction))
            trained_models[[model.index]] <- train.results
        }
        model.index <- model.index + 1
    }

    train_all <- data.frame(rmse = sampleSize_train_rmse, type = rep("train",
        100), fraction = rep(seq(0.1, 1, 0.1), each = 10))
```

```r
    test_all <- data.frame(rmse = sampleSize_test_rmse, type = rep("test",
        100), fraction = rep(seq(0.1, 1, 0.1), each = 10))
    model_all <- rbind(train_all, test_all)
    model_summary <- summarySE(model_all, measurevar = "rmse",
        groupvars = c("type", "fraction"))

    p01 <- ggplot(model_summary, aes(x = fraction, y = rmse,
        colour = type)) + geom_errorbar(aes(ymin = rmse - se,
        ymax = rmse + se), width = 0.1) + geom_line() + geom_point()

    p01 <- p01 + geom_point(size = 1.5) + scale_x_continuous(name = "Sample Size Fraction",
        breaks = seq(0, 1, 0.1), limits = c(0, 1)) + scale_y_continuous(name = "Mean RMSE") +
        ggtitle(title) + theme_bw()

    list(figure = p01, model_summary = model_summary, trained_models = trained_models)

}
```

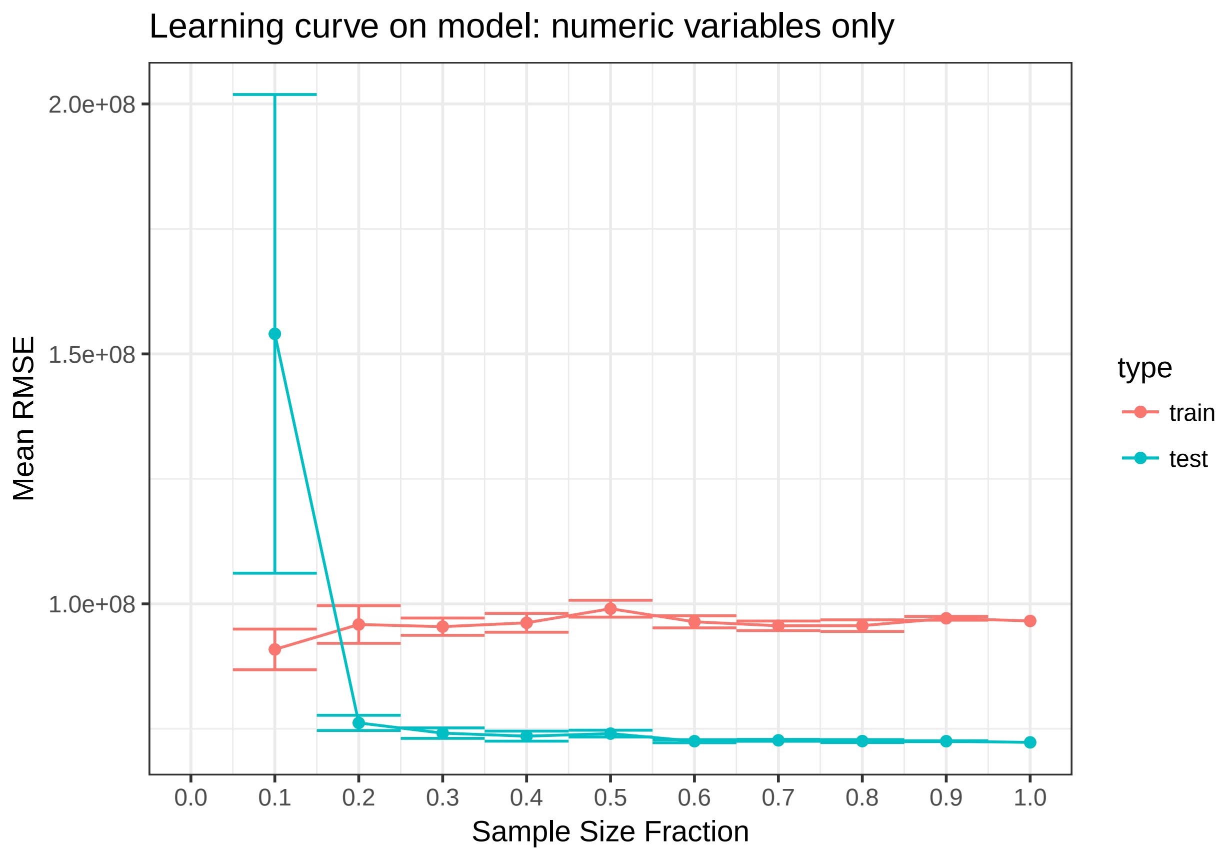
## 1. Numeric variables

Use Linear Regression to predict `Gross` based on available *numeric* variables. You can choose to include all or a subset of them.

```r
# TODO: Build & evaluate model 1 (numeric variables only)
feature.set1 <- final.dataset[, c(2, 5, 15, 16, 17, 22, 24, 25,
    27, 28, 29, 36, 37)]
sapply(feature.set1, class)
```

```
##               Year            Runtime          Metascore         imdbRating
##          "numeric"          "numeric"          "numeric"          "numeric"
##          imdbVotes       tomatoRating        tomatoFresh       tomatoRotten
##          "numeric"          "numeric"          "numeric"          "numeric"
##    tomatoUserMeter   tomatoUserRating tomatoUserReviews             Budget
##          "numeric"          "numeric"          "numeric"          "numeric"
##              Gross
##          "numeric"
```

```r
feature.set1.results <- model_evaluation(data = feature.set1,
    modelName = "numeric variables only")
print(feature.set1.results$figure)
```



```r
model.summary <- feature.set1.results$model_summary
print(model.summary[model.summary$type == "test", ])
```

```
##    type fraction  N      rmse           sd           se           ci
## 11 test      0.1 10 154010811 1.513865e+08 4.787260e+07 1.082954e+08
## 12 test      0.2 10  76195106 4.802906e+06 1.518812e+06 3.435792e+06
## 13 test      0.3 10  74143777 3.309240e+06 1.046474e+06 2.367288e+06
## 14 test      0.4 10  73540783 3.175103e+06 1.004056e+06 2.271332e+06
## 15 test      0.5 10  74053522 2.131416e+06 6.740128e+05 1.524723e+06
## 16 test      0.6 10  72535035 9.796370e+05 3.097884e+05 7.007901e+05
## 17 test      0.7 10  72708846 5.980267e+05 1.891126e+05 4.278025e+05
## 18 test      0.8 10  72556059 9.460709e+05 2.991739e+05 6.767783e+05
## 19 test      0.9 10  72537672 2.823731e+05 8.929422e+04 2.019976e+05
## 20 test      1.0 10  72295468 1.701896e-07 5.381866e-08 1.217463e-07
```

**Q**: List the numeric variables you used.

**A**: In this section I used the following numeric variables:
Year, Runtime, Metascore, imdbRating, imdbVotes, tomatoRating, tomatoFresh, tomatoRotten, tomatoUserMeter, tomatoUserRating, tomatoUserReviews and Budget.
To avoid collinearity issue, variables tomatoMeter and tomatoReviews were not included in the features list. Since tomatoMeter = 100 * tomotoFresh / (tomotoFresh + tomotoRotten), and tomatoReviews = tomotoFresh + tomotoRotten.

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

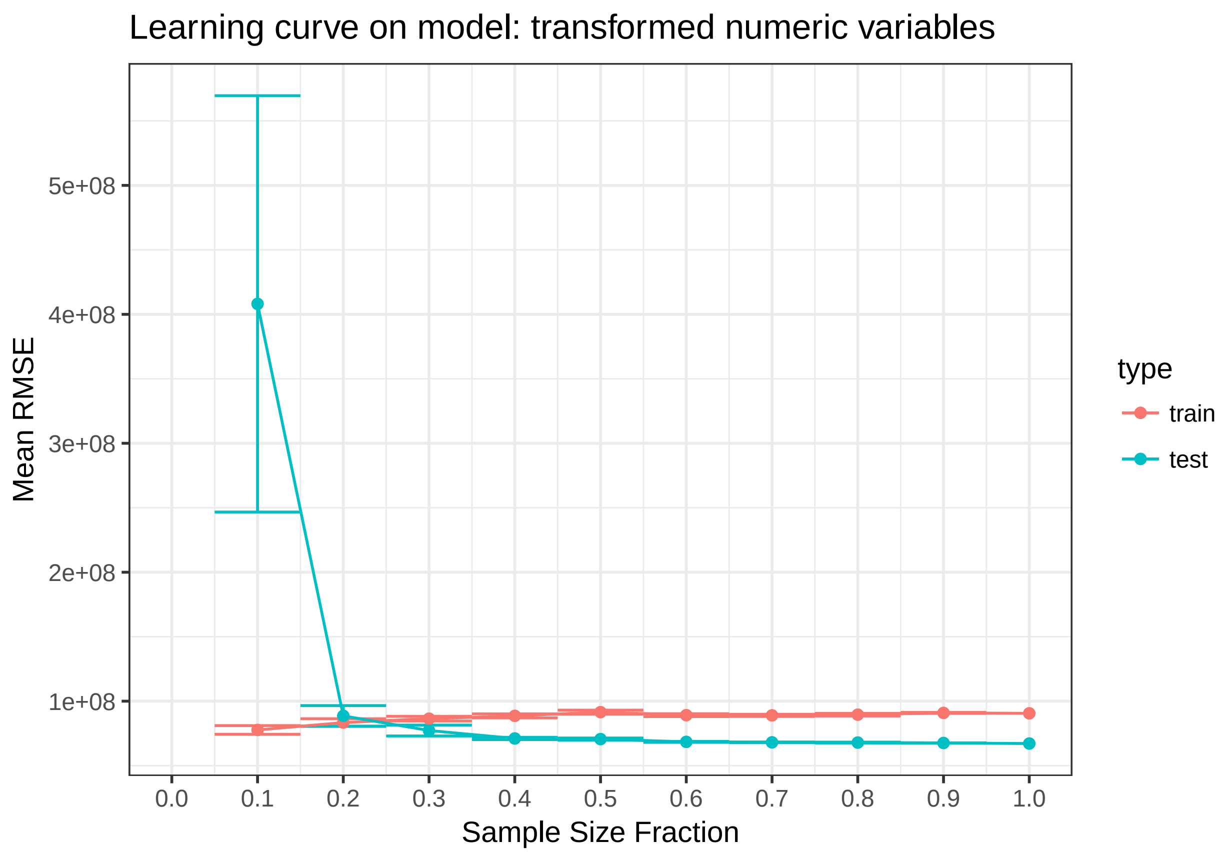
**A**: The best mean test RMSE value I observed is 72295468 at training set size = 100%.

## 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`).

```r
# TODO: Build & evaluate model 2 (transformed numeric
# variables only) Only keep numeric variables names
numeric.col.names <- colnames(feature.set1)
numeric.col.names <- numeric.col.names[-length(numeric.col.names)]

# power transformations
log.transform <- function(x) log10(x)
sqrt.transform <- function(x) x^(1/2)
cube.root.transform <- function(x) x^(1/3)

# function list
function.list <- list()
function.list[[1]] <- log.transform
function.list[[2]] <- sqrt.transform
function.list[[3]] <- cube.root.transform
function.names <- c(".log", ".sqrt", ".cub")

# Create power transformed features
transformed.features <- data.frame(matrix(NA, nrow = dim(feature.set1)[1],
    ncol = 0))
transformed.features.names <- c()
for (i in 1:3) {
    for (j in seq(1, length(numeric.col.names))) {
        new.col.name <- paste(numeric.col.names[j], function.names[i],
            sep = "")
        new.col <- function.list[[i]](feature.set1[, numeric.col.names[j]] +
            1e-05)
        transformed.features <- cbind(transformed.features, new.col)
        transformed.features.names <- c(transformed.features.names,
            new.col.name)
    }
}

colnames(transformed.features) <- transformed.features.names

# binning transformations budget more than 3M
is.budget.more.than.3M <- rep(0, dim(feature.set1)[1])
is.budget.more.than.3M[feature.set1$Budget > 3e+06] <- 1

# year later than 2007
is.year.later.than.2007 <- rep(0, dim(feature.set1)[1])
is.year.later.than.2007[feature.set1$Year > 2007] <- 1

# run time more than 90 mins
is.runtime.more.than.90 <- rep(0, dim(feature.set1)[1])
is.runtime.more.than.90[feature.set1$Runtime > 90] <- 1

# metascore more than 50
```

```
is.metascore.more.than.50 <- rep(0, dim(feature.set1)[1])
is.metascore.more.than.50[feature.set1$Metascore > 50] <- 1

binning.transformation <- data.frame(is.budget.more.than.3M = is.budget.more.than.3M,
    is.year.later.than.2007 = is.year.later.than.2007)
binning.transformation$is.runtime.more.than.90 <- is.runtime.more.than.90
binning.transformation$is.metascore.more.than.50 <- is.metascore.more.than.50

# Combine power transformations and binning transformations
feature.set2 <- cbind(feature.set1, transformed.features, binning.transformation)

# Modelling
options(warn = -1)
feature.set2.results <- model_evaluation(data = feature.set2,
    modelName = "transformed numeric variables")
print(feature.set2.results$figure)
```



```
model.summary.2 <- feature.set2.results$model_summary
print(model.summary.2[model.summary.2$type == "test", ])
```

```
##    type fraction  N      rmse           sd           se           ci
## 11 test      0.1 10 408116276 5.105239e+08 1.614418e+08 3.652068e+08
## 12 test      0.2 10  88612118 2.516547e+07 7.958020e+06 1.800229e+07
## 13 test      0.3 10  77235788 1.335761e+07 4.224048e+06 9.555460e+06
## 14 test      0.4 10  71091621 2.798326e+06 8.849084e+05 2.001802e+06
## 15 test      0.5 10  70578514 2.705707e+06 8.556196e+05 1.935546e+06
## 16 test      0.6 10  68442871 1.165312e+06 3.685040e+05 8.336140e+05
## 17 test      0.7 10  68063227 8.535512e+05 2.699166e+05 6.105937e+05
## 18 test      0.8 10  67892833 1.066193e+06 3.371597e+05 7.627083e+05
## 19 test      0.9 10  67570075 3.767915e+05 1.191519e+05 2.695404e+05
## 20 test      1.0 10  67146558 4.469933e-03 1.413517e-03 3.197597e-03
```

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

**A**: In this section I employed both power transforms and non-numeric transformations.
(1) Power transformation:
I used log10 transformation, square root transformation and cube root transformation. The reason why I chose them is from Project 1 we saw some numeric variables are skewed strongly to the right, and using these common transformations could improve the distribution of the data somewhat and make it closing to normal distribution.
(2) Non-numeric transformations:
I binarized numeric variables Year, Runtime, Metascore and Budget using following arbitary cutoffs:
If Year $>$ 2005 then 1 else 0;
If Runtime $>$ 90 then 1 else 0;
If Metascore $>$ 50 then 1 else 0;
If Budget $>$ 3 millions then 1 else 0.
My motivation is to check if newer, longer runtime, higher metascore and more budget movies will result in increasing gross revenue.

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

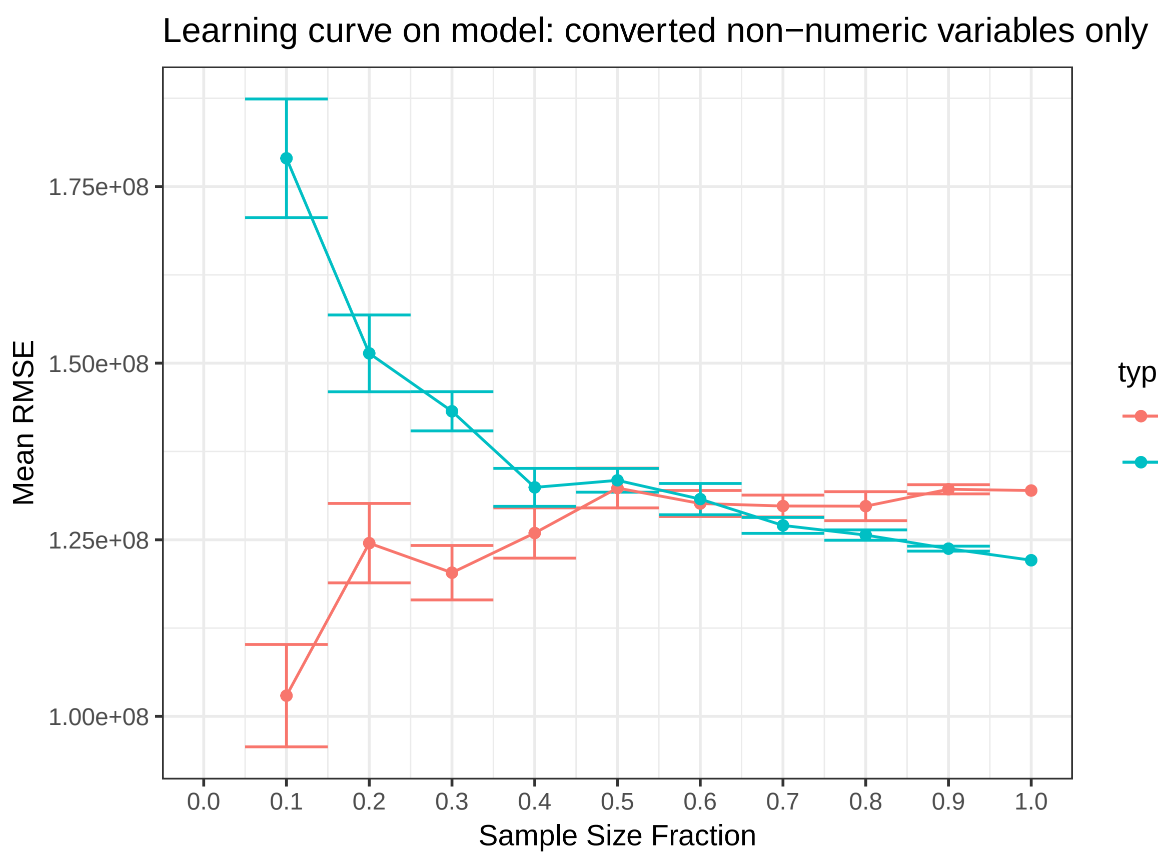
**A**: Comparing to Task1, the best mean test RMSE value was reduced from 72295468 to 67146558 at training set size = 100%.

## 3. Non-numeric variables

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

```r
# TODO: Build & evaluate model 3 (converted non-numeric
# variables only) Categorical variables
categorical.variables <- final.dataset[, c(3, 6, 7, 8, 9, 11,
    12, 13, 33, 37)]
# sapply(categorical.variables, class)
categorical.variables[is.na(categorical.variables)] <- "N/A"

# Convert Awards
awards.normination <- c()

for (i in seq_along(categorical.variables$Awards)) {
    if (categorical.variables$Awards[i] == "N/A") {
        awards.normination <- c(awards.normination, 0)
    } else {
        award.num <- gregexpr("[0-9]+", categorical.variables$Awards[i])
        total.awards.normination <- sum(as.numeric(unlist(regmatches(categorical.variables$Awards[i],
            award.num))))
        awards.normination <- c(awards.normination, total.awards.normination)
    }
}
```

```r
categorical.variables$awards.normination <- awards.normination

# Convert Rated
rated <- factor(tolower(categorical.variables$Rated))
ranks <- rank(-table(rated), ties.method = "first")
categorical.variables$rated.num <- ranks[as.character(rated)]

# Convert production
production <- categorical.variables$Production
production <- gsub("/", " ", production)
production <- gsub("First Independent", "FirstIndependent", production)
production <- gsub("First Run", "FirstRun", production)
production <- gsub("First Look", "FirstLook", production)
production <- gsub("Lions Gate", "Lionsgate", production)
production <- gsub("Metro-Goldwyn-Mayer", "MGM", production)
production <- gsub("Mirimax", "Miramax", production)
production.abbr <- sapply(strsplit(production, " "), function(x) {
    as.character(x[1])
})
production.abbr <- as.factor(tolower(production.abbr))
ranks <- rank(-table(production.abbr), ties.method = "first")
categorical.variables$production.num <- ranks[as.character(production.abbr)]

# one-hot encoding function
one.hot.encoding <- function(categorical.V) {
    t1 <- strsplit(as.character(categorical.V), ",\\s?")  # split the strings
    lvl <- unique(unlist(t1))  # get unique elements
    t2 <- lapply(t1, factor, levels = lvl)  # convert to factor
    results <- as.data.frame(t(sapply(t2, table)))
}

# top 10 features by encoding
top10.features <- function(categorical.V, feature.name) {
    corp <- Corpus(VectorSource(categorical.V))
    dtm <- DocumentTermMatrix(corp)
    df <- data.frame(as.matrix(dtm))
    sorted_df <- sort(colSums(df), decreasing = TRUE)[1:10]
    name_list <- names(sorted_df)
    sub_df <- df[, names(sorted_df)]
    names(sub_df) <- paste(feature.name, name_list, sep = "")
    sub_df
}

# Convert Writer, remove everything in the brackets
writer <- gsub("\\s*\\([^\\)]+\\)", "", tolower(as.character(categorical.variables$Writer)))
writer <- gsub(" ", "", writer)
writer <- gsub(",", " ", writer)  # reserve whole writer's name
writer.top10 <- top10.features(writer, "writer.")
categorical.variables <- cbind(categorical.variables, writer.top10)

# Convert Director, remove everything in the brackets
director <- gsub("\\s*\\([^\\)]+\\)", "", tolower(as.character(categorical.variables$Director)))
director <- gsub(" ", "", director)
director <- gsub(",", " ", director)  # reserve whole director's name
director.top10 <- top10.features(director, "director.")
categorical.variables <- cbind(categorical.variables, director.top10)

# Convert Actors, remove everything in the brackets
actors <- gsub("\\s*\\([^\\)]+\\)", "", tolower(as.character(categorical.variables$Actors)))
actors <- gsub(" ", "", actors)
actors <- gsub(",", " ", actors)  # reserve whole actors's name
actors.top10 <- top10.features(actors, "actors.")
categorical.variables <- cbind(categorical.variables, actors.top10)

# Convert Genre, Language and Country
categorical.variables$Genre <- gsub(", ", ",", categorical.variables$Genre)
categorical.variables$Genre <- gsub(" ", "_", categorical.variables$Genre)
categorical.variables$Genre <- gsub("-", "_", categorical.variables$Genre)
categorical.variables$Language <- gsub(", ", ",", categorical.variables$Language)
categorical.variables$Language <- gsub(" ", "_", categorical.variables$Language)
categorical.variables$Language <- gsub("-", "_", categorical.variables$Language)
categorical.variables$Country <- gsub(", ", ",", categorical.variables$Country)
categorical.variables$Country <- gsub(" ", "_", categorical.variables$Country)
categorical.variables$Country <- gsub("-", "_", categorical.variables$Country)
genre <- one.hot.encoding(tolower(categorical.variables$Genre))
categorical.variables <- cbind(categorical.variables, genre)
categorical.variables[, c("n/a")] <- NULL
language <- one.hot.encoding(tolower(categorical.variables$Language))
categorical.variables <- cbind(categorical.variables, language)
categorical.variables[, c("n/a")] <- NULL
country <- one.hot.encoding(tolower(categorical.variables$Country))
categorical.variables <- cbind(categorical.variables, country)
categorical.variables[, c("n/a")] <- NULL

# Final feature set3
feature.set3 <- categorical.variables
feature.set3[, 1:9] <- NULL
# print(colnames(feature.set3))

# Modelling
options(warn = -1)
feature.set3.results <- model_evaluation(data = feature.set3,
    modelName = "converted non-numeric variables only")
print(feature.set3.results$figure)
```



```
model.summary.3 <- feature.set3.results$model_summary
print(model.summary.3[model.summary.3$type == "test", ])
```

```
##    type fraction  N      rmse           sd           se           ci
## 11 test      0.1 10 179004686 2.656019e+07 8.399069e+06 1.900001e+07
## 12 test      0.2 10 151391350 1.718981e+07 5.435894e+06 1.229685e+07
## 13 test      0.3 10 143189712 8.775233e+06 2.774972e+06 6.277424e+06
## 14 test      0.4 10 132418221 8.489254e+06 2.684538e+06 6.072846e+06
## 15 test      0.5 10 133412912 5.310874e+06 1.679446e+06 3.799170e+06
## 16 test      0.6 10 130755208 7.000275e+06 2.213681e+06 5.007695e+06
## 17 test      0.7 10 127030796 3.564644e+06 1.127240e+06 2.549993e+06
## 18 test      0.8 10 125662999 2.327007e+06 7.358644e+05 1.664641e+06
## 19 test      0.9 10 123739766 1.110174e+06 3.510679e+05 7.941708e+05
## 20 test      1.0 10 122101156 1.779842e-07 5.628354e-08 1.273222e-07
```

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

**A**: I used categorical variables Awards, Rated, Production, Writers, Actors, Directors, Genre, Language and Country and encoded them in 4 different ways:
(1) Awards
For each cell I extracted the numbers using regular expression, and sum them up as the total awards and nominations.
(2) Rated, Production
For these two columns, each cell only has one value thus I directly converted them into dummy variables. Note that in 'Production' the same company may have different names and I treated them with same dummy variable.
(3) Genre, Language and Country

For these three columns, each cell has multiple values yet the total number of different values is few. So I used one-hot encoding to convert these categorical variables.
(4) Writers, Actors, Directors
For these three columns, each cell has multiple values and the total amounts of different values is tremendous.Thus I first computed the appearence frequencies of each actors/directors/writers, and selected top 10 actors/directors/writers to carry out one-hot encoding, respectively.

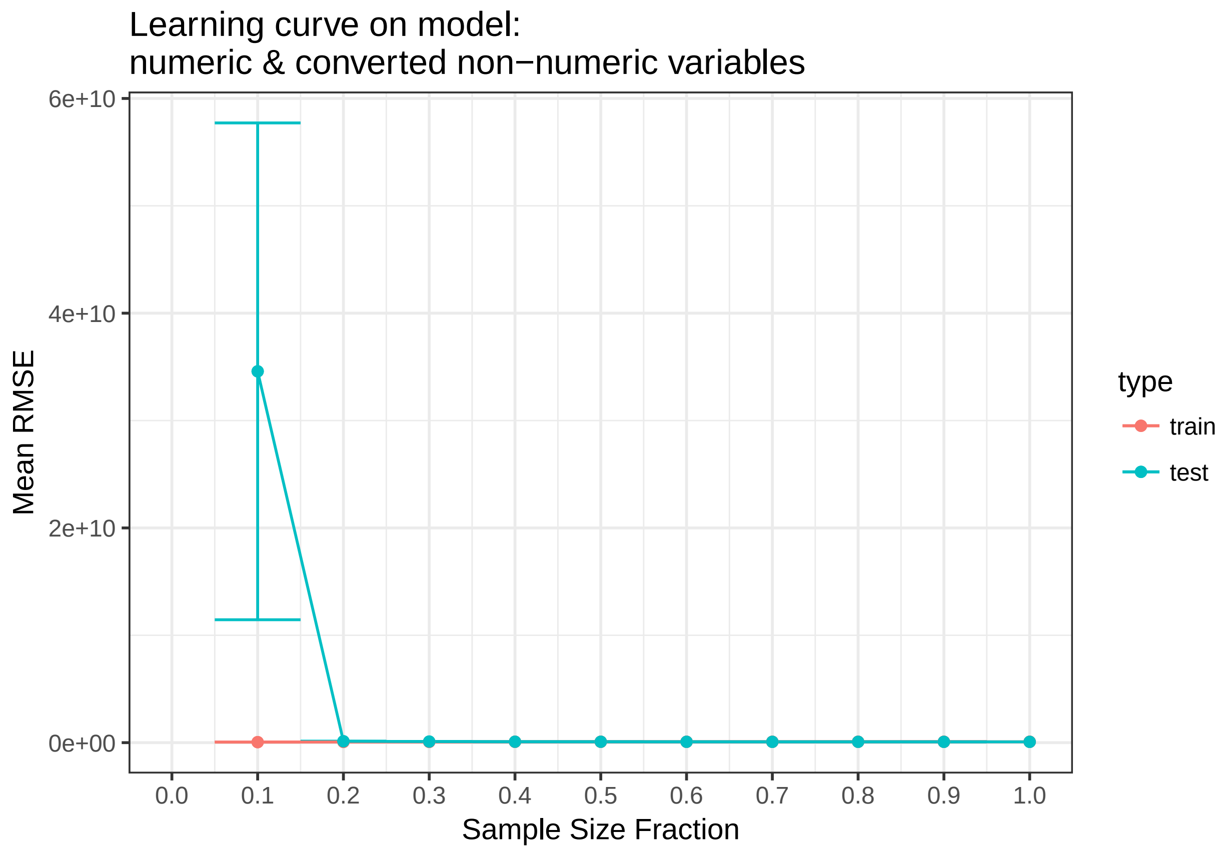
**Q**: What is the best mean test RMSE value you observed, and at what training set size? How does this compare with Task 2?

**A**: The best mean test RMSE value I observed is 122101156 at training set size = 100%. Comparing with Task 2, the best mean test RMSE value of Task 3 is almost double that of Task 2 (122101156 VS. 67146558).

## 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)
feature.set3.1 <- feature.set3
feature.set3.1$Gross <- NULL
feature.set4 <- cbind(feature.set2, feature.set3.1)

feature.set4.results <- model_evaluation(data = feature.set4,
    modelName = "\nnumeric & converted non-numeric variables")
print(feature.set4.results$figure)
```



```r
model.summary.4 <- feature.set4.results$model_summary
print(model.summary.4[model.summary.4$type == "test", ])
```

```
##    type fraction  N        rmse           sd           se           ci
## 11 test      0.1 10 34585217294 7.316872e+10 2.313798e+10 5.234175e+10
## 12 test      0.2 10   128217923 2.957795e+07 9.353369e+06 2.115879e+07
## 13 test      0.3 10   101047832 2.507416e+07 7.929147e+06 1.793698e+07
## 14 test      0.4 10    85577763 7.337511e+06 2.320325e+06 5.248939e+06
## 15 test      0.5 10    83691552 4.780535e+06 1.511738e+06 3.419789e+06
## 16 test      0.6 10    78608415 3.357161e+06 1.061628e+06 2.401568e+06
## 17 test      0.7 10    76967079 2.882575e+06 9.115502e+05 2.062070e+06
## 18 test      0.8 10    77512935 2.073266e+06 6.556242e+05 1.483125e+06
## 19 test      0.9 10    76067944 1.013291e+06 3.204307e+05 7.248646e+05
## 20 test      1.0 10    75711552 6.865272e-03 2.170990e-03 4.911120e-03
```

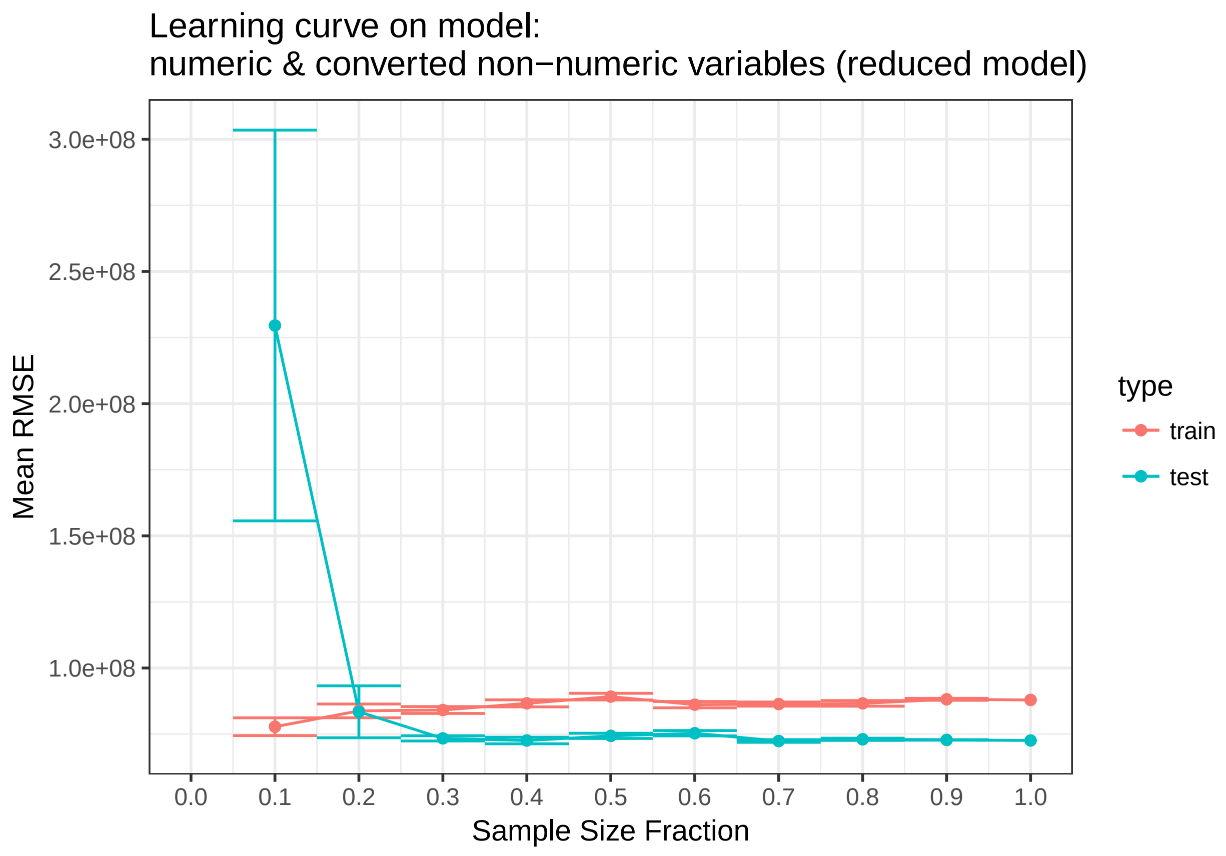
```r
trained_models.4 <- feature.set4.results$trained_models
coef <- summary(trained_models.4[[10]])$coef
p.value.list <- data.frame(coef[coef[, 4] <= 1, 4])
colnames(p.value.list) <- c("p.value")
sig.feature <- rownames(p.value.list)[p.value.list$p.value <
    0.05]
print("Features have p-value < 0.05:")
```

```
## [1] "Features have p-value < 0.05:"
```

```r
full.model <- as.data.frame(p.value.list[p.value.list$p.value <
    0.05, 1])
colnames(full.model) <- c("p.value")
rownames(full.model) <- sig.feature
print(full.model)
```

```
##                          p.value
## (Intercept)         1.244939e-08
## Year                1.998710e-08
## imdbRating          2.931619e-03
## tomatoRating        4.591190e-04
## tomatoFresh         7.914720e-05
## tomatoUserReviews   1.083594e-03
## Budget              3.596931e-35
## Year.log            2.008377e-08
## imdbRating.log      6.145577e-03
## tomatoRating.log    2.180296e-03
## tomatoFresh.log     9.150499e-03
## Budget.log          1.292109e-05
## imdbRating.sqrt     4.460990e-03
## imdbVotes.sqrt      2.203204e-02
## tomatoRating.sqrt   9.998701e-04
## tomatoFresh.sqrt    7.040046e-04
## tomatoUserReviews.sqrt 9.725443e-03
## Budget.sqrt         2.601751e-12
## imdbRating.cub      5.013949e-03
## imdbVotes.cub       3.868407e-02
## tomatoRating.cub    1.289122e-03
## tomatoFresh.cub     8.630857e-04
## Budget.cub          2.314580e-09
## writer.davids          1.792251e-03
## writer.weiss           3.444764e-02
## sci_fi                 3.860551e-02
## documentary            9.302838e-03
## animation              1.629277e-15
## mandarin               3.427985e-02
## greek                  7.188254e-03
## icelandic              5.639738e-04
## old_english            3.964819e-03
## sindarin               6.530477e-06
## xhosa                  4.707733e-02
## aramaic                3.744806e-07
## north_american_indian  1.645286e-08
## syriac                 1.450028e-03
## australia              3.966962e-02
## china                  5.000778e-04
## bahamas                1.414302e-02
```

```
sig.feature <- sig.feature[-1]

feature.set4.1 <- feature.set4[, sig.feature]
feature.set4.1$Gross <- feature.set4$Gross
feature.set4.1.results <- model_evaluation(data = feature.set4.1,
    modelName = "\nnumeric & converted non-numeric variables (reduced model)")
print(feature.set4.1.results$figure)
```



```r
model.summary.4.1 <- feature.set4.1.results$model_summary
print(model.summary.4.1[model.summary.4.1$type == "test", ])
```

```
##    type fraction  N      rmse           sd           se           ci
## 11 test      0.1 10 229566079 2.337108e+08 7.390584e+07 1.671866e+08
## 12 test      0.2 10  83439165 3.107909e+07 9.828070e+06 2.223264e+07
## 13 test      0.3 10  73384676 3.089525e+06 9.769936e+05 2.210113e+06
## 14 test      0.4 10  72577348 3.931462e+06 1.243237e+06 2.812398e+06
## 15 test      0.5 10  74325954 3.130357e+06 9.899057e+05 2.239322e+06
## 16 test      0.6 10  75328825 3.193530e+06 1.009883e+06 2.284514e+06
## 17 test      0.7 10  72391129 1.602456e+06 5.067411e+05 1.146328e+06
## 18 test      0.8 10  73031318 1.321966e+06 4.180425e+05 9.456778e+05
## 19 test      0.9 10  72793123 2.849342e+05 9.010410e+04 2.038296e+05
## 20 test      1.0 10  72577211 4.449756e-03 1.407136e-03 3.183164e-03
```

```r
trained_models.4.1 <- feature.set4.1.results$trained_models
coef <- summary(trained_models.4.1[[10]])$coef
p.value.list <- data.frame(coef[coef[, 4] <= 1, 4])
colnames(p.value.list) <- c("p.value")
sig.feature <- rownames(p.value.list)[p.value.list$p.value <
    0.05]
print("Reduced Model:")
```

```
## [1] "Reduced Model:"
```

```r
reduced.model <- as.data.frame(p.value.list[p.value.list$p.value <
    0.05, 1])
colnames(reduced.model) <- c("p.value")
rownames(reduced.model) <- sig.feature
print(reduced.model)
```

```
##                            p.value
## (Intercept)           2.220970e-08
## Year                  3.016328e-08
## imdbRating            2.165101e-03
## tomatoRating          9.273749e-03
## tomatoFresh           5.430082e-06
## tomatoUserReviews     7.138978e-09
## Budget                1.321420e-57
## Year.log              3.097608e-08
## imdbRating.log        4.804142e-03
## tomatoRating.log      2.504570e-02
## tomatoFresh.log       9.452877e-03
## Budget.log            1.117240e-06
## imdbRating.sqrt       3.330477e-03
## imdbVotes.sqrt        3.528448e-34
## tomatoRating.sqrt     1.499785e-02
## tomatoFresh.sqrt      1.304148e-04
## tomatoUserReviews.sqrt 2.515756e-14
## Budget.sqrt           2.026477e-19
## imdbRating.cub        3.789323e-03
## imdbVotes.cub         6.629513e-19
## tomatoRating.cub      1.773319e-02
## tomatoFresh.cub       4.110511e-04
## Budget.cub            6.342618e-14
## writer.davids          5.530630e-04
## writer.weiss           1.923883e-03
## sci_fi                 6.906620e-04
## documentary            2.157881e-04
## animation              2.268503e-29
## mandarin               2.313356e-02
## greek                  5.376250e-03
## icelandic              3.968956e-03
## old_english            8.990932e-04
## sindarin               4.628032e-11
## aramaic                4.587571e-08
## north_american_indian  7.142104e-12
## syriac                 1.285610e-03
## australia              3.868184e-02
## china                  4.292526e-03
## bahamas                3.247914e-02
```

```
sig.feature <- sig.feature[-1]
```

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

**A**: I first used all features in Task 2 and 3 and the best mean test RMSE value is 75711552 at training set size = 100%. It is worse than that of Task 2 (best mean test RMSE = 67146558), yet is much better than that of Task 3 (best mean test RMSE = 122101156). Based on this linear regression results, I then select features from Tasks 2 & 3 that have p-values < 0.05 and built a reduced model. The best mean test RMSE value of this reduced model is 72577211. Again, it is worse than that of Task 2 and much better than that of Task 3.

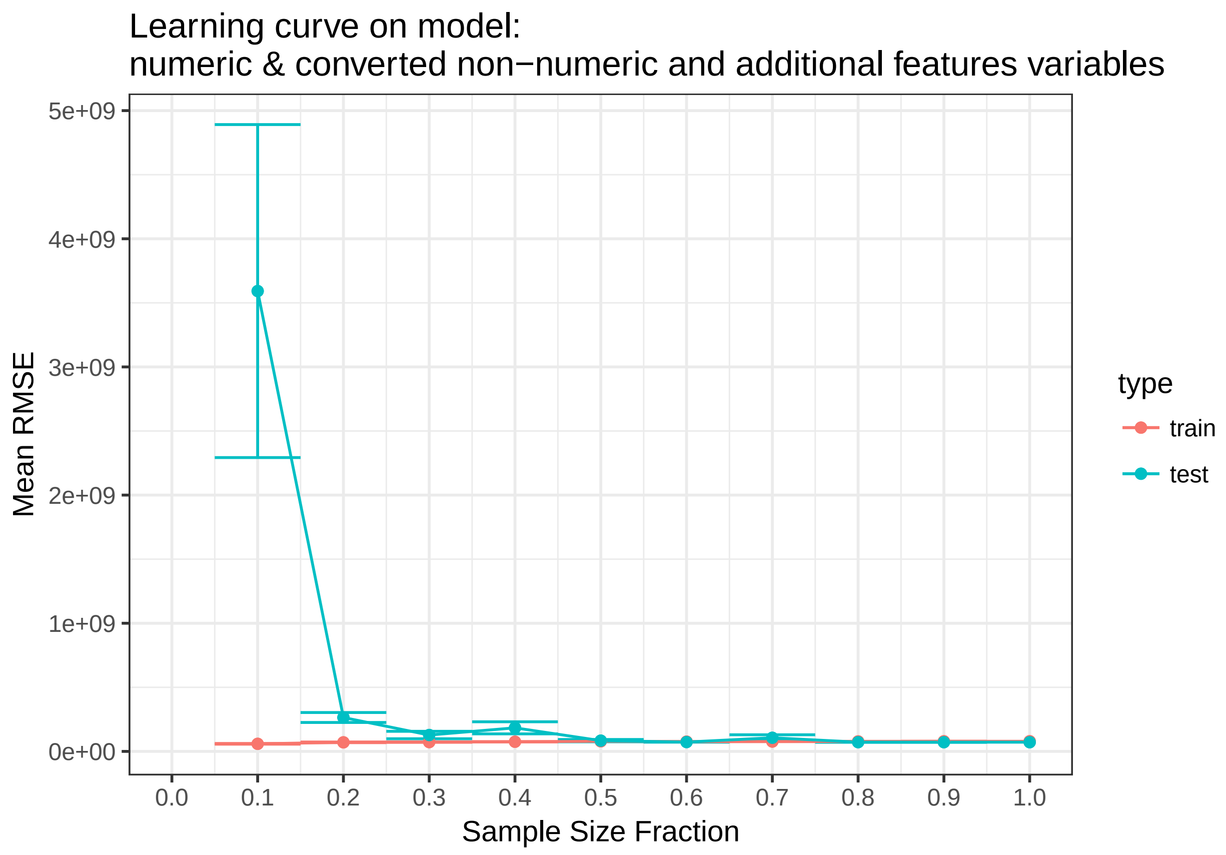
## 5. Additional features

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) population[order(population$age),]
reduced.model.sorted <- as.data.frame(reduced.model[order(reduced.model$p.value),
    ])
colnames(reduced.model.sorted) <- c("p.value")
rownames(reduced.model.sorted) <- rownames(reduced.model)[order(reduced.model$p.value)]
print(reduced.model.sorted)
```

```
##                              p.value
## Budget                 1.321420e-57
## imdbVotes.sqrt         3.528448e-34
## animation              2.268503e-29
## Budget.sqrt            2.026477e-19
## imdbVotes.cub          6.629513e-19
## tomatoUserReviews.sqrt 2.515756e-14
## Budget.cub             6.342618e-14
## north_american_indian  7.142104e-12
## sindarin               4.628032e-11
## tomatoUserReviews      7.138978e-09
## (Intercept)            2.220970e-08
## Year                   3.016328e-08
## Year.log               3.097608e-08
## aramaic                4.587571e-08
## Budget.log              1.117240e-06
## tomatoFresh             5.430082e-06
## tomatoFresh.sqrt        1.304148e-04
## documentary             2.157881e-04
## tomatoFresh.cub         4.110511e-04
## writer.davids           5.530630e-04
## sci_fi                  6.906620e-04
## old_english             8.990932e-04
## syriac                  1.285610e-03
## writer.weiss            1.923883e-03
## imdbRating              2.165101e-03
## imdbRating.sqrt         3.330477e-03
## imdbRating.cub          3.789323e-03
## icelandic               3.968956e-03
## china                   4.292526e-03
## imdbRating.log          4.804142e-03
## greek                   5.376250e-03
## tomatoRating            9.273749e-03
## tomatoFresh.log         9.452877e-03
## tomatoRating.sqrt       1.499785e-02
## tomatoRating.cub        1.773319e-02
## mandarin                2.313356e-02
## tomatoRating.log        2.504570e-02
## bahamas                 3.247914e-02
## australia               3.868184e-02
```

```
interaction.features <- data.frame(matrix(NA, nrow = dim(feature.set4.1)[1],
    ncol = 0))
interaction.features.names <- c()
for (i in 1:9) {
    for (j in seq(i + 1, 10)) {
        feature_i <- rownames(reduced.model.sorted)[i]
        feature_j <- rownames(reduced.model.sorted)[j]
        new.col.name <- paste(feature_i, feature_j, sep = "_")
        new.col <- feature.set4.1[, feature_i] * feature.set4.1[,
            feature_j]
        interaction.features <- cbind(interaction.features, new.col)
        interaction.features.names <- c(interaction.features.names,
            new.col.name)
    }
}

colnames(interaction.features) <- interaction.features.names

# Add interatction features and modelling
feature.set5 <- feature.set4.1[, sig.feature]
feature.set5$Gross <- feature.set4$Gross
feature.set5 <- cbind(feature.set5, interaction.features)

feature.set5.results <- model_evaluation(data = feature.set5,
    modelName = "\nnumeric & converted non-numeric and additional features variables")
print(feature.set5.results$figure)
```



```
model.summary.5 <- feature.set5.results$model_summary
print(model.summary.5[model.summary.5$type == "test", ])
```

```
##    type fraction  N       rmse           sd           se           ci
## 11 test      0.1 10 3591987342 4.108478e+09 1.299215e+09 2.939028e+09
## 12 test      0.2 10  264515120 1.226248e+08 3.877735e+07 8.772047e+07
## 13 test      0.3 10  127668084 9.171273e+07 2.900211e+07 6.560733e+07
## 14 test      0.4 10  183836803 1.492189e+08 4.718715e+07 1.067447e+08
## 15 test      0.5 10   83780629 2.887447e+07 9.130909e+06 2.065555e+07
## 16 test      0.6 10   72876002 2.910895e+06 9.205059e+05 2.082329e+06
## 17 test      0.7 10  106071141 7.432198e+07 2.350267e+07 5.316674e+07
## 18 test      0.8 10   72102303 2.359726e+06 7.462110e+05 1.688047e+06
## 19 test      0.9 10   71981390 5.428639e+05 1.716686e+05 3.883415e+05
## 20 test      1.0 10   71682610 6.504097e-03 2.056776e-03 4.652750e-03
```

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

**A**: Based on the reduced model in Task 4, I sorted the p-values of the features in ascending order and selected the top 10 features. Then I calculated their interaction terms and added them into the reduced model in Task 4. My motivation is to check if the relationships of some key features are additive or not.

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

**A**: The final mean test RMSE value is 71682610 at training set size = 100%. It is better than that of Task 4 (best mean test RMSE = 72577211), Task 3 (best mean test RMSE = 122101156) and Task 1 (best mean test RMSE = 72295468), yet still worse than that of Task 2 (best mean test RMSE = 67146558).

The course of this project teaches me a lot and I believe the most important thing I learned is that, instead of the model itself, the following four steps are the key factors for building and tuning up a useful model:

(1) Data cleaning:
The initial input data is not very clean. It includes missing values, mismatching values in different columns, misspelling words and a few not obvious mistakes you could not be aware of until starting encoding. Thus a clean and well-prepared dataset is a good beginning for the entire project.
(2) Encoding:
For various types of categorical data, we might use different encoding strategies. A proper encoding strategy could help us to avoid multicollinearity and overfitting.
(3) Data size:
The training set size plays a vital role in building up models. From Task 1, 2, 3 and 4 we could clearly see when training set size = 10%, we always have the best mean training RMSE and the worst mean testing RMSE.
(4) Feature engineering:
Besides barely using the features in the initial dataset, it is worthy to spend some times on feature engineering, since the distribution of some variables are highly skewed right or left, and the relationships of some variables are not additive. One good example is Task 2, in which we performed transformation on numerical variables and obtained the best mean test RMSE value among all 5 tasks.