A novel study of the Buzzard Simulation

R Markdown

```
rm(list=ls())
file.path = file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/PROJECT_DATASETS/BUZZARD/Buzzard_DC1.Rdata"
load(url(file.path))
rm(file.path)
objects()
```

```
## [1] "df"
```

Introduction and Data

‘’’ The BUZZARD data set were produced from a simulation with Large Synoptic Survey Telescope (LSST) Data Challenge 1 (DC1), BUZZARD implies that data comes from a simulation so the models we create can be finetuned when actual data is presented. A portion of our predictor space includes variables denoted as mag.u, mag.i, mag.r, mag.z, mag.y, and mag.g. These are essentially a way of measuring the “brightness” of galaxies. These predictors are valuable because in this study we are attempting to analyze redshift, a way of using visual cues to approximate how far a galaxy is. Essentially based on the kinds of redhsift we see coming from a galaxy, we can determine how far away it is based on the wavelength. We will be using both redshift and the mag’s to model mass of a galaxy. We will first convert the mag predictors to colors however, We will be attempting to relate photoemetry to mass first and then in a different part of the study, we will predict redshift from photometry alone and then use those predictions of redhsift and our original photoemtry used to make those predictions to predict mass. Some high level questions we can answer are: How effectively can we predict redshift from photoemtry? From the predicted redshift and photemtry can we predict galaxy mass?

Our data contains 111,171 galaxie with 14 variables. We will disregard the error terms and proceed with EDA with our mag converted to colors and an r band along with the rest of our predictor space. ‘’’

Exploratory Data Analysis

```
dfTask1=data.frame(df[1:6],df[,13])
dfTask1Col=data.frame(df[1:6][-1] - df[1:6][-ncol(df[1:6])], df[,13])
dfTask2=data.frame(df[1:6][-1] - df[1:6][-ncol(df[1:6])],df[3],df[13:14])
library(tidyverse)
```

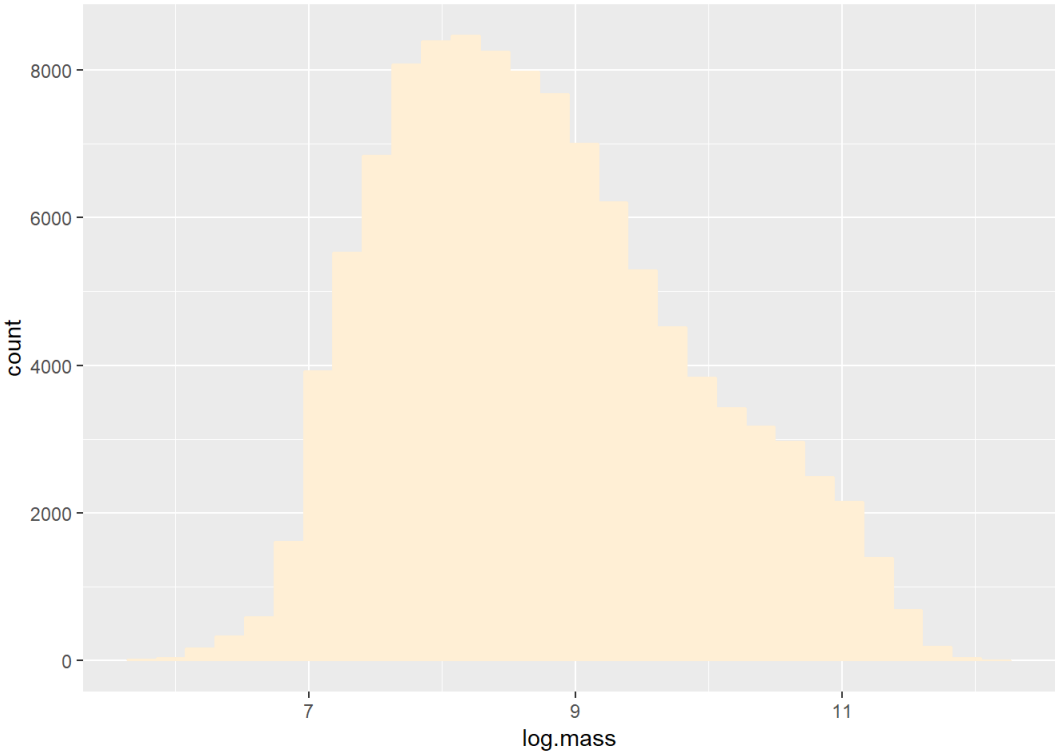
```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
ggplot(data=df,aes(x=log.mass))+geom_histogram(color="papayawhip",fill="papayawhip")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



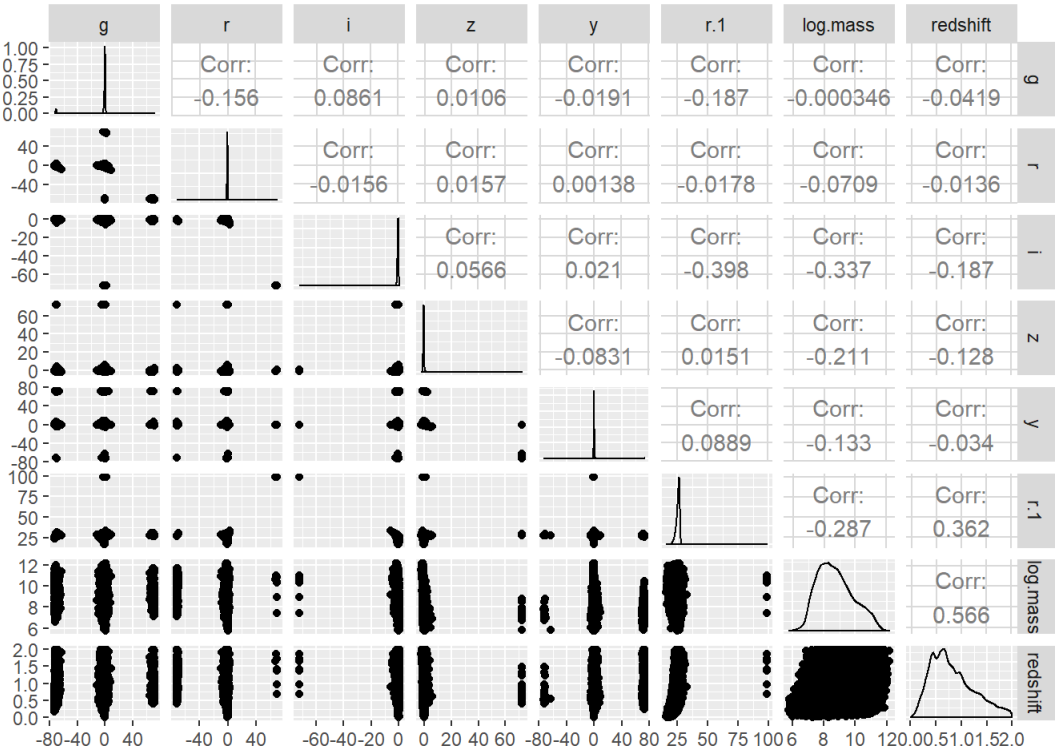
First, let's look at the general distribution of the log.mass galaxies

```
library(GGally)
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

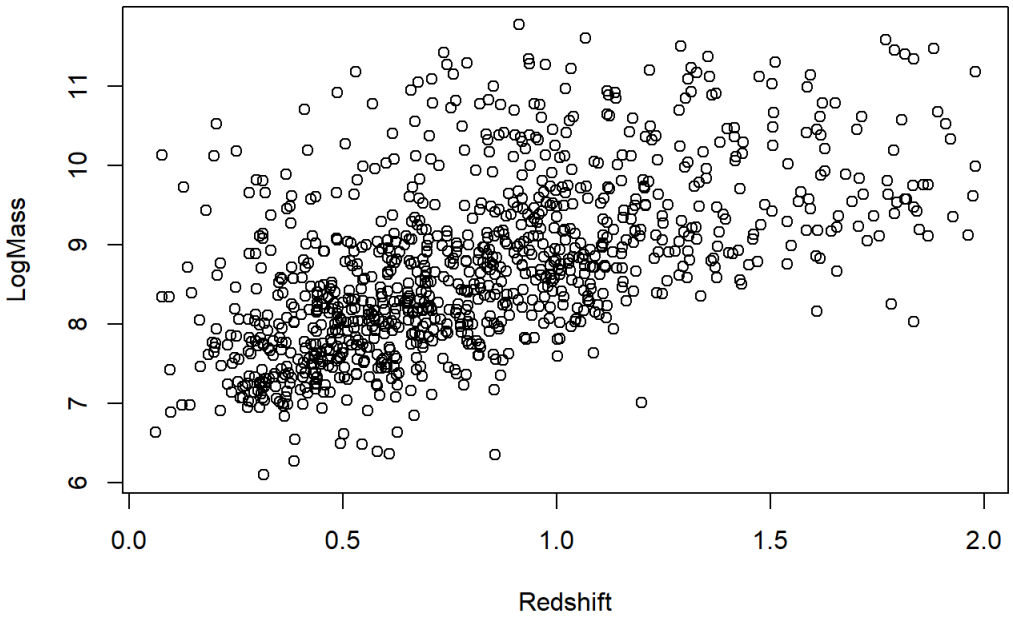
```
ggpairs(dfTask2,progress=FALSE)
```



From the pairs plots we notice right away that the variable most correlated with log.mass seems to be redshift. In fact, redshift and log.mass have a correlation coefficent just under 0.6,which means that redhsift is likely critical in generating the model predicting log.mass. The other good news is that none of the color bands and our mag have strong correlation with each other, this means that we can likely use all of our predictor space in the pairs plot without having to worry about correlations among predictors. Let's explore the relationship between redshift and log.mass more. Let's not include all the data however, as the visual plots can be slightly crowded.

```
set.seed(103)
ind=sample(1:111171, 1000, replace=FALSE)

plot(dfTask2[,8][ind],dfTask2[,7][ind], xlab="Redshift", ylab="LogMass")
```



There is certainly a correlation between our redshift and log.mass. Let's see how well we can predict log.mass with a simple linear model. Later in our analysis, we will use predicted redshift, but for now let us use the given redshift sample.

```
tempFrame=data.frame(dfTask2[,8][ind],dfTask2[,7][ind])
 colnames(tempFrame) <- c("redshift", "log.mass")
linearMod <- lm(log.mass ~ redshift, data=tempFrame)
set.seed(290)
testIndex = sample(1:111171, 1000, replace=FALSE)
pri = data.frame(dfTask2[,8][testIndex])
colnames(pri) <- c("redshift")
pred = predict(linearMod, newdata=pri)
mse = mean((pred-dfTask2[,7][testIndex]))^2
mse
```

```
## [1] 0.0007887128
```

This mse indicates we have a great way of effectively modeling mass from redshift. Now let's explore the relationship between our colors and r-band. Let's start off with a correlation table.

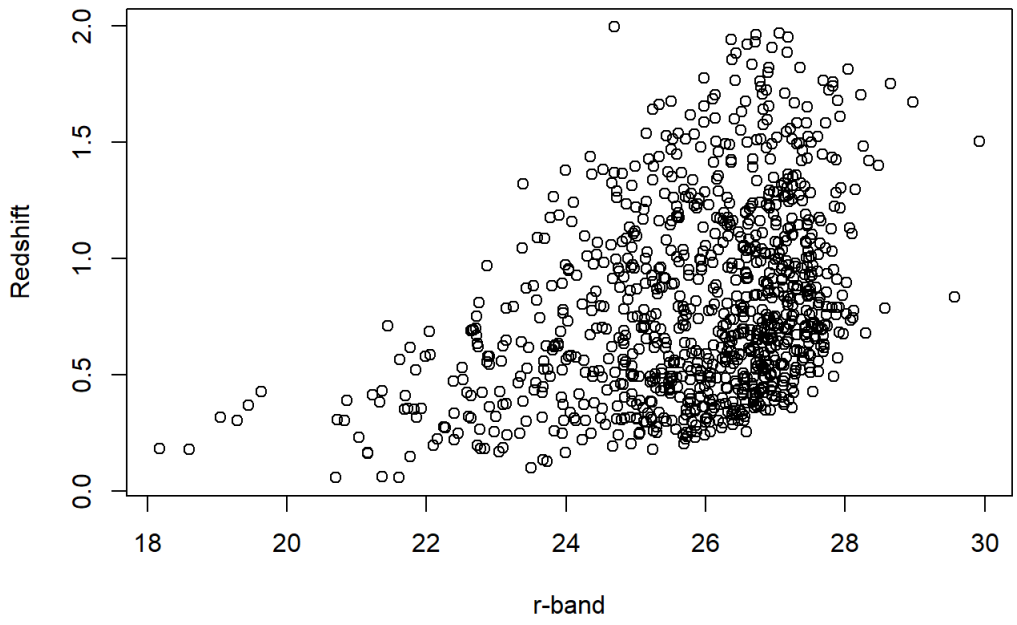
```
onlyPred = data.frame(dfTask2[,1:6],dfTask2[,8])
colnames(onlyPred) <- c("col.u.g","col.g.r","col.r.i","col.i.z","col.z.y","r.band", "redshift")
cor(onlyPred)
```

```
##              col.u.g     col.g.r     col.r.i     col.i.z     col.z.y
## col.u.g   1.00000000 -0.15573216  0.08613305  0.01058620 -0.01905953
## col.g.r  -0.15573216  1.00000000 -0.01556467  0.01570172  0.00138113
## col.r.i   0.08613305 -0.01556467  1.00000000  0.05663049  0.02100961
## col.i.z   0.01058620  0.01570172  0.05663049  1.00000000 -0.08305187
## col.z.y  -0.01905953  0.00138113  0.02100961 -0.08305187  1.00000000
## r.band   -0.18731772 -0.01775553 -0.39810295  0.01508493  0.08892831
## redshift -0.04188639 -0.01360255 -0.18732249 -0.12842290 -0.03403849
##                r.band    redshift
## col.u.g  -0.18731772 -0.04188639
## col.g.r  -0.01775553 -0.01360255
## col.r.i  -0.39810295 -0.18732249
## col.i.z   0.01508493 -0.12842290
## col.z.y   0.08892831 -0.03403849
## r.band    1.00000000  0.36210994
## redshift  0.36210994  1.00000000
```

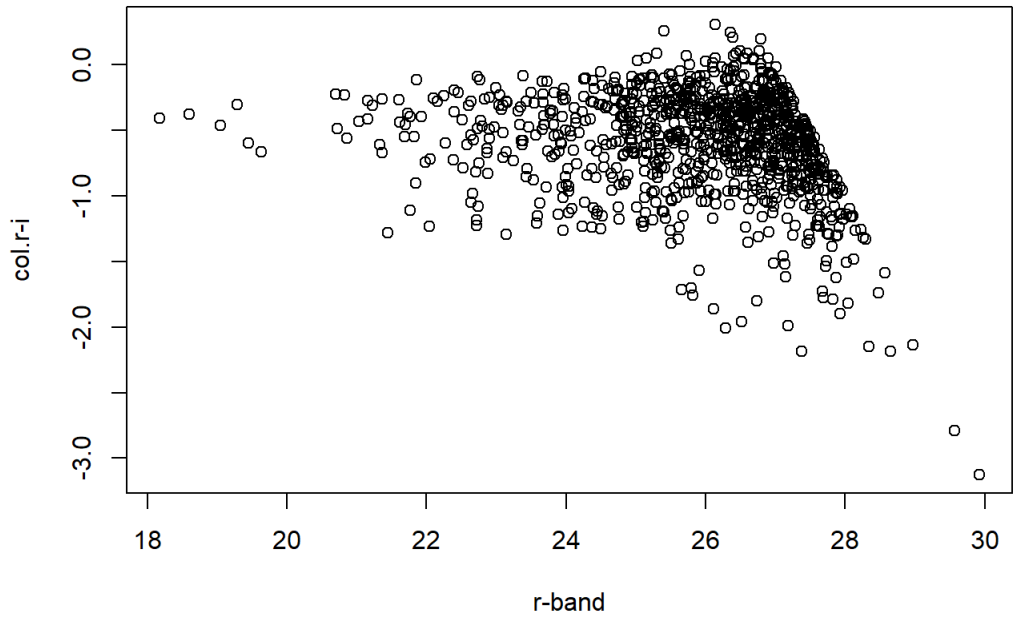
There are not any correlations that stand out from our table, however this simply checks for linear relation ships. Let's see visually how our 2 highest correlations look on a plot. mag.r and redshift as well as col.r -i and mag.r.

```
set.seed(180)
ind=sample(1:111171, 1000, replace=FALSE)

plot(onlyPred[,6][ind],onlyPred[,7][ind], xlab="r-band", ylab="Redshift")
```

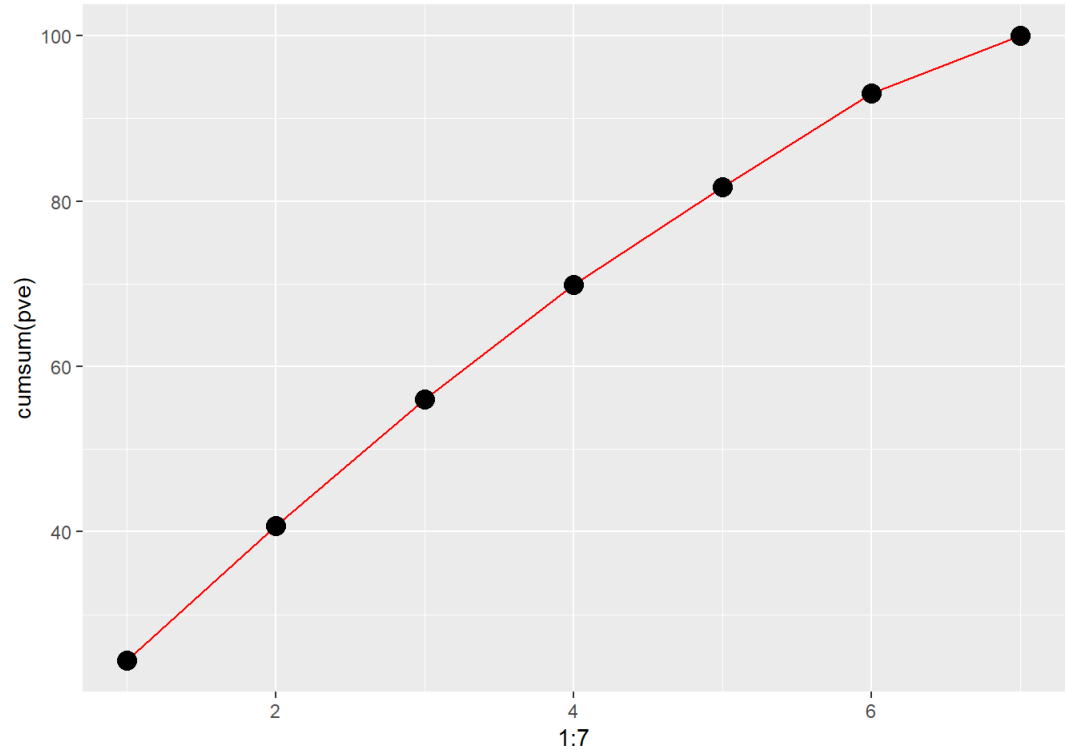



```
plot(onlyPred[,6][ind],onlyPred[,3][ind], xlab="r-band", ylab="col.r-i")
```



Obviously there is not a linear relationship here, but there is some kind of noticeable relationship with th
e both pairs of plots. Let's employ PCA to really weed out the important variables in our predictor space.

```
library(tidyverse)
v=prcomp(onlyPred,scale.=TRUE)
pve =100*v$sdev ^2/sum(v$sdev ^2)
g=data.frame((1:7),pve,cumsum(pve))
ggplot(g, mapping=aes(x=1:7,y=cumsum(pve)))+geom_line(color="red")+geom_point(size=4)
```



Our principal component analysis yields interesting results. There is no noticeable "shoulder" from our cumulative graph. This implies that modeling would best be done using ALL of our predictor space. This conclusion is as expected too, since we did not see any signs of significant collinearity from our earlier correlation table. We will hold to the assumption that the entire predictor space is useful for modeling. As a roadmap of what we are going to do in our modeling: We will first generate the best model for predicting redshift from photemtry and then we will use our best predictions to create a predictor space with our predicted redshifts along with the original colors and band. This will be our final predictor space to attempt to model log.mass(finally!). Our obvious measure of effectiveness will be with the mse(essentially how little error our models generate).

Modeling

We'll kick things off with Ridge regression. First, let's set up the predictor/response split we will use from now on.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded glmnet 2.0-16
```

```
library(Matrix)
library(dbplyr)
```

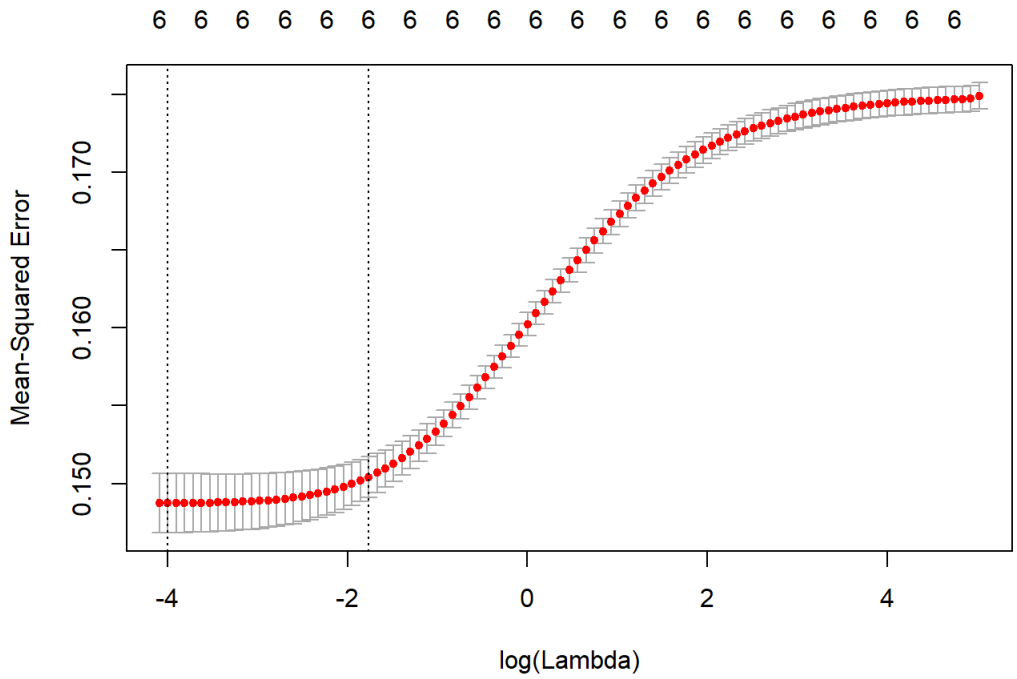
```
##
## Attaching package: 'dbplyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     ident, sql
```

```
library(dplyr)
set.seed(47)
frac = 0.8
len = nrow(onlyPred)
ind = sample(len, frac * len)
pred.train = onlyPred[ind,1:6]
resp.train = onlyPred[ind,7]
RedShiftTrain = data.frame(pred.train,resp.train)
pred.test = onlyPred[-ind,1:6]
resp.test = onlyPred[-ind,7]
x=model.matrix(resp.train~.,pred.train)[,-1]
xTest=model.matrix(resp.test~.,pred.test)[,-1]
y=resp.train
grid=10^seq(10,-2,length=100)
ridge.mod=glmnet(x,y,alpha=0,lambda=grid)
dim(coef(ridge.mod))
```

```
## [1]   7 100
```

```
set.seed(1)
cv.out=cv.glmnet(x,y,alpha=0)
plot(cv.out)
```




```
bestlam =cv.out$lambda.min
ridge.pred=predict(ridge.mod ,s=bestlam ,newx=xTest)
ridgeMean=mean((ridge.pred-resp.test)^2)
ridgeMean
```

```
## [1] 0.1464227
```

```
mseFrameRedShift = data.frame("RidgeMean", ridgeMean)
```

A Ridge model yields a favorable mse. 0.146 is defintely a good place to start predicting redshift from. How ever, let's just save that mse into a mse dataFrame and continue with other modeling techniques. Perhaps we can do even better.

```
library(tree)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(tidyverse)
library(dplyr)
library(FNN)
tree.star=tree(resp.train~.,data=RedShiftTrain)
summary(tree.star)
```

```
##
## Regression tree:
## tree(formula = resp.train ~ ., data = RedShiftTrain)
## Variables actually used in tree construction:
## [1] "col.i.z" "col.z.y" "col.g.r" "col.r.i" "r.band"
## Number of terminal nodes:  10
## Residual mean deviance:  0.08702 = 7738 / 88930
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.14500 -0.16630 -0.03434  0.00000  0.10600  1.60800
```

```
treePred=predict(tree.star,newdata=pred.test)
mseTree=mean((treePred-resp.test)^2)

mat=xgb.DMatrix(data=as.matrix(pred.train),label=resp.train)
set.seed(103)
out=xgb.cv(params=list(objective="reg:linear"),data=mat,nfold=10,nround=20,verbose=0)
numTrees=which.min(out$evaluation_log$test_rmse_mean)
fit=xgboost(mat,nrounds=numTrees, params=list(objective="reg:linear"))
```

```
## [1]  train-rmse:0.414409
## [2]  train-rmse:0.342341
## [3]  train-rmse:0.298503
## [4]  train-rmse:0.272743
## [5]  train-rmse:0.256575
## [6]  train-rmse:0.247200
## [7]  train-rmse:0.241939
## [8]  train-rmse:0.238627
## [9]  train-rmse:0.236359
## [10] train-rmse:0.234726
## [11] train-rmse:0.233648
## [12] train-rmse:0.232836
## [13] train-rmse:0.232082
## [14] train-rmse:0.231436
## [15] train-rmse:0.231094
## [16] train-rmse:0.230590
## [17] train-rmse:0.230185
## [18] train-rmse:0.229906
## [19] train-rmse:0.229374
## [20] train-rmse:0.228790
```

```
pred=predict(fit,newdata=xgb.DMatrix(data=as.matrix(pred.test)))
mseBoost=mean((pred-resp.test)^2)
mseBoost
```

```
## [1] 0.0554249
```

```
mseFrame=array()
arr=seq(1,3)
ii=1
for(ii in arr){
  v=knn.reg(pred.train, y= resp.train, k = ii, algorithm="brute")
  mse2=mean((v$pred-resp.train)^2)
  mseFrame[ii]=mse2


}
out1=knn.reg(pred.test, y= resp.test, k = which.min(mseFrame), algorithm="brute")
dfT=data.frame(mseFrame,arr)
mcrKNeares=mean((out1$pred-resp.test)^2)
mcrKNeares
```

```
## [1] 0.07237175
```

Our MSE improved even more! Standard decisicion trees had an MSE of 0.087. K-Nearest-neighbors yielded an MS E of only 0.072 and Boosting yielded the lowest of the bunch, 0.055! All of these models would be great for generating a set of redshift predictions. We have reached a good stopping point, our MSEs are low and we hav e a handful of ideal models to choose from. Most of the other models do not yield comparable results. Let's add these to our mse data frame and finally model with support vector machines.

```
library(e1071)
svmmodel <- svm(resp.train ~., RedShiftTrain)
predictedY <- predict(svmmodel, pred.test)
svmMse = mean((predictedY-resp.test)^2)
```

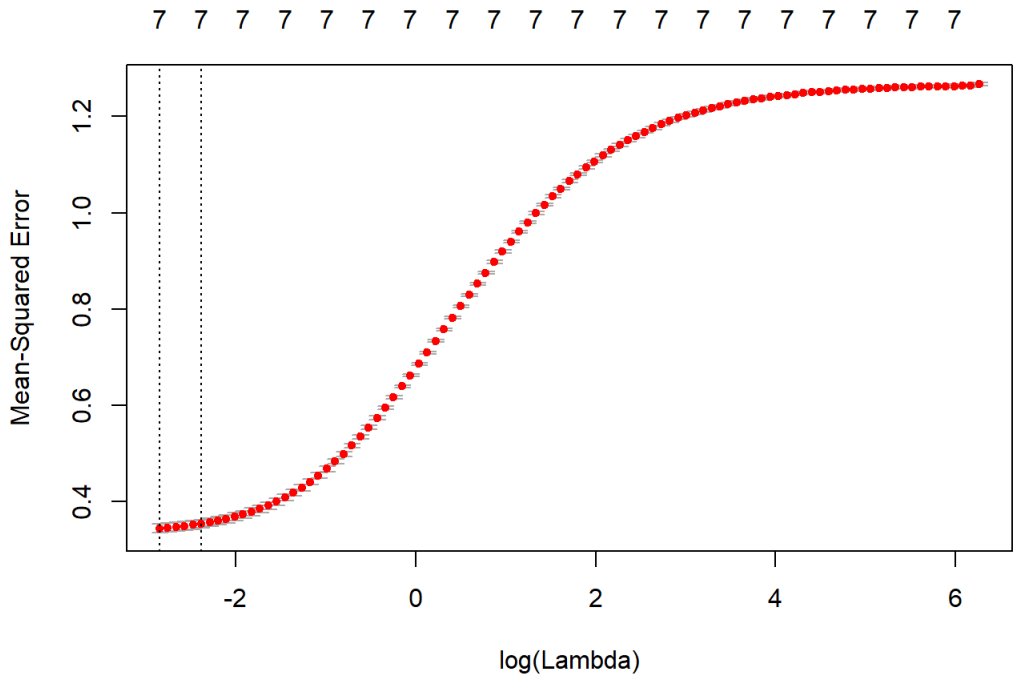
Our svm model yielded an mse of 0.0712! Let's take a look at all the models and respective mse's we have so far.

So now that we have gathered the most noteable mses and their respective models, we can pick a model to gene rate a set of predicted redshifts from. I will employ Boosting because it produced the lowest MSE. Let's cre ate a prediction frame for redshifts, generate a new dataframe with our colors, band and red shift, and fina lly also include mass as our new response! Then we will rinse, repeat, and recycle to find the best model fo r predicting mass.

```
predRedShift=predict(fit,newdata=xgb.DMatrix(data=as.matrix(onlyPred[,1:6])))
MassPred = data.frame(onlyPred[,1:6], predRedShift, df[,13])
set.seed(47)
frac = 0.8
len = nrow(MassPred)
ind = sample(len, frac * len)
pred.trainMass = MassPred[ind,1:7]
resp.trainMass = MassPred[ind,8]
RedShiftTrain = data.frame(pred.trainMass,resp.trainMass)
pred.testMass = MassPred[-ind,1:7]
resp.testMass = MassPred[-ind,8]
xFin=model.matrix(resp.trainMass~.,pred.trainMass)[,-1]
xTestFin=model.matrix(resp.testMass~.,pred.testMass)[,-1]
y=resp.trainMass
grid=10^seq(10,-2,length=100)
ridge.mod=glmnet(xFin,y,alpha=0,lambda=grid)
dim(coef(ridge.mod))
```

```
## [1]   8 100
```

```
set.seed(1)
cv.out=cv.glmnet(xFin,y,alpha=0)
plot(cv.out)
```




```
bestlam =cv.out$lambda.min
ridge.pred=predict(ridge.mod ,s=bestlam ,newx=xTestFin)
ridgeMean=mean((ridge.pred-resp.testMass)^2)
ridgeMean
```

```
## [1] 0.356788
```

```
mseFrameRedShift = data.frame("RidgeMean", ridgeMean)
```

Our mse for ridge modeling came up weak, 0.357. The plot shows high MSE on the y-axis. We definetely will attempt the other models previously used as well. But first, let's look at a variable importance plot with random forest using a smaller portion of our training data to see if maybe some variables are not needed.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

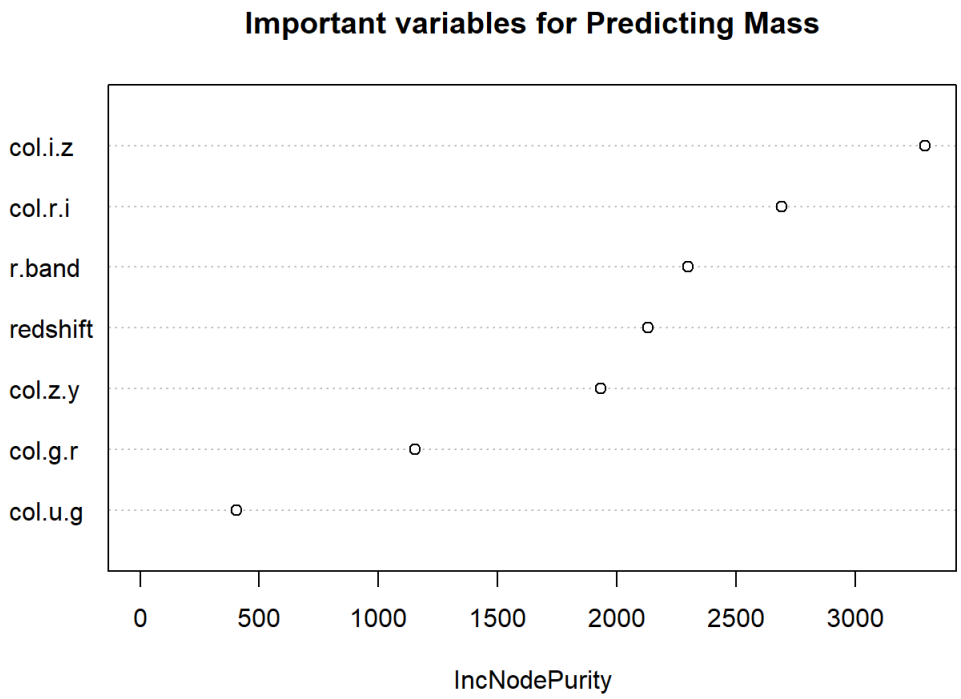
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

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

```
colnames(MassPred) <- c("col.u.g","col.g.r","col.r.i","col.i.z","col.z.y","r.band", "redshift","log.mass")
set.seed(47)
frac = 0.1
len = nrow(MassPred)
ind = sample(len, frac * len)
pred.trainMassSmall = MassPred[ind,1:7]
resp.trainMassSmall = MassPred[ind,8]
smallerFrame=data.frame(pred.trainMassSmall,resp.trainMassSmall)
rf1 <- randomForest(resp.trainMassSmall~.,
                data=smallerFrame)
rf1$importance
```

```
##          IncNodePurity
## col.u.g       405.5208
## col.g.r      1154.1899
## col.r.i      2690.8880
## col.i.z      3290.7928
## col.z.y      1930.5806
## r.band       2297.6263
## redshift     2131.4234
```

```
varImpPlot(rf1,main="Important variables for Predicting Mass")
```



It seems that col.u-g is least important for predicting mass and col.i-z is most important. However since al l of them have purity scores above 400, I not drop any from the model. Let's continue with trees, boosting, and k-nearest neighbors.

```
library(tree)
library(xgboost)
library(tidyverse)
library(dplyr)
library(FNN)
tree.star=tree(resp.trainMass~.,data=pred.trainMass)
summary(tree.star)
```

```
##
## Regression tree:
## tree(formula = resp.trainMass ~ ., data = pred.trainMass)
## Variables actually used in tree construction:
## [1] "col.i.z"      "col.z.y"      "col.r.i"      "r.band"
## [5] "predRedShift"
## Number of terminal nodes:  12
## Residual mean deviance:  0.4475 = 39790 / 88920
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.05400 -0.42650 -0.02444  0.00000  0.40650  3.42900
```

```
treePred=predict(tree.star,newdata=pred.testMass)
mseTree=mean((treePred-resp.testMass)^2)

mat=xgb.DMatrix(data=as.matrix(pred.trainMass),label=resp.trainMass)
set.seed(103)
out=xgb.cv(params=list(objective="reg:linear"),data=mat,nfold=10,nround=20,verbose=0)
numTrees=which.min(out$evaluation_log$test_rmse_mean)
fit=xgboost(mat,nrounds=numTrees, params=list(objective="reg:linear"))
```

```
## [1]  train-rmse:5.835098
## [2]  train-rmse:4.102436
## [3]  train-rmse:2.894197
## [4]  train-rmse:2.052901
## [5]  train-rmse:1.472911
## [6]  train-rmse:1.078497
## [7]  train-rmse:0.816346
## [8]  train-rmse:0.648325
## [9]  train-rmse:0.546577
## [10] train-rmse:0.487193
## [11] train-rmse:0.454608
## [12] train-rmse:0.436597
## [13] train-rmse:0.426226
## [14] train-rmse:0.420527
## [15] train-rmse:0.416903
## [16] train-rmse:0.414569
## [17] train-rmse:0.413074
## [18] train-rmse:0.411717
## [19] train-rmse:0.411102
## [20] train-rmse:0.409865
```

```
pred=predict(fit,newdata=xgb.DMatrix(data=as.matrix(pred.testMass)))
mseBoost=mean((pred-resp.testMass)^2)
mseBoost
```

```
## [1] 0.1900889
```

```
mseFrame=array()
arr=seq(1,3)
ii=1
for(ii in arr){
  v=knn.reg(pred.trainMass, y= resp.trainMass, k = ii, algorithm="brute")
  mse2=mean((v$pred-resp.trainMass)^2)
  mseFrame[ii]=mse2


}
out2=knn.reg(pred.testMass, y= resp.testMass, k = which.min(mseFrame), algorithm="brute")
mcrKNeares=mean((out2$pred-resp.testMass)^2)
mcrKNeares
```

```
## [1] 0.2486489
```

Our MSE for the deciscion tree model was 0.44, for Boositng was 0.19, and for KNN was 0.24. Boosting again p roduced the best MSE from our modeling. Even when our predictor space and response variable changed, Boostin g produced the most accurate model to determine log.mass. An mse of 0.19 leaves much to be desired. Looking back at our variable importance plot, we noticed that col.u-g was given a noticeably less purity score. Let' s drop this predictor from our predictor space and rerun the same analyses.

```
library(tree)
library(xgboost)
library(tidyverse)
library(dplyr)
library(FNN)
tree.star=tree(resp.trainMass~.,data=pred.trainMass[,-1])
summary(tree.star)
```

```
##
## Regression tree:
## tree(formula = resp.trainMass ~ ., data = pred.trainMass[, -1])
## Variables actually used in tree construction:
## [1] "col.i.z"      "col.z.y"      "col.r.i"      "r.band"
## [5] "predRedShift"
## Number of terminal nodes:  12
## Residual mean deviance:  0.4475 = 39790 / 88920
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.05400 -0.42650 -0.02444  0.00000  0.40650  3.42900
```

```
treePred=predict(tree.star,newdata=pred.testMass[,-1])
mseTree=mean((treePred-resp.testMass)^2)

mat=xgb.DMatrix(data=as.matrix(pred.trainMass[,-1]),label=resp.trainMass)
set.seed(103)
out=xgb.cv(params=list(objective="reg:linear"),data=mat,nfold=10,nround=20,verbose=0)
numTrees=which.min(out$evaluation_log$test_rmse_mean)
fit=xgboost(mat,nrounds=numTrees, params=list(objective="reg:linear"))
```

```
## [1]  train-rmse:5.835098
## [2]  train-rmse:4.102437
## [3]  train-rmse:2.894198
## [4]  train-rmse:2.052902
## [5]  train-rmse:1.472914
## [6]  train-rmse:1.078518
## [7]  train-rmse:0.816306
## [8]  train-rmse:0.648587
## [9]  train-rmse:0.546508
## [10] train-rmse:0.487129
## [11] train-rmse:0.454729
## [12] train-rmse:0.436549
## [13] train-rmse:0.426303
## [14] train-rmse:0.420587
## [15] train-rmse:0.417374
## [16] train-rmse:0.414648
## [17] train-rmse:0.412902
## [18] train-rmse:0.411553
## [19] train-rmse:0.410798
## [20] train-rmse:0.409978
```

```
pred=predict(fit,newdata=xgb.DMatrix(data=as.matrix(pred.testMass[,-1])))
mseBoost=mean((pred-resp.testMass)^2)
mseBoost
```

```
## [1] 0.1909936
```

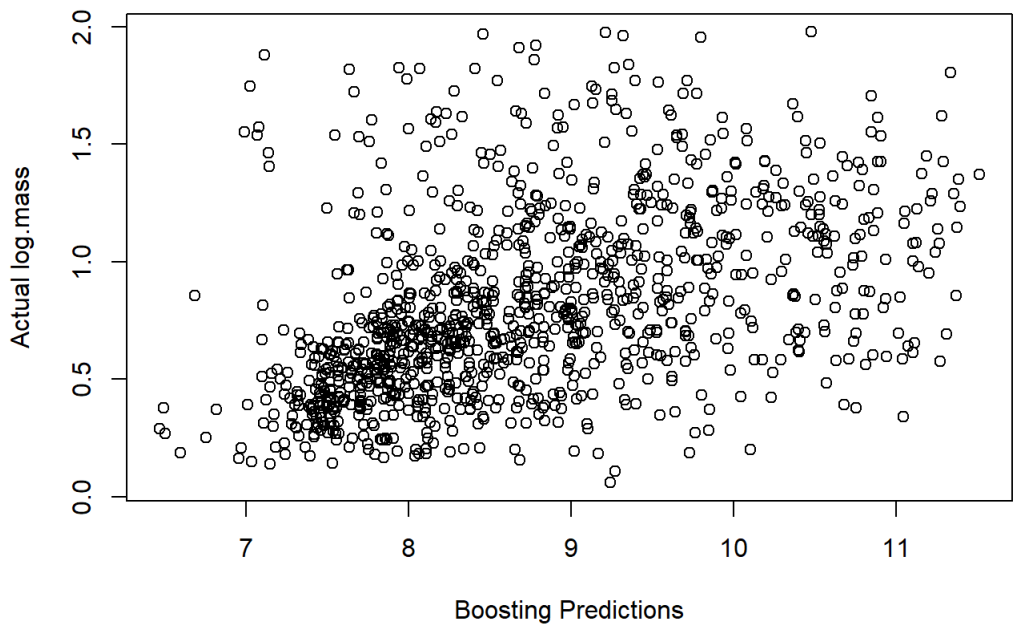
```
mseFrame=array()
arr=seq(1,3)
ii=1
for(ii in arr){
  v=knn.reg(pred.trainMass[,-1], y= resp.trainMass, k = ii, algorithm="brute")
  mse2=mean((v$pred-resp.trainMass)^2)
  mseFrame[ii]=mse2


}
out2=knn.reg(pred.testMass[,-1], y= resp.testMass, k = which.min(mseFrame), algorithm="brute")
mcrKNeares=mean((out2$pred-resp.testMass)^2)
mcrKNeares
```

```
## [1] 0.2354284
```

Our new Mse's after dropping the col.u-g are 0.448, 0.19, and 0.235. Not so "new" though! Clearly the column was not helpful in predicting the mass, so whether we used it in the model does not matter. Since we are goin g with boosting let's look at the plot of observed vs predicted for our boosting predictions.

```
library(tidyverse)
library(dplyr)
ind = sample(length(pred), (0.05 * length(pred)))
plot(pred[ind],resp.test[ind],xlab="Boosting Predictions",ylab="Actual log.mass")
```



Conclusion

The plot displays a relative strong amount of scatter. We have reason to believe the boosting model for pred icting mass from colors and mag.r along with a predicted redshift will be a good model to tune for real gala xy data. Boosting also was effective in predicting the redshift predictor space we would use in the model. T he mse values for KNN, Boosting, Ridge, and Trees were all low when it came to predicting redshift, but did not keep their strength when predicting mass. The Buzzard simulation provided us with the task of modeling r edshift from colors and from there using all of the prior to model mass and in this study we came up with an effective model.