Project 2

Rodrigo De Luna Lara

30 November 2016

1 Introduction and Dataset Cleanup

This second part of the project consists on fitting a linear model to the movies dataset in order to predict the Profit (the difference between the Budget and the Revenue), trying to tune the model in order to reduce the Mean Square Error (MSE) and improve the prediction ability of the model. The MSE is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2$$

In all cases linear models are fitted using the *lm* function from the default R environment.

The dataset has to be prepared with the following conditions:

1. All rows without revenue and budget information have to be dropped
2. All movies released prior to 2000 have to be dropped to partially account for the inflation.
3. Non-relevant numeric and categorical variables have to be dropped from the dataset.

The dataset was recreated completely from the movies_merged file, taking some code from the previous project submission. The variables that were removed from the dataset are:

- tomatoConsensus: not relevant to linear fitting
- Plot: not relevant to linear fitting
- Date: the release date is used instead, as it contains also month and day
- BoxOffice: highly sparse, contains virtually no information
- DVD: not relevant in predicting

```r
# Remove unused columns
movies_merged$tomatoConsensus = NULL
movies_merged$Plot = NULL
movies_merged$Date = NULL
movies_merged$BoxOffice = NULL
movies_merged$DVD = NULL
```

Then, as stated in the requirements, all films prior to 2000 and posterior to 2016 are removed from the dataset. Additionally, films without Gross and Budget information are removed from the dataset.

```r
# Remove films prior to 2000 and posterior to 2016
movies_merged = movies_merged[which(movies_merged$Year >= 2000
                                  & movies_merged$Year <= 2016),]
# Remove all rows without gross or budget
movies_merged = movies_merged[which(!is.na(movies_merged$Gross)
                                  & !is.na(movies_merged$Budget)),]
```

As the release date was kept, it is converted from a date to three separate values for the year, month and day, converting them to three useful numeric variables.

```r
# Extract release day, month and year from release date
movies_merged$Day = as.numeric(format(movies_merged$Released,"%d"))
movies_merged$Month = as.numeric(format(movies_merged$Released,"%m"))
# For the year, use the release year if available, if not keep the original value
# from the year variable
movies_merged$Year[which(!is.na(movies_merged$Released))] =
  as.numeric(format(movies_merged$Released[which(!is.na(movies_merged$Released))],"%Y"))
# Remove the released variable as it is no longer of use
movies_merged$Released = NULL
```

The runtime variable is then converted to numeric from the string:

```r
# Convert runtime to numeric
movies_merged$Runtime = as.numeric(lapply(movies_merged$Runtime, function(x) {
  time = NA
  if (length(grep("min",x)) != 0){
    if (length(grep("h",x)) != 0){
      pos = gregexpr("h",x)[[1]][1]
      hrs = as.numeric(substr(x,1,pos-1)) * 60
      x = substr(x,pos+1,nchar(x))
    } else {
      hrs = 0
    }
    pos = gregexpr("m",x)[[1]][1]
    min = as.numeric(substr(x,1,pos-1))
    time = hrs + min
  }
  return(time)
}))
```

Another variable is transformed from string to numeric is the Awards, for the initial model the total number of wins and nominations is retrieved from the dataset to be added as a numeric variable.

```r
# Extract number of wins and nominations for different categories
# Remove punctuation
movies_merged$Awards = gsub("[^[:alnum:][:space:]']", "", movies_merged$Awards)
# Convert everything to lower case
movies_merged$Awards = gsub(pattern = '([[:upper:]])',
                            perl = TRUE,
                            replacement = '\\L\\1', movies_merged$Awards)

library(stringr)

# Split the awards variable
awards.split = strsplit(movies_merged$Awards,split = " ")
# Convert the all values to numeric, dropping strings
awards.numeric = lapply(awards.split,as.numeric)
# Create the new variable
movies_merged$Awards.sum = as.numeric(lapply(awards.numeric, function(x){
  return(sum(x[which(!is.na(x))]))
}))
```

The last part of the preprocessing is to calculate the profit and drop any variables directly related to it (Domestic Gross and Gross variables).

```
# Calculate the profit
movies_merged$Profit = movies_merged$Gross - movies_merged$Budget
# Drop the Gross variables as they directly predict the profit
movies_merged$Domestic_Gross = NULL
movies_merged$Gross = NULL
```

2 Numeric Model (Assignment 1)

2.1 First Numeric Model

The numeric model is obtained by getting all numeric variables from the dataset:

```
# Get all the numeric variables from the dataframe
ColumnClass = cbind(sapply(movies_merged, class))
ColumnNumeric = which(ColumnClass == "numeric")
# Get which column corresponds to the Profit (the response variable)
ResponseIndex = which(row.names(ColumnClass) == "Profit")
PredictorIndex = ColumnNumeric[!ColumnNumeric %in% ResponseIndex]
# Build the numeric model
numericModel = movies_merged[,c(ColumnNumeric)]
numericModel[is.na(numericModel)] = 0
```

Then the fractions of training data are defined, the graphs presented throughout this report are all based on the fraction of training data with respect to the resulting dataset.

```
#Define the splits for training data
trainingSplits = seq(0.05,0.95,by=0.05)
```

The initial numeric model, as well as the rest of the models in this report are trained with the following code, just changing the formula as appropriate.

```
trainingMSE_all = rep(0,length(trainingSplits))
validationMSE_all = rep(0,length(trainingSplits))

for(j in 1:length(trainingSplits)){
  trainingMSE = rep(0,10)
  validationMSE = rep(0,10)
  for(i in 1:10){
    trainingIndex = sample(nrow(numericModel),
                           floor(trainingSplits[j]*nrow(numericModel)))
    model = lm(Profit ~ .,data=numericModel[trainingIndex,])
    trainingMSE[i] = mean(model$residuals^2)
    validationResponse =
      predict(model,newdata=numericModel[-trainingIndex,])
    validationMSE[i] =
      mean((numericModel[-trainingIndex,"Profit"]-validationResponse)^2)
  }
  trainingMSE_all[j] = mean(trainingMSE)
  validationMSE_all[j] = mean(validationMSE)
}
```

Figure 1 shows the resulting MSE for this assignment.

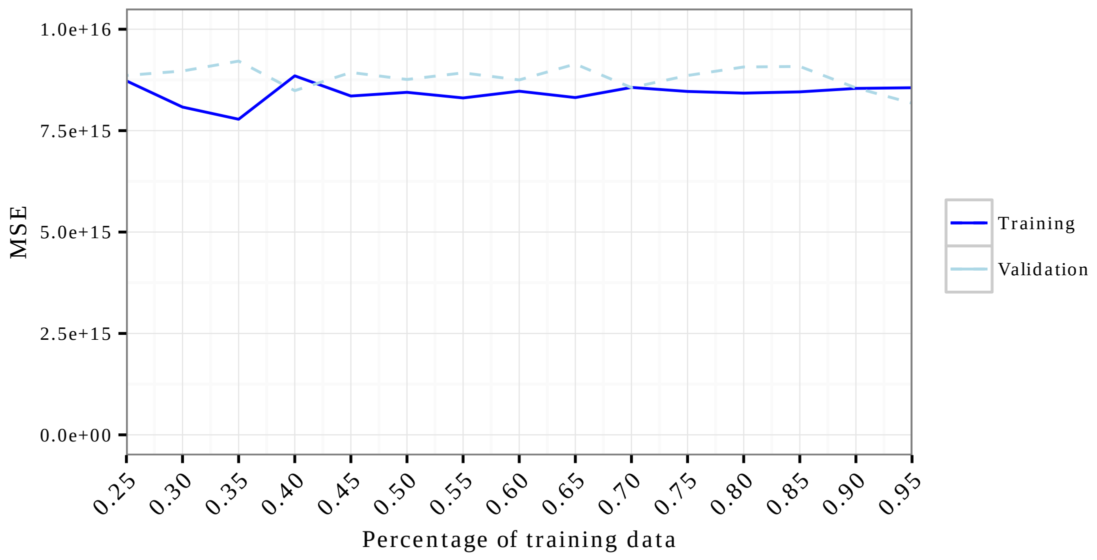

Figure 1: MSE for Numeric Model 1 (Assignment 1)

3 Numeric Model Improvement (Assignment 2)

3.1 Second Numeric Model

As first step for improvement, it was decided to bin the awards variable by type of award or nomination, splitting it into the following categories:

1. General wins
2. General nominations
3. Oscar wins
4. Oscar nominations
5. Golden Globe wins
6. Golden Globe nominations
7. BAFTA wins
8. BAFTA nominations

The purpose behind this transformation is to give different weights to the different types of awards, an Oscar win should not be weighted the same as a generic award, the same can be said about Golden Globes and BAFTAs, they each should have a different weight.

This was achieved in code using the power of regular expressions to extract the information from the awards string:

```
# Extract specific number of wins
Wins.regex = str_match(movies_merged$Awards,"([0-9]{1,})\\s{1,}win")
movies_merged$Wins = as.numeric(Wins.regex[,2])

# Extract specific number of nominations
Nominations.regex = str_match(movies_merged$Awards,"([0-9]{1,})\\s{1,}nomination")
movies_merged$Nominations = as.numeric(Nominations.regex[,2])
```

```
# Extract specific number of Oscar wins and nominations
Oscars.regex = str_match(movies_merged$Awards,
  "(won\\s{0,}([0-9]{0,})\\s{0,}oscar(s)?){0,}
   (nominated\\s{0,}for\\s{0,}([0-9])\\s{0,}oscar(s)?){0,}")
movies_merged$Oscar.Wins = as.numeric(Oscars.regex[,3])
movies_merged$Oscar.Nominations = as.numeric(Oscars.regex[,6])

# Extract specific number of Golden Globe wins and nominations
GoldenGlobe.regex = str_match(movies_merged$Awards,
  "(won\\s{0,}([0-9]{0,})\\s{0,}golden globe(s)?){0,}
   (nominated\\s{0,}for\\s{0,}([0-9])\\s{0,}golden globe(s)?){0,}")
movies_merged$GoldenGlobe.Wins = as.numeric(GoldenGlobe.regex[,3])
movies_merged$GoldenGlobe.Nominations = as.numeric(GoldenGlobe.regex[,6])

# Extract specific number of BAFTA wins and nominations
Bafta.regex = str_match(movies_merged$Awards,
  "(won\\s{0,}([0-9]{0,})\\s{0,}bafta){0,}
   (nominated\\s{0,}for\\s{0,}([0-9])\\s{0,}bafta){0,}")
movies_merged$Bafta.Wins = as.numeric(Bafta.regex[,3])
movies_merged$Bafta.Nominations = as.numeric(Bafta.regex[,5])

# Remove Awards sum variable
movies_merged$Awards.sum = NULL
```

The model was trained using the formula *Profit* ., with the new variables. The resulting MSE can be seen in Figure 2, comparing it with the previous model from Assignment 1.

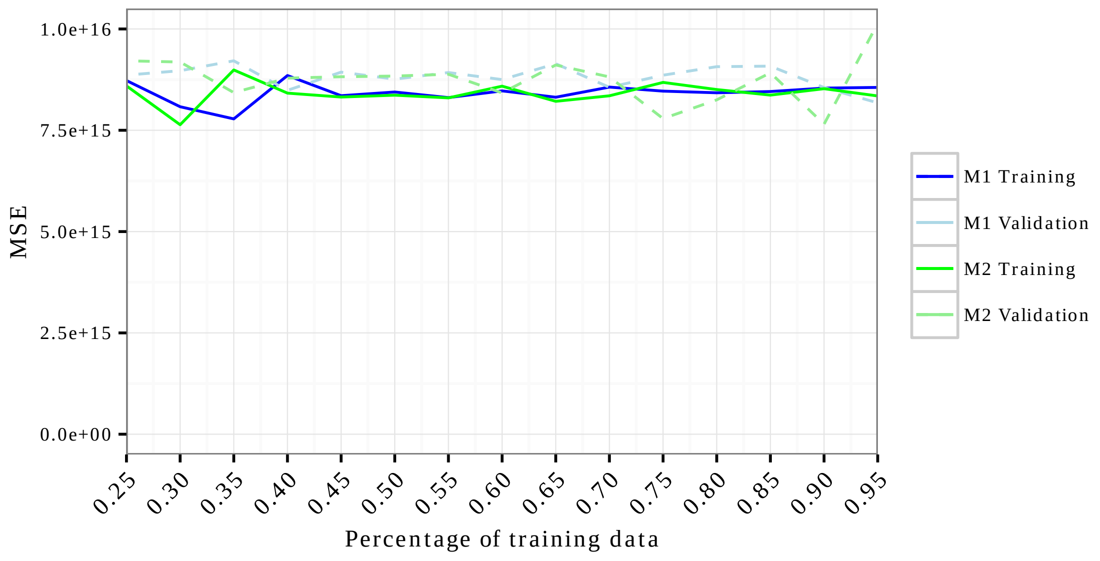

Figure 2: MSE for Numeric Model 2 (Assignment 2)

There is a noticeable improvement in the validation MSE, indicating better predictability.

3.2 Third Numeric Model

The next improvement takes the previous model and uses feature selection by means of the *stepAIC* function in the *MASS* package for each iteration. The resulting formula is then stored in a matrix. The purpose of this is to extract only the relevant variables in order to try to reduce the error in the model by diminishing overfitting.

```r
# Create a matrix to store the formulas
formulas = matrix(0,nrow=10,ncol=length(trainingSplits))

# ... code for model training ...
model3 = lm(formula = Profit ~ . + Budget*. ,
            data = numericModel[trainingIndex, ])
model3 = stepAIC(model3, direction = "both", trace = FALSE)
formulas[i,j] = as.character(formula(model3))[[3]]
# ... code for model training ...
```

Using this approach, selecting the best model by feature selection at each iteration yields the result shown in Figure 3. In this case the performance is not compared versus the previous model as this is only an intermediate step for another model.

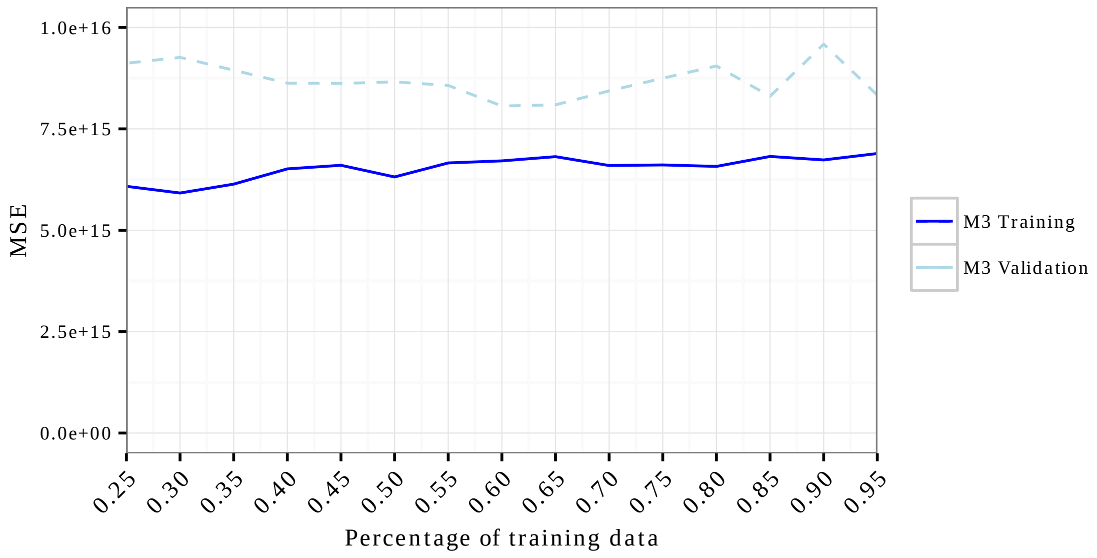

Figure 3: MSE for Numeric Model 3 (Assignment 2)

3.3 Fourth Numeric Model

The resulting formulas from the previous model are used to create a term frequency table, in order to get the most common terms across all trainings for the linear model.

```r
# Extract the individual terms from each formula
formulas.split = lapply(strsplit(formulas,split="+ "),str_trim)
term.frequency = cbind(sort(table(abind(formulas.split))))
term.frequency = term.frequency[-c(length(term.frequency)-1,length(term.frequency)),]
# Then get the top 80% of terms to create the new formula
```

```
term.frequency =
  term.frequency[rev(cumsum(rev(term.frequency)) < sum(term.frequency)*0.80)]
```

The formula used for this model is then created from the term frequency table.

```
formula.model4 = paste("Profit ~",paste(names(term.frequency),collapse=" + "))
```

The resulting MSE can be seen in Figure 4, in comparison with the second model.

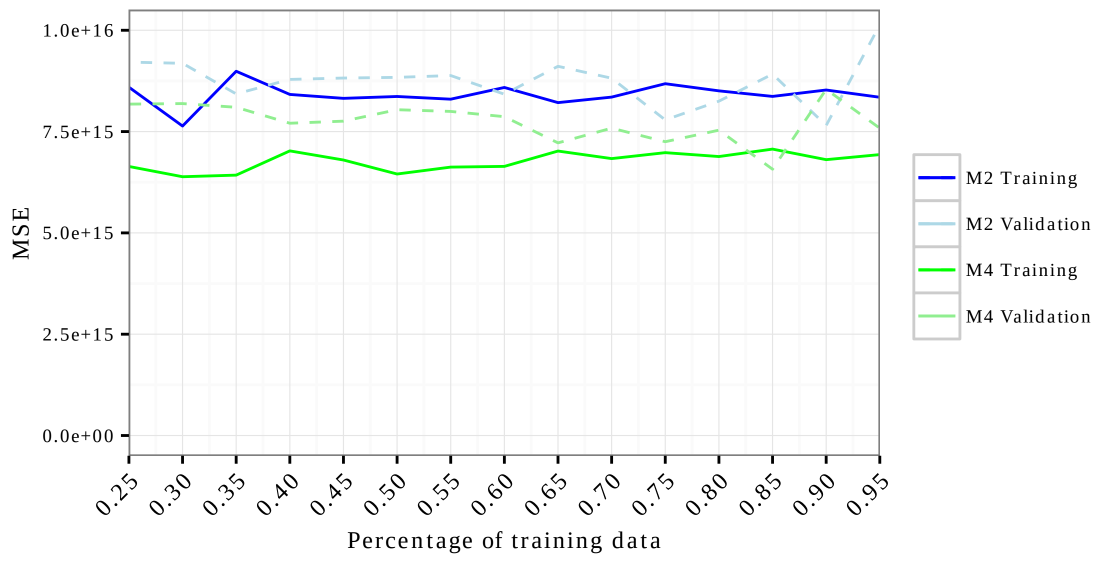

Figure 4: MSE for Numeric Model 4 (Assignment 2)

3.4 Fifth Numeric Model

To further improve the model, it was decided to normalize all variables related to ratings, so they are all within the same range. This homogenization of variables has the purpose of removing any bias the difference between rating scales could have on the model. Each variable was rescaled to a 0-1 scale with their maximum value (as the minimum in all case is 0).

```
# Rescale all rating variables
numericModel$Metascore = numericModel$Metascore/100
numericModel$imdbRating = numericModel$imdbRating/5
numericModel$tomatoMeter = numericModel$tomatoMeter/100
numericModel$tomatoRating = numericModel$tomatoRating/10
numericModel$tomatoUserMeter = numericModel$tomatoUserMeter/100
numericModel$tomatoUserRating = numericModel$tomatoUserRating/5
# Set runtime on an hourly basis
numericModel$Runtime = numericModel$Runtime/60
```

The formula for this case is the one from the previous model (model 4). The resulting MSE versus model 4 can be seen in Figure 5.

It can be seen that there is not a significant improvement because of these variable transformations. So both models are almost equivalent between each other.

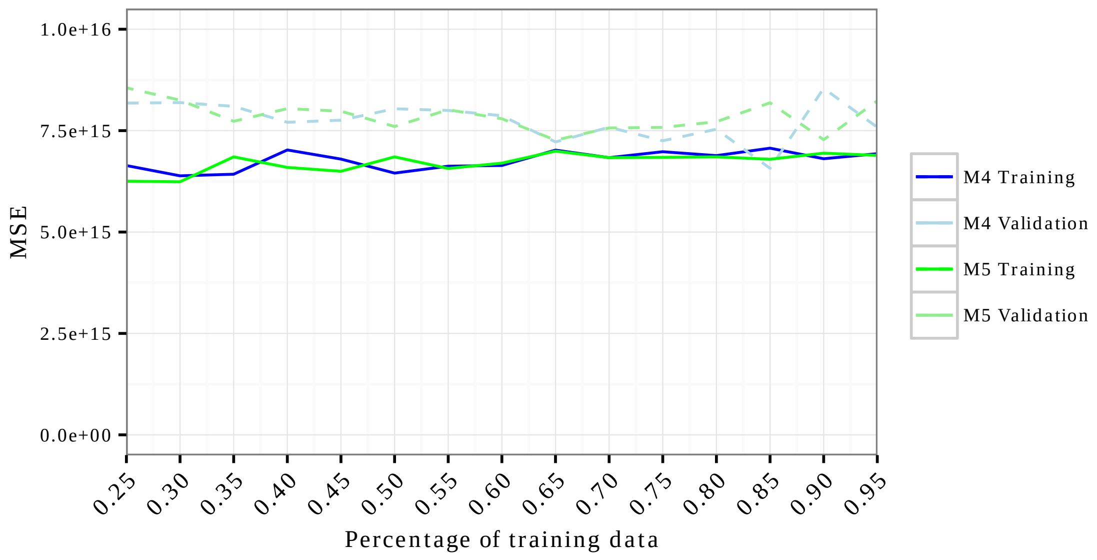

Figure 5: MSE for Numeric Model 5 (Assignment 2)

3.5 Sixth Numeric Model

For the next iteration of the numeric model it was decided to include interactions between the Budget and all of the other variables, and then use stepwise term selection using the *stepAIC* on each iteration of the model training as with the third model.

```r
# Create a matrix to store the formulas
formulas2 = matrix(0,nrow=10,ncol=length(trainingSplits))

# ... code for model training ...
model6 = lm(formula = Profit ~ . + Budget:.,data = numericModel[trainingIndex, ])
model6 = stepAIC(model6, direction = "both", trace = FALSE)
formulas2[i,j] = as.character(formula(model6))[[3]]
# ... code for model training ...
```

The resulting plot from averaging the performance across 10 samplings for each training set fraction can be seen in Figure 6. It must be noted that in this case the validation error resulted in very large values, outside the range of the rest of the models.

3.6 Seventh Numeric Model

As with the fourth model, a term frequency table is obtained and the top 80% of most common terms is used to create an unique formula, which serves as basis for this model.

```r
# Extract top 80% of terms across formulas
formulas.split2 = lapply(strsplit(formulas2,split="+ "),str_trim)
term.frequency2 = cbind(sort(table(abind(formulas.split2))))
term.frequency2 = term.frequency2[-c(length(term.frequency2)-1,length(term.frequency2)),]
term.frequency2 = term.frequency2[rev(cumsum(rev(term.frequency2)) < sum(term.frequency2)*0.80)]
```

The formula used for this model is:

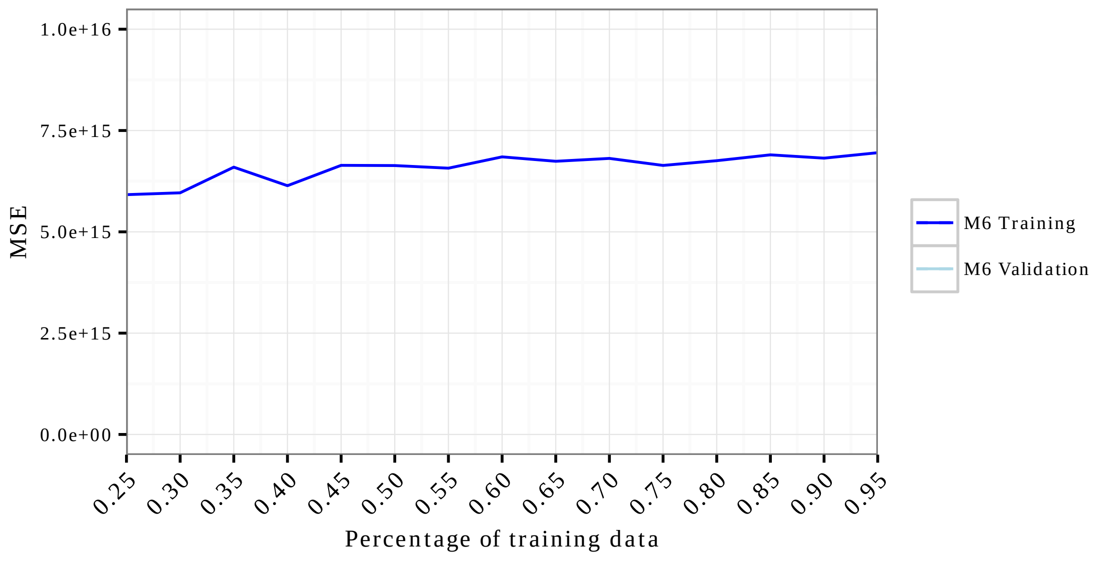

Figure 6: MSE for Numeric Model 6 (Assignment 2)

```
formula.model7 = paste("Profit ~",paste(names(term.frequency2),collapse=" + "))
```

The resulting MSE plot can be seen in Figure 7, with respect to Model 5.

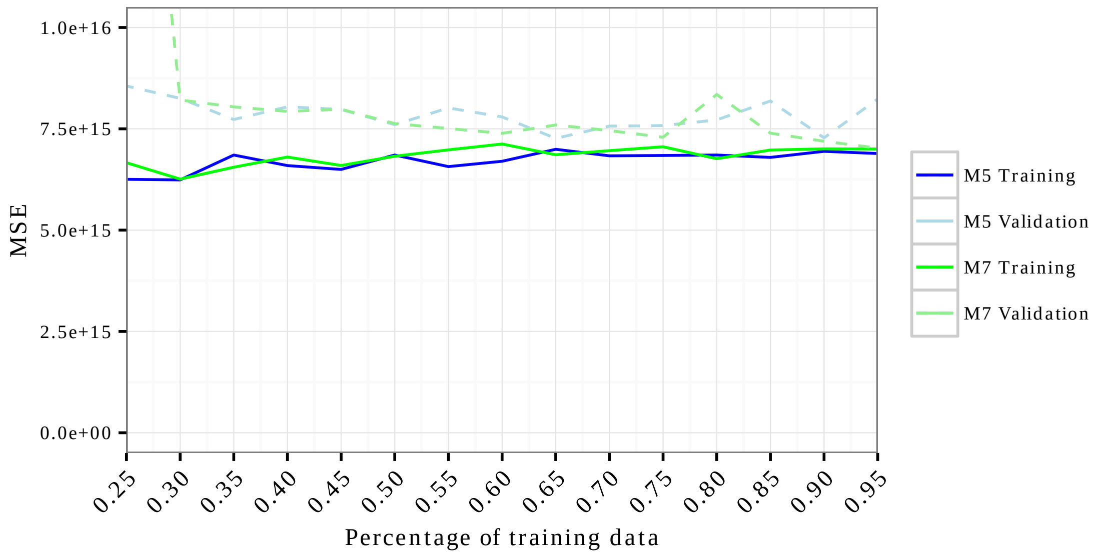

Figure 7: MSE for Numeric Model 7 (Assignment 2)

It can be seen that for most part both models perform almost the same, so it can be seen that Model 5 is the best numeric model to use.

4 Categorical Variable Featurization (Assignment 3)

First, the categorical model was defined with all non-numeric columns in the original dataset.

```r
# Get all the numeric variables from the dataframe
ColumnClass = cbind(sapply(movies_merged, class))
ColumnNumeric = which(ColumnClass == "numeric")
# Extract categorical variables
categoricalModel = movies_merged[,-c(ColumnNumeric)]
categoricalModel$Profit = movies_merged$Profit
```

All of the relevant categorical variables were one-hot encoded in order to train a linear model based on categorical data.

4.1 Genre Featurization

The first variable to be encoded was the genre:

```r
# Featurize genres
# Split the strings by commas to separate each genre
genres.split = lapply(strsplit(movies_merged$Genre,split=","),str_trim)
# Get the list of unique genres
unique.genres = str_trim(sort(unique(abind(genres.split))))
# Create a binary vector for one-hot encoding
Genre.binary = (lapply(genres.split,function (x){
  as.numeric(unique.genres %in% x)
}))
# Create a matrix based on the one-hot encoded binary vectors
genres.matrix = do.call(rbind,Genre.binary)
# Assign the variable names
colnames(genres.matrix) = paste("is.genre.",unique.genres,sep="")
row.names(genres.matrix) = categoricalModel$Title
# Add the new variables to the categorical model
categoricalModel = cbind(categoricalModel,genres.matrix)
categoricalModel$Genre = NULL
```

4.2 Directors Featurization

The top 200 directors were one-hot encoded:

```r
# Featurize directors
# Split the director string by commas, separating the directors if there is
# more than one
directors.split = lapply(strsplit(categoricalModel$Director,split=","),str_trim)
# Get the unique directors
unique.directors = str_trim(sort(unique(abind(directors.split))))
# Get the frequency of directors
directors.frequency = cbind(sort(table(abind(directors.split)),decreasing = TRUE))
# Retrieve the top 200 directors
directors.top200 = names(directors.frequency[1:200,])
# Encode them in a binary vector
```

```r
Director.binary = (lapply(directors.split,function (x){
  as.numeric(directors.top200 %in% x)
}))
# Create the feature matrix
directors.matrix = do.call(rbind,Director.binary)
# Encode the "Other" directors by checking which rows are empty
directors.matrix =
  cbind(directors.matrix,as.numeric(rowSums(directors.matrix) == 0))
# Process the strings to remove unnecessary characters
directors.top200 = gsub("[^[:alnum:][:space:]']", "", directors.top200)
directors.top200 = paste("is.director.",str_replace(directors.top200," ",""),sep="")
# Assign the variable names
colnames(directors.matrix) = c(directors.top200,"is.director.Other")
row.names(directors.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,directors.matrix)
categoricalModel$Director = NULL
```

4.3 Writers Featurization

The top 200 writers were one-hot encoded:

```r
# Featurize writers
# Split the writer string by commas, separating the writers if there is
# more than one
writers.split = lapply(strsplit(categoricalModel$Writer,split=","),str_trim)
# Get the unique writers
unique.writers = str_trim(sort(unique(abind(writers.split))))
# Create the frequency table
writers.frequency = cbind(sort(table(abind(writers.split)),decreasing = TRUE))
# Get the top 200 writers
writers.top200 = names(writers.frequency[1:200,])
# Encode them in a binary vector
Writer.binary = (lapply(writers.split,function (x){
  as.numeric(writers.top200 %in% x)
}))
# Create the matrix
writers.matrix = do.call(rbind,Writer.binary)
# Add the "Other" category by checking which rows are empty
writers.matrix = cbind(writers.matrix,as.numeric(rowSums(writers.matrix) == 0))
# Remove unnecesary characters
writers.top200 = gsub("[^[:alnum:][:space:]']", "", writers.top200)
writers.top200 = paste("is.writer.",str_replace(writers.top200," ",""),sep="")
# Assign the variable names
colnames(writers.matrix) = c(writers.top200,"is.writer.Other")
row.names(writers.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,writers.matrix)
categoricalModel$Writer = NULL
```

4.4 Language Featurization

The top 10 languages were one-hot encoded:

```
# Featurize languages
# Split the langauges by comma in case there is more than one
languages.split = lapply(strsplit(categoricalModel$Language,split=","),str_trim)
# Remove anything between parentheses, including the parentheses
languages.split = lapply(languages.split,function(x){
  return(gsub("\\s*\\([^\\)]+\\)","",x))
})
# Get the unique languages
unique.languages = str_trim(sort(unique(abind(languages.split))))
# Create a frequency table
languages.frequency = cbind(sort(table(abind(languages.split)),decreasing = TRUE))
# Get the top 10 languages
languages.top10 = names(languages.frequency[1:10,])
# Encode them in a binary vector
languages.binary = (lapply(languages.split,function (x){
  as.numeric(languages.top10 %in% x)
}))
# Create the feature matrix
languages.matrix = do.call(rbind,languages.binary)
# Add the "Other" variable by checking which rows are empty
languages.matrix = cbind(languages.matrix,as.numeric(rowSums(languages.matrix) == 0))
# Remove unnecessary characters
languages.top10 = gsub("[^[:alnum:][:space:]']", "", languages.top10)
languages.top10 = paste("is.language.",str_replace(languages.top10," ",""),sep="")
# Assign the variable names
colnames(languages.matrix) = c(languages.top10,"is.language.Other")
row.names(languages.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,languages.matrix)
categoricalModel$Language = NULL
```

4.5 Actors Featurization

The top 200 actors were one-hot encoded:

```
# Featurize actors
# Split by commas to separate the different actors
actors.split = lapply(strsplit(categoricalModel$Actors,split=","),str_trim)
# Get the unique actors
unique.actors = str_trim(sort(unique(abind(actors.split))))
# Create a frequency table
actors.frequency = cbind(sort(table(abind(actors.split)),decreasing = TRUE))
# Get the top 200 actors
actors.top200 = names(actors.frequency[1:200,])
# Encode them in a binary vector
Actors.binary = (lapply(actors.split,function (x){
  as.numeric(actors.top200 %in% x)
}))
# Create the feature matrix
```

```r
actors.matrix = do.call(rbind,Actors.binary)
# Create the "Other" actors variable
actors.matrix = cbind(actors.matrix,as.numeric(rowSums(actors.matrix) == 0))
# Remove unnecessary characters
actors.top200 = gsub("[^[:alnum:][:space:]']", "", actors.top200)
actors.top200 = paste("is.actor.",str_replace(actors.top200," ",""),sep="")
# Assign the variable names
colnames(actors.matrix) = c(actors.top200,"is.actor.Other")
row.names(actors.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,actors.matrix)
categoricalModel$Actors = NULL
```

4.6 Country Featurization

The top 50 countries were one-hot encoded:

```r
# Featurize countries
countries.split = lapply(strsplit(categoricalModel$Country,split=","),str_trim)
# Get the unique countries
unique.countries = str_trim(sort(unique(abind(countries.split))))
# Create a frequency table
countries.frequency = cbind(sort(table(abind(countries.split)),decreasing = TRUE))
# Get the top 50 countries
countries.top50 = names(countries.frequency[1:50,])
# Encode them in a binary vector
countries.binary = (lapply(countries.split,function (x){
  as.numeric(countries.top50 %in% x)
}))
# Create the feature matrix
countries.matrix = do.call(rbind,countries.binary)
# Add the "Other" category
countries.matrix = cbind(countries.matrix,as.numeric(rowSums(countries.matrix) == 0))
# Remove unnecessary characters
countries.top50 = gsub("[^[:alnum:][:space:]']", "", countries.top50)
countries.top50 = paste("is.country.",str_replace(countries.top50," ",""),sep="")
# Assign the variable names
colnames(countries.matrix) = c(countries.top50,"is.country.Other")
row.names(countries.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,countries.matrix)
categoricalModel$Country = NULL
```

4.7 Production Featurization

The top 100 production companies were one-hot encoded:

```r
# Featurize production
# The string was separated by commas in case there is more than one company
productions.split = lapply(strsplit(as.character(categoricalModel$Production),
                                    split=","),str_trim)
```

```r
# Remove anything between parentheses, including the parentheses
productions.split = lapply(productions.split,function(x){
  return(gsub("\\s*\\([^\\)]+\\)","",x))
})
# Get the unique production companies
unique.productions = str_trim(sort(unique(abind(productions.split))))
# Create the frequency table
productions.frequency = cbind(sort(table(abind(productions.split)),decreasing = TRUE))
# Get the top 100 companies
productions.top100 = names(productions.frequency[1:100,])
# Encode them in a binary vector
productions.binary = (lapply(productions.split,function (x){
  as.numeric(productions.top100 %in% x)
}))
# Create the matrix
productions.matrix = do.call(rbind,productions.binary)
# Add the "Other" category
productions.matrix = cbind(productions.matrix,as.numeric(rowSums(productions.matrix) == 0))
# Remove unnecessary characters
productions.top100 = gsub("[^[:alnum:][:space:]']", "", productions.top100)
productions.top100 = paste("is.production.",str_replace(productions.top100," ",""),sep="")
# Assign the variable names
colnames(productions.matrix) = c(productions.top100,"is.production.Other")
row.names(productions.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,productions.matrix)
categoricalModel$Production = NULL
```

4.8 Ratings Featurization

All of the features are one-hot encoded:

```r
# Featurize ratings
# Replace NOT RATED with UNRATED
categoricalModel$Rated
[which(is.na(categoricalModel$Rated) | categoricalModel$Rated == "NOT RATED")] = "UNRATED"
# Convert to character from factor
categoricalModel$Rated = as.character(categoricalModel$Rated)
# Split ratings by commas
ratings.split = lapply(strsplit(categoricalModel$Rated,split=","),str_trim)
# Get the unique ratings
unique.ratings = str_trim(sort(unique(abind(ratings.split))))
# Create the frequency table
ratings.frequency = cbind(sort(table(abind(ratings.split)),decreasing = TRUE))
# Encode them in a binary vector
ratings.binary = (lapply(ratings.split,function (x){
  as.numeric(unique.ratings%in% x)
}))
# Create the feature matrix
ratings.matrix = do.call(rbind,ratings.binary)
# Assign the names to the variables
ratings = paste("is.rating.",str_replace(unique.ratings," ",""),sep="")
```

```
colnames(ratings.matrix) = ratings
row.names(ratings.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,ratings.matrix)
categoricalModel$Rated = NULL
```

4.9 Tomato Image Featurization

The tomato image (fresh/rotten) was also one hot-encoded for consistency:

```
# Featurize tomatoImage
categoricalModel$tomatoImage = as.character(categoricalModel$tomatoImage)
# Make all "NA" observations be "none"
categoricalModel$tomatoImage[which(is.na(categoricalModel$tomatoImage))] = "none"
tomatoImages.split = lapply(strsplit(categoricalModel$tomatoImage,split=","),str_trim)
# Get the uniqe ratings
unique.tomatoImages = str_trim(sort(unique(abind(tomatoImages.split))))
# Create the frequency table
tomatoImages.frequency = cbind(sort(table(abind(tomatoImages.split)),decreasing = TRUE))
# Create the binary vector
tomatoImages.binary = (lapply(tomatoImages.split,function (x){
  as.numeric(unique.tomatoImages%in% x)
}))
# Create the feature matrix
tomatoImages.matrix = do.call(rbind,tomatoImages.binary)
# Assign the variable names
tomatoImages = paste("is.tomatoImage.",str_replace(unique.tomatoImages," ",""),sep="")
colnames(tomatoImages.matrix) = tomatoImages
row.names(tomatoImages.matrix) = categoricalModel$Title
# Add the new variables to the model
categoricalModel = cbind(categoricalModel,tomatoImages.matrix)
categoricalModel$tomatoImage = NULL
```

5 Categorical Model Training (Assignment 4)

The model was trained using only the categorical variables present in the dataset. The resulting MSE can be seen in Figure 8.

6 Complete Model (Assignment 5)

6.1 First Model

The numeric and categorical models were combined into a single model:

```
totalModel = cbind(numericModel,categoricalModel)
totalModel[is.na(totalModel)] = 0
```

The complete model was trained as is with both sets of variables, the resulting MSE can be seen in Figure 9.

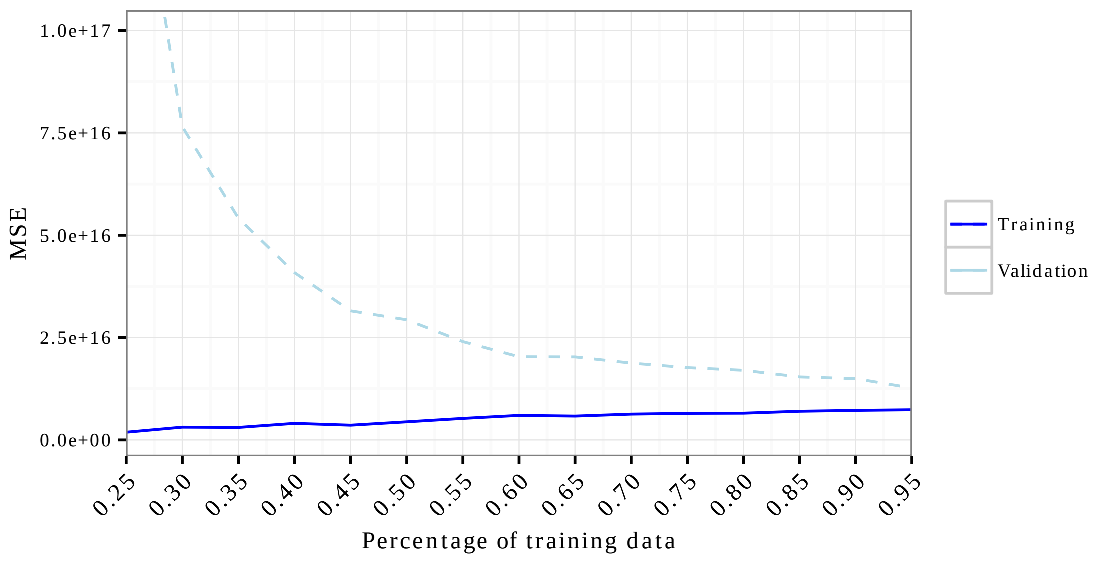

Figure 8: MSE for Categorical Model (Assignment 4)

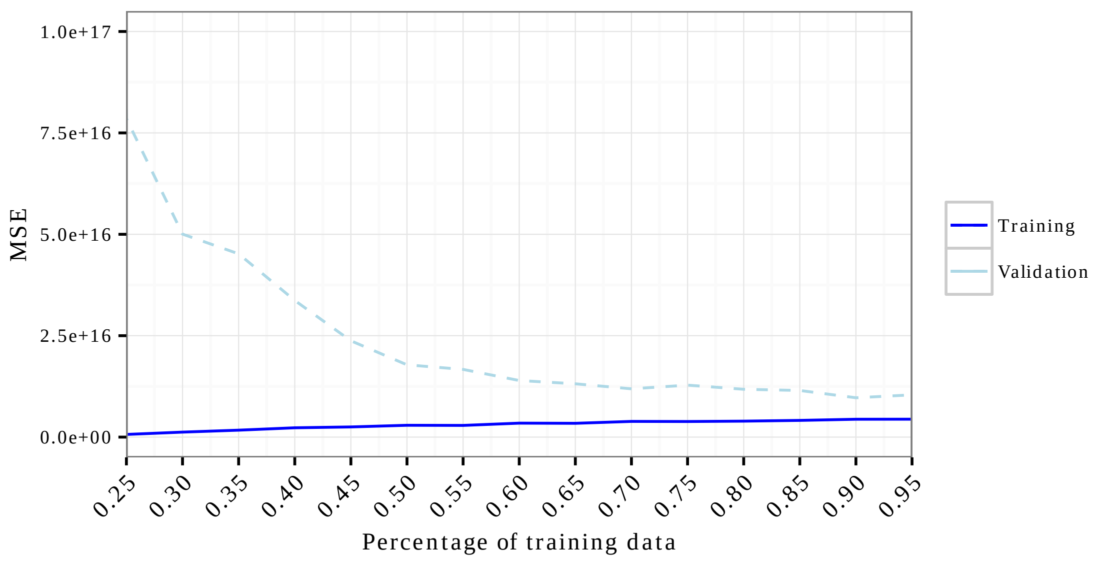

Figure 9: MSE for Complete Model 1 (Assignment 5)

6.2 Second Model

The second model includes the interactions between Budget and all variables and Wins and all variables according to the formula $Profit\ . + Budget : . + Wins : .$, additionally the formula for each iteration was retrieved, obtaining the coefficients with a significance value lower than 0.001.

```r
formulas = matrix(0,nrow=10,ncol=length(trainingSplits))

# ... code for model training ...
model = lm(Profit ~ . + Budget:. + Wins:.,data=totalModel[trainingIndex,])
formulas[i,j] =
  paste(row.names(data.frame(summary(model)$coef[summary(model)$coef[,4] <= .001, 4]))
        ,collapse=" + ")
# ... code for model training ...
```

This was used to create the formula for the second model, the minimum frequency for the terms used to create the final formula was calculated by trial and error, looking at how the MSE behaved with varying frequencies.

```r
# Split the formulas separating by the plus in between terms
formulas.split = lapply(strsplit(formulas,split="+ "),str_trim)
# Create a frequency table
term.frequency = cbind(sort(table(abind(formulas.split))))
# When the model is overfitted, the terms appear as numbers so they have
# to be removed from the formulas
numbers.informula = lapply(rownames(term.frequency),function(x){
  return(as.numeric(x[[1]]))
})
numbers.informula = as.numeric(numbers.informula)
term.frequency = term.frequency[is.na(numbers.informula),]
term.frequency = cbind(term.frequency[-length(term.frequency)])
# By trial and error it was determined that using terms with a frequency of
# at least 25 times
formula.model2 = paste("Profit ~",
                       paste(names(term.frequency[term.frequency > 25,]),collapse=" + "))
```

With this formula, the resulting MSE vs the baseline can be seen in Figure 10.

6.3 Third Model

The formula for this model only considers the original terms, but also makes term selection based on the level of significance. The formula being used in this case is $Profit\tilde{}.$, and as in the previous model is calculated for each iteration:

```r
# ... code for model training ...
model = lm(Profit ~ .,data=totalModel[trainingIndex,])
formulas[i,j] = paste(row.names(
  data.frame(summary(model)$coef[summary(model)$coef[,4] <= .05, 4]))
                      ,collapse=" + ")
# ... code for model training ...
```

With this, the formula for model 3 is created:

```
# Split the formulas separating by the plus in between terms
formulas.split = lapply(strsplit(formulas,split="+ "),str_trim)
# Create a frequency table
term.frequency = cbind(sort(table(abind(formulas.split))))
# When the model is overfitted, the terms appear as numbers so they have
# to be removed from the formulas
numbers.informula = lapply(rownames(term.frequency),function(x){
  return(as.numeric(x[[1]]))
})
numbers.informula = as.numeric(numbers.informula)
term.frequency = term.frequency[is.na(numbers.informula),]
term.frequency = cbind(term.frequency[-length(term.frequency)])
# By trial and error it was determined that using 30% of terms gave good results
formula.model3 = paste("Profit ~",
  paste(names(term.frequency[rev(cumsum(rev(term.frequency))
  < sum(term.frequency)*0.30),]),collapse=" + "))
formula.model3 = str_replace(formula.model3,"\\+ \\(Intercept\\)","")
```

The resulting MSE vs the previous model can be seen in Figure 11.

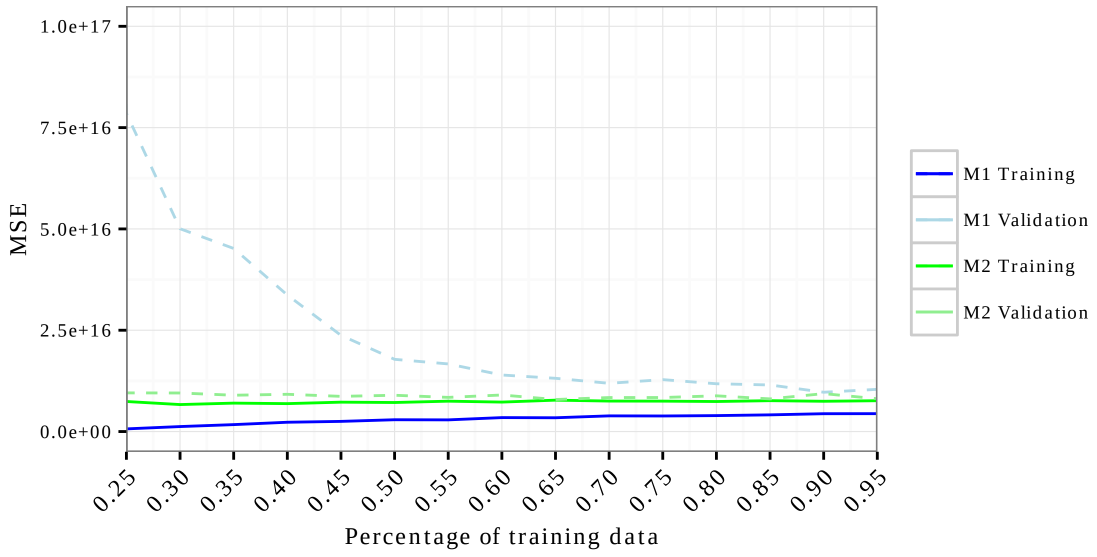

Figure 10: MSE for Complete Model 2 (Assignment 5)

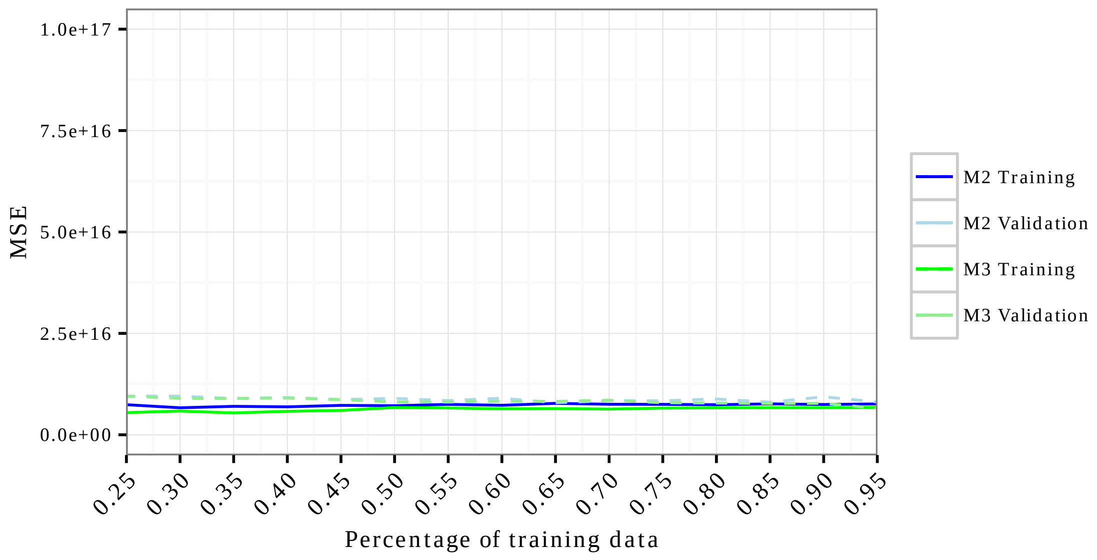

Figure 11: MSE for Complete Model 3 (Assignment 5)

7 Table of MSE for all models

7.1 Numeric Models

Table 1: Numeric Model Training MSE

Training data %	Model 1	Model 2	Model 3	Model 4
0.05	7.403044e+15	6.093351e+15	2.628045e+15	4.339169e+15
0.10	7.284860e+15	6.610856e+15	4.500640e+15	4.969227e+15
0.15	8.277958e+15	6.855798e+15	5.082692e+15	5.885706e+15
0.20	7.739299e+15	7.243535e+15	6.010567e+15	5.829971e+15
0.25	8.726676e+15	8.595509e+15	6.082732e+15	6.638975e+15
0.30	8.080951e+15	7.636304e+15	5.917694e+15	6.385705e+15
0.35	7.780629e+15	8.988317e+15	6.137104e+15	6.425117e+15
0.40	8.850768e+15	8.416663e+15	6.512699e+15	7.023354e+15
0.45	8.353474e+15	8.319376e+15	6.601719e+15	6.798171e+15
0.50	8.444129e+15	8.365822e+15	6.313924e+15	6.452408e+15
0.55	8.305686e+15	8.299421e+15	6.659464e+15	6.623449e+15
0.60	8.471920e+15	8.588009e+15	6.710249e+15	6.640683e+15
0.65	8.315243e+15	8.214470e+15	6.813971e+15	7.019119e+15
0.70	8.565235e+15	8.350616e+15	6.595754e+15	6.832770e+15
0.75	8.465766e+15	8.680945e+15	6.609893e+15	6.979784e+15
0.80	8.425168e+15	8.504843e+15	6.573751e+15	6.883637e+15
0.85	8.456144e+15	8.367914e+15	6.819105e+15	7.067266e+15
0.90	8.541003e+15	8.525616e+15	6.732853e+15	6.806963e+15
0.95	8.557196e+15	8.346369e+15	6.891342e+15	6.931467e+15

Table 2: Numeric Model Training MSE

Training data %	Model 5	Model 6	Model 7
0.05	2.589921e+15	2.880782e+15	3.551948e+15
0.10	5.395460e+15	4.656436e+15	5.353388e+15
0.15	5.523210e+15	4.964993e+15	6.401572e+15
0.20	5.696929e+15	5.748753e+15	5.619389e+15
0.25	6.253101e+15	5.916998e+15	6.663530e+15
0.30	6.238515e+15	5.962471e+15	6.258705e+15
0.35	6.852750e+15	6.595514e+15	6.552958e+15
0.40	6.592288e+15	6.138435e+15	6.802951e+15
0.45	6.497425e+15	6.640339e+15	6.595091e+15
0.50	6.852896e+15	6.634850e+15	6.823290e+15
0.55	6.566995e+15	6.570299e+15	6.980213e+15
0.60	6.698844e+15	6.849625e+15	7.124565e+15
0.65	6.994242e+15	6.740782e+15	6.859262e+15
0.70	6.833349e+15	6.811514e+15	6.960731e+15
0.75	6.841386e+15	6.638450e+15	7.056920e+15
0.80	6.852806e+15	6.755103e+15	6.762119e+15
0.85	6.794009e+15	6.899332e+15	6.976376e+15
0.90	6.944376e+15	6.817915e+15	7.005406e+15
0.95	6.889693e+15	6.952056e+15	7.001813e+15

Table 3: Numeric Model Validation MSE

Training data %	Model 1	Model 2	Model 3	Model 4
0.05	8.650820e+16	1.087591e+16	8.845975e+16	5.507387e+16
0.10	9.827824e+15	1.055482e+16	4.410791e+16	1.348686e+16
0.15	8.930642e+15	9.644179e+15	1.095255e+16	9.067296e+15
0.20	9.142866e+15	9.554840e+15	9.430141e+15	1.384599e+16
0.25	8.860268e+15	9.214017e+15	9.116422e+15	8.177279e+15
0.30	8.970097e+15	9.184457e+15	9.261192e+15	8.190392e+15
0.35	9.214187e+15	8.429815e+15	8.945929e+15	8.099174e+15
0.40	8.485702e+15	8.786802e+15	8.625727e+15	7.705562e+15
0.45	8.934488e+15	8.821422e+15	8.619369e+15	7.756916e+15
0.50	8.761230e+15	8.837242e+15	8.657673e+15	8.039378e+15
0.55	8.924118e+15	8.882061e+15	8.570510e+15	7.997009e+15
0.60	8.750020e+15	8.422077e+15	8.063960e+15	7.867313e+15
0.65	9.141353e+15	9.110969e+15	8.091608e+15	7.220379e+15
0.70	8.560275e+15	8.816662e+15	8.436445e+15	7.584576e+15
0.75	8.859427e+15	7.787915e+15	8.748195e+15	7.249376e+15
0.80	9.069328e+15	8.250821e+15	9.049532e+15	7.534886e+15
0.85	9.082475e+15	8.917980e+15	8.303364e+15	6.570917e+15
0.90	8.553614e+15	7.650075e+15	9.587199e+15	8.542539e+15
0.95	8.176727e+15	1.013664e+16	8.333060e+15	7.582620e+15

Table 4: Numeric Model Validation MSE

Training data %	Model 5	Model 6	Model 7
0.05	1.182590e+17	5.188623e+180	1.221692e+17
0.10	1.104669e+16	1.390405e+157	3.772188e+16
0.15	1.001334e+16	2.465442e+135	9.593989e+15
0.20	9.072990e+15	4.594064e+132	8.446429e+15
0.25	8.556148e+15	4.592350e+149	2.143835e+16
0.30	8.251306e+15	1.242117e+71	8.210492e+15
0.35	7.729781e+15	2.908156e+86	8.040662e+15
0.40	8.042432e+15	6.901467e+88	7.929398e+15
0.45	7.978286e+15	1.997642e+71	7.982407e+15
0.50	7.601514e+15	4.270149e+71	7.628648e+15
0.55	8.015042e+15	3.716836e+71	7.506908e+15
0.60	7.792878e+15	1.573552e+71	7.387671e+15
0.65	7.263974e+15	3.800503e+37	7.593913e+15
0.70	7.566426e+15	1.142194e+37	7.455996e+15
0.75	7.579433e+15	3.650086e+64	7.291961e+15
0.80	7.718696e+15	1.337192e+37	8.347168e+15
0.85	8.186542e+15	1.210032e+38	7.392004e+15
0.90	7.276376e+15	1.618742e+38	7.192453e+15
0.95	8.234948e+15	7.537479e+15	7.023825e+15

7.2 Categoric Model

Table 5: Categoric Model MSE

Training data %	Training	Validation
0.05	3.321517e+13	7.453019e+17
0.10	2.912862e+14	1.454124e+17
0.15	6.071417e+14	1.776710e+17
0.20	1.912319e+15	1.836019e+17
0.25	1.881484e+15	1.623128e+17
0.30	3.120686e+15	7.653742e+16
0.35	3.058285e+15	5.417938e+16
0.40	4.046888e+15	4.086504e+16
0.45	3.605032e+15	3.152988e+16
0.50	4.426652e+15	2.935086e+16
0.55	5.258861e+15	2.401642e+16
0.60	5.995245e+15	2.031970e+16
0.65	5.824013e+15	2.027988e+16
0.70	6.303421e+15	1.877083e+16
0.75	6.492682e+15	1.766063e+16
0.80	6.535272e+15	1.700846e+16
0.85	7.002953e+15	1.538947e+16
0.90	7.215613e+15	1.496859e+16
0.95	7.366833e+15	1.268344e+16

7.3 Complete Model

Table 6: Complete Model Training MSE

Training data %	Model 1	Model 2	Model 3
0.05	0.000000e+00	4.457204e+15	3.872271e+15
0.10	2.628059e+13	6.185447e+15	4.282958e+15
0.15	2.757025e+14	5.977401e+15	4.347325e+15
0.20	5.633127e+14	6.530892e+15	5.463591e+15
0.25	6.536620e+14	7.415654e+15	5.455580e+15
0.30	1.224423e+15	6.654948e+15	5.840719e+15
0.35	1.705675e+15	7.014032e+15	5.387850e+15
0.40	2.293208e+15	6.887522e+15	5.780075e+15
0.45	2.501384e+15	7.249840e+15	5.998236e+15
0.50	2.913403e+15	7.161792e+15	6.715695e+15
0.55	2.880153e+15	7.491548e+15	6.601623e+15
0.60	3.437786e+15	7.270491e+15	6.390618e+15
0.65	3.400710e+15	7.744203e+15	6.443847e+15
0.70	3.867972e+15	7.518351e+15	6.329616e+15
0.75	3.840900e+15	7.498909e+15	6.573838e+15
0.80	3.935189e+15	7.408120e+15	6.631733e+15
0.85	4.118368e+15	7.619381e+15	6.667521e+15
0.90	4.402277e+15	7.467236e+15	6.671212e+15
0.95	4.409291e+15	7.603515e+15	6.752558e+15

Table 7: Complete Model Validation MSE

Training data %	Model 1	Model 2	Model 3
0.05	5.334652e+18	1.619870e+16	1.072500e+16
0.10	5.105980e+18	1.320286e+16	1.032225e+16
0.15	4.776636e+18	1.058458e+16	1.052006e+16
0.20	8.460765e+17	9.440529e+15	9.400392e+15
0.25	7.851364e+16	9.539184e+15	9.519662e+15
0.30	5.004592e+16	9.510402e+15	8.970832e+15
0.35	4.517994e+16	8.961243e+15	8.951620e+15
0.40	3.368454e+16	9.189389e+15	9.044215e+15
0.45	2.374219e+16	8.683422e+15	8.644415e+15
0.50	1.782049e+16	8.956046e+15	8.088787e+15
0.55	1.668501e+16	8.427477e+15	8.065184e+15
0.60	1.394554e+16	9.017802e+15	8.266942e+15
0.65	1.313842e+16	7.903003e+15	8.238622e+15
0.70	1.189912e+16	8.388517e+15	8.519451e+15
0.75	1.281028e+16	8.394439e+15	7.894991e+15
0.80	1.179096e+16	8.812385e+15	7.810978e+15
0.85	1.149057e+16	8.056316e+15	7.728129e+15
0.90	9.698543e+15	9.351782e+15	7.749135e+15
0.95	1.041449e+16	8.090453e+15	6.532870e+15