Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics - Summer 2018

Due: Sunday, July 24, 2018 at 11:59 PM UTC-12:00 on T-Square

Mohitdeep Singh (msingh74)

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')
options(warn=-1)
```

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
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_mov <-subset(df, df$Type=="movie")
cat("df has ", dim(df_mov)[1], "rows and", dim(df_mov)[2], "columns", end="\n", file="")
```

```
## df has  40000 rows and 39 columns
```

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_mov_gross = df_mov[!is.na(df_mov$Gross), ]
cat("df has", dim(df_mov_gross)[1], "rows and", dim(df_mov_gross)[2], "columns", end="\n", file="")
```

```
## df has 4558 rows and 39 columns
```

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
df_mov_gross_2k = df_mov_gross[df_mov_gross$Year >=2000 ,]
cat("df has", dim(df_mov_gross_2k)[1], "rows and", dim(df_mov_gross_2k)[2], "columns", end="\n", file="
```

```
## df has 3332 rows and 39 columns
```

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
X= data.frame(do.call('rbind', strsplit(as.character(df_mov_gross_2k$Released),'-',fixed=TRUE)))
df_mov_gross_2k$releasedYear = X$X1
cat("df_mov_gross_2k", dim(df_mov_gross_2k)[1], "rows and", dim(df_mov_gross_2k)[2], "columns", end="\n
```

```
## df_mov_gross_2k 3332 rows and 40 columns
```

```r
df_movie_mismatch = df_mov_gross_2k[abs(df_mov_gross_2k$Year - df_mov_gross_2k$Date) <= 1,]
df_movie_mismatch$releasedYear = as.numeric(as.character(df_movie_mismatch$releasedYear))
df_movie_mismatch_2 = df_movie_mismatch[abs(df_movie_mismatch$Year - df_movie_mismatch$releasedYear) <=
cat("df_movie_gross_2000_match3 has", dim(df_movie_mismatch_2)[1], "rows and", dim(df_movie_mismatch_2)
```

```
## df_movie_gross_2000_match3 has 3179 rows and 40 columns
```

```r
#df_movie_gross_2000_match2 = df_mov_gross_2k[abs(df_mov_gross_2k$Year - df_mov_gross_2k$Date) <= 1,]
#df_movie_gross_2000_match2$releasedYear = as.numeric(as.character(df_movie_gross_2000_match2$releasedY
#df_movie_gross_2000_match3 = df_movie_gross_2000_match2[abs(df_movie_gross_2000_match2$Year - #df_movi
#cat("df_movie_gross_2000_match3 has", dim(df_movie_gross_2000_match3)[1], "rows and", dim(df_movie_gro
```

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.

```r
# TODO: Exclude the `Domestic_Gross` column
df_movie_mismatch_2$Domestic_Gross <- NULL
cat("df_movie_gross_2000_match3 has", dim(df_movie_mismatch_2)[1], "rows and", dim(df_movie_mismatch_2)
```

```
## df_movie_gross_2000_match3 has 3179 rows and 39 columns
```

6. Process `Runtime` column

```r
# TODO: Replace df$Runtime with a numeric column containing the runtime in minutes
num_rows = dim(df_movie_mismatch_2)[1]
for (i in seq(1, num_rows)){
  con = unlist(strsplit(df[i, "Runtime"], split=" "))
  if (con == "N/A"){
    df_movie_mismatch_2[i,"Runtime"] = 0
    next()
  }
```

```
  cur = 0
  for (j in seq(1, length(con) - 1)){
    if (substr(con[j + 1], 1, 1) == "h"){
      cur = cur + as.numeric(con[j]) * 60
    }

    if (substr(con[j + 1],1, 3) == "min"){
      cur = cur + as.numeric(con[j])
    }
  }
  df_movie_mismatch_2[i,"Runtime"]= cur
}
df_movie_mismatch_2$Runtime =as.numeric((df_movie_mismatch_2$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
df = df_movie_mismatch_2
```

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

Final preprocessed dataset

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

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

```
## After preprocessing, dataset has 3179 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] "Gross"             "Date"              "releasedYear"
```

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 (%).

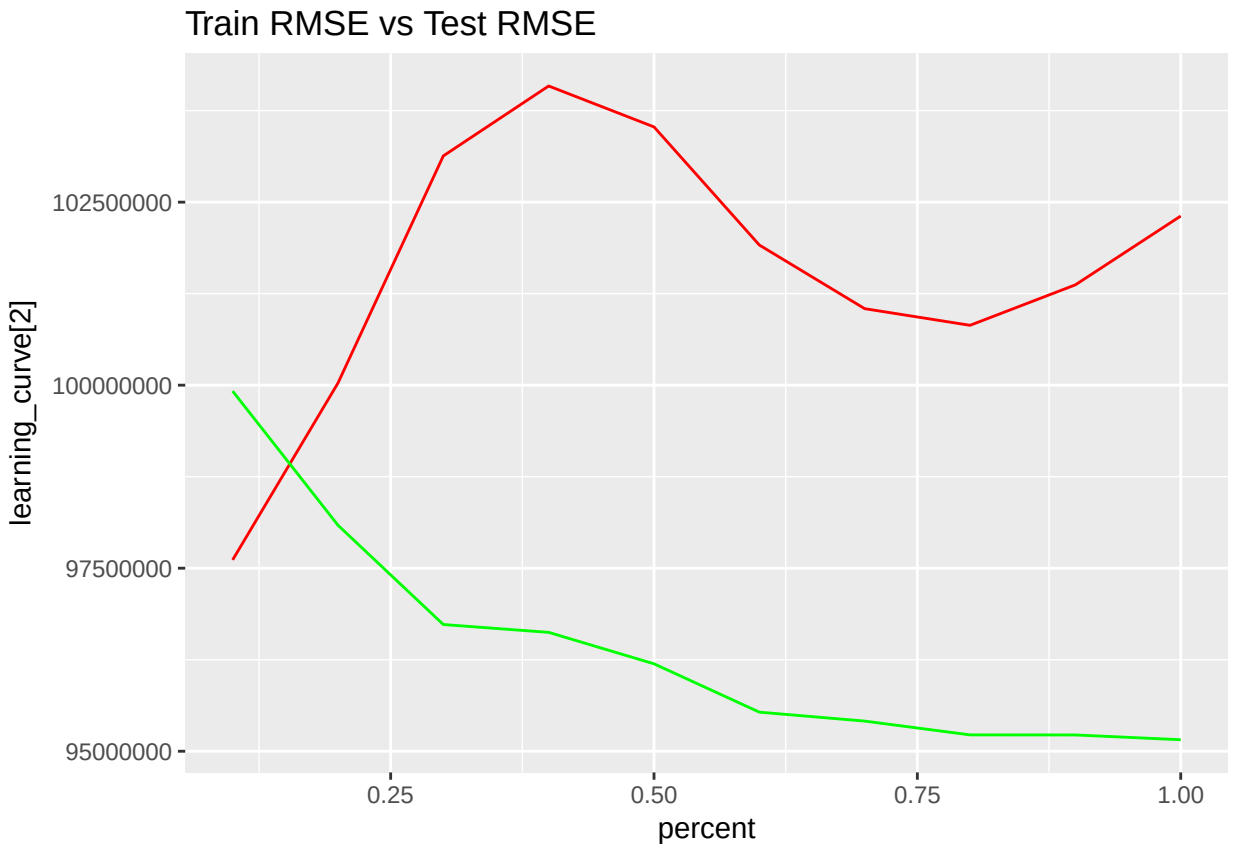
1. Numeric variables

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)

df2 <- df[sample(nrow(df)),]
test = df2[1: (dim(df2)[1] * 0.2),]
train = df2[((dim(df2)[1] * 0.2)) : dim(df2)[1],]

feature_set = c("Gross", "Runtime", "imdbVotes", "tomatoRating", "imdbRating","tomatoFresh","Budget" )
this_train = subset(train, select = feature_set)
this_train = na.omit(this_train)

this_test = subset(test, select = feature_set)
this_test = na.omit(this_test)

getLearningCurve <- function(train_rmse, test_rmse){

  learning_curve = matrix(c(0), nrow = 10, ncol = 3)
  colnames = c("sample_percentage", "train", "")
  for (row in 1: 10){
    learning_curve[row,1] = row * 0.1
    learning_curve[row, 2] = mean(train_rmse[row,])
    learning_curve[row, 3] = mean(test_rmse[row,])
  }
  learning_curve = data.frame(learning_curve)
  return(learning_curve)
}
plot_learning_curves <- function(learning_curve){
```

```
  percent = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)

  g <- ggplot(learning_curve, aes(percent))
  g <- g + geom_line(aes(y=learning_curve[2]), colour="red")
  g <- g + geom_line(aes(y=learning_curve[3]), colour="green")
  g = g + labs(title = "Train RMSE vs Test RMSE")
return(g)
}
trainModel <- function(train, test){
  sample_size = as.integer(c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0) * dim(train)[1])

  train_rmses = matrix(c(0), nrow = 10, ncol = 10)
  test_rmses = matrix(c(0), nrow = 10, ncol = 10)
  for (j in 1:10){
    train = train[sample(nrow(train)),]
    for (i in 1:10){
        cur = train[1:sample_size[i],]
        fit = lm(Gross~., data = cur, na.action = na.omit)
        cur$pred_train = predict(fit, cur)
        cur$resid = (cur$pred_train - cur$Gross) * (cur$pred_train - cur$Gross)
        cur_RMSE = sqrt(sum(cur$resid) / dim(cur)[1])
        train_rmses[i,j] = (cur_RMSE)
        test$pred_test = predict(fit, test)
        test$resid = (test$pred_test - test$Gross) * (test$pred_test - test$Gross)
        test_rmse = sqrt(sum(test$resid) / dim(test)[1])
        test_rmses[i, j] = test_rmse
    }
  }
  lc = getLearningCurve(train_rmses, test_rmses)
  return(lc)

}

lc_1 = trainModel(this_train, this_test)
g = plot_learning_curves(lc_1)
g
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```




```
head(lc_1, 10)
```

```
##     X1        X2       X3
## 1  0.1  97613573 99918449
## 2  0.2 100029510 98086566
## 3  0.3 103131616 96730480
## 4  0.4 104086511 96623370
## 5  0.5 103527384 96193750
## 6  0.6 101914148 95533476
## 7  0.7 101045071 95411469
## 8  0.8 100817810 95223283
## 9  0.9 101370390 95222025
## 10 1.0 102310277 95156871
```

```
print (min(lc_1[,3]))
```

```
## [1] 95156871
```

```
print (which.min(lc_1[,3]))
```

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

Q: List the numeric variables you used.

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

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

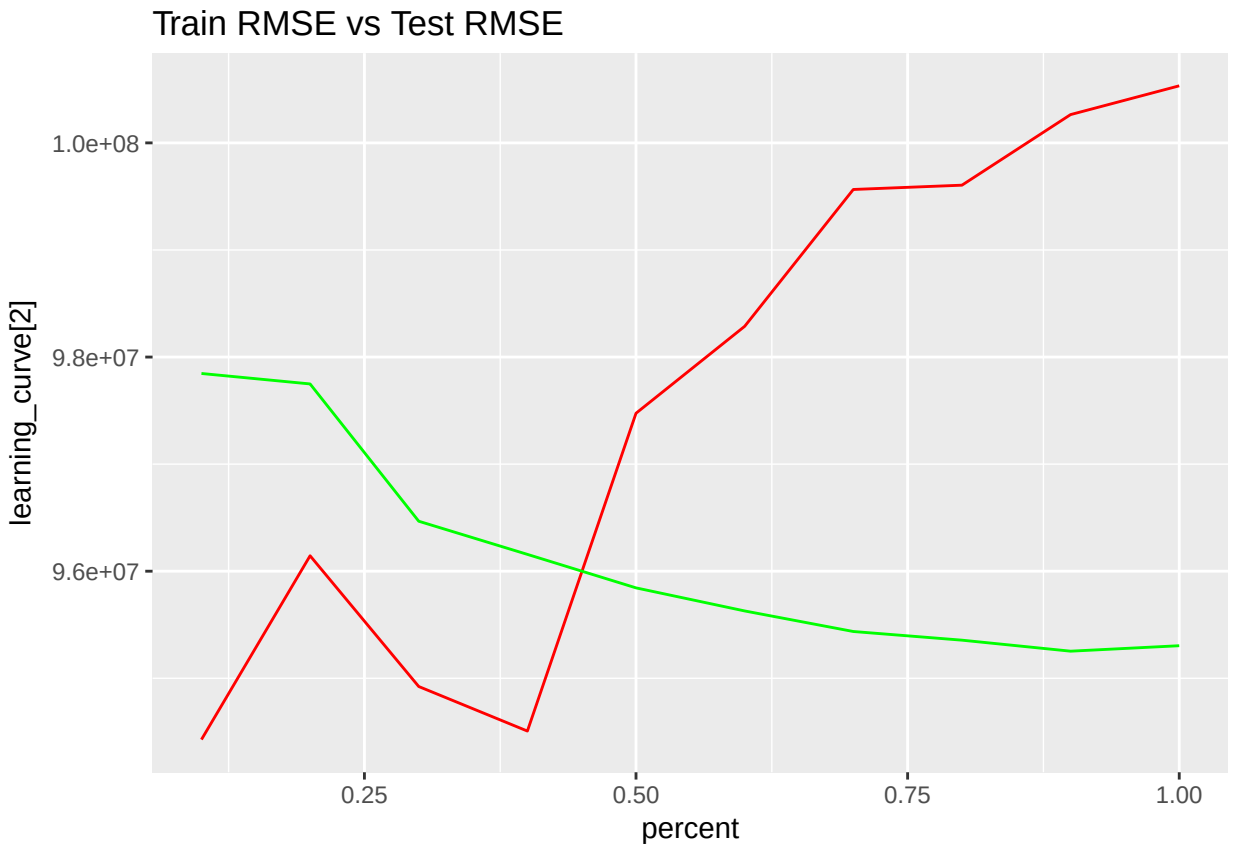
A: RMSE is 103499940 corresponding to 100% trainset size

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)

selected = c("Gross", "Runtime", "imdbVotes", "tomatoRating", "imdbRating","tomatoFresh","Budget" )
train2 = subset(train, select = selected)
train2 = na.omit(train2)

train2$logBudget = log(train2$Budget)
train2$bintomatoRating = cut(train2$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))

test2 = subset(test, select = selected)
test2 = na.omit(test2)

test2$logBudget = log(test2$Budget)
test2$bintomatoRating = cut(test2$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))

lc_2 = trainModel(train2, test2)
g = plot_learning_curves(lc_2)
g
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```



```
head(lc_2, 10)
```

```
##     X1        X2       X3
## 1  0.1  94427806 97846435
## 2  0.2  96144348 97748869
## 3  0.3  94923232 96466650
## 4  0.4  94507345 96157243
## 5  0.5  97475283 95844468
## 6  0.6  98286893 95628872
## 7  0.7  99564885 95436955
## 8  0.8  99605077 95355973
## 9  0.9 100264268 95253126
## 10 1.0 100532965 95304176
```

```
print (min(lc_2[,3]))
```

```
## [1] 95253126
```

```
print (which.min(lc_2[,3]))
```

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

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

A: I used the log transformation to scale "budget" feature. I also "binned" tomatoRatings. Since, we want to improve prediction of "Gross" variable, we observed (in project 1) that log transforming "budget" variable has positive impact in predicting "Gross". Binning tomato ratings help in differentiating tomato values even further. This increases the overall power of this feature.

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

A: The test rmse dropped to 102218111 from 103499940 (first task)

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)

selected = c("Gross", "Runtime", "imdbVotes", "tomatoRating","imdbRating","tomatoFresh","Budget", "Genre
train3 = subset(train, select = selected)
train3 = na.omit(train3)
test3 = subset(test, select = selected)
test3 = na.omit(test3)

total = rbind(train3, test3)
library(tm)
```

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

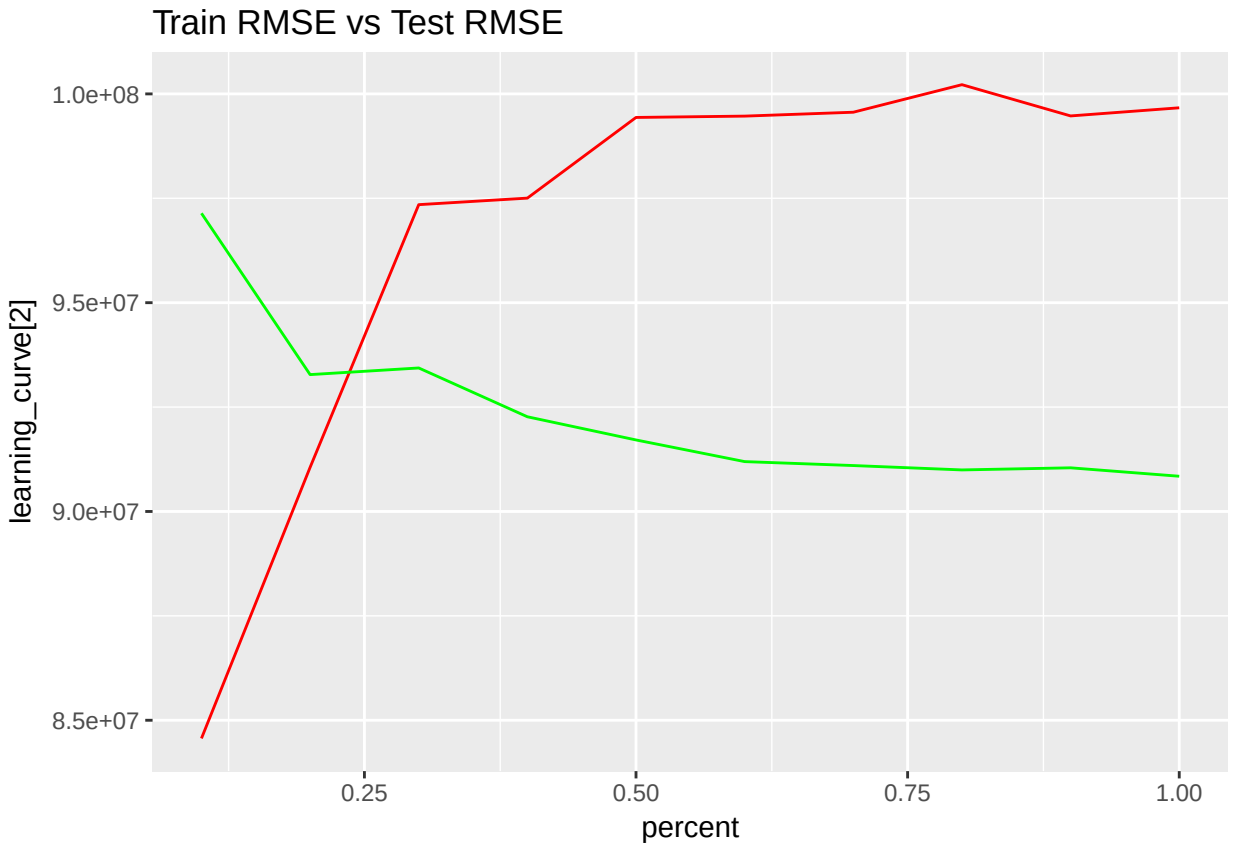
```
##
## Attaching package: 'NLP'
```

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

```
##     annotate
```

```
myCorpus <- Corpus(VectorSource(total$Genre))
myTDM <- DocumentTermMatrix(myCorpus, control = list(minWordLength = 1))
x = as.matrix(myTDM)
total = cbind(total, x)
total$Genre = NULL

train3 = total[1: (dim(train3)[1]),]
test3 = total[(dim(train3)[1]): (dim(total)[1]),]


lc_3 = trainModel(train3, test3)
g = plot_learning_curves(lc_3)
g
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```




```
print (min(lc_3[,3]))
```

```
## [1] 90843584
```

```
print (which.min(lc_3[,3]))
```

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

```
head(lc_3, 10)
```

```
##     X1        X2       X3
## 1  0.1  84565067 97141680
```

```
## 2  0.2  91056035 93277782
## 3  0.3  97347805 93436592
## 4  0.4  97503953 92268106
## 5  0.5  99436063 91712541
## 6  0.6  99467699 91194121
## 7  0.7  99561194 91099671
## 8  0.8 100220723 90994809
## 9  0.9  99472566 91045694
## 10 1.0  99667426 90843584
```

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

A: I used genre (categorical variable). Basically, I converted this feature into binary numbers (0/1). So, if movie belongs to a particular genre, the value is 1 else 0 (one hot encoding approach)

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

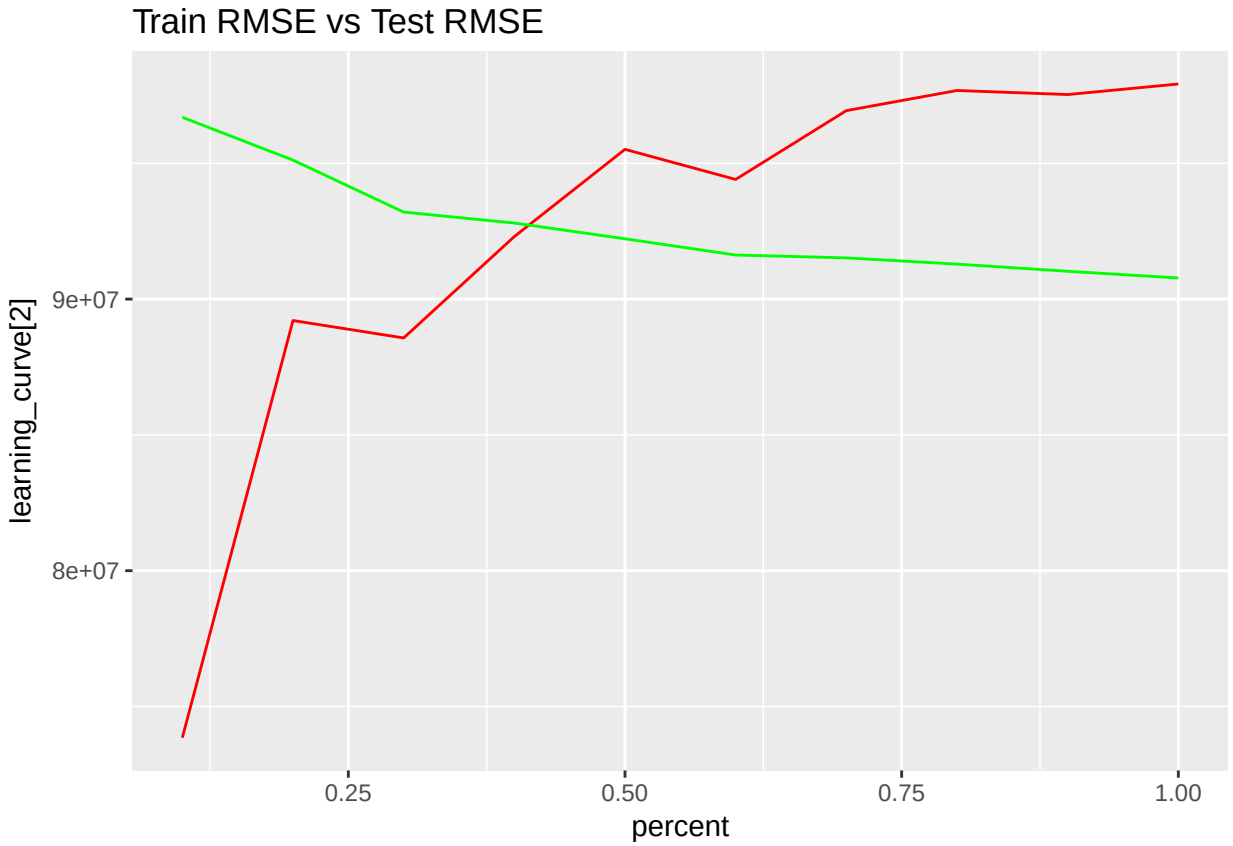
A: Improved from 102218111(Task 2) to 100087639

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)
library(tm)
selected = c("Gross", "Runtime", "imdbVotes", "tomatoRating","imdbRating","tomatoFresh","Budget", "Genre
train4 = subset(train, select = selected)
train4 = na.omit(train4)
test4 = subset(test, select = selected)
test4 = na.omit(test4)

final = rbind(train4, test4)

corpus <- Corpus(VectorSource(final$Genre))
tdm <- DocumentTermMatrix(corpus, control = list(minWordLength = 1))
X = as.matrix(tdm)
final = cbind(final, X)
final$Genre = NULL
train4 = final[1: (dim(train4)[1]),]
test4 = final[(dim(train4)[1]): (dim(final)[1]),]

train4$logBudget = log(train4$Budget)
train4$bintomatoRating = cut(train4$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))
test4$logBudget = log(test4$Budget)
test4$bintomatoRating = cut(test4$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))

lc_4 <- trainModel(train4, test4)
g <- plot_learning_curves(lc_4)
g
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```




```
print (min(lc_4[,3]))
```

```
## [1] 90777210
```

```
print (which.min(lc_4[,3]))
```

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

```
head(lc_4, 10)
```

```
##     X1       X2       X3
## 1  0.1 73848909 96694722
## 2  0.2 89210645 95117830
## 3  0.3 88569953 93203481
## 4  0.4 92303448 92801222
## 5  0.5 95513757 92220406
## 6  0.6 94406508 91623958
## 7  0.7 96939918 91517041
## 8  0.8 97681628 91288885
## 9  0.9 97531712 91024484
## 10 1.0 97920650 90777210
```

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

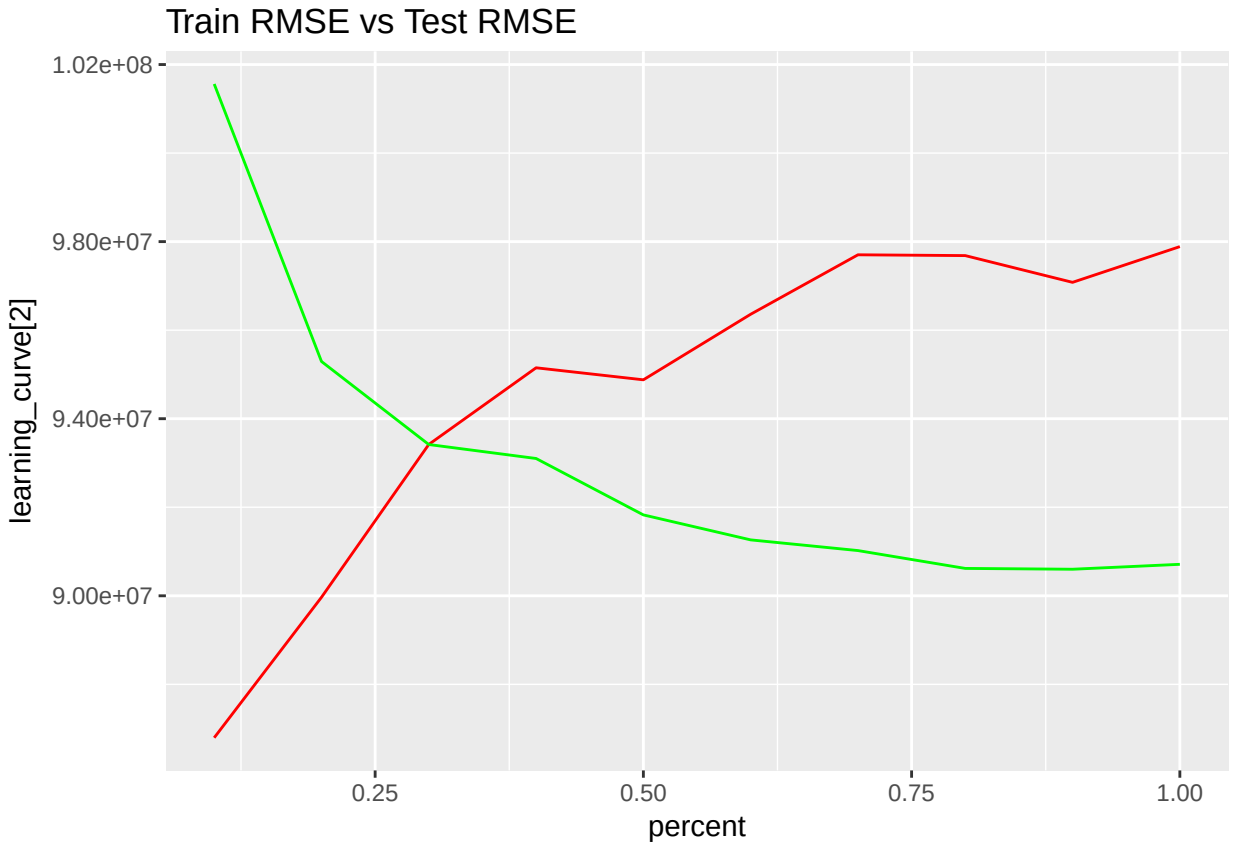
A: The RMSE further dropped from 102218111(Task 2), 100087639(task 3) to 98254740

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)
selected = c("Gross", "Runtime", "imdbVotes", "tomatoRating","imdbRating","tomatoFresh","Budget", "Genre
train5 = subset(train, select = selected)
train5 = na.omit(train5)
test5 = subset(test, select = selected)
test5 = na.omit(test5)

final = rbind(train5, test5)

myCorpus <- Corpus(VectorSource(final$Genre))
myTDM <- DocumentTermMatrix(myCorpus, control = list(minWordLength = 1))
x = as.matrix(myTDM)
final = cbind(final, x)
final$Genre = NULL

train5 = final[1: (dim(train5)[1]),]
test5 = final[(dim(train5)[1]): (dim(final)[1]),]

train5$logBudget = log(train5$Budget)
train5$bintomatoRating = cut(train5$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))
test5$logBudget = log(test5$Budget)
test5$bintomatoRating = cut(test5$tomatoRating, 5, include.lowest=TRUE, labels=c(1, 2,3,4,5))

train5$test <- ifelse(train5$imdbRating > 7 & train5$Budget > 5000000, 1, 0)
test5$test <- ifelse(test5$imdbRating > 7 & test5$Budget > 5000000, 1, 0)

lc_5 <- trainModel(train5, test5)
g <- plot_learning_curves(lc_5)
g
```

```
## Don't know how to automatically pick scale for object of type data.frame. Defaulting to continuous.
```




```
print (min(lc_5[,3]))
```

```
## [1] 90599497
```

```
print (which.min(lc_5[,3]))
```

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

```
head(lc_5, 20)
```

```
##     X1       X2        X3
## 1  0.1 86793856 101560615
## 2  0.2 89966835  95292783
## 3  0.3 93418042  93417319
## 4  0.4 95150293  93101461
## 5  0.5 94878713  91827643
## 6  0.6 96359047  91262716
## 7  0.7 97702443  91021553
## 8  0.8 97684381  90618598
## 9  0.9 97080502  90599497
## 10 1.0 97887186  90710935
```

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

A: I added a new binary feature. This binary feature indicates the current "perception" of movie based on imdbratings and budget of the movie. Basically, if the imdbrating is higher than 7 and the budget is greater than 5M, then feature would be 1. The reason to choose this is that this helps us in understanding the sentiment around the movie. Both budget and imdbRatings are known apriori which means that we can actually predict how much gross this movie would make before the movie is released. Note: while imdbRatings can accumulate forever but the rating process does start before movie is released.

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

A:The final RMSE obtained is 98293912. What I learnt from this project: Importance of feature engineering. Even simpler models can give good performance with time spend improving the quality of features.