Project 2: Modeling and Evaluation

CSE6242 - Data and Visual Analytics - Fall 2017

Due: Sunday, April 22, 2018 at 11:59 PM UTC-12:00 on T-Square

James Corwell

GTID: jcorwell3

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

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

Setup

Same as Project 1, load the dataset into memory:

```
load('movies_merged')
```

This creates an object of the same name (`movies_merged`). For convenience, you can copy it to `df` and start using it:

```
df = movies_merged
cat("Dataset has", dim(df)[1], "rows and", dim(df)[2], "columns", end="\n", file="")
```

```
## Dataset has 40789 rows and 39 columns
```

```
colnames(df)
```

```
##  [1] "Title"             "Year"              "Rated"
##  [4] "Released"          "Runtime"           "Genre"
##  [7] "Director"          "Writer"            "Actors"
## [10] "Plot"              "Language"          "Country"
## [13] "Awards"            "Poster"            "Metascore"
## [16] "imdbRating"        "imdbVotes"         "imdbID"
## [19] "Type"              "tomatoMeter"       "tomatoImage"
## [22] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [25] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [28] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [31] "DVD"               "BoxOffice"         "Production"
## [34] "Website"           "Response"          "Budget"
## [37] "Domestic_Gross"    "Gross"             "Date"
```

Load R packages

Load any R packages that you will need to use. You can come back to this chunk, edit it and re-run to load any additional packages later.

```
library(tm)
```

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

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

```
df=df[df$Type=="movie", ]
```

2. Drop rows with missing `Gross` value

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

```
df=df[!is.na(df$Gross),]
```

3. Exclude movies released prior to 2000

Inflation and other global financial factors may affect the revenue earned by movies during certain periods of time. Taking that into account is out of scope for this project, so let's exclude all movies that were released prior to the year 2000 (you may use `Released`, `Date` or `Year` for this purpose).

```
df = df[!(df$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.

```
#rows with na value for release date
naReleased = rownames(df[is.na(df$Released),])

#rid na dates from movies data
dates = setdiff(rownames(df),naReleased)
df = df[dates,]
print('Rows minus N/A rows')
```

```
## [1] "Rows minus N/A rows"
```

```
dim(df)[1]
```

```
## [1] 3122
```

```
#rid mismatched dates from dadta
df=df[abs((df$Year-as.numeric(substr(df$Released, 1, 4))))<=1,]
print('Rows minus mismatached rows')
```

```
## [1] "Rows minus mismatached rows"
```

```
dim(df)[1]
```

```
## [1] 3050
```

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.

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

6. Process `Runtime` column

```
## case 1 hr, case 2 hr/min, case 3 min, case 4 N/A

##iterates over each element and changes accordingly
minRun = c()
for(i in seq(df$Runtime)){
   runtime = strsplit(df$Runtime[i], ' ')[[1]]

      if (length(runtime) ==  2){

           #case 3 only minutes

          if(runtime[2] == "min")
          {minRun = suppressWarnings(c(minRun, as.integer(runtime[1])))}

          # case 1 hours

          else{minRun = suppressWarnings(c(minRun, (as.integer(runtime[1])) * 60))}
      }

      # case 2 hours/minutes

      else if( length((runtime) == 4)) {
        minRun = suppressWarnings(c(minRun, as.integer(runtime[1]) * 60  + as.integer(ru
ntime[3])))
      }

       # case 1 N/A
      else minRun = c(minRun, NA)

}

#replace original runtime data.
 df$Runtime <- minRun
```

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).

```
#no rows with na
df <- na.omit(df)


#copy into seperate dataframe for use in later parts
df2 = df
```

***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!)

```
cat(dim(df)[1],"rows",dim(df)[2],"columns\n")
```

```
## 2642 rows 38 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"
```

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

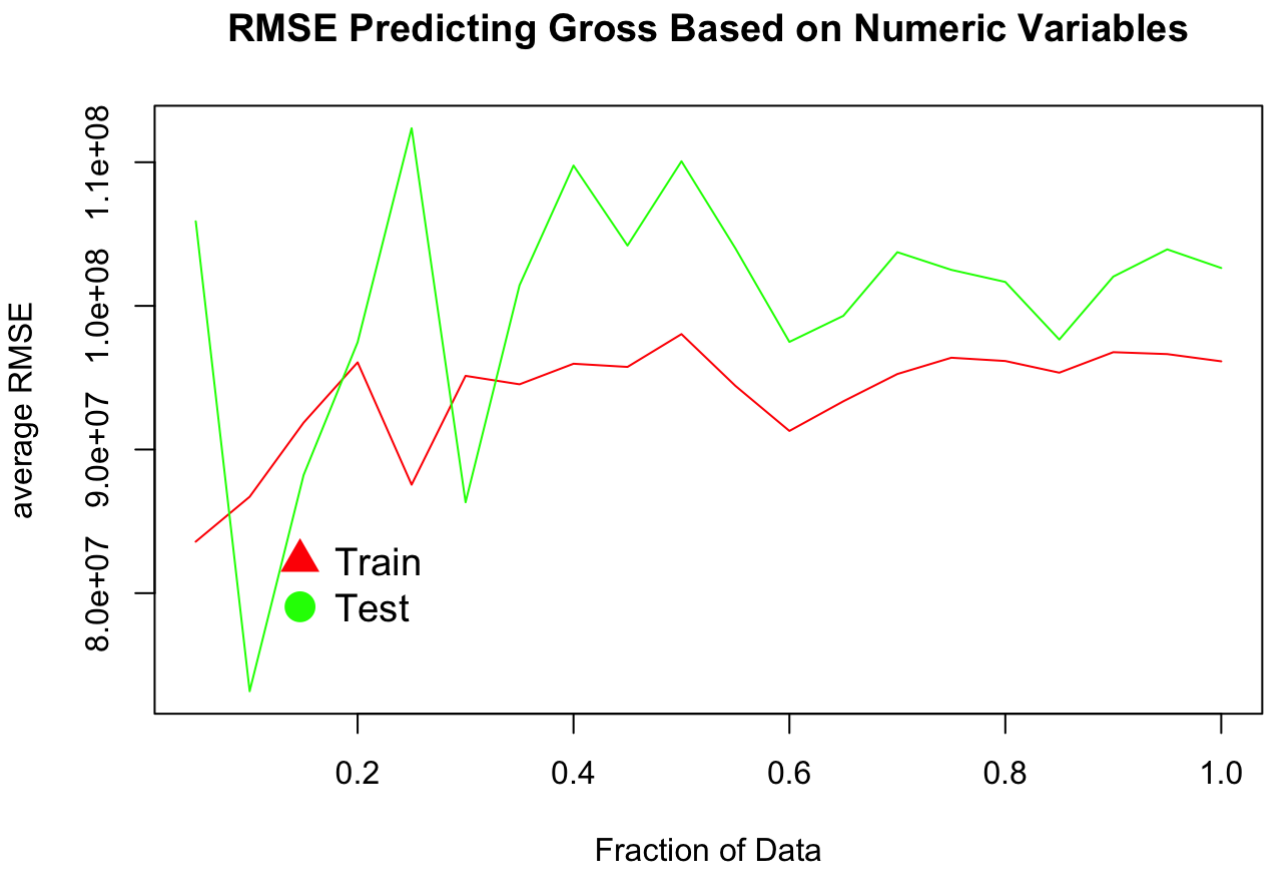
Each of the following tasks is worth 20 points, for a total of 100 points for this project. Remember to build each model as specified, evaluate it using the strategy outlined above, and plot the training and test errors by training set size (%).

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.

```
#use only numeric variables
  df<-df[,which(sapply(df,is.numeric))]

#remove tomato rotten (giving NA values)
  df <- subset(df, select = -tomatoRotten)


RMSE = function(predict, true){sqrt(mean((predict - true)^2)) }
calcRMSE<-function(train,test,percent){

  train_rmse = c()
  test_rmse = c()
  for(i in 1:10){

  #subset randomly
  train_subset = as.data.frame(head(train[sample(nrow(train)),],percent*nrow(train)))
  test_subset = as.data.frame(head(test[sample(nrow(test)),],percent*nrow(test)))

  #train model on all available numeric variables
  model = lm(Gross~., train_subset)

  #predict test and train values

  test_predict  <- suppressWarnings(predict(model,test_subset))
  train_predict <- suppressWarnings(predict(model,train_subset))

  #Calculate RMSE
  RMSE_test = RMSE(test_predict,test_subset$Gross)
  RMSE_train = RMSE(train_predict, train_subset$Gross)

  #append to array of trial values to be averaged upon return
  train_rmse <- c(train_rmse,RMSE_train)
  test_rmse <- c(test_rmse,RMSE_test)
  }
  return(c(mean(train_rmse),mean(test_rmse)))
  }

##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(df), 0.8*nrow(df))  # row indices for training data
train <- df[trainingRowIndex, ]  # training data
test  <- df[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}
```

```
plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="RMSE Predicting Gross Based on Numeric Variables",ylim=c(min(cbind(arrTrain,arrTe
st)),max(cbind(arrTrain,arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```



Q: List the numeric variables you used.

A: [1] “Year” “Runtime” “imdbRating” “imdbVotes” “tomatoMeter” “tomatoRating”
[7] “tomatoReviews” “tomatoFresh” “tomatoUserMeter” “tomatoUserRating” “tomatoUserReviews” “Budget”
[13] “Gross” “Date”

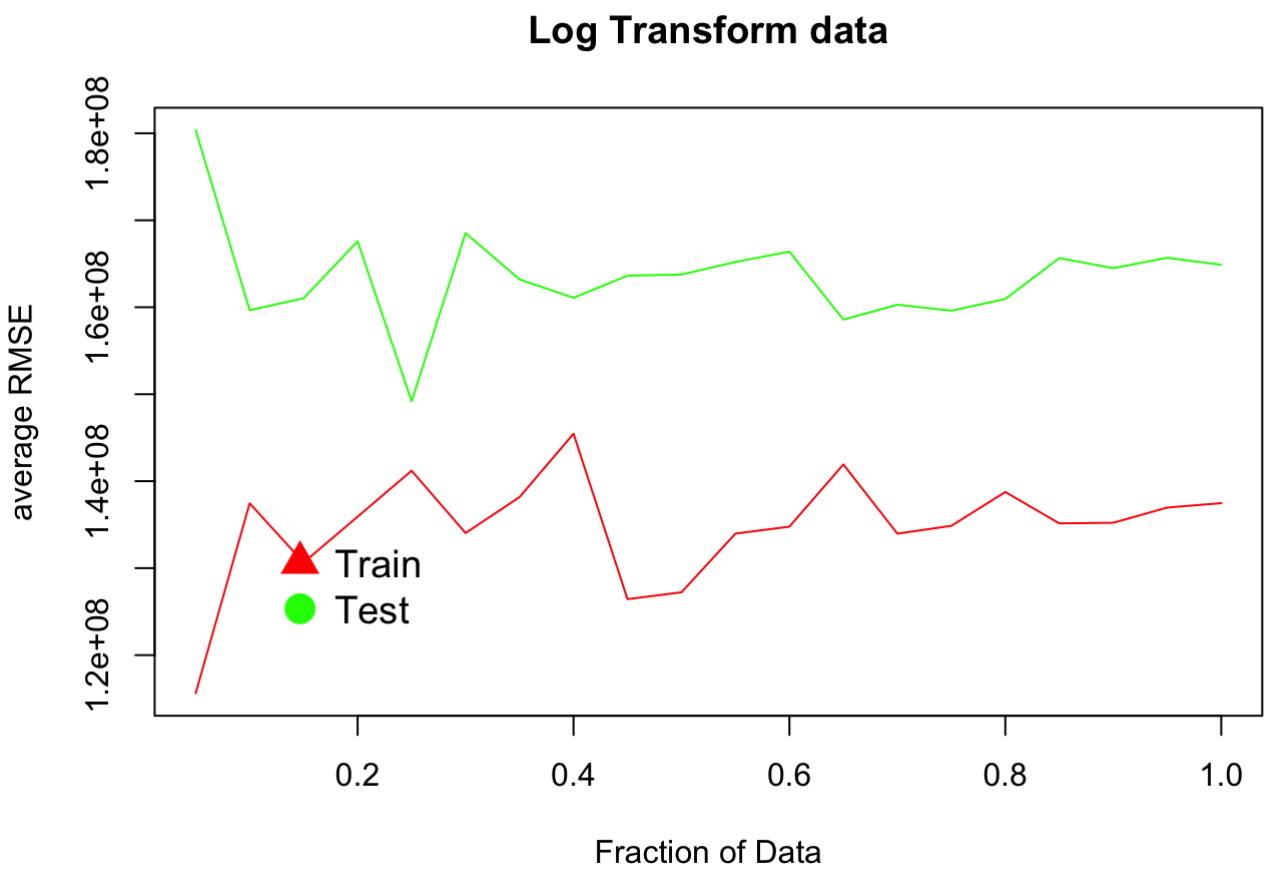
Q: What is the best mean test RMSE value you observed, and at what training set size?

A: Though my code changes for each run, in one run I found the lowest RMSE of the train data at .45% of data with 9.5e+07.

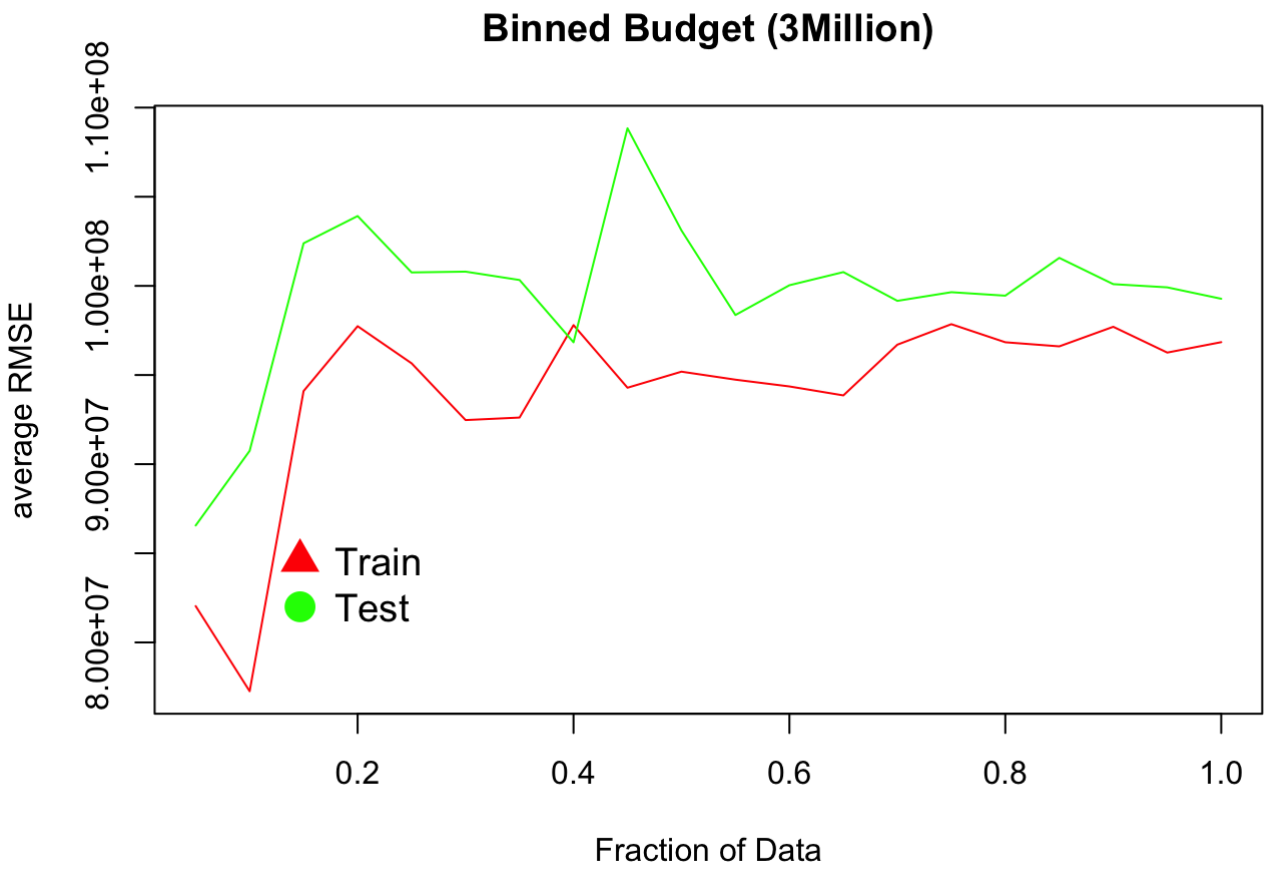
2. Feature transformations

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`).

```
#Logarithm transformation of all of the data

dflog <- log(df)
dflog$Gross <- df$Gross #Use Gross, not log(Gross)
#remove -inf
dflog<-dflog[!(dflog$tomatoMeter=="-Inf"),]
dflog<-dflog[!(dflog$tomatoFresh=="-Inf"),]
dflog<-dflog[!(dflog$Gross=="-Inf"),]
#names(dflog) <- paste0(names(df), "_log")

##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(dflog), 0.8*nrow(dflog))  # row indices for training d
ata
train <- dflog[trainingRowIndex, ]  # training data
test  <- dflog[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main = "Log Transform data",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrain,arrT
est))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```


Log Transform data
average RMSE
1.2e+08
1.4e+08
1.6e+08
1.8e+08
0.2
0.4
0.6
0.8
1.0
Fraction of Data
Train
Test

```
#now lets try some binning (classifies it into high(1) or low(0) budget move)
dfbin = df

for(i in 1:nrow(dfbin)){
  if(dfbin$Budget[i]>=3000000)
    {dfbin$highbudget[i]<- 1}
  else
  {dfbin$highbudget[i] <- 0}
}

#we can select which bins we want to analyze but "cutting"" the data according to each b
in.
dfbin_highbudget <- dfbin[which(dfbin$highbudget==1),]
dfbin_lowbudget <- dfbin[which(dfbin$highbudget==0),]

##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(dfbin), 0.8*nrow(dfbin))  # row indices for training d
ata
train <- dfbin[trainingRowIndex, ]  # training data
test  <- dfbin[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="Binned Budget (3Million)",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrain,
arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```

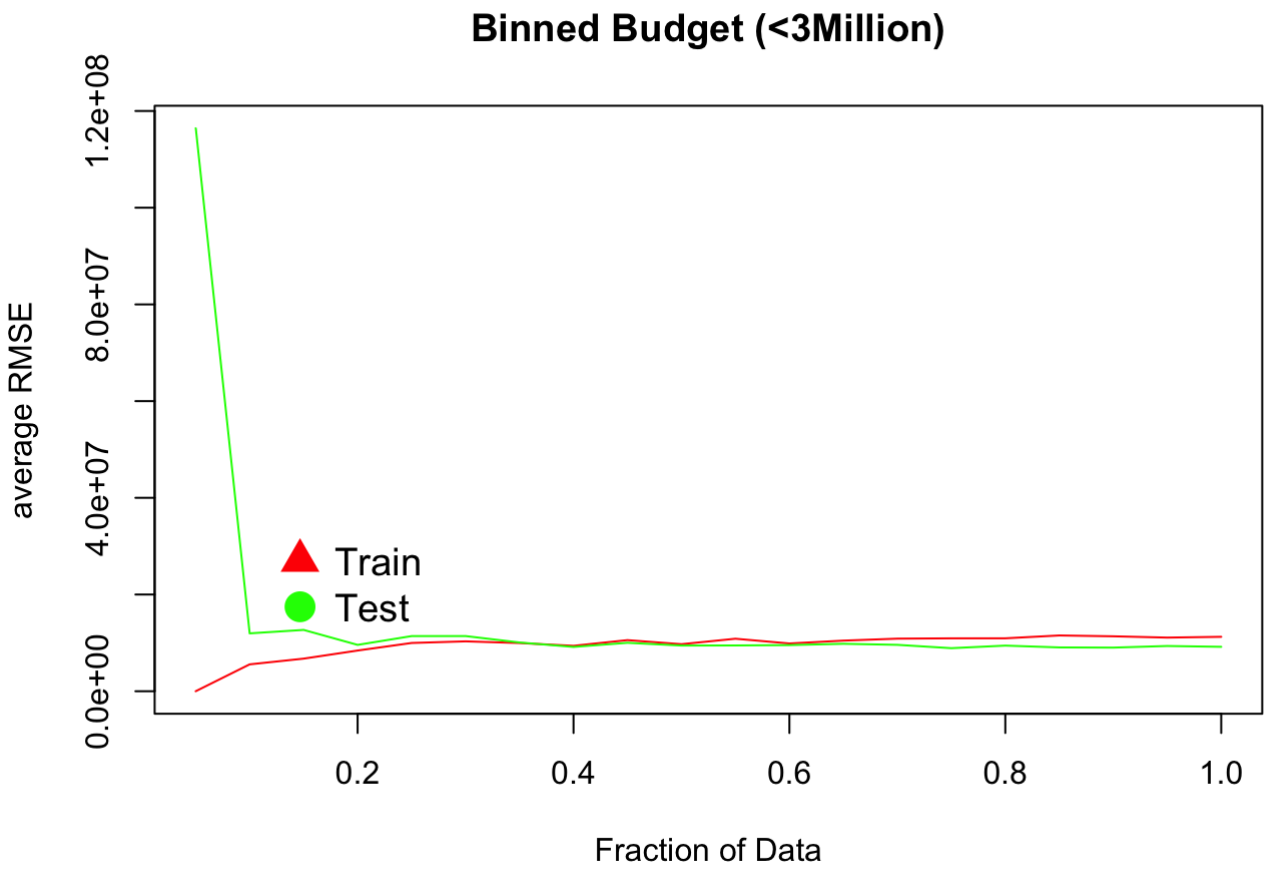


#High budget only data

```
##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(dfbin_highbudget), 0.8*nrow(dfbin_highbudget))  # row
 indices for training data
train <- dfbin_highbudget[trainingRowIndex, ]  # training data
test  <- dfbin_highbudget[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="Binned Budget (>3Million)",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrai
n,arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```

Low budget only data

```
##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(dfbin_lowbudget), 0.8*nrow(dfbin_lowbudget))  # row in
dices for training data
train <- dfbin_lowbudget[trainingRowIndex, ]  # training data
test  <- dfbin_lowbudget[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="Binned Budget (<3Million)",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrai
n,arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```



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

A: First, I wanted to try a log transformation over the data to see if a simple transform could generate a lower RMSE, and it turns out that this wasn't the case. Then I chose to conduct binning based on budget. Out of curiosity, I broke the data up into low budget (under 3 million) and high budget datasets to determine behavior.

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

A: The Log transformed data performed significantly worse than than the data from task 1, while the binned data performed marginally better for the majority of runs. After seperating the data into two different sets based on their budget and plotting, we can see that >3 million is roughly equal to the non-transformed data, while <3 million is significantly lower, we could assume this as an outcome, but it's interesting to verify. ## 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.

```
#Convert Awards to 2 numeric columns: wins and nominations

#initialize an empty data frame for awards and nominations
awardsDF = data.frame(matrix(ncol = 2, nrow = dim(df2)[1] ))
colnames(awardsDF) = c("win", "nomination")

#loop over every element in the data

for (i in 1:dim(awardsDF)[1]){

  #split phrases into words
  disambig = unlist(strsplit(df2[i, "Awards"], split=" "))



  if(length(disambig) < 2) next

  for (j in 1:(length(disambig) - 1)){

    if (grepl("w",disambig[j+1])) {
      numWins = suppressWarnings(as.numeric(disambig[j]))
      awardsDF[i, "win"] = numWins

    }


    if (grepl("n",disambig[j+1])) {
        numNominations = suppressWarnings(as.numeric(disambig[j]))
        awardsDF[i, "nomination"] = numNominations
    }
  }


}

df2 = merge(awardsDF,df2,by=0,all=TRUE)
```

```
# Replace Genre with a collection of binary columns

##manipulate genre as document
genre = VCorpus(VectorSource(df2$Genre))
##remove NAs && punctuation && lowercase
genre = tm_map(tm_map(tm_map(genre, removeWords, c("N/A")), removePunctuation), content_
transformer(tolower))


##frequency of occurences (generate docterm mat.)
genreDTM = DocumentTermMatrix(genre)

genreMatrix = as.data.frame(as.matrix(genreDTM))

df2 = merge(genreMatrix, df2, by=0, all=TRUE)
```

```
## Warning in merge.data.frame(genreMatrix, df2, by = 0, all = TRUE): column
## name 'Row.names' is duplicated in the result
```

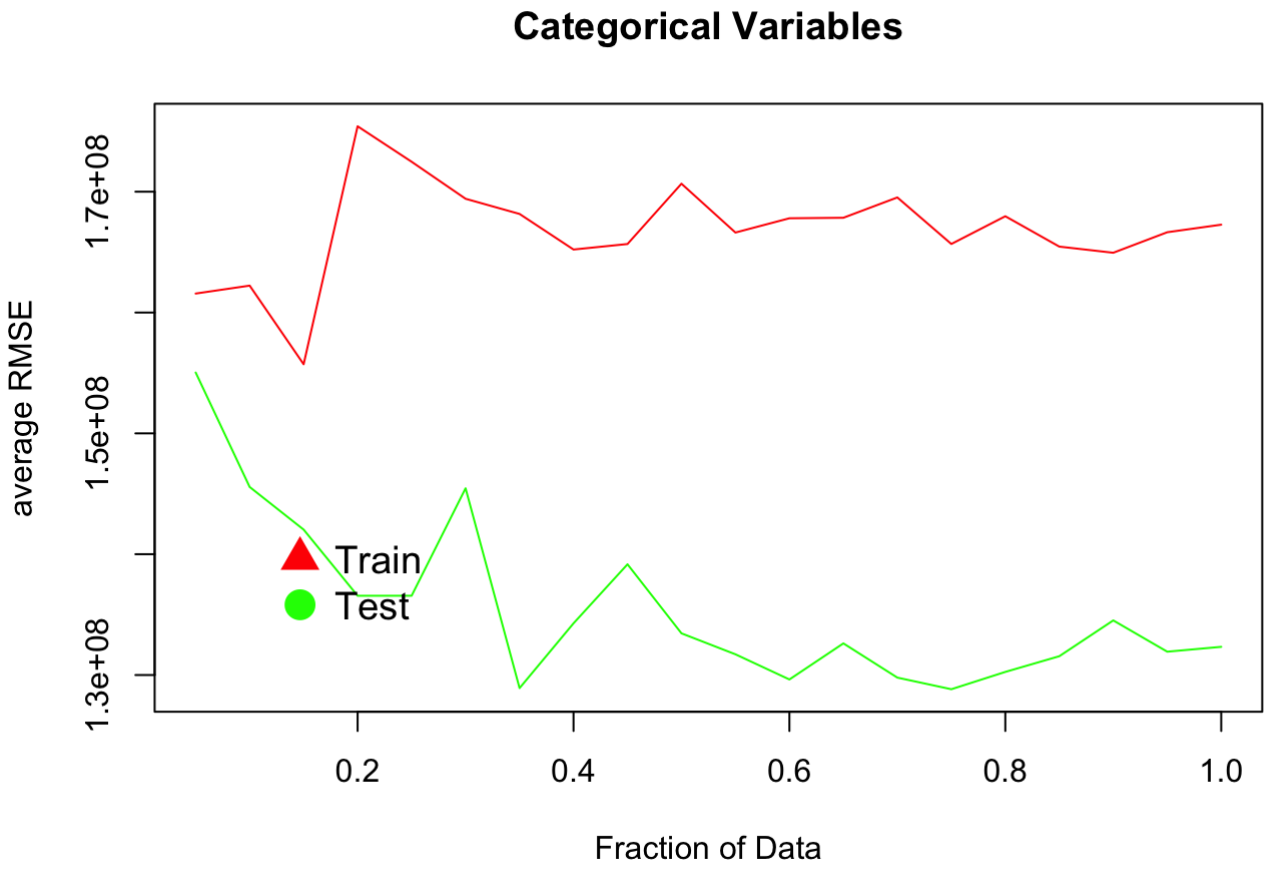
```
colnames(df2)
```

```
##  [1] "Row.names"         "action"            "adventure"
##  [4] "animation"         "biography"         "comedy"
##  [7] "crime"             "documentary"       "drama"
## [10] "family"            "fantasy"           "history"
## [13] "horror"            "music"             "musical"
## [16] "mystery"           "news"              "romance"
## [19] "scifi"             "short"             "sport"
## [22] "thriller"          "war"               "western"
## [25] "Row.names"         "win"               "nomination"
## [28] "Title"             "Year"              "Rated"
## [31] "Released"          "Runtime"           "Genre"
## [34] "Director"          "Writer"            "Actors"
## [37] "Plot"              "Language"          "Country"
## [40] "Awards"            "Poster"            "Metascore"
## [43] "imdbRating"        "imdbVotes"         "imdbID"
## [46] "Type"              "tomatoMeter"       "tomatoImage"
## [49] "tomatoRating"      "tomatoReviews"     "tomatoFresh"
## [52] "tomatoRotten"      "tomatoConsensus"   "tomatoUserMeter"
## [55] "tomatoUserRating"  "tomatoUserReviews" "tomatoURL"
## [58] "DVD"               "BoxOffice"         "Production"
## [61] "Website"           "Response"          "Budget"
## [64] "Gross"             "Date"
```

```
#use only numeric variables
df3 <-df2[,which(sapply(df2,is.numeric))]
df3 <- subset(df3, select = -Year)
df3 <- subset(df3, select = -Runtime)
df3 <- subset(df3, select = -imdbRating)
df3 <- subset(df3, select = -imdbVotes)
df3 <- subset(df3, select = -tomatoMeter)
df3 <- subset(df3, select = -tomatoRating)
df3 <- subset(df3, select = -tomatoReviews)
df3 <- subset(df3, select = -tomatoFresh)
df3 <- subset(df3, select = -tomatoRotten)
df3 <- subset(df3, select = -tomatoUserMeter)
df3 <- subset(df3, select = -tomatoUserRating)
df3 <- subset(df3, select = -tomatoUserReviews)
df3 <- subset(df3, select = -Budget)
df3 <- subset(df3, select = -Date)

colnames(df3)
```

```
##  [1] "action"      "adventure"   "animation"   "biography"   "comedy"
##  [6] "crime"       "documentary" "drama"       "family"      "fantasy"
## [11] "history"     "horror"      "music"       "musical"     "mystery"
## [16] "news"        "romance"     "scifi"       "short"       "sport"
## [21] "thriller"    "war"         "western"     "win"         "nomination"
## [26] "Gross"
```

```
options(warn = -1)

#fix N/A values
df3$win[is.na(df3$win)] <- 0
df3$nomination[is.na(df3$nomination)] <- 0
df3=df3[!is.na(df3$Gross),]

##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(df3), 0.8*nrow(df3))  # row indices for training data
train <- df3[trainingRowIndex, ]  # training data
test  <- df3[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()
#percentages <- c(.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,.8,.85,.9,.95,1)

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="Categorical Variables",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrain,arr
Test))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```



Q:

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

A: I used Genre seperating each genre into a binary column, as well as seperating the awards columns into wins and nominations.

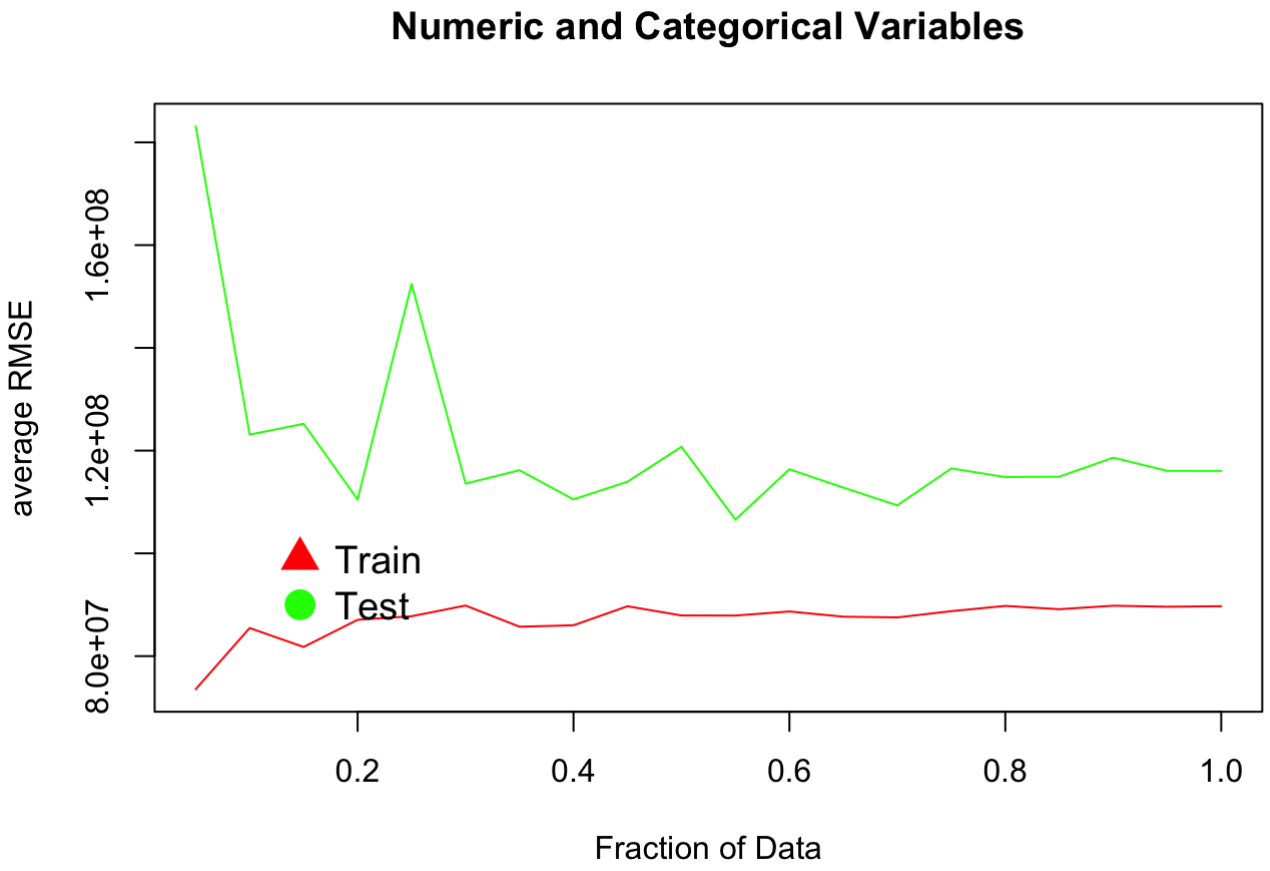
Q: What is the best mean test RMSE value you observed, and at what training set size? How does this compare with Task 2?

A: The best (lowest) RMSE for the training curve I observed was 1.3e+08, at 5% of data. This is much worse than task 2. Task 2 is a refinement from a large numeric dataset; this uses much less features, and thus has a more difficult time modeling for gross.

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**.

```
#trying to combine the binned data with the df2 gives us only two observations/ thus thi
s is no good.
options(warn = -1)
df2 <- df2[,which(sapply(df2,is.numeric))]
#fix N/A values
df2$win[is.na(df2$win)] <- 0
df2$nomination[is.na(df2$nomination)] <- 0
df2=df2[!is.na(df2$Gross),]

#BINNING
for(i in 1:nrow(df2)){
  if(df2$Budget[i]>=3000000)
    {df2$highbudget[i]<- 1}
  else
  {df2$highbudget[i] <- 0}
}


##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(df2), 0.8*nrow(df2))  # row indices for training data
train <- df2[trainingRowIndex, ]  # training data
test  <- df2[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main="Numeric and Categorical Variables",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind
(arrTrain,arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```



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

A: Better than RMSE for binning on the numeric only data, by roughly half an order of magnitude. This shows significant improvement over Numeric or categorial alone.

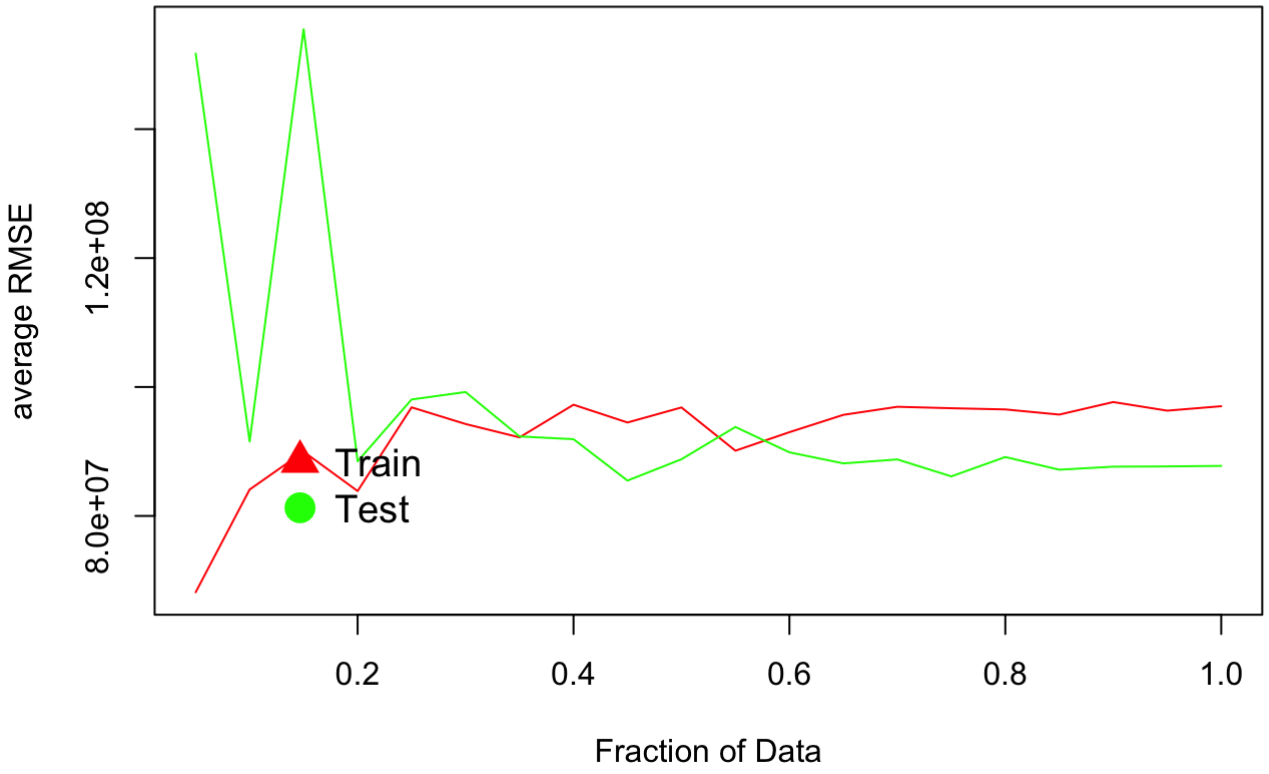
5. Additional features

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`).

```
#lets create some new features based off of genre and budget
df2$romcom <- df2$romance*df2$comedy
df2$budget_comedy <- df2$comedy*df2$highbudget
df2$budget_drama <- df2$drama*df2$highbudget
df2$budget_romance <- df2$romance*df2$highbudget
df2$budget_shorts <- df2$short*df2$highbudget
df2$budget_action <-df2$action*df2$highbudget
df2$budget_adventure  <-df2$adventure*df2$highbudget
df2$budget_animation  <-df2$animation*df2$highbudget
df2$budget_biography  <-df2$biography*df2$highbudget
df2$budget_crime  <-df2$crime*df2$highbudget
df2$budget_documentary  <-df2$documentary*df2$highbudget
df2$budget_family  <-df2$family*df2$highbudget
df2$budget_fantasy  <-df2$fantasy*df2$highbudget
df2$budget_history <-df2$history*df2$highbudget
df2$budget_horror <-df2$horror*df2$highbudget
df2$budget_music <-df2$music*df2$highbudget
df2$budget_muscial <-df2$musical*df2$highbudget
df2$budget_mystery <-df2$mystery*df2$highbudget
df2$budget_news <-df2$news*df2$highbudget
df2$budget_scifi <-df2$scifi*df2$highbudget
df2$budget_sport <-df2$sport*df2$highbudget
df2$budget_thriller <-df2$thriller*df2$highbudget
df2$budget_war <-df2$war*df2$highbudget
df2$busget_western <-df2$western*df2$highbudget


##Train test Split (80/20 split)
trainingRowIndex <- sample(1:nrow(df2), 0.8*nrow(df2))  # row indices for training data
train <- df2[trainingRowIndex, ]  # training data
test  <- df2[-trainingRowIndex, ] # test data

arrTrain = c()
arrTest = c()

#run thorugh the model with different train and test sizes
for(percent in percentages){

arrTrain <- c(arrTrain,calcRMSE(train,test,percent)[1])
arrTest <- c(arrTest,calcRMSE(train,test,percent)[2])
}

plot(percentages,arrTrain,type="l",col="red",ylab="average RMSE",xlab="Fraction of Data"
,main = "5. Additional Features",ylim=c(min(cbind(arrTrain,arrTest)),max(cbind(arrTrain,
arrTest))))
lines(percentages,arrTest,col="green")
legend("bottomleft",,
  legend = c("Train", "Test"),
  col = c("red","green",
  rgb(0.8,0.4,0.1,0.7)),
  pch = c(17,19),
  bty = "n",
  pt.cex = 2,
```

```
  cex = 1.2,
  text.col = "black",
  horiz = F ,
  inset = c(0.1, 0.1))
```

5. Additional Features



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

A: I decided to create new variables for each genre of whether or not it was a high budget movie or not, This created 23 new features to model with. I also created a new genre called “romcom” as this is a very popular movie genre. I chose them because it is a simple to way engineer many new features, and gives the model more weight on aspects of budget and genre which I believe matter most when trying to determine the potential Gross of your movie.

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

A:The final RMSE is about 9.5e+07, though again this will vary by run. I learned that you have to attempt many things to try and get better RMSE values when conducting linear regression on your data. The best improvement I found was using both numeric and categorical data. By adding additional features I was able to ascertain the robustnes of the original model. Many runs have both part 4 and 5 having a train RMSE of 9*10^7. Many useful techniques have been covered for application to any dataset in the future.

Taking this project further, I want to develop cross-validated feature engineering code examples to work with, instead of engineering features based on a hunch.