Homework 2

David Contento, Mark Vandre, Pujan Thakrar, and John Macke

May 17, 2019

Multiple Regression

We gathered our dataset from the UC Irvine data machine learning repository. Our data set consists of responses from a survey that was given to several highschool students in Portugal regarding their home, social, and academic life. There are 26 explanatory variables (mostly indicator), and our dependent variabl, which we are trying to predict, is whether the student recived a passing grade (1 if final exam score is greater than 60% and 0 otherwise). We start with setting up a mutiple regression model.

```
#setwd("C:/Users/David/Desktop")
load("data.Rdata")

#Importing and Splitting data
set.seed(42)
data=data[-c(1,15,16,17)]
train=sample(1:nrow(data), round(nrow(data)*.6), replace=F)
datatrain=data[train,]
datatest=data[-train,]
```

First we split the data and performed a backword stepwise regression in order to see which variables where the best at explaining the variation in the dependent variable. The output shows us the model that was selected from the backword step wise function that uses AIC in its selection process.

```
#multiple regression
#using all predictors
reg1=lm(finalscore~., data=datatrain)
summary(reg1)
```

```
##
## Call:
## lm(formula = finalscore ~ ., data = datatrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7509 -0.3507 -0.1296  0.3946  0.9012
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.173183   0.591620   1.983  0.04867 *
## age         -0.031027   0.027886  -1.113  0.26713
## Medu         0.080825   0.037504   2.155  0.03229 *
## Fedu         0.006022   0.037789   0.159  0.87354
## traveltime  -0.071536   0.048174  -1.485  0.13906
## studytime    0.001223   0.039539   0.031  0.97536
## failures    -0.110080   0.046606  -2.362  0.01909 *
## famrel      -0.036259   0.034312  -1.057  0.29184
## freetime     0.021251   0.032740   0.649  0.51700
## goout       -0.067867   0.032351  -2.098  0.03711 *
## Dalc         0.044373   0.045046   0.985  0.32573
## Walc        -0.051070   0.034881  -1.464  0.14465
```

```
## health         0.018881   0.023175   0.815  0.41616
## absences      -0.005697   0.003493  -1.631  0.10444
## school_MS     -0.071360   0.105259  -0.678  0.49856
## Pstatus_T      0.059731   0.102839   0.581  0.56199
## famsize_LE3    0.068349   0.068789   0.994  0.32156
## schoolsup_yes -0.284533   0.095260  -2.987  0.00315 **
## famsup_yes    -0.118412   0.065922  -1.796  0.07389 .
## activities_no -0.022092   0.062669  -0.353  0.72480
## paid_no        0.024196   0.068807   0.352  0.72545
## internet_yes   0.062486   0.081994   0.762  0.44686
## nursery_yes   -0.071582   0.076892  -0.931  0.35295
## higher_yes    -0.018106   0.126979  -0.143  0.88675
## romantic_yes  -0.059444   0.069389  -0.857  0.39260
## address_R      0.044770   0.079456   0.563  0.57372
## sex_F          0.017300   0.073365   0.236  0.81381
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4486 on 210 degrees of freedom
## Multiple R-squared:  0.2397, Adjusted R-squared:  0.1455
## F-statistic: 2.546 on 26 and 210 DF,  p-value: 0.0001343
```

```r
#Performing Backword Stepwise
library(MASS)
stepback=stepAIC(reg1, direction="backward")
```

```r
#Final model from backword stepwise
stepback$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## finalscore ~ age + Medu + Fedu + traveltime + studytime + failures +
##     famrel + freetime + goout + Dalc + Walc + health + absences +
##     school_MS + Pstatus_T + famsize_LE3 + schoolsup_yes + famsup_yes +
##     activities_no + paid_no + internet_yes + nursery_yes + higher_yes +
##     romantic_yes + address_R + sex_F
##
## Final Model:
## finalscore ~ age + Medu + traveltime + failures + goout + absences +
##     schoolsup_yes + famsup_yes
##
##
##               Step Df     Deviance Resid. Df Resid. Dev       AIC
## 1                                        210   42.25848 -354.6485
## 2      - studytime  1 0.0001924685       211   42.25867 -356.6474
## 3     - higher_yes  1 0.0039781913       212   42.26265 -358.6251
## 4           - Fedu  1 0.0041885497       213   42.26684 -360.6016
## 5          - sex_F  1 0.0094065519       214   42.27624 -362.5489
## 6        - paid_no  1 0.0241993694       215   42.30044 -364.4132
## 7  - activities_no  1 0.0274145214       216   42.32786 -366.2597
## 8      - address_R  1 0.0723696007       217   42.40023 -367.8548
## 9      - school_MS  1 0.0843981101       218   42.48462 -369.3835
## 10     - Pstatus_T  1 0.0819035955       219   42.56653 -370.9271
```

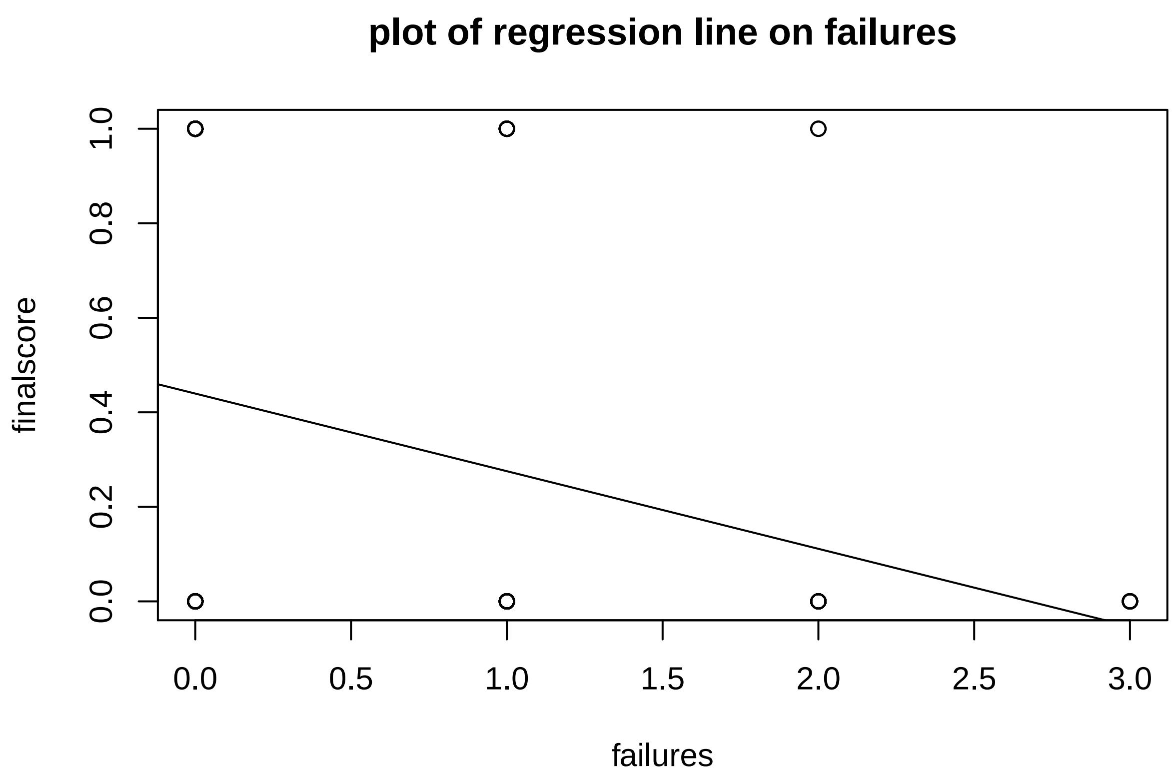
```
## 11      - freetime  1 0.0982373469       220   42.66477 -372.3808
## 12   - famsize_LE3  1 0.1531380914       221   42.81790 -373.5316
## 13  - internet_yes  1 0.1486670018       222   42.96657 -374.7101
## 14        - health  1 0.1233505039       223   43.08992 -376.0307
## 15  - romantic_yes  1 0.1199872862       224   43.20991 -377.3717
## 16        - famrel  1 0.1417607676       225   43.35167 -378.5954
## 17   - nursery_yes  1 0.2095871782       226   43.56126 -379.4524
## 18          - Dalc  1 0.2312919666       227   43.79255 -380.1974
## 19          - Walc  1 0.1441478900       228   43.93670 -381.4185
```

```r
#final backward regression
reg3=lm(finalscore ~ age + Medu + traveltime + failures + goout + absences +
    schoolsup_yes + famsup_yes, data = datatrain)
summary(reg3)
```

```
##
## Call:
## lm(formula = finalscore ~ age + Medu + traveltime + failures +
##     goout + absences + schoolsup_yes + famsup_yes, data = datatrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7408 -0.3712 -0.1572  0.4134  0.9323
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.395356   0.443047   3.149  0.00185 **
## age           -0.042236   0.024848  -1.700  0.09054 .
## Medu           0.078642   0.027673   2.842  0.00489 **
## traveltime    -0.068485   0.041932  -1.633  0.10380
## failures      -0.100664   0.041142  -2.447  0.01517 *
## goout         -0.079433   0.025996  -3.056  0.00251 **
## absences      -0.006031   0.003186  -1.893  0.05964 .
## schoolsup_yes -0.268553   0.088688  -3.028  0.00274 **
## famsup_yes    -0.108192   0.059854  -1.808  0.07199 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.439 on 228 degrees of freedom
## Multiple R-squared:  0.2095, Adjusted R-squared:  0.1817
## F-statistic: 7.551 on 8 and 228 DF,  p-value: 6.098e-09
```

From the output above we can see that most the coefficients are significant, which hopefully will result in a low error when we go to classify the students as pass or failed.

```r
#plotting regression
attach(data)
plot(failures,finalscore, main="plot of regression line on failures")
abline(lm(finalscore~failures))
```



Unfortunately since this model is still just a linear regression we can see from the plot above that it does not do a good job at fitting the data. Especially when the response variable in binary. This will result in less interpretable coefficients and possibly more classification error. Next we validate the mutiple regression model using our testing data and looked at the confusion matrix. Since our regression results will not be binary we selected a threshold and of .5 and rounded all values above the threshold to 1 and all values below the threshold to zero.

```
#confusion matrix (multiple regression)
pred=predict(reg3, newdata = datatest, type="response", se=TRUE)
pred=pred$fit

#setting and using threshold
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
threshold=.5
m=ifelse(pred>=threshold,1,0)

confusionMatrix(factor(m), factor(datatest$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 66 32
```

```
##          1 29 31
##
##                Accuracy : 0.6139
##                  95% CI : (0.5333, 0.6902)
##     No Information Rate : 0.6013
##     P-Value [Acc > NIR] : 0.4058
##
##                   Kappa : 0.1883
##  Mcnemar's Test P-Value : 0.7979
##
##             Sensitivity : 0.6947
##             Specificity : 0.4921
##          Pos Pred Value : 0.6735
##          Neg Pred Value : 0.5167
##              Prevalence : 0.6013
##          Detection Rate : 0.4177
##    Detection Prevalence : 0.6203
##       Balanced Accuracy : 0.5934
##
##        'Positive' Class : 0
##
```

We can see that our model had an accuracy of 61.4%, which is not great. Using the accruacy results from our confusion matrix we can now go on to fit other models and see how well the perform compared to each other. Next we fit a logistic regression model. We again selected our variables using the backword stepwise method.

Logistic Regression

```r
#Logistic Regression
library(stats)
mylogit=glm(finalscore ~., data = datatrain, family = binomial(link="logit"))
```

```r
#Performing Backword Stepwise
stepback=stepAIC(mylogit, direction="backward")
```

It is interesting to note that the backword stepwise regression selected different variables for our logistic regression than it did for our multiple regression model. After selecting the variables for out Logistic model we fit the model and looked at how significant our vaiables where

```r
#Final model from Backword Stepwise
stepback$anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## finalscore ~ age + Medu + Fedu + traveltime + studytime + failures +
##     famrel + freetime + goout + Dalc + Walc + health + absences +
##     school_MS + Pstatus_T + famsize_LE3 + schoolsup_yes + famsup_yes +
##     activities_no + paid_no + internet_yes + nursery_yes + higher_yes +
##     romantic_yes + address_R + sex_F
##
## Final Model:
## finalscore ~ age + Medu + traveltime + failures + goout + absences +
##     schoolsup_yes + famsup_yes
##
```

```
##
##                Step Df    Deviance Resid. Df Resid. Dev      AIC
## 1                                         210   242.8449 296.8449
## 2  - activities_no  1 0.006492719       211   242.8514 294.8514
## 3      - studytime  1 0.017541087       212   242.8690 292.8690
## 4           - Fedu  1 0.041242973       213   242.9102 290.9102
## 5        - paid_no  1 0.075033629       214   242.9853 288.9853
## 6          - sex_F  1 0.077146668       215   243.0624 287.0624
## 7     - higher_yes  1 0.154120701       216   243.2165 285.2165
## 8      - school_MS  1 0.249932665       217   243.4665 283.4665
## 9      - Pstatus_T  1 0.309341237       218   243.7758 281.7758
## 10     - address_R  1 0.381404788       219   244.1572 280.1572
## 11   - famsize_LE3  1 0.390921398       220   244.5481 278.5481
## 12   - nursery_yes  1 0.531517767       221   245.0796 277.0796
## 13      - freetime  1 0.607524628       222   245.6872 275.6872
## 14  - internet_yes  1 0.878736086       223   246.5659 274.5659
## 15  - romantic_yes  1 0.949697280       224   247.5156 273.5156
## 16        - health  1 1.073055744       225   248.5887 272.5887
## 17        - famrel  1 1.309435937       226   249.8981 271.8981
## 18          - Dalc  1 1.564718992       227   251.4628 271.4628
## 19          - Walc  1 1.819543911       228   253.2824 271.2824
```

```
#Logistic Model
mylogit=glm(finalscore ~age + Medu + traveltime + failures + goout + absences +
    schoolsup_yes + famsup_yes
, data = data, family = binomial(link="logit"))
summary(mylogit)
```

```
##
## Call:
## glm(formula = finalscore ~ age + Medu + traveltime + failures +
##     goout + absences + schoolsup_yes + famsup_yes, family = binomial(link = "logit"),
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6426  -0.9816  -0.5271   1.0732   2.2927
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    2.62551    1.78902   1.468 0.142220
## age           -0.14573    0.10283  -1.417 0.156422
## Medu           0.32665    0.11211   2.914 0.003573 **
## traveltime    -0.19241    0.17626  -1.092 0.275010
## failures      -0.87013    0.25069  -3.471 0.000519 ***
## goout         -0.16836    0.10620  -1.585 0.112897
## absences      -0.02390    0.01626  -1.470 0.141449
## schoolsup_yes -1.68849    0.44583  -3.787 0.000152 ***
## famsup_yes    -0.46465    0.23778  -1.954 0.050689 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 526.43  on 394  degrees of freedom
```

```
## Residual deviance: 456.08  on 386  degrees of freedom
## AIC: 474.08
##
## Number of Fisher Scoring iterations: 5
```

As we can see most of our variables are not significant, which is not promising. Next we validated our model using the same testing data as before. First we predict values using the model, then we set a threshold (since our results are not binary) and rounded values above the threshold to one and below to zero. Lastly, we looked at the confusion matrix for our model to see how well it performed

```
# Predict the fitted values given the model and hypothetical data
predictedvalues=predict(mylogit, newdata = datatest, type="response", se=TRUE)
predicted=predictedvalues$fit

#Creating and setting up threshold
library(caret)
threshold=.5
t=ifelse(predicted>=threshold,1,0)

#Conufusion matrix (logit)
confusionMatrix(factor(t), factor(datatest$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 66 27
##          1 29 36
##
##                Accuracy : 0.6456
##                  95% CI : (0.5656, 0.7199)
##     No Information Rate : 0.6013
##     P-Value [Acc > NIR] : 0.1453
##
##                   Kappa : 0.2647
##  Mcnemar's Test P-Value : 0.8937
##
##             Sensitivity : 0.6947
##             Specificity : 0.5714
##          Pos Pred Value : 0.7097
##          Neg Pred Value : 0.5538
##              Prevalence : 0.6013
##          Detection Rate : 0.4177
##    Detection Prevalence : 0.5886
##       Balanced Accuracy : 0.6331
##
##        'Positive' Class : 0
##
```

From the output above we can see that the Logistic model did perform better at classifying students when compared to our multiple regression model. This is expected since logisitc regression model are better are predicting binary variables. Next we went on to fit a ridge regression.

Ridge Regression

```
library(caret)
library(glmnet)
# put into matrix for glmnet function
x = as.matrix(datatrain[,-ncol(data)])
y = as.matrix(datatrain[,ncol(data)])
```

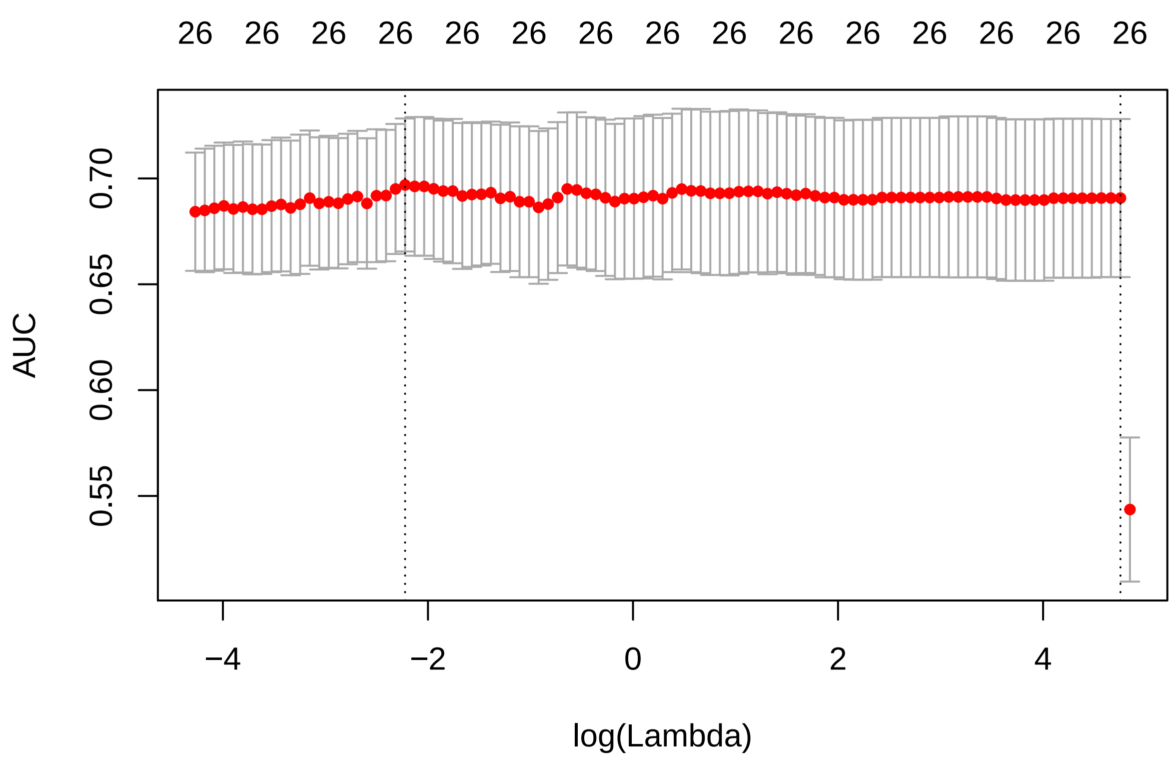
```
# run ridge regression with cross validation
cv.ridge <- cv.glmnet(x, y, family='binomial', alpha=0, parallel=TRUE, standardize=TRUE, type.measure='a
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
best_lambda = cv.ridge$lambda.min
print(best_lambda)
```

```
## [1] 0.108278
```

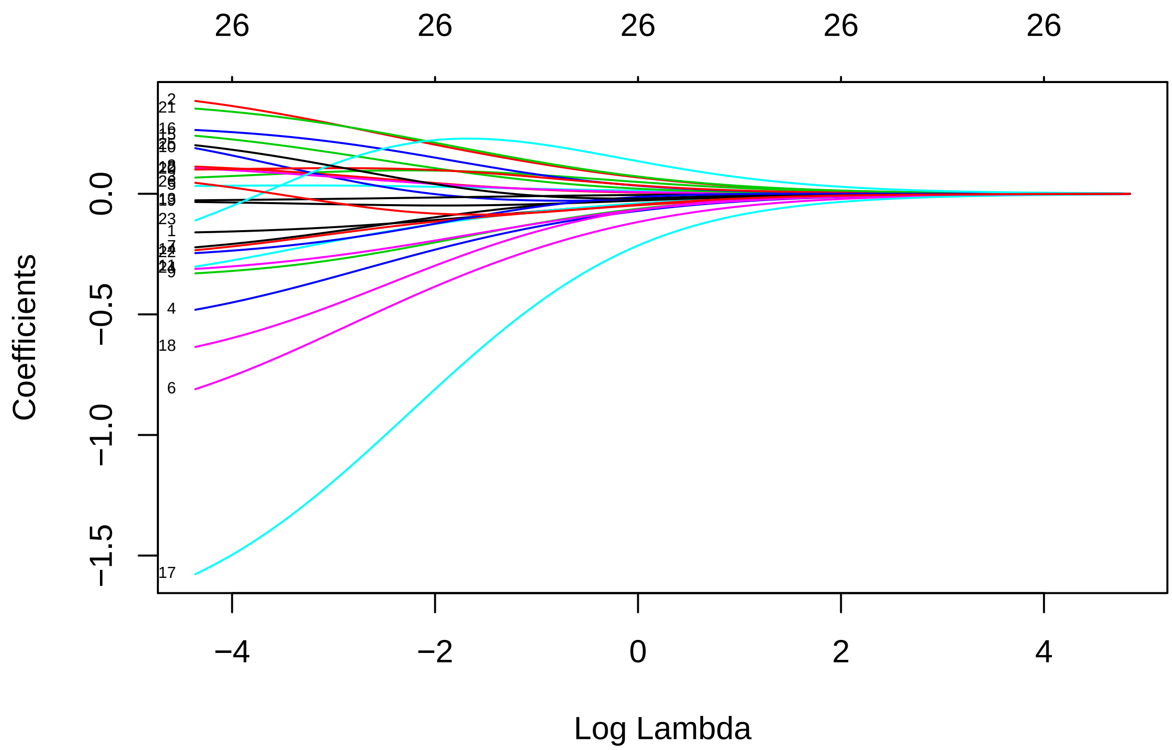
```
fit = cv.ridge$glmnet.fit
plot(cv.ridge)
```



The above plot shows the AUC (area under the ROC curve) on the y-axis and the log(lambda) on the x-axis. The function uses 10-fold cross validation to find the best lambda by selecting the lambda with the smallest mean cross-validated error. The best lambda is 0.53.

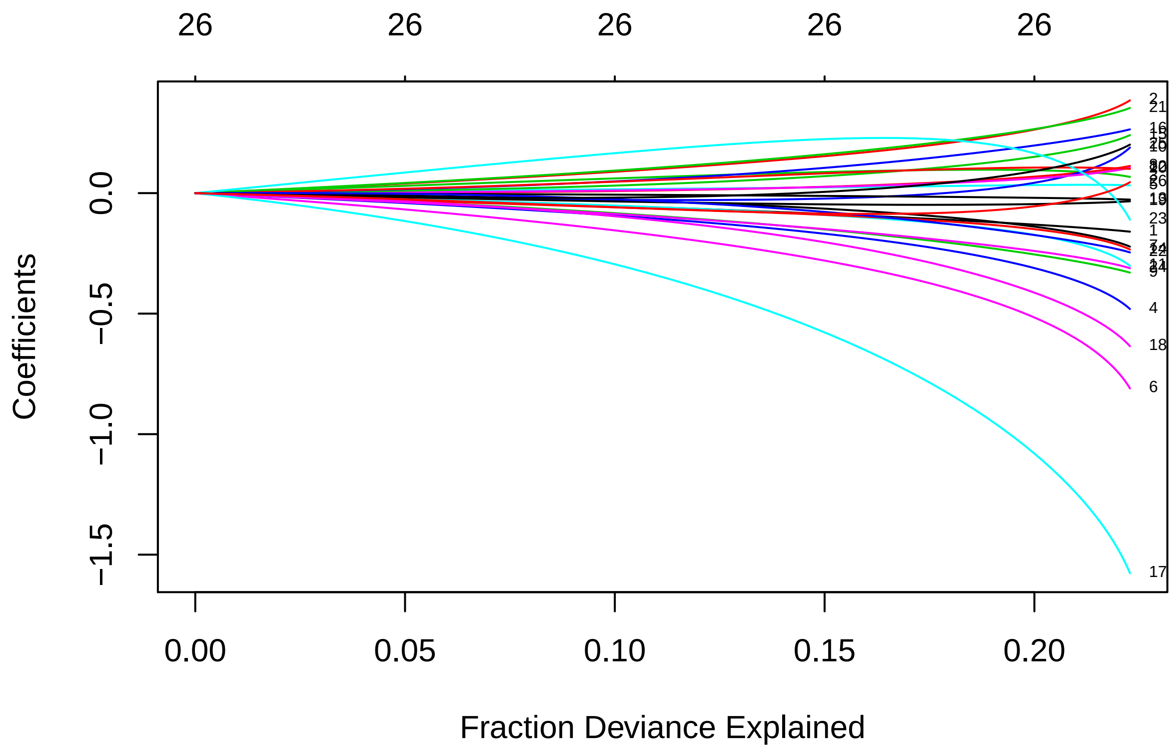
```
# run ridge regression
ridge = glmnet(x,y, family = "binomial", alpha = 0)

# get coefficients from optimal lambda found using cv.glmnet
```

```r
ridge.coef = coef(ridge, s = best_lambda, exact = T)


# plot decay of parameters
plot(ridge,xvar="lambda",label=T)
```



The plot above shows us lambda on the x-axis and the coefficients of the predictor variables on the y-axis. The numbers refer to the actual coefficient of a particular variable. Across the top of the plot is the number of variables used in the model. Remember this number never changes when doing ridge regression. We can see that most of the coefficeints of the predictors decay to zero at about the same rate. However, a few of them decay slower.

```r
# plot the deviance explained
plot(ridge,xvar='dev',label=T)
```



The second plot shows us the deviance explained on the x-axis and the coefficients of the predictor variables on the y-axis.

```
# predict
test.x = as.matrix(datatest[,1:(ncol(datatest)-1)])
ridge.pred = predict(ridge, newx = test.x, type = "response", s = best_lambda)

# calculate mean squared error
threshold = 0.5
t = ifelse(ridge.pred >= threshold, 1, 0)
ridge.resid = t-datatest$finalscore
mse.ridge = mean(ridge.resid^2)
mse.ridge
```

```
## [1] 0.3860759
```

```
#creating threshold and calculating confusion matrix
threshold = 0.5
t = ifelse(ridge.pred >= threshold, 1, 0)
confusionMatrix(factor(t), factor(datatest$finalscore))
```

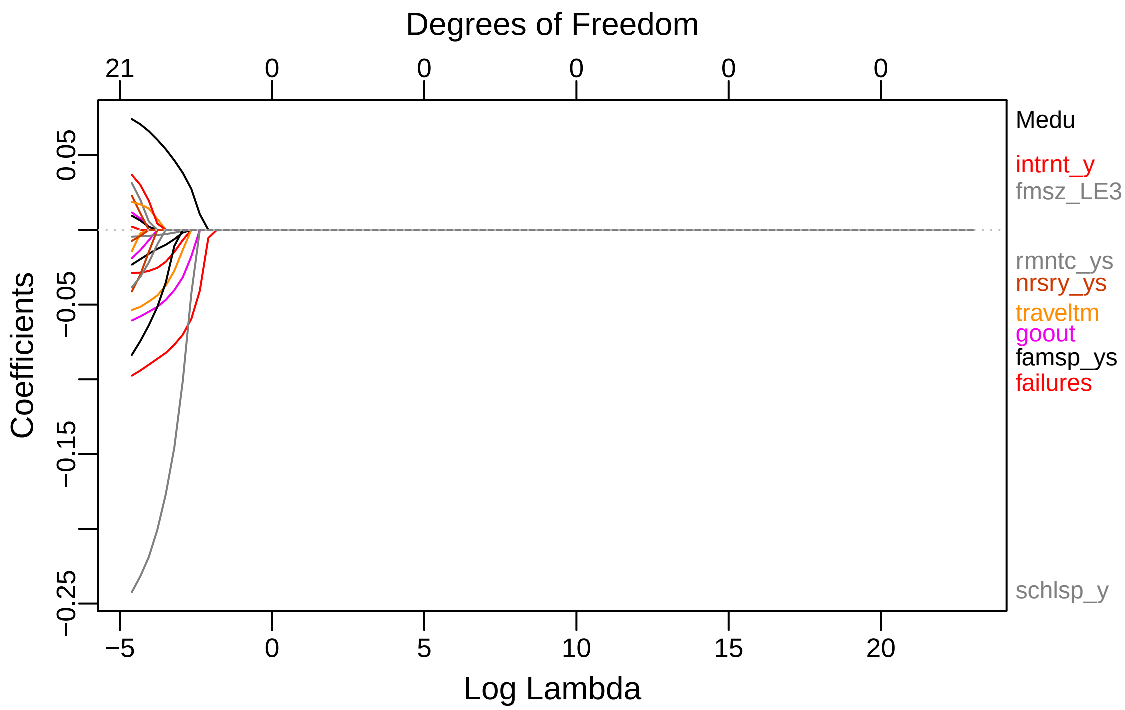
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 72 38
##          1 23 25
##
```

```
##                Accuracy : 0.6139
##                  95% CI : (0.5333, 0.6902)
##     No Information Rate : 0.6013
##     P-Value [Acc > NIR] : 0.40578
##
##                   Kappa : 0.1612
##  Mcnemar's Test P-Value : 0.07305
##
##             Sensitivity : 0.7579
##             Specificity : 0.3968
##          Pos Pred Value : 0.6545
##          Neg Pred Value : 0.5208
##              Prevalence : 0.6013
##          Detection Rate : 0.4557
##    Detection Prevalence : 0.6962
##       Balanced Accuracy : 0.5774
##
##        'Positive' Class : 0
##
```

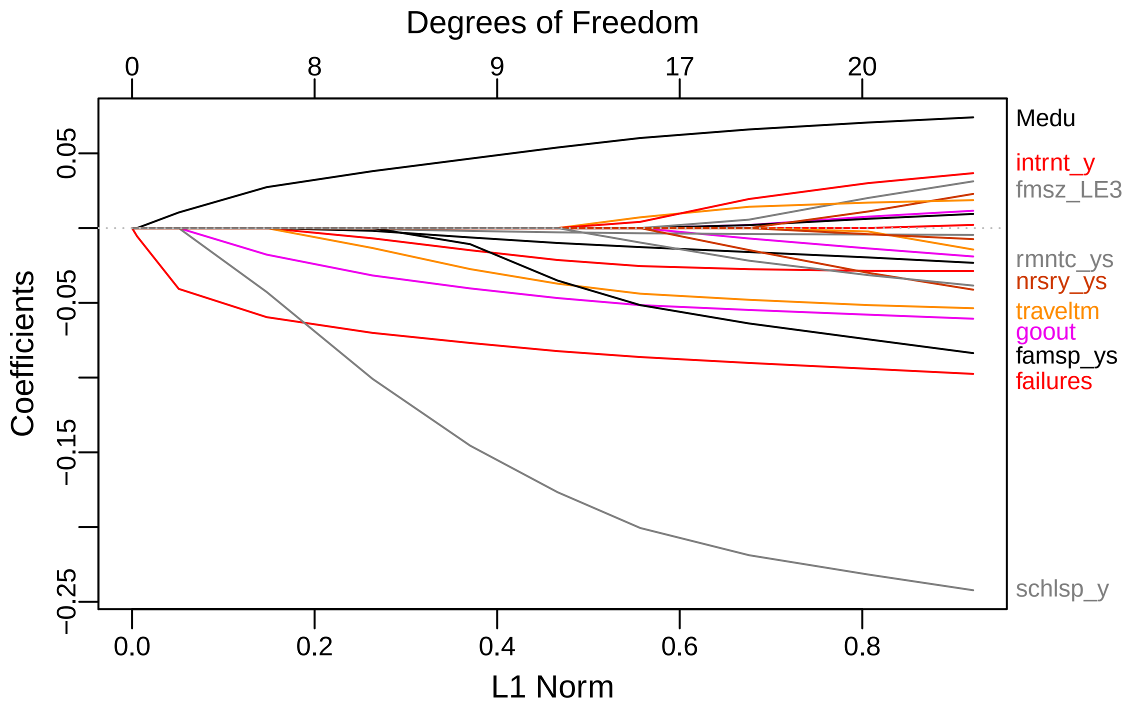
LASSO

```r
#load in libraries
library(plotmo)
library(glmnet)
library(caret)
```

```r
#setting up LASSO model
grid = 10^seq(10,-2, length =100)

lasso.mod = glmnet(x, y, alpha = 1, lambda = grid)
#plot(lasso.mod, xvar = "lambda", label = TRUE)
plot_glmnet(lasso.mod, xvar = "lambda", label = 10)
```

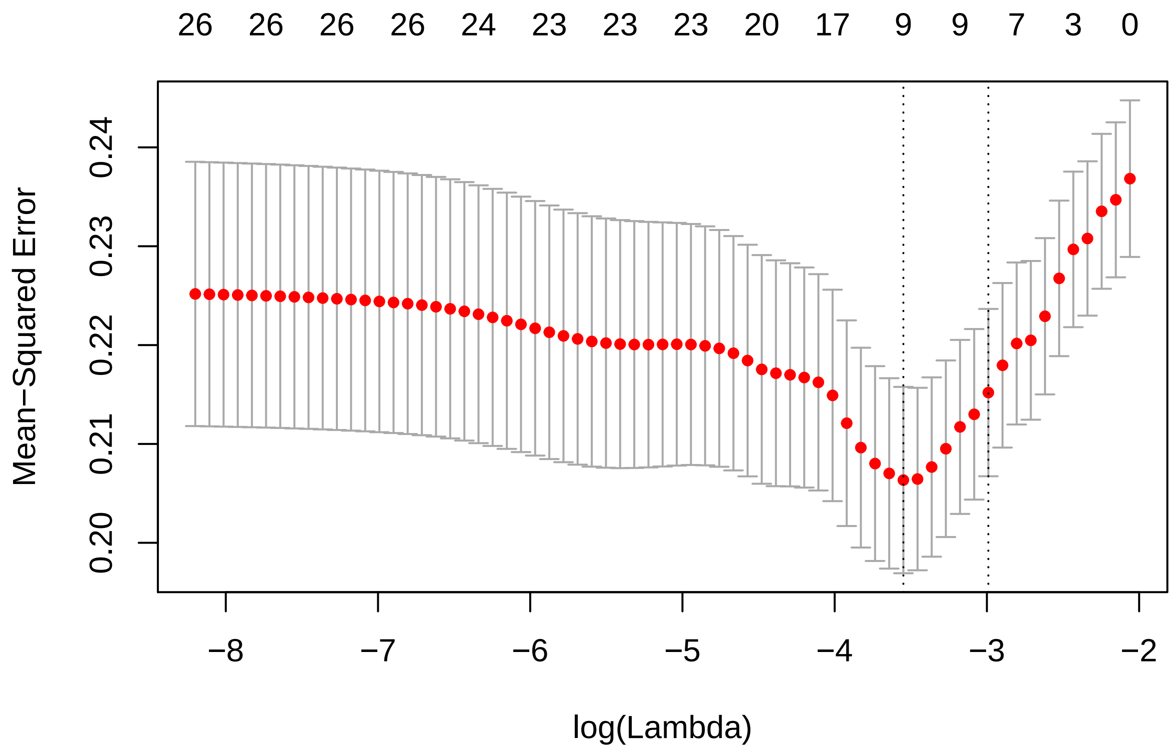



```
plot_glmnet(lasso.mod, xvar = "norm", label = 10)
```



As we can see, the coefficients reduce to zero before a log-lambda of 0.

```
#determing best lambda to use
cv.out = cv.glmnet(x, y, alpha = 1, nfolds = nrow(x), grouped = FALSE)
plot(cv.out)
```




```r
bestlam = cv.out$lambda.min

lasso.pred = predict(lasso.mod, s = bestlam, newx = x)
```

As shown in the plot above, the minimum lambda occurs near a log-lambda between -3 and -4. As shown in the output below, this corresponds with a lambda value of about 0.029. This is the value of lambda we will use in fitting and evaluating our lasso regression. For this cross-validation, we used leave-one-out cross-validation.

```r
print(paste("Minimum Lambda:", as.character(cv.out$lambda.min), sep = " "))
```

```
## [1] "Minimum Lambda: 0.0287595958551548"
```

```r
out = glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef = predict(out, type = "coefficients", s = bestlam)
lasso.coef
```

```
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)    0.927671223
## age           -0.022330351
## Medu           0.055430758
## Fedu           .
## traveltime    -0.038775196
## studytime      .
## failures      -0.083263142
## famrel         .
## freetime       .
```

```
## goout         -0.047942980
## Dalc           .
## Walc          -0.010627191
## health         .
## absences      -0.002985701
## school_MS      .
## Pstatus_T      .
## famsize_LE3    .
## schoolsup_yes -0.182426816
## famsup_yes    -0.039045134
## activities_no  .
## paid_no        0.001723661
## internet_yes   0.000999334
## nursery_yes    .
## higher_yes     .
## romantic_yes  -0.002317134
## address_R      .
## sex_F          .
```

From the coefficients above, we can see the variables that lasso preserves and being predictive. The variables kept include age, mother's education, travel time, failures, going out with friends, weekend alcohol consumption, nubmber of absences, school support, family support, internet connection, and romantic relationships.

```
#Setting Threshold and calculating confusion matrix
threshold = 0.5
t2 = ifelse(lasso.pred >= threshold, 1, 0)
conf = confusionMatrix(factor(t2), factor(y))
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134  53
##          1  14  36
##
##                Accuracy : 0.7173
##                  95% CI : (0.6554, 0.7737)
##     No Information Rate : 0.6245
##     P-Value [Acc > NIR] : 0.001668
##
##                   Kappa : 0.3396
##  Mcnemar's Test P-Value : 3.443e-06
##
##             Sensitivity : 0.9054
##             Specificity : 0.4045
##          Pos Pred Value : 0.7166
##          Neg Pred Value : 0.7200
##              Prevalence : 0.6245
##          Detection Rate : 0.5654
##    Detection Prevalence : 0.7890
##       Balanced Accuracy : 0.6549
##
##        'Positive' Class : 0
##
```

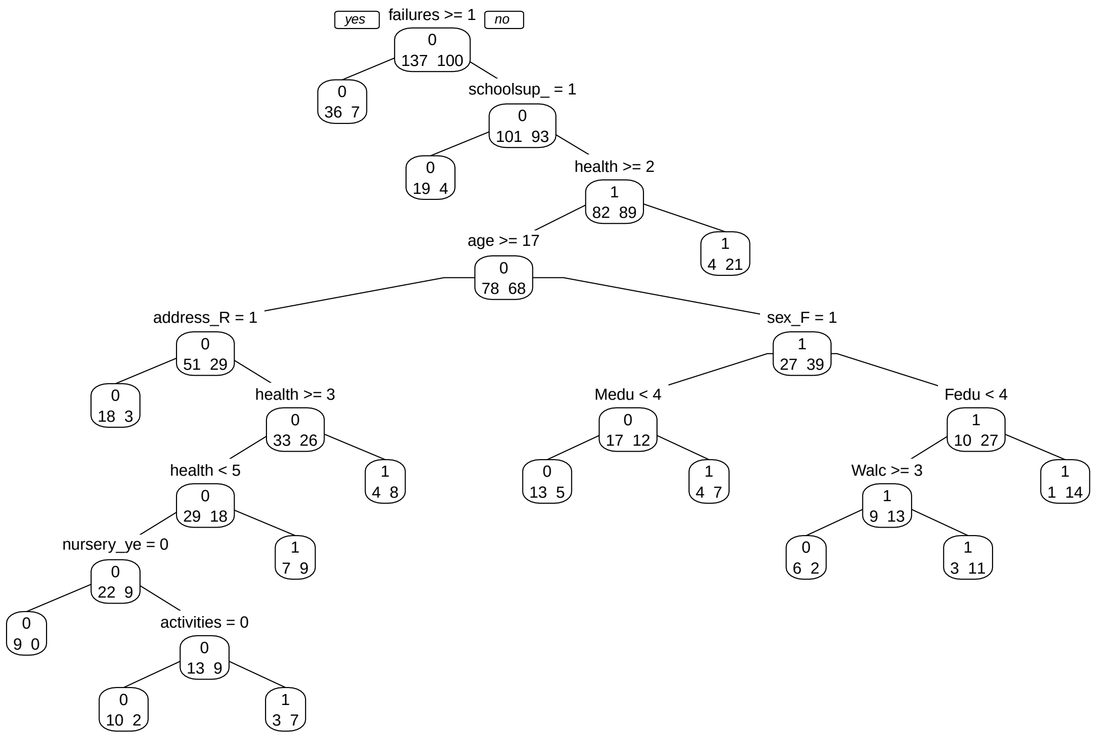
From the confusion matrix and results metrics above, we can see that we have an overall accuracy of about 71.7%. In some contexts, this score might be poor. In this context, however, this result may be acceptable given we have relatively few observations to work with.

Random Forest

```
library(rpart)
library(rpart.plot)
library(caret)
library(randomForest)
library(adabag)
######################################
#split into training and testing sets#
######################################

set.seed(1)
train=sample(1:nrow(data), round(nrow(data)*.6), replace=F)
datatrain=data[train,]
datatest=data[-train,]

train.df <- datatrain
valid.df <- datatest
```

Below, we visualize the importance of the variables in the dataset by creating a default decision tree. The decision tree below shows the optimal splits in the data for each variable.

```
################
#Decision Trees#
################

class.tree <- rpart(train.df$finalscore ~ ., data = train.df, method = "class")
## plot tree
prp(class.tree, type = 1, extra = 1, split.font = 1, varlen = -10)
```



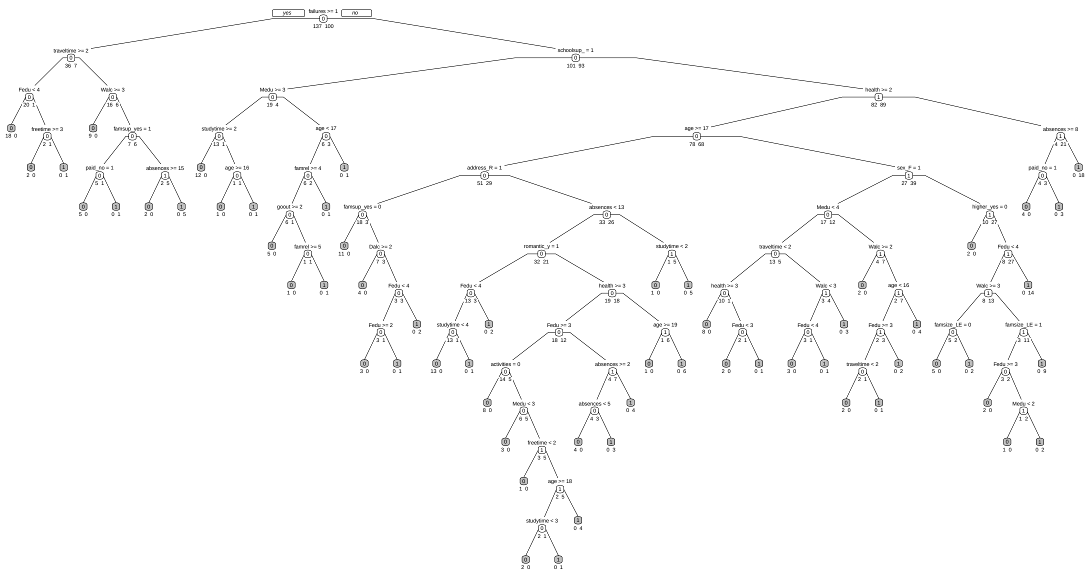
Each node is split by variables in the dataset.

If we want a fuller decision tree, we can use a complexity parameter of 0 and a minplit parameter that indicated the number of observations that must be present in order to attempt a split. Setting the minsplit equal to one gives us the "fully grown" tree.

```
deeper.ct <- rpart(train.df$finalscore ~ ., data = train.df, method = "class", cp = 0, minsplit = 1)
# count number of leaves
length(deeper.ct$frame$var[deeper.ct$frame$var == "<leaf>"])
```

```
## [1] 59
```

```
# plot tree
prp(deeper.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10,
    box.col=ifelse(deeper.ct$frame$var == "<leaf>", 'gray', 'white'))
```



Now, we want to take these trees and fit them to the training set.

```
################################
#Fitting trees to training set#
################################

# fit default tree to the training set
class.tree.ct.point.pred.train <- predict(class.tree,train.df,type = "class")
# generate confusion matrix for training data

confusionMatrix(class.tree.ct.point.pred.train, as.factor(train.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111  23
##          1  26  77
##
##                Accuracy : 0.7932
##                  95% CI : (0.736, 0.843)
##     No Information Rate : 0.5781
##     P-Value [Acc > NIR] : 2.162e-12
##
##                   Kappa : 0.5779
##  Mcnemar's Test P-Value : 0.7751
##
```

```
##              Sensitivity : 0.8102
##              Specificity : 0.7700
##           Pos Pred Value : 0.8284
##           Neg Pred Value : 0.7476
##               Prevalence : 0.5781
##           Detection Rate : 0.4684
##     Detection Prevalence : 0.5654
##        Balanced Accuracy : 0.7901
##
##         'Positive' Class : 0
##
```

```
### repeat the code for the validation set, and the deeper tree
```

With the default tree, we see that accuracy is very high compared to the other models we have fit so far. Now, we do this again for the deeper tree.

```
# fit deeper tree to training set
deeper.ct.point.pred.train <- predict(deeper.ct,train.df,type = "class")
# generate confusion matrix for training data
confusionMatrix(deeper.ct.point.pred.train, as.factor(train.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   0   1
##          0 137   0
##          1   0 100
##
##                 Accuracy : 1
##                   95% CI : (0.9846, 1)
##      No Information Rate : 0.5781
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                    Kappa : 1
##   Mcnemar's Test P-Value : NA
##
##              Sensitivity : 1.0000
##              Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 1.0000
##               Prevalence : 0.5781
##           Detection Rate : 0.5781
##     Detection Prevalence : 0.5781
##        Balanced Accuracy : 1.0000
##
##         'Positive' Class : 0
##
```

Accuracy with the full tree is 100%. Although this is a significantly better model for the training set, we are at risk of overfitting the model. This may result in poor predictions for out-of-sample data.

Because of this, we will prune our tree. The method we choose to prune our tree is the cross validation method using 5 folds. We will also increase the minimum split to 5 observations and print out the complexity parameter that occurs at each step.

```r
############
#Prune tree#
############


cv.ct <- rpart(finalscore ~ ., data = train.df, method = "class",
               cp = 0.00001, minsplit = 5, xval = 5)
# use printcp() to print the table.
printcp(cv.ct)
```

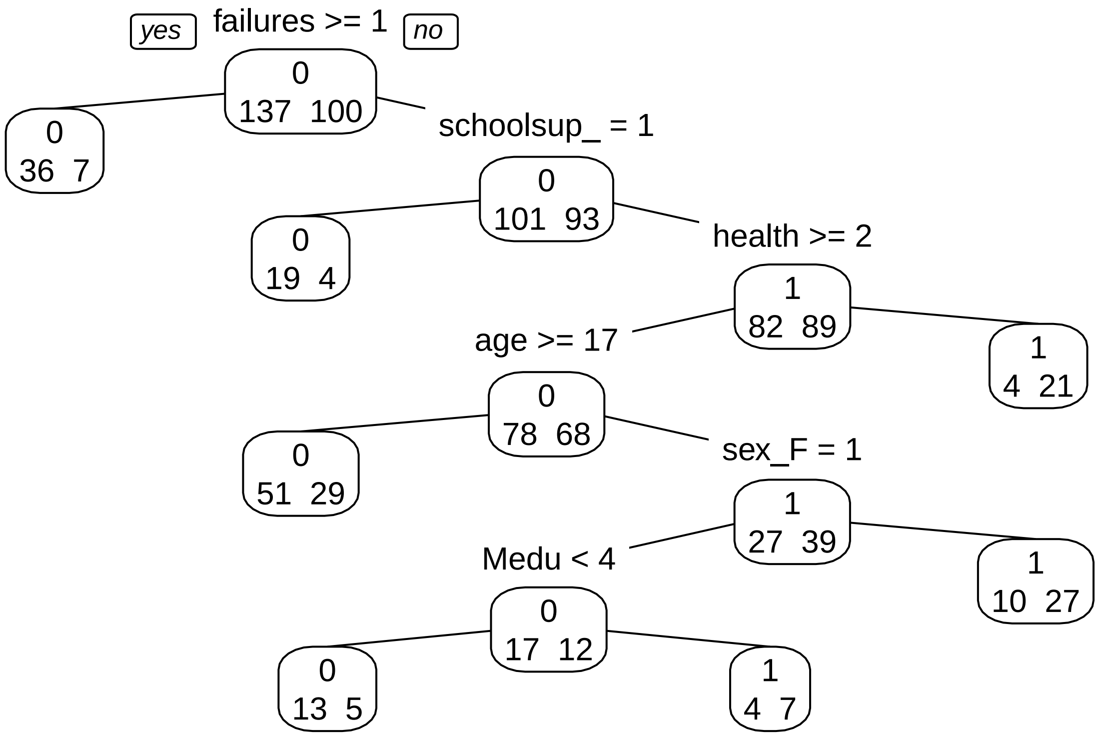
```
##
## Classification tree:
## rpart(formula = finalscore ~ ., data = train.df, method = "class",
##     cp = 1e-05, minsplit = 5, xval = 5)
##
## Variables actually used in tree construction:
##  [1] absences      activities_no address_R     age           Dalc
##  [6] failures      famrel        famsize_LE3   famsup_yes    Fedu
## [11] health        higher_yes    Medu          paid_no       romantic_yes
## [16] schoolsup_yes sex_F         traveltime    Walc
##
## Root node error: 100/237 = 0.42194
##
## n= 237
##
##          CP nsplit rel error xerror     xstd
## 1 0.0725000      0      1.00   1.00 0.076030
## 2 0.0500000      4      0.71   1.00 0.076030
## 3 0.0300000      5      0.66   0.92 0.075025
## 4 0.0200000      6      0.63   0.90 0.074715
## 5 0.0150000     18      0.37   0.93 0.075171
## 6 0.0100000     23      0.29   0.93 0.075171
## 7 0.0066667     32      0.19   0.94 0.075311
## 8 0.0050000     38      0.15   0.94 0.075311
## 9 0.0000100     40      0.14   0.94 0.075311
```

We can prune the tree by using the lowest xerror.

```r
# prune by lower cp
pruned.ct <- prune(cv.ct,
                   cp = cv.ct$cptable[which.min(cv.ct$cptable[,"xerror"]),"CP"])
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
```

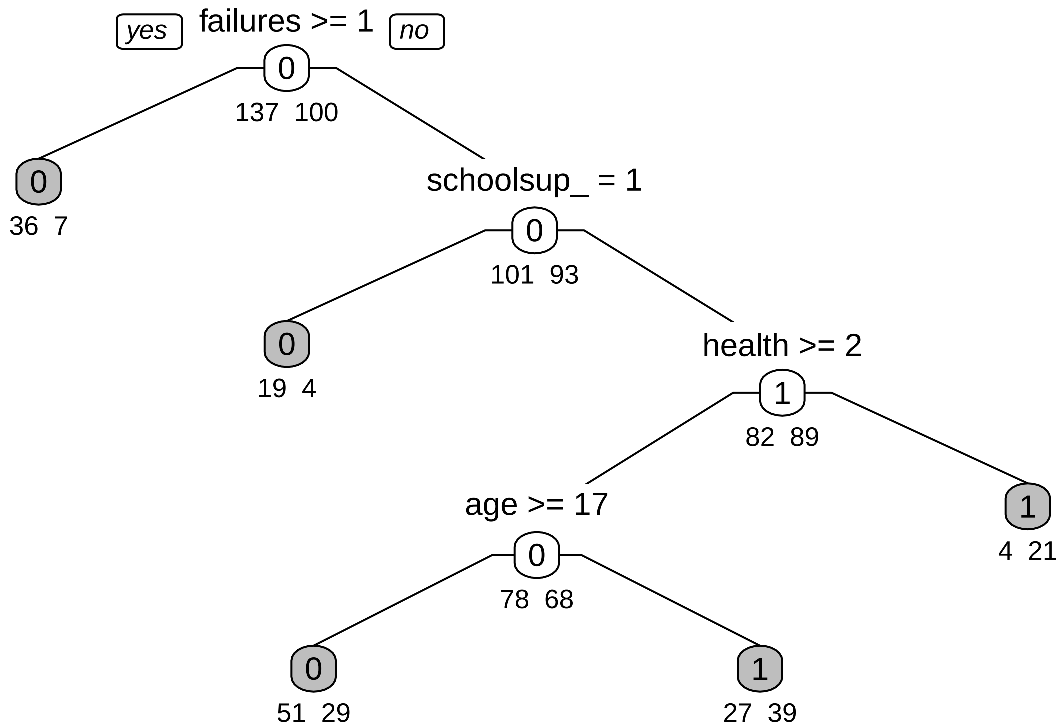
```
## [1] 7
```

```r
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```



By adding one xstd to the minimum xerror, we can get the best pruned tree.

```
#best pruned trees
best.pruned.ct <- prune(cv.ct, cp = 0.05)
prp(best.pruned.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10,
    box.col=ifelse(best.pruned.ct$frame$var == "<leaf>", 'gray', 'white'))
```



We can fit our trees to the training sets

```
pruned.ct.point.pred.train <- predict(pruned.ct,train.df,type = "class")
# generate confusion matrix for training data
confusionMatrix(pruned.ct.point.pred.train, as.factor(train.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 119  45
##          1  18  55
##
##                Accuracy : 0.7342
##                  95% CI : (0.6731, 0.7893)
##     No Information Rate : 0.5781
##     P-Value [Acc > NIR] : 4.268e-07
##
##                   Kappa : 0.4345
##  Mcnemar's Test P-Value : 0.001054
##
##             Sensitivity : 0.8686
##             Specificity : 0.5500
##          Pos Pred Value : 0.7256
##          Neg Pred Value : 0.7534
##              Prevalence : 0.5781
##          Detection Rate : 0.5021
```

```
##    Detection Prevalence : 0.6920
##       Balanced Accuracy : 0.7093
##
##        'Positive' Class : 0
##
```

```
best.pruned.ct.point.pred.train <- predict(best.pruned.ct,train.df,type = "class")
# generate confusion matrix for training data
confusionMatrix(best.pruned.ct.point.pred.train, as.factor(train.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 106  40
##          1  31  60
##
##                Accuracy : 0.7004
##                  95% CI : (0.6377, 0.758)
##     No Information Rate : 0.5781
##     P-Value [Acc > NIR] : 6.839e-05
##
##                   Kappa : 0.3783
##  Mcnemar's Test P-Value : 0.3424
##
##             Sensitivity : 0.7737
##             Specificity : 0.6000
##          Pos Pred Value : 0.7260
##          Neg Pred Value : 0.6593
##              Prevalence : 0.5781
##          Detection Rate : 0.4473
##    Detection Prevalence : 0.6160
##       Balanced Accuracy : 0.6869
##
##        'Positive' Class : 0
##
```

Since the pruned tree has the highest accuracy on the training set while still adhering to the parsimony principle, we choose this to fit to the testing set.

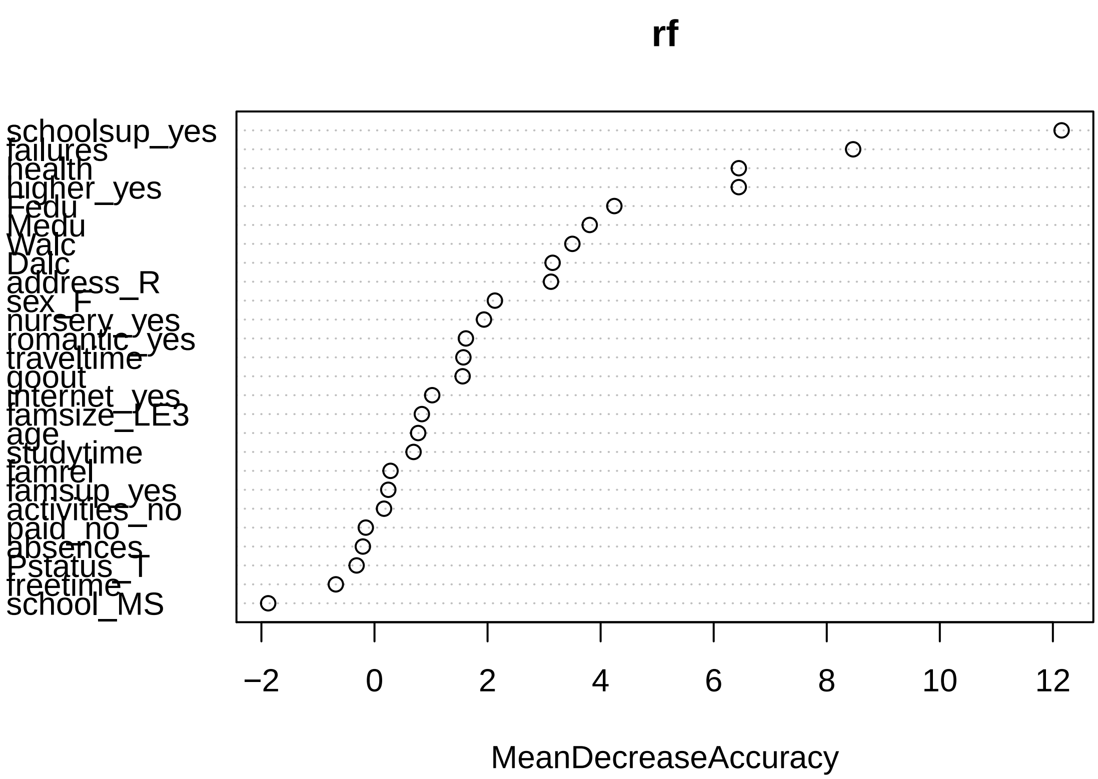
```
pruned.ct.point.pred.valid <- predict(pruned.ct,valid.df,type = "class")
# generate confusion matrix for training data
confusionMatrix(pruned.ct.point.pred.valid, as.factor(valid.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 84 29
##          1 22 23
##
##                Accuracy : 0.6772
##                  95% CI : (0.5983, 0.7493)
##     No Information Rate : 0.6709
##     P-Value [Acc > NIR] : 0.4701
##
```

```
##                  Kappa : 0.2431
##  Mcnemar's Test P-Value : 0.4008
##
##             Sensitivity : 0.7925
##             Specificity : 0.4423
##          Pos Pred Value : 0.7434
##          Neg Pred Value : 0.5111
##              Prevalence : 0.6709
##          Detection Rate : 0.5316
##    Detection Prevalence : 0.7152
##       Balanced Accuracy : 0.6174
##
##        'Positive' Class : 0
##
```

Now, we will use the random forests method instead.

```r
## random forest
rf <- randomForest(as.factor(finalscore) ~ ., data = train.df, ntree = 500,
                   mtry = 4, nodesize = 5, importance = TRUE)

## variable importance plot
varImpPlot(rf, type = 1)
```




```r
## confusion matrix
rf.pred <- predict(rf, valid.df)
confusionMatrix(rf.pred, as.factor(valid.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 88 26
##          1 18 26
##
##                Accuracy : 0.7215
##                  95% CI : (0.6447, 0.7898)
##     No Information Rate : 0.6709
##     P-Value [Acc > NIR] : 0.1009
##
##                   Kappa : 0.3437
##  Mcnemar's Test P-Value : 0.2913
##
##             Sensitivity : 0.8302
##             Specificity : 0.5000
##          Pos Pred Value : 0.7719
##          Neg Pred Value : 0.5909
##              Prevalence : 0.6709
##          Detection Rate : 0.5570
##    Detection Prevalence : 0.7215
##       Balanced Accuracy : 0.6651
##
##        'Positive' Class : 0
##
```

The random forest method yields a significantly higher accuracy rate.

We can also try the boosted tree method below.

```
#Boosting Random Forest
train.df$finalscore <- as.factor(train.df$finalscore)
boost <- boosting(finalscore ~ ., data = train.df)
pred <- predict(boost, valid.df)
confusionMatrix(as.factor(pred$class), as.factor(valid.df$finalscore))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 76 24
##          1 30 28
##
##                Accuracy : 0.6582
##                  95% CI : (0.5787, 0.7317)
##     No Information Rate : 0.6709
##     P-Value [Acc > NIR] : 0.6668
##
##                   Kappa : 0.2481
##  Mcnemar's Test P-Value : 0.4962
##
##             Sensitivity : 0.7170
##             Specificity : 0.5385
##          Pos Pred Value : 0.7600
##          Neg Pred Value : 0.4828
```

```
##              Prevalence : 0.6709
##          Detection Rate : 0.4810
##    Detection Prevalence : 0.6329
##       Balanced Accuracy : 0.6277
##
##        'Positive' Class : 0
##
```

The boosted method does not do as good of a job as the random forests method.

Based on our results we believe that the best model would be a Random Forest model. This model not only has the highest accuracy but also does a better job at selecting variables in order to satisfy the variance-bias tradeoff. Although Randome Forest was the best model in this case, it may not always be the best at predicting or the least complex (parsimony principle).