#aaldabbagh3

title: ‘Project 2: Modeling and Evaluation’ subtitle: ’
CSE6242 - Data and Visual Analytics

Due: Friday, April 21, 2017 at 11:59 PM UTC-12:00 on T-Square

’ output: pdf_document: default html_document: default —

Data

We will use the same dataset as Project 1: `movies_merged` (https://s3.amazonaws.com/content.udacity-data.com/courses/gt-cs6242/project/movies_merged).

Objective

Your goal in this project is to build a linear regression model that can predict the `Gross` revenue earned by a movie based on other variables. You may use R packages to fit and evaluate a regression model (no need to implement regression yourself). Please stick to linear regression, however.

Instructions

You should be familiar with using an RMarkdown (http://rmarkdown.rstudio.com) Notebook by now. Remember that you have to open it in RStudio, and you can run code chunks by pressing *Cmd+Shift+Enter*.

Please complete the tasks below and submit this R Markdown file (as **pr2.Rmd**) containing all completed code chunks and written responses, as well as a PDF export of it (as **pr2.pdf**) which should include all of that plus output, plots and written responses for each task.

*Note that **Setup** and **Data Preprocessing** steps do not carry any points, however, they need to be completed as instructed in order to get meaningful results.*

Setup

Same as Project 1, load the dataset into memory:

```
setwd("C:/Users/AMD/Desktop/Classes/GaTech/Data Visualization/P2")
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.

```
suppressWarnings(library(ggplot2))
suppressWarnings(library(caret))
```

```
## Loading required package: lattice
```

```
suppressWarnings(library(pls))
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:caret':
##
##     R2
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
suppressWarnings(library(reshape2))
```

If you are using any non-standard packages (ones that have not been discussed in class or explicitly allowed for this project), please mention them below. Include any special instructions if they cannot be installed using the regular `install.packages('<pkg name>')` command.

Non-standard packages used: None

Data Preprocessing

Before we start building models, we should clean up the dataset and perform any preprocessing steps that may be necessary. Some of these steps can be copied in from your Project 1 solution. It may be helpful to print the dimensions of the resulting dataframe at each step.

1. Remove non-movie rows

```
# TODO: Remove all rows from df that do not correspond to movies
df = subset(df, Type=="movie")
rows_left = nrow(df)
rows_left
```

```
## [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.

```
# TODO: Remove rows with missing Gross value
newdf = df[complete.cases(df[,38]),]
```

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
newdf = newdf[newdf$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
secondYearColumn = na.omit(newdf$Released)
secondYearColumnLength = length(na.omit(secondYearColumn))
newYear = vector(mode="numeric", length=secondYearColumnLength)

i=1
for(releaseDate in secondYearColumn){
    newYear[i] = as.numeric(substr(secondYearColumn[i],1,4))
    i=i+1
}
noMismatch = subset(newdf, (!is.na(newdf[,4])) )
#noMismatch$Year2 = newYear
noMismatch = noMismatch[noMismatch$Year==newYear,] #This is the dataframe where Released year an
d Year Match

ReleasedMissing = subset(newdf, (is.na(newdf[,4])) )#This is the dataframe where released is mis
sing

dfNoMismatch = rbind(noMismatch[,1:39],ReleasedMissing) #This is the joined data frame
```

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
dfNoMismatch = dfNoMismatch[ , !(names(dfNoMismatch) %in% c("Domestic_Gross"))]
```

6. Process `Runtime` column

```
runtime_Categories = unique(gsub('[0-9]+', '', dfNoMismatch$Runtime))
changeToInt = function(number){
  category = unique(gsub('[0-9]+', '', number))
  if(category==" min"){
    numberList = na.omit(as.numeric(unlist(strsplit(number, "[^0-9]+"))))
    return (numberList[1])
  }
  if(category==" h  min"){
    numberList = na.omit(as.numeric(unlist(strsplit(number, "[^0-9]+"))))
    return (numberList[1]*60+numberList[2])
  }
  if(category==", min"){
    return (1618)
  }
  else{
    return (NaN)
  }
}
runtime = dfNoMismatch$Runtime
temp_runtime = vector(mode="numeric", length=length(runtime))

i = 1
for(tm in runtime){
  temp_runtime[i] = changeToInt(tm)
  i = i+1
}

dfNoMismatch$Runtime=temp_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


f=function(x){
   x<-as.numeric(as.character(x))
   x[is.na(x)] =median(x, na.rm=TRUE)
   x
}
dfOnlyNumeric=suppressWarnings(data.frame(apply(dfNoMismatch,2,f)))
dfOnlyNumeric$Title = dfNoMismatch$Title
dfOnlyNumeric <- dfOnlyNumeric[,colSums(is.na(dfOnlyNumeric))<nrow(dfOnlyNumeric)]
# Set gross values equal to zero to mean value of gross
index <- dfOnlyNumeric$Gross == 0
dfOnlyNumeric$Gross[index] <- mean(dfOnlyNumeric$Gross)
```

***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
colnames(dfNoMismatch)
```

```
##  [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"
```

```
#dimensions
length(dfNoMismatch[1,])
```

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

```
length(dfNoMismatch[,1])
```

```
## [1] 2742
```

Evaluation Strategy

In each of the tasks described in the next section, you will build a regression model. In order to compare their performance, use the following evaluation procedure every time:

1. Randomly divide the rows into two sets of sizes 5% and 95%.
2. Use the first set for training and the second for testing.
3. Compute the Root Mean Squared Error (RMSE) on the train and test sets.
4. Repeat the above data partition and model training and evaluation 10 times and average the RMSE results so the results stabilize.
5. Repeat the above steps for different proportions of train and test sizes: 10%-90%, 15%-85%, …, 95%-5% (total 19 splits including the initial 5%-95%).
6. Generate a graph of the averaged train and test RMSE as a function of the train set size (%).

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

Tasks

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

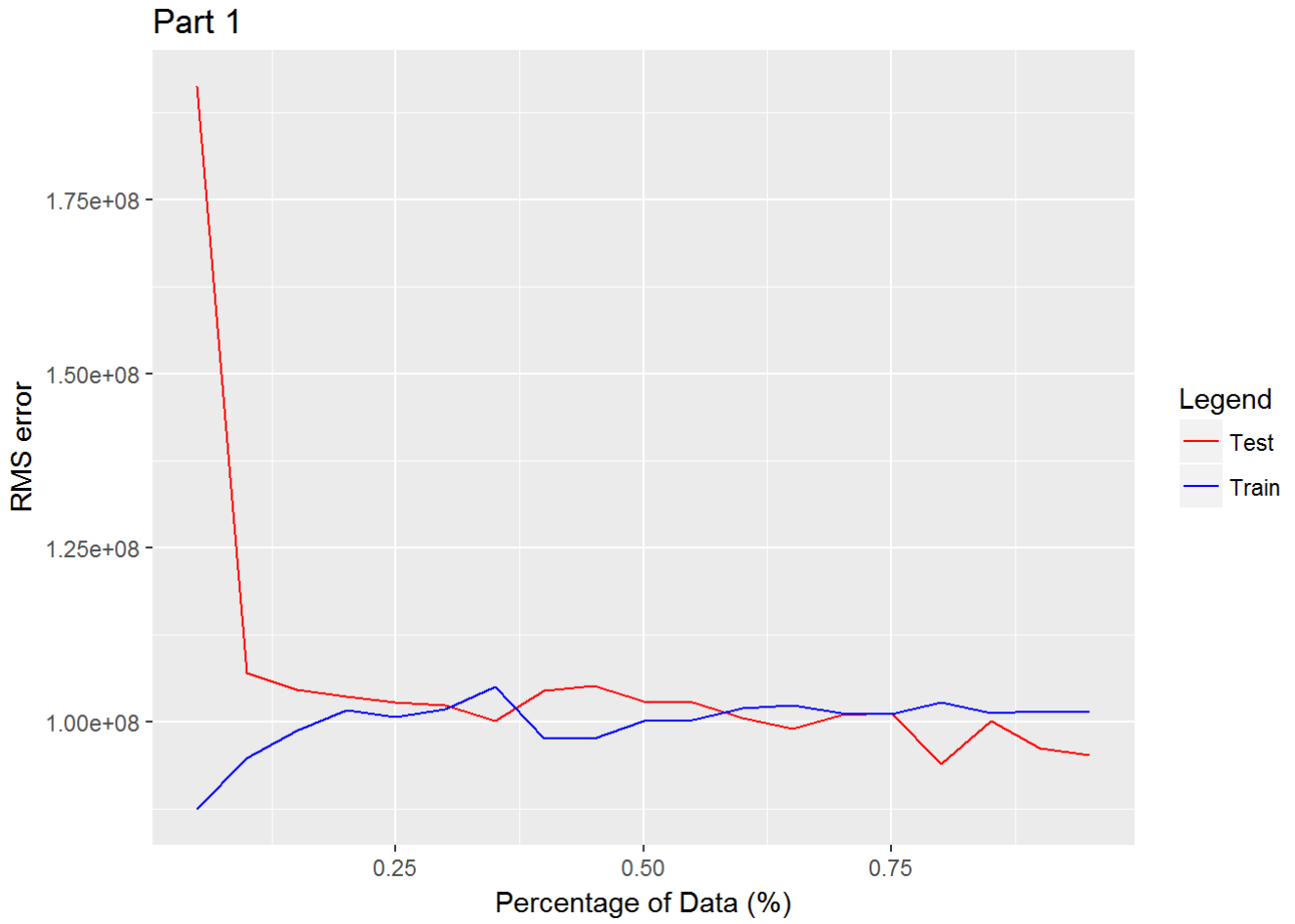
1. Numeric variables

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

```
# TODO: Build & evaluate model 1 (numeric variables only)
# Make gross the last column
colnames(dfOnlyNumeric)
```

```
##  [1] "Title"             "Year"              "Runtime"
##  [4] "Metascore"         "imdbRating"        "imdbVotes"
##  [7] "tomatoMeter"       "tomatoRating"      "tomatoReviews"
## [10] "tomatoFresh"       "tomatoRotten"      "tomatoUserMeter"
## [13] "tomatoUserRating"  "tomatoUserReviews" "Budget"
## [16] "Gross"             "Date"
```

```
tempColumn = dfOnlyNumeric$Gross
dfOnlyNumeric[,16] = dfOnlyNumeric$Date
colnames(dfOnlyNumeric)[16] = "Date"
dfOnlyNumeric[,17] = tempColumn
colnames(dfOnlyNumeric)[17] = "Gross"

onlyNumeric = dfOnlyNumeric[,3:17]
#Split Data to train and Test Data
set.seed(5)

percentages = seq(0.05, 0.95, 0.05)
learnCurve = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
for (perc in percentages) {
  currentRow = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
  for (i in 1:10) {
      onlyNumeric = onlyNumeric[sample(nrow(onlyNumeric)),]
      numericDataIndex <- createDataPartition(onlyNumeric$Gross, p = perc, list = F)
      numericDataTrain <- onlyNumeric[numericDataIndex,]
      numericDataTest <- onlyNumeric[-numericDataIndex,]
      #summary(dfOnlyNumeric)

      #fit model
      fit = suppressWarnings(lm(Gross~.,numericDataTrain))

      #base <- lm(Gross~.,numericDataTrain)
      #fit <- step(base) #http://machinelearningmastery.com/linear-regression-in-r/

      #summary(fit)
      #fit <- pcr(Gross~.,data = numericDataTrain, validation="CV")
      #predict on test data
      predictions_Test <- suppressWarnings(predict(fit, numericDataTest))

      #predict on train data
      prediction_Train <- suppressWarnings(predict(fit, numericDataTrain))

      #RMSE Test
      rmse_test <- sqrt(mean((numericDataTest$Gross - predictions_Test)^2))

      #RMSE Test
      rmse_train <- sqrt(mean((numericDataTrain$Gross - prediction_Train)^2))
      currentRow = rbind(currentRow,c(perc,rmse_test,rmse_train))
  }
  rowMean = colMeans(currentRow)

  learnCurve = rbind(learnCurve,rowMean)
}
colnames(learnCurve) = c('percentage','rmsetest','rmsetrain')
ggplot(learnCurve) + geom_line(aes(y = rmsetest,x=percentage, colour = "blue"))+
  geom_line(aes(y = rmsetrain,x=percentage, colour = "red"))+
  labs(x="Percentage of Data (%)", y="RMS error")+
  scale_color_manual(labels = c("Test", "Train"), values = c("red", "blue"))+
  guides(color=guide_legend("Legend"))+
  ggtitle("Part 1")
```



Q: List all the numeric variables you used.

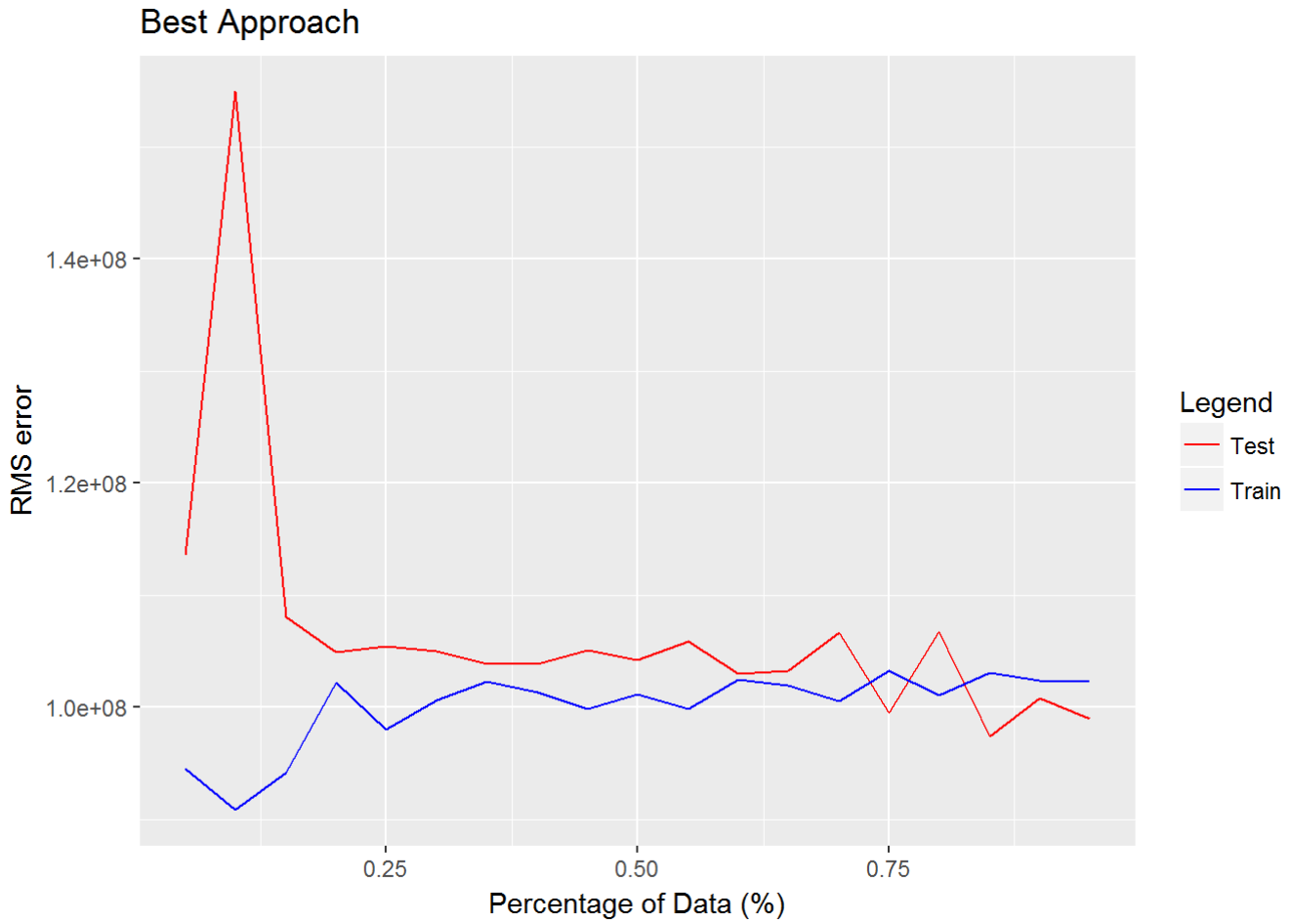
A: The variables used are: Runtime,Metascore, imdbRating, imdbVotes, tomatoMeter, tomatoRating, tomatoReviews, tomatoFresh, tomatoRotten, tomatoMeter, tomatoUserRating, tomatoUserReviews, Budget, Date

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)
onlyNumeric2 = onlyNumeric
onlyNumeric2$Runtime = as.numeric(cut(onlyNumeric2$Runtime,seq(0,240,20),labels=1:12))
onlyNumeric2$Metascore = as.numeric(cut(onlyNumeric2$Metascore,seq(0,100,10),labels=1:10))
onlyNumeric2$imdbRating = as.numeric(cut(onlyNumeric2$imdbRating,seq(0,10,1),labels=1:10))
onlyNumeric2$imdbVotes =
as.numeric(cut(onlyNumeric2$imdbVotes,seq(0,1680000,1000),labels=1:1680))
#onlyNumeric2$imdbVotes = log(onlyNumeric2$imdbVotes+1)
onlyNumeric2$tomatoMeter = as.numeric(cut(onlyNumeric2$tomatoMeter,seq(0,100,10),labels=1:10))
onlyNumeric2$tomatoRating = as.numeric(cut(onlyNumeric2$tomatoRating,seq(0,10,1),labels=1:10))
onlyNumeric2$tomatoReviews =
as.numeric(cut(onlyNumeric2$tomatoReviews,seq(0,360,20),labels=1:18))
onlyNumeric2$tomatoFresh = as.numeric(cut(onlyNumeric2$tomatoFresh,seq(0,360,20),labels=1:18))
onlyNumeric2$tomatoRotten = as.numeric(cut(onlyNumeric2$tomatoRotten,seq(0,260,20),labels=1:13))
onlyNumeric2$tomatoUserMeter =
as.numeric(cut(onlyNumeric2$tomatoUserMeter,seq(0,100,10),labels=1:10))
onlyNumeric2$tomatoUserReviews = as.numeric(cut(onlyNumeric2$tomatoUserReviews,seq(0,35000000,10
00),labels=1:35000))
#onlyNumeric2$tomatoUserReviews = log(onlyNumeric2$tomatoUserReviews+1)
onlyNumeric2$Budget = as.numeric(cut(onlyNumeric2$Budget,seq(0,500000000,500000),labels=1:1000))
#onlyNumeric2$Budget = log(onlyNumeric2$Budget+1)

for(i in 1:ncol(onlyNumeric2)){
  onlyNumeric2[is.na(onlyNumeric2[,i]), i] <- mean(onlyNumeric2[,i], na.rm = TRUE)
}

set.seed(5)

percentages = seq(0.05, 0.95, 0.05)
learnCurve = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
for (perc in percentages) {
  currentRow = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
  for (i in 1:10) {
      onlyNumeric2 = onlyNumeric2[sample(nrow(onlyNumeric2)),]
      numericDataIndex <- createDataPartition(onlyNumeric2$Gross, p = perc, list = F)
      numericDataTrain <- onlyNumeric2[numericDataIndex,]
      numericDataTest <- onlyNumeric2[-numericDataIndex,]
      #summary(dfOnlyNumeric)

      #fit model
      fit = suppressWarnings(lm(Gross~.,numericDataTrain))

      #base <- lm(Gross~.,numericDataTrain)
      #fit <- step(base) #http://machinelearningmastery.com/linear-regression-in-r/

      #summary(fit)
      #fit <- pcr(Gross~.,data = numericDataTrain, validation="CV")
      #predict on test data
      predictions_Test <- suppressWarnings(predict(fit, numericDataTest))

      #predict on train data
      prediction_Train <- suppressWarnings(predict(fit, numericDataTrain))
```

```
      #RMSE Test
      rmse_test <- sqrt(mean((numericDataTest$Gross - predictions_Test)^2))

      #RMSE Test
      rmse_train <- sqrt(mean((numericDataTrain$Gross - prediction_Train)^2))
      currentRow = rbind(currentRow,c(perc,rmse_test,rmse_train))
  }
  rowMean = colMeans(currentRow)

  learnCurve = rbind(learnCurve,rowMean)
}
colnames(learnCurve) = c('percentage','rmsetest','rmsetrain')
ggplot(learnCurve) + geom_line(aes(y = rmsetest,x=percentage, colour = "blue"))+
  geom_line(aes(y = rmsetrain,x=percentage, colour = "red"))+
  labs(x="Percentage of Data (%)", y="RMS error")+
  scale_color_manual(labels = c("Test", "Train"), values = c("red", "blue"))+
  guides(color=guide_legend("Legend"))+
  ggtitle("Best Approach")
```



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

A: I tried a combination of bins and transformations such as taking the log of large values compared to other values. Most numeric values are in the range of 10s or 100s. However, imdbVotes, tomatoUserReviews and Budget are in the range of millions so I thought of taking the log of the large values and convert the smaller values into bins of increments of 1 incase of 10s range or increments of 10 and 20 in case of 100s and 200-300s respectivly. The second approach I also tried is using bins for all values in proportion to their scale so the

millions values had bins of width 1000. A combination of binning only the variables of ranges upto 500 were binned while keeping other values as is. Finally, a fourth attepmt was tried by keeping all variables as is and taking the log of the larger variables. It was found that using all bins vs. the other approaches yielded better results so I went with that approach.

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)
onlyNumeric3 = onlyNumeric
# Add Awards
newAwards = gsub('s', '', dfNoMismatch$Awards)
numAward= vector(mode="numeric", length=length(newAwards))
numNomination = vector(mode="numeric", length=length(newAwards))

for(i in seq(1,length(newAwards))){
    current = newAwards[i]
    noNumbers = gsub('[0-9]+', '', newAwards[i])
    numberList = na.omit(as.numeric(unlist(strsplit(newAwards[i], "[^0-9]+"))))

    if (noNumbers==" win."){
      numAward[i] = numberList[1]
      numNomination[i]= 0
    }
    if (noNumbers=="N/A"){
      numAward[i] =0
      numNomination[i]=0
    }

if (noNumbers== " win &  nomination."){
      numAward[i] = numberList[1]
      numNomination[i]=numberList[2]
    }

if (noNumbers== " nomination."){
      numAward[i] = 0
      numNomination[i]= numberList[1]
    }

if (noNumbers== "Nominated for  Ocar. Another  win &  nomination." ){
      numAward[i] = numberList[2]
      numNomination[i]= numberList[1]+numberList[3]
    }
if (noNumbers=="Won  Ocar. Another  win &  nomination."){
      numAward[i] = numberList[1]+numberList[2]
      numNomination[i]= numberList[3]
    }
if (noNumbers=="Won  Golden Globe. Another  win &  nomination."){
      numAward[i] = numberList[1]+numberList[2]
      numNomination[i]=numberList[3]
    }
 if (noNumbers=="Nominated for  Primetime Emmy. Another  win &  nomination."){
      numAward[i] = numberList[2]
      numNomination[i]= numberList[1]+numberList[3]
    }
 if (noNumbers=="Nominated for  Golden Globe. Another  nomination."){
      numAward[i] = 0
      numNomination[i]=numberList[1]+numberList[2]
    }

 if (noNumbers== "Nominated for  Golden Globe. Another  win &  nomination."){
```

```
      numAward[i] = numberList[2]
      numNomination[i]= numberList[1]+numberList[3]
    }
 if (noNumbers=="Nominated for  BAFTA Film Award. Another  win &  nomination."){
     numAward[i] = numberList[2]
      numNomination[i]= numberList[1]+numberList[3]
    }
if (noNumbers=="Won  Primetime Emmy. Another  win &  nomination."){
     numAward[i] = numberList[1]+numberList[2]
      numNomination[i]= numberList[3]
    }

if (noNumbers== "Nominated for  Ocar. Another  win."){
      numAward[i] = numberList[2]
      numNomination[i]= numberList[1]
    }
 if (noNumbers=="Nominated for  Ocar. Another  nomination."){
      numAward[i] = 0
      numNomination[i]=numberList[1]+numberList[2]
    }
 if (noNumbers== "Won  Primetime Emmy. Another  nomination."){
      numAward[i] =numberList[1]
      numNomination[i]=numberList[2]
    }
 if (noNumbers== "Nominated for  BAFTA Film Award. Another  nomination."){
      numAward[i] = 0
      numNomination[i]= numberList[1]+numberList[2]
    }
 if (noNumbers== "Nominated for  BAFTA Film Award. Another  win."){
      numAward[i] = numberList[2]
      numNomination[i]=numberList[1]
    }
 if (noNumbers=="Nominated for  Primetime Emmy. Another  nomination."){
      numAward[i] = 0
      numNomination[i]= numberList[1]+numberList[2]
    }
 if (noNumbers=="Won  Ocar. Another  win."){
      numAward[i] = numberList[1]+numberList[2]
      numNomination[i]=0
    }
if (noNumbers=="Won  BAFTA Film Award. Another  win."){
      numAward[i] = numberList[1]+numberList[2]
      numNomination[i]= 0
    }
if (noNumbers== "Nominated for  Primetime Emmy. Another  win."){
      numAward[i] = numberList[2]
      numNomination[i]=numberList[1]
    }
 if (noNumbers== "Won  Golden Globe. Another  nomination."){
      numAward[i] = numberList[1]
      numNomination[i]=numberList[2]
    }
 if (noNumbers=="Won  BAFTA Film Award. Another  win &  nomination."){
      numAward[i] = numberList[1]+numberList[2]
```

```
      numNomination[i]=numberList[3]
    }
if (noNumbers=="Won  Ocar. Another  nomination."){
      numAward[i] = numberList[1]
      numNomination[i]=numberList[2]
    }
if (noNumbers=="Won  BAFTA Film Award. Another  nomination."){
      numAward[i] = numberList[1]
      numNomination[i]= numberList[2]
    }
if (noNumbers=="Nominated for  Golden Globe. Another  win."){
      numAward[i] = numberList[2]
      numNomination[i]=numberList[1]
    }
}
onlyNumeric3$numawards = numAward
onlyNumeric3$numNomination = numNomination

# Add Genres
suppressWarnings(library('tm'))
```

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

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

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

```
splitGenresToVector = function(genreString){
  stringVector = strsplit(genreString,", ")
  return (stringVector)
}

genreColumn = vector(length=length(dfNoMismatch$Genre))

i = 1
for (entry in dfNoMismatch$Genre){
  genreColumn[i] = splitGenresToVector(entry)
  i=i+1
}

allGenres = c()
for(i in seq(1,length(dfNoMismatch$Genre))){
  temp = genreColumn[[i]]
  for(genre in temp){
    allGenres = c(allGenres,genre)
  }
}
uniqueGenres = unique(allGenres)

onlyNumeric3$Documentary = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Biography = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Romance = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$short = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Thriller = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Drama = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Documentary = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$War = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Comedy = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Horror = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$scifi = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Adventure = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Family = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$History = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Crime = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Action = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Music = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Mystery = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Fantasy = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Sport = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Animation = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Musical = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Talkshow = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Adult = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$western = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Filmnoir = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$Realitytv = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$News = vector(mode="numeric", length=length(dfNoMismatch$Genre))
onlyNumeric3$GameShow = vector(mode="numeric", length=length(dfNoMismatch$Genre))

for(i in seq(1,length(dfNoMismatch$Genre))){
```

```
temp = genreColumn[[i]]
for(genre in temp){
  if (genre =="Documentary"){
    onlyNumeric3$Documentary[i] = 1
  } else{
    onlyNumeric3$Documentary[i] = 0
  }
  if (genre=="Biography"){
      onlyNumeric3$Biography[i] = 1
  }else{
      onlyNumeric3$Biography[i] = 0
  }
  if (genre=="Romance"){
    onlyNumeric3$Romance[i] = 1
  }else{
    onlyNumeric3$Romance[i] = 0
  }
  if(genre=="Short"){
    onlyNumeric3$short[i] = 1
  }else{
    onlyNumeric3$short[i] = 0
  }
if(genre== "Thriller"){
  onlyNumeric3$Thriller[i] = 1
}else{
  onlyNumeric3$Thriller[i] = 0}

  if(genre=="Drama" ){
   onlyNumeric3$Drama[i] = 1
  }else{onlyNumeric3$Drama[i] = 0}
if(genre=="War"){
  onlyNumeric3$War[i] = 1}
  else{onlyNumeric3$War[i] = 0}

  if(genre=="Comedy"){
    onlyNumeric3$Comedy[i]=1}
  else{onlyNumeric3$Comedy[i]=0
  }
  if(genre=="Horror"){
    onlyNumeric3$Horror[i] = 1
  }else{onlyNumeric3$Horror[i] = 0}

  if(genre=="Sci-Fi"){
    onlyNumeric3$scifi[i] = 1
  }else{onlyNumeric3$scifi[i] = 0}

  if(genre=="Adventure"){
    onlyNumeric3$Adventure[i]=1}
  else{onlyNumeric3$Adventure[i]=0
  }

  if(genre=="Family"){
    onlyNumeric3$Family[i]= 1}
  else{onlyNumeric3$Family[i]= 0
```

```
  }
  if(genre=="History"){onlyNumeric3$History[i] = 1
  }else{onlyNumeric3$History[i] = 0
  }
  if(genre=="Crime"){onlyNumeric3$Crime[i]=1
  }else{onlyNumeric3$Crime[i]=0
  }
  if(genre=="Action"){onlyNumeric3$Action[i] = 1
  }else{onlyNumeric3$Action[i] = 0
  }
  if(genre=="Music"){
    onlyNumeric3$Music[i] = 1
  }else{onlyNumeric3$Music[i] = 0}

  if(genre=="Mystery"){onlyNumeric3$Mystery[i] = 1
  }else{onlyNumeric3$Mystery[i] = 0}

  if(genre=="Fantasy"){
    onlyNumeric3$Fantasy[i] = 1
  }else{onlyNumeric3$Fantasy[i] = 0}

  if(genre=="Sport"){
    onlyNumeric3$Sport[i] = 1
  }else{onlyNumeric3$Sport[i] = 0}

  if(genre=="Animation"){
    onlyNumeric3$Animation[i] = 1
  }else{onlyNumeric3$Animation[i] = 0}
  if(genre=="Musical"){onlyNumeric3$Musical[i] = 1
  }else{
    onlyNumeric3$Musical[i] = 0}
  if(genre=="Talk-Show"){onlyNumeric3$Talkshow[i] = 1
  }else{onlyNumeric3$Talkshow[i] = 0}

  if(genre=="Adult"){onlyNumeric3$Adult[i] = 1
  }else{onlyNumeric3$Adult[i] = 0}

  if(genre=="Western"){onlyNumeric3$western[i] = 1
  }else{onlyNumeric3$western[i] = 0}

  if(genre=="Film-Noir"){onlyNumeric3$Filmnoir[i] = 1
  }else{onlyNumeric3$Filmnoir[i] = 0}

  if(genre=="Reality-TV"){onlyNumeric3$Realitytv[i] = 1
  }else{onlyNumeric3$Realitytv[i] = 0}

  if(genre=="News"){onlyNumeric3$News[i] = 1
  }else{onlyNumeric3$News[i] = 0}

  if(genre=="Game-Show"){
    onlyNumeric3$GameShow[i] = 1
  }else{onlyNumeric3$GameShow[i] = 0}

}
```

```
}
# Add top 30 Occuring Actors by summing the total of them in a movie
ActorColumn = vector(length=length(dfNoMismatch$Actors))
ActorNum = vector(length=length(dfNoMismatch$Actors))

i = 1
for (entry in dfNoMismatch$Actors){
  ActorColumn[i] = splitGenresToVector(entry)
  ActorNum[i] = 0
  i=i+1
}

allActors = c()
for(i in seq(1,length(dfNoMismatch$Actors))){
  temp = ActorColumn[[i]]
  for(actr in temp){
    allActors = c(allActors,actr)
  }
}
maximum_actors = sort(table(allActors), decreasing = TRUE)
topthirtyActors = c("Robert De Niro","Mark Wahlberg","Owen Wilson","Adam Sandler","Bruce
Willis","Johnny Depp","Matt Damon","Ben Stiller","Samuel L. Jackson","Cameron Diaz","Jack
Black","Will Ferrell","George Clooney","Gerard Butler","Jason Statham","Nicolas Cage","Robert Do
wney Jr.","Steve Carell","Brad Pitt","Charlize Theron","Angelina Jolie Pitt","Christian Bale","G
reg Kinnear","Nicole Kidman","Woody Harrelson","Ben Affleck","Channing Tatum","Dwayne
Johnson","Eddie Murphy")

for(i in seq(1,length(dfNoMismatch$Actors))){
  temp = ActorColumn[[i]]
  for(actr in temp){
    if (actr %in% topthirtyActors){
      ActorNum[i] = ActorNum[i] + 1
    }

  }

}

onlyNumeric3$actorNumbers = ActorNum

# Add Number of Languages
languageColumn = vector(length=length(dfNoMismatch$Language))
numlanguageColumn = vector(length=length(dfNoMismatch$Language))

i = 1
for (entry in dfNoMismatch$Language){
  languageColumn[i] = splitGenresToVector(entry)
  numlanguageColumn[i] = length(languageColumn[[i]])
  i=i+1
}
onlyNumeric3$NumLanguages = numlanguageColumn

onlyNonNumerical = onlyNumeric3[,15:47]
```

```
for(i in 1:ncol(onlyNonNumerical)){
  onlyNonNumerical[is.na(onlyNonNumerical[,i]), i] <- mean(onlyNonNumerical[,i], na.rm = TRUE)
}

set.seed(5)

percentages = seq(0.05, 0.95, 0.05)
learnCurve = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
for (perc in percentages) {
  currentRow = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
  for (i in 1:10) {
      onlyNonNumerical = onlyNonNumerical[sample(nrow(onlyNonNumerical)),]
      numericDataIndex <- createDataPartition(onlyNonNumerical$Gross, p = perc, list = F)
      numericDataTrain <- onlyNonNumerical[numericDataIndex,]
      numericDataTest <- onlyNonNumerical[-numericDataIndex,]
      #summary(dfOnlyNumeric)

      #fit model
      fit = suppressWarnings(lm(Gross~.,numericDataTrain))

      #base <- lm(Gross~.,numericDataTrain)
      #fit <- step(base) #http://machinelearningmastery.com/linear-regression-in-r/

      #summary(fit)
      #fit <- pcr(Gross~.,data = numericDataTrain, validation="CV")
      #predict on test data
      predictions_Test <- suppressWarnings(predict(fit, numericDataTest))

      #predict on train data
      prediction_Train <- suppressWarnings(predict(fit, numericDataTrain))

      #RMSE Test
      rmse_test <- sqrt(mean((numericDataTest$Gross - predictions_Test)^2))

      #RMSE Test
      rmse_train <- sqrt(mean((numericDataTrain$Gross - prediction_Train)^2))
      currentRow = rbind(currentRow,c(perc,rmse_test,rmse_train))
  }
  rowMean = colMeans(currentRow)

  learnCurve = rbind(learnCurve,rowMean)
}
colnames(learnCurve) = c('percentage','rmsetest','rmsetrain')
ggplot(learnCurve) + geom_line(aes(y = rmsetest,x=percentage, colour = "blue"))+
  geom_line(aes(y = rmsetrain,x=percentage, colour = "red"))+
  labs(x="Percentage of Data (%)", y="RMS error")+
  scale_color_manual(labels = c("Test", "Train"), values = c("red", "blue"))+
  guides(color=guide_legend("Legend"))+
  ggtitle("Non Numerical")
```


```
# Adding Director did not improve results
```

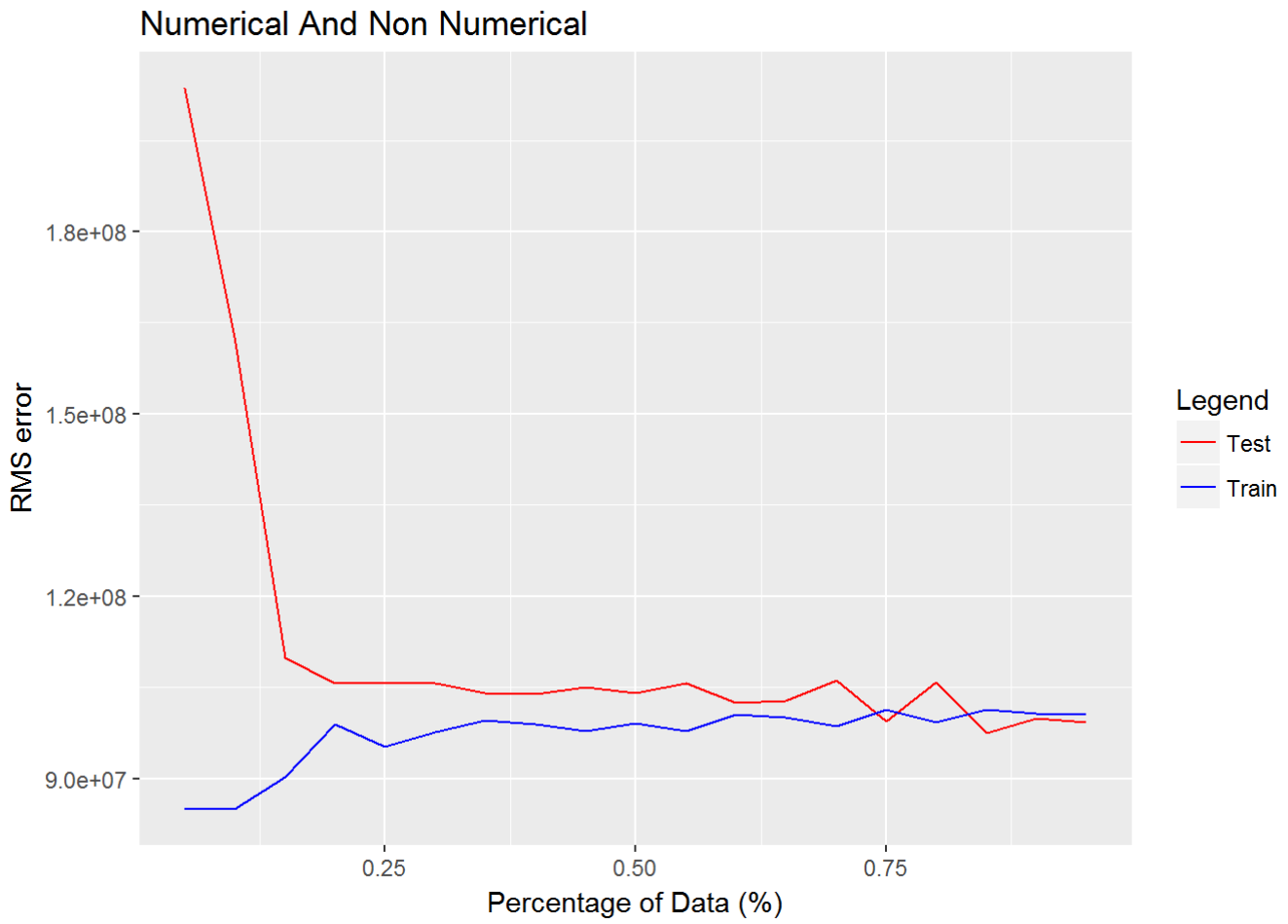
Q: Explain which categorical variables you used, and how you encoded them into features.

A: I have used genres which I added using one hot encoding, I also used languages where I counted the total number of languages and added that as a feature. I used the awards where I added two features one corrosponding to the total number of awards and the other to the total number of nominations. I used actors as a feature but one hot encoding will not work since there are more than 5000 different ones so I looked at top 30 actors and saw how many of them are participating in the film as a feature. Finally, adding directors in a similar fashion did not seem to have a big effect on the results so they were neglected.

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**.

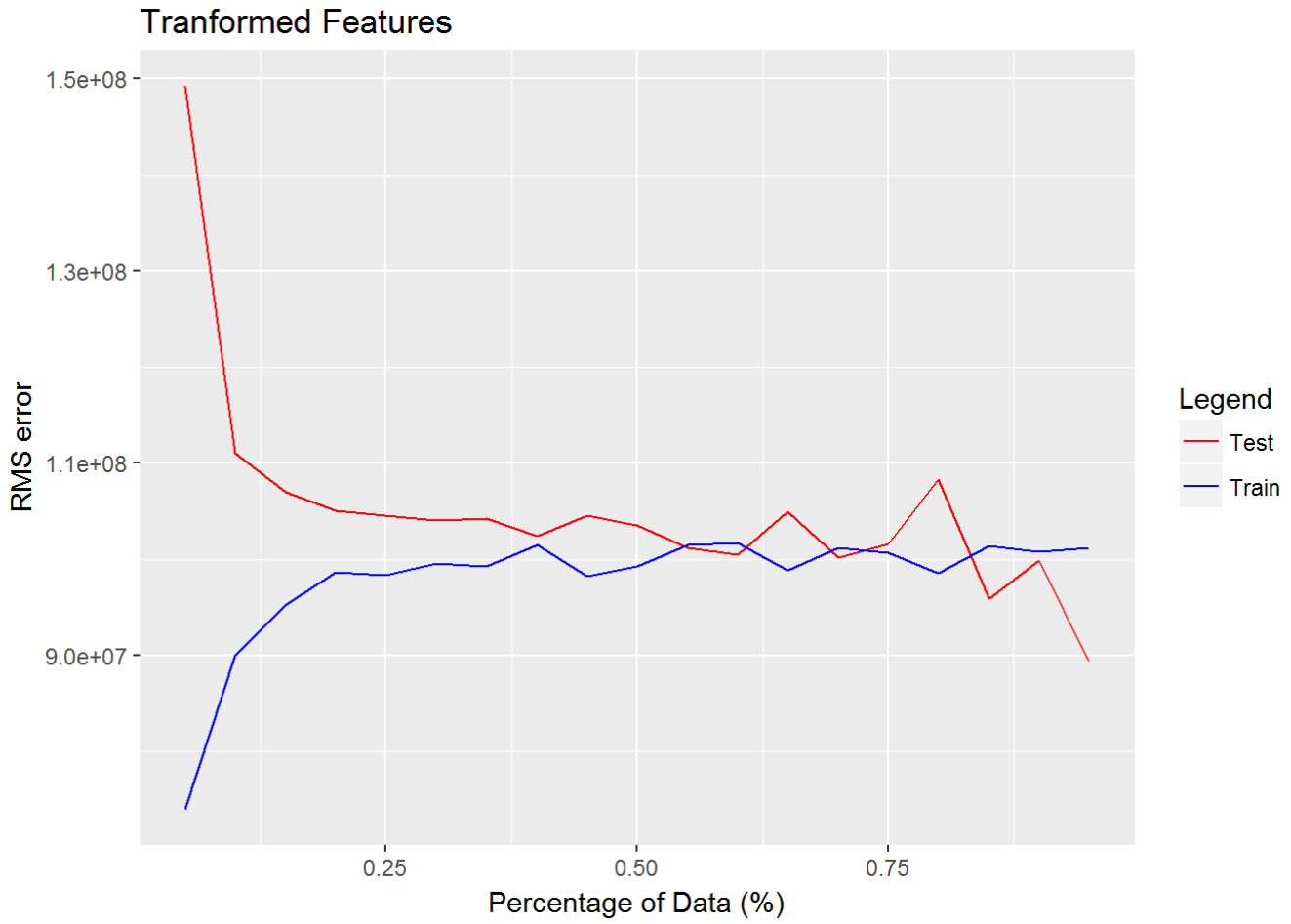
```r
# TODO: Build & evaluate model 4 (numeric & converted non-numeric variables)
allCombined = onlyNumeric3
for(i in 1:ncol(allCombined)){
  allCombined[is.na(allCombined[,i]), i] <- mean(allCombined[,i], na.rm = TRUE)
}

set.seed(5)

percentages = seq(0.05, 0.95, 0.05)
learnCurve = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
for (perc in percentages) {
  currentRow = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
  for (i in 1:10) {
      allCombined = allCombined[sample(nrow(allCombined)),]
      numericDataIndex <- createDataPartition(allCombined$Gross, p = perc, list = F)
      numericDataTrain <- allCombined[numericDataIndex,]
      numericDataTest <- allCombined[-numericDataIndex,]
      #summary(dfOnlyNumeric)

      #fit model
      fit = suppressWarnings(lm(Gross~.,numericDataTrain))

      #base <- lm(Gross~.,numericDataTrain)
      #fit <- step(base) #http://machinelearningmastery.com/linear-regression-in-r/

      #summary(fit)
      #fit <- pcr(Gross~.,data = numericDataTrain, validation="CV")
      #predict on test data
      predictions_Test <- suppressWarnings(predict(fit, numericDataTest))

      #predict on train data
      prediction_Train <- suppressWarnings(predict(fit, numericDataTrain))

      #RMSE Test
      rmse_test <- sqrt(mean((numericDataTest$Gross - predictions_Test)^2))

      #RMSE Test
      rmse_train <- sqrt(mean((numericDataTrain$Gross - prediction_Train)^2))
      currentRow = rbind(currentRow,c(perc,rmse_test,rmse_train))
  }
  rowMean = colMeans(currentRow)

  learnCurve = rbind(learnCurve,rowMean)
}
colnames(learnCurve) = c('percentage','rmsetest','rmsetrain')
ggplot(learnCurve) + geom_line(aes(y = rmsetest,x=percentage, colour = "blue"))+
  geom_line(aes(y = rmsetrain,x=percentage, colour = "red"))+
  labs(x="Percentage of Data (%)", y="RMS error")+
  scale_color_manual(labels = c("Test", "Train"), values = c("red", "blue"))+
  guides(color=guide_legend("Legend"))+
  ggtitle("Numerical And Non Numerical")
```



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(5)

percentages = seq(0.05, 0.95, 0.05)
learnCurve = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
for (perc in percentages) {
  currentRow = data.frame(percentage= numeric(0), rmsetest= numeric(0), rmsetrain= numeric(0))
  for (i in 1:10) {
      allCombined = allCombined[sample(nrow(allCombined)),]
      numericDataIndex <- createDataPartition(allCombined$Gross, p = perc, list = F)
      numericDataTrain <- allCombined[numericDataIndex,]
      numericDataTest <- allCombined[-numericDataIndex,]
      #summary(dfOnlyNumeric)

      #fit model
      fit = pcr(Gross~.,data = numericDataTrain,validation="CV")

      #base <- lm(Gross~.,numericDataTrain)
      #fit <- step(base) #http://machinelearningmastery.com/linear-regression-in-r/

      #summary(fit)
      #fit <- pcr(Gross~.,data = numericDataTrain, validation="CV")
      #predict on test data
      predictions_Test <- suppressWarnings(predict(fit, numericDataTest,ncomp=30))

      #predict on train data
      prediction_Train <- suppressWarnings(predict(fit, numericDataTrain,ncomp=30))

      #RMSE Test
      rmse_test <- sqrt(mean((numericDataTest$Gross - predictions_Test)^2))

      #RMSE Test
      rmse_train <- sqrt(mean((numericDataTrain$Gross - prediction_Train)^2))
      currentRow = rbind(currentRow,c(perc,rmse_test,rmse_train))
  }
  rowMean = colMeans(currentRow)

  learnCurve = rbind(learnCurve,rowMean)
}
colnames(learnCurve) = c('percentage','rmsetest','rmsetrain')
ggplot(learnCurve) + geom_line(aes(y = rmsetest,x=percentage, colour = "klj"))+
  geom_line(aes(y =rmsetrain, x=percentage, colour = "lkj"))+
  labs(x="Percentage of Data (%)", y="RMS error")+
  scale_color_manual(labels = c("Test", "Train"), values = c("red", "blue"))+
  guides(color=guide_legend("Legend"))+
  ggtitle("Tranformed Features")
```



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

A: I have decided to tranform my features to try and reduce my dimensionality as explained below.

Since many of the features seemed redundant, I have used Principal Component Regression which creates a linear regression model using the outputs of a Principal Component Analysis (PCA) to estimate the coefficients of the model. This model is especially useful when the dataset contains highly correlated predictors which is the case when looking at the columns of ratings of different imdb and rottontomato components. I tried predicting by variying the number of components I tried 30, 20, 10 and 5. Beyond 20, the accuracy seemed to drop where as using 30 components resulted in a higher accuracy. By inspecting the resulting eigenvalues, components with zero or near zero (<0.01 of total eigenvalues) tell us that this component is insignificant and can be removed.