Annand Final Project Code

Joseph Annand

2023-12-15

## Load Libraries

library(ISLR2)  
library(MASS)

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:ISLR2':  
##   
## Boston

library(leaps)

## Warning: package 'leaps' was built under R version 4.3.2

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.2

## Loading required package: Matrix

## Loaded glmnet 4.1-8

library(pls)

## Warning: package 'pls' was built under R version 4.3.2

##   
## Attaching package: 'pls'

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

library(boot)  
library(tree)

## Warning: package 'tree' was built under R version 4.3.2

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.3.2

## randomForest 4.7-1.1

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

library(gbm)

## Warning: package 'gbm' was built under R version 4.3.2

## Loaded gbm 2.1.8.1

library(BART)

## Warning: package 'BART' was built under R version 4.3.2

## Loading required package: nlme

## Loading required package: nnet

## Loading required package: survival

##   
## Attaching package: 'survival'

## The following object is masked from 'package:boot':  
##   
## aml

## Load wine data sets

# Import wine quality data  
wine.data <- read.csv("winequality-white.csv", sep=";", na.strings = "?", stringsAsFactors = T)  
View(wine.data)

The data set consists of 4898 observations with 12 attributes connected to each observation. Each observation is a record of 12 different characteristics of a different type of wine. The characteristics of interest include fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxides, total sulfur dioxides, total sulfur dioxides, density, pH, sulphates, alcohol, and quality.

For the purpose of this research, we are interested in determining the best regression model to predict the quality of the wine using some, or all, of the other characteristics in the data set.

white\_cor <- cor(wine.data, use="complete.obs")  
print(white\_cor[12, ])

## fixed.acidity volatile.acidity citric.acid   
## -0.113662831 -0.194722969 -0.009209091   
## residual.sugar chlorides free.sulfur.dioxide   
## -0.097576829 -0.209934411 0.008158067   
## total.sulfur.dioxide density pH   
## -0.174737218 -0.307123313 0.099427246   
## sulphates alcohol quality   
## 0.053677877 0.435574715 1.000000000

The table shows the correlations between each attribute and quality. The correlation to quality is 1.000 as expected. There are not any other attributes particularly strongly correlated to the quality of white wine. Alcohol and density have the correlation coefficients of greatest magnitude. Chlorides, volatile acidity, and total sulfur dioxide are next strongly correlated to quality.

## Track test errors across models

model.errors <- data.frame(model = c("Least Squares", "Ridge Regression", "The LASSO", "PLS",  
 "Best Subset", "Forward Stepwise", "Backward Stepwise", "Tree",   
 "Pruned Tree", "Bagging", "RF", "Boosting"),  
 test.error = rep(NA, 12))

### Multiple Linear Regression

lm.wine <- lm(quality ~ ., data = wine.data)  
summary(lm.wine)

##   
## Call:  
## lm(formula = quality ~ ., data = wine.data)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.8348 -0.4934 -0.0379 0.4637 3.1143   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.502e+02 1.880e+01 7.987 1.71e-15 \*\*\*  
## fixed.acidity 6.552e-02 2.087e-02 3.139 0.00171 \*\*   
## volatile.acidity -1.863e+00 1.138e-01 -16.373 < 2e-16 \*\*\*  
## citric.acid 2.209e-02 9.577e-02 0.231 0.81759   
## residual.sugar 8.148e-02 7.527e-03 10.825 < 2e-16 \*\*\*  
## chlorides -2.473e-01 5.465e-01 -0.452 0.65097   
## free.sulfur.dioxide 3.733e-03 8.441e-04 4.422 9.99e-06 \*\*\*  
## total.sulfur.dioxide -2.857e-04 3.781e-04 -0.756 0.44979   
## density -1.503e+02 1.907e+01 -7.879 4.04e-15 \*\*\*  
## pH 6.863e-01 1.054e-01 6.513 8.10e-11 \*\*\*  
## sulphates 6.315e-01 1.004e-01 6.291 3.44e-10 \*\*\*  
## alcohol 1.935e-01 2.422e-02 7.988 1.70e-15 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.7514 on 4886 degrees of freedom  
## Multiple R-squared: 0.2819, Adjusted R-squared: 0.2803   
## F-statistic: 174.3 on 11 and 4886 DF, p-value: < 2.2e-16

The F-statistic for the multiple linear regression with all predictors is well above 1, meaning our regression fit is statistically significant. The predictors that are statistically significant and have a p-value less than 0.05 are residual sugar, free sulfur dioxide, density, pH, sulphates, and alcohol.

# Create training and test data  
set.seed(10)  
train.wine <- sample(1:nrow(wine.data), 0.5 \* nrow(wine.data))  
test.wine <- (-train.wine)

# Train a linear model with statistically significant predictors  
lm.train <- lm(quality ~ volatile.acidity + residual.sugar + free.sulfur.dioxide  
 + density + pH + sulphates + alcohol, data = wine.data,  
 subset = train.wine)  
summary(lm.train)

##   
## Call:  
## lm(formula = quality ~ volatile.acidity + residual.sugar + free.sulfur.dioxide +   
## density + pH + sulphates + alcohol, data = wine.data, subset = train.wine)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.6963 -0.5029 -0.0299 0.4676 2.7588   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.145e+02 2.001e+01 5.721 1.19e-08 \*\*\*  
## volatile.acidity -1.824e+00 1.532e-01 -11.912 < 2e-16 \*\*\*  
## residual.sugar 6.649e-02 8.020e-03 8.291 < 2e-16 \*\*\*  
## free.sulfur.dioxide 2.928e-03 9.647e-04 3.035 0.002432 \*\*   
## density -1.133e+02 2.002e+01 -5.660 1.69e-08 \*\*\*  
## pH 3.764e-01 1.071e-01 3.515 0.000448 \*\*\*  
## sulphates 6.084e-01 1.384e-01 4.397 1.14e-05 \*\*\*  
## alcohol 2.389e-01 2.897e-02 8.245 2.66e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.7496 on 2441 degrees of freedom  
## Multiple R-squared: 0.2748, Adjusted R-squared: 0.2727   
## F-statistic: 132.1 on 7 and 2441 DF, p-value: < 2.2e-16

A multiple linear regression model was trained using half of the observations from the white wine data set. Unlike the first model that included all the possible predictor variables, the trained model used the seven statistically significant predictors from the first model. The F-statistic for the trained model is still much greater than one. All the p-values for the predictors are less than 0.05, so each predictor is significant in this model.

# Use trained linear model to predict the wine quality  
lm.predict <- predict(lm.train, wine.data[test.wine, ])  
lm.mse <- mean((lm.predict - wine.data$quality[test.wine])^2)  
lm.mse

## [1] 0.5698636

model.errors[model.errors$model == "Least Squares", "test.error"] <- lm.mse

The test mean squared error calculated using the trained model is approximately 0.570, meaning that on average, the prediction of the quality rating for a white wine is off by 0.755 units.

# Train the first linear model  
first.lm.wine <- lm(quality ~ ., data = wine.data, subset = train.wine)  
summary(first.lm.wine)

##   
## Call:  
## lm(formula = quality ~ ., data = wine.data, subset = train.wine)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.7823 -0.4940 -0.0422 0.4655 2.7949   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.971e+02 3.262e+01 6.040 1.77e-09 \*\*\*  
## fixed.acidity 1.173e-01 3.218e-02 3.646 0.000272 \*\*\*  
## volatile.acidity -1.741e+00 1.608e-01 -10.827 < 2e-16 \*\*\*  
## citric.acid -9.765e-02 1.340e-01 -0.729 0.466327   
## residual.sugar 9.651e-02 1.233e-02 7.828 7.31e-15 \*\*\*  
## chlorides -6.428e-01 8.578e-01 -0.749 0.453708   
## free.sulfur.dioxide 3.384e-03 1.200e-03 2.820 0.004842 \*\*   
## total.sulfur.dioxide -1.155e-04 5.495e-04 -0.210 0.833529   
## density -1.975e+02 3.305e+01 -5.977 2.61e-09 \*\*\*  
## pH 7.697e-01 1.581e-01 4.868 1.20e-06 \*\*\*  
## sulphates 7.251e-01 1.419e-01 5.109 3.48e-07 \*\*\*  
## alcohol 1.330e-01 4.123e-02 3.225 0.001276 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.7478 on 2437 degrees of freedom  
## Multiple R-squared: 0.2795, Adjusted R-squared: 0.2762   
## F-statistic: 85.93 on 11 and 2437 DF, p-value: < 2.2e-16

# Use first linear model to predict the wine quality  
first.lm.predict <- predict(first.lm.wine, wine.data[test.wine, ])  
first.lm.mse <- mean((first.lm.predict - wine.data$quality[test.wine])^2)  
first.lm.mse

## [1] 0.5731913

Rounding to three decimal places, the test mean squared error for the trained linear model with all predictors is 0.573, which is slightly greater than that of the trained model with seven predictors. The F-statistic is noticeably lower for the model with all predictors, but it is still well above 1.

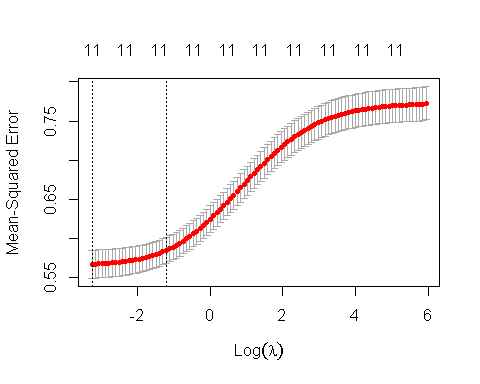
### Ridge Regression

# Create matrix of x, the predictors, and vector of y, the response  
x <- model.matrix(quality ~ ., wine.data)[, -1]  
y <- wine.data$quality  
  
y.test <- y[test.wine]

# Create a lambda grid and use it to form ridge regression model  
lambda.grid <- 10^seq(10, -2, length = 100)  
ridge.mod <- glmnet(x[train.wine, ], y[train.wine], alpha = 0, lambda = lambda.grid,  
 thresh = 1e-12)  
summary(ridge.mod)

## Length Class Mode   
## a0 100 -none- numeric  
## beta 1100 dgCMatrix S4   
## df 100 -none- numeric  
## dim 2 -none- numeric  
## lambda 100 -none- numeric  
## dev.ratio 100 -none- numeric  
## nulldev 1 -none- numeric  
## npasses 1 -none- numeric  
## jerr 1 -none- numeric  
## offset 1 -none- logical  
## call 6 -none- call   
## nobs 1 -none- numeric

# Determine the best lambda, or tuning parameter, using cross-validation  
set.seed(2)  
cv.out <- cv.glmnet(x[train.wine, ], y[train.wine], alpha = 0)  
plot(cv.out)



bestlam.ridge <- cv.out$lambda.min  
bestlam.ridge

## [1] 0.0384732

Cross-validation is used to determine the best lambda, or tuning parameter for the ridge regression model. The plot shows the mean squared errors plotted over the log of the tuning parameter. The tuning parameter that minimizes the mean squared error is 0.0384732.

# Predict the response of test data using ridge regression with best tuning parameter  
ridge.pred <- predict(ridge.mod, s = bestlam.ridge, newx = x[test.wine, ])  
ridge.mse <- mean((ridge.pred - y.test)^2)  
ridge.mse

## [1] 0.5737655

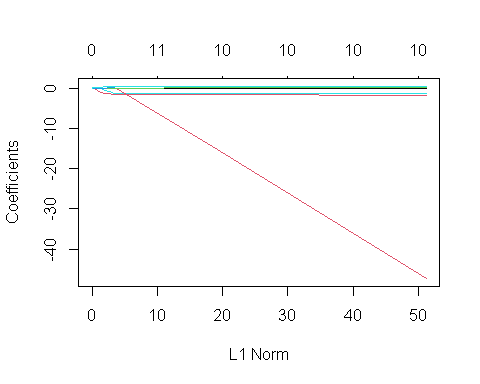
model.errors[model.errors$model == "Ridge Regression", "test.error"] <- ridge.mse

The ridge regression model yields a test mean squared error of 0.574, which is very similar to the test error the linear model with all predictors yielded.

### The Lasso

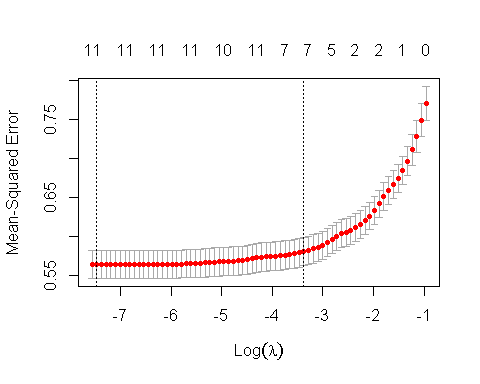
lasso.mod <- glmnet(x[train.wine, ], y[train.wine], alpha = 1, lambda = lambda.grid)  
plot(lasso.mod)

## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):  
## collapsing to unique 'x' values



A Lasso model is trained using the grid of lambda values. A plot of coefficients over L1 show that many coefficients converge to zero or near zero.

# Perform cross-validation to determine best tuning parameter  
set.seed(2)  
cv.out <- cv.glmnet(x[train.wine, ], y[train.wine], alpha = 1)  
plot(cv.out)



bestlam.lasso <- cv.out$lambda.min  
bestlam.lasso

## [1] 0.0005713149

The tuning parameter that minimizes the mean squared error for the Lasso model is 0.0005713149. With a tuning parameter so close to zero, the shrinkage effect on the coefficients is minimal and the model is likely to not change the test error significantly compared to least squares regression.

# Predict the response of test data and calculate MSE  
lasso.pred <- predict(lasso.mod, s = bestlam.lasso, newx = x[test.wine, ])  
lasso.mse <- mean((lasso.pred - y.test)^2)  
lasso.mse

## [1] 0.575157

model.errors[model.errors$model == "The LASSO", "test.error"] <- lasso.mse

The Lasso model is used to predict the quality of white wine using the best tuning parameter and the test data. The mean squared error for the Lasso model is 0.575, which is worse than the multiple linear regression and the ridge regression models.

lasso.coef <- predict(lasso.mod, type = "coefficients", s = bestlam.lasso)[1:11, ]  
lasso.coef

## (Intercept) fixed.acidity volatile.acidity   
## 4.910463e+01 0.000000e+00 -1.720231e+00   
## citric.acid residual.sugar chlorides   
## -2.552551e-02 3.824724e-02 -1.245298e+00   
## free.sulfur.dioxide total.sulfur.dioxide density   
## 2.961531e-03 -1.019355e-04 -4.727139e+01   
## pH sulphates   
## 2.011136e-01 4.309143e-01

lasso.coef[lasso.coef != 0]

## (Intercept) volatile.acidity citric.acid   
## 4.910463e+01 -1.720231e+00 -2.552551e-02   
## residual.sugar chlorides free.sulfur.dioxide   
## 3.824724e-02 -1.245298e+00 2.961531e-03   
## total.sulfur.dioxide density pH   
## -1.019355e-04 -4.727139e+01 2.011136e-01   
## sulphates   
## 4.309143e-01

Looking at the coefficients of the Lasso model, the coefficient for fixed acidity is zero, which essentially means that the model removed fixed acidity as a predictor of quality.

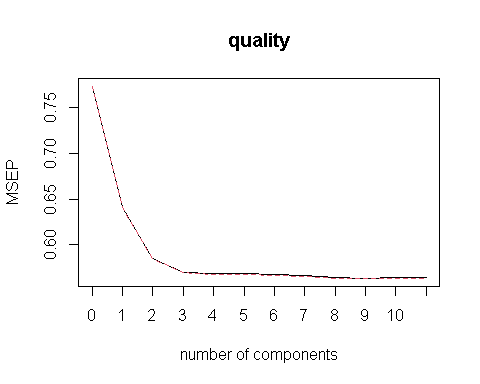
### Partial Least Squares Regression

# Create PLS model on the wine wine quality data  
set.seed(2)  
pls.fit <- plsr(quality ~ ., data = wine.data, subset = train.wine, scale = T,  
 validation = "CV")  
summary(pls.fit)

## Data: X dimension: 2449 11   
## Y dimension: 2449 1  
## Fit method: kernelpls  
## Number of components considered: 11  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 0.8792 0.8002 0.7651 0.7549 0.7539 0.7538 0.7533  
## adjCV 0.8792 0.8002 0.7648 0.7546 0.7537 0.7535 0.7530  
## 7 comps 8 comps 9 comps 10 comps 11 comps  
## CV 0.7525 0.7509 0.7508 0.7509 0.7509  
## adjCV 0.7522 0.7507 0.7506 0.7507 0.7507  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps  
## X 26.89 38.95 45.44 55.72 65.78 72.58 75.07 79.53  
## quality 17.47 25.07 27.06 27.26 27.31 27.39 27.71 27.89  
## 9 comps 10 comps 11 comps  
## X 85.17 93.34 100.00  
## quality 27.95 27.95 27.95

A partial least squares model is trained using the training data set and fit using a cross-validation method. The The root mean squared error determined through cross-validation gradually decreases from 1 component to 8 components. The value of the root mean squared error does not change much with the number of components greater than 8. The summary of the partial least squares fit provides the amount of variance that is explained in the predictors and the response. The highest amount of variance explained in the response is 27.95% while the highest amount of variance in the predictors is 100% when 11 components are used.

# Plot MSEP over the number of components  
validationplot(pls.fit, val.type = "MSEP")  
axis(side=1, at=seq(1, 20, by=1))



The figure shows the mean squared error plotted over the number of components used within the partial least squares model. As the number fo components increases from 0 to 3, the mean squared error reduces significantly. After more than 3 components are used in the model, the mean squared error does not change much.

# Predict quality of the wine using PLS  
pls.pred <- predict(pls.fit, newdata = x[test.wine, ], ncomp=3)  
pls.mse <- mean((pls.pred - y.test)^2)  
pls.mse

## [1] 0.5783051

model.errors[model.errors$model == "PLS", "test.error"] <- pls.mse

The partial least squares model is used to predict the quality of the white wine with 3 as the number of components. The test mean squared error for PLS

## Best Subset Selection

# Function to predict response using best subset selection  
predict.regsubsets <- function(object, newdata, id, ...) {  
 form <- as.formula(object$call[[2]])  
 mat <- model.matrix(form, newdata)  
 coefi <- coef(object, id = id)  
 xvars <- names(coefi)  
 mat[, xvars] %\*% coefi  
}

### Validation Set Approach

# Use validation set approach to determine best subset selection model  
regfit.best <- regsubsets(quality ~ ., data = wine.data[train.wine, ], nvmax = 11)  
  
# Create test matrix  
test.mat <- model.matrix(quality ~ ., data = wine.data[test.wine, ])  
  
# Compute test MSE for all possible amounts of variables used in the model  
val.errors <- rep(NA, 13)  
for (i in 1:11) {  
 coefi <- coef(regfit.best, id = i)  
 pred <- test.mat[, names(coefi)] %\*% coefi  
 val.errors[i] <- mean((wine.data$quality[test.wine] - pred)^2)  
}  
  
# Get coefficient estimates for model with best subset collection  
best.subset <- which.min(val.errors)  
val.errors[best.subset]

## [1] 0.5726076

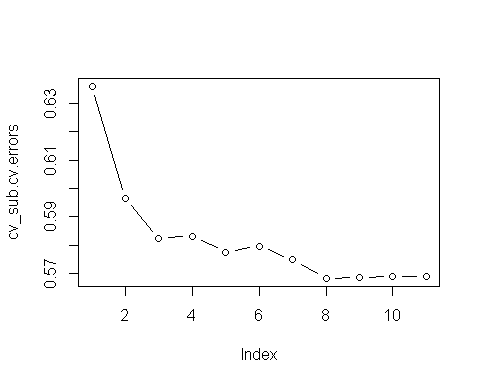
coef(regfit.best, best.subset)

## (Intercept) fixed.acidity volatile.acidity residual.sugar   
## 2.047419e+02 1.199790e-01 -1.735146e+00 9.955035e-02   
## free.sulfur.dioxide density pH sulphates   
## 3.123752e-03 -2.054633e+02 8.047257e-01 7.251004e-01   
## alcohol   
## 1.290130e-01

The best subset selection model is used to select the best set of predictors that minimizes the test mean squared error. A validation set approach is used to find the best subset. This approach finds that the best subset of predictors is fixed acidity, volatile acidity, residual sugar, free sulfur dioxide, density, pH, sulphates, and alcohol; which is a total of 8 predictors from the original 11. The minimmized test MSE is 0.573.

### K-fold Cross-validation approach

# Best subset selection using cross-validation method  
  
k <- 10  
n <- nrow(wine.data)  
set.seed(11)  
folds <- sample(rep(1:k, length = n))  
cv\_sub.errors <- matrix(NA, k, 11,  
 dimnames = list(NULL, paste(1:11)))  
  
for (j in 1:k) {  
 cv\_sub.fit <- regsubsets(quality ~ .,  
 data = wine.data[folds != j, ],  
 nvmax = 11)  
 for (i in 1:11) {  
 pred.cv\_sub <- predict.regsubsets(cv\_sub.fit, wine.data[folds == j, ], id = i)  
 cv\_sub.errors[j, i] <- mean((wine.data$quality[folds == j] - pred.cv\_sub)^2)  
 }  
}  
  
cv\_sub.cv.errors <- apply(cv\_sub.errors, 2, mean)  
par(mfrow = c(1,1))  
plot(cv\_sub.cv.errors, type = "b")



which.min(cv\_sub.cv.errors)

## 8   
## 8

cv\_sub.mse <- cv\_sub.cv.errors[["8"]]  
cv\_sub.mse

## [1] 0.5683492

model.errors[model.errors$model == "Best Subset", "test.error"] <- cv\_sub.mse

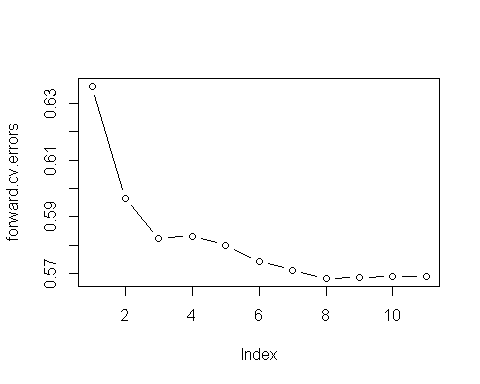
Best subset selection is performed again using a k-fold cross-validation approach this time. With k = 10, the cross-validation also determines that the best subset contains 8 predictors. The test MSE is calculated to be 0.568.

## Forward Stepwise Selection

k <- 10  
n <- nrow(wine.data)  
set.seed(11)  
folds <- sample(rep(1:k, length = n))  
f.cv.errors <- matrix(NA, k, 11,  
 dimnames = list(NULL, paste(1:11)))  
  
for (j in 1:k) {  
 fstep.fit <- regsubsets(quality ~ .,  
 data = wine.data[folds != j, ],  
 nvmax = 11,  
 method = "forward")  
 for (i in 1:11) {  
 pred.forward <- predict.regsubsets(fstep.fit, wine.data[folds == j, ], id = i)  
 f.cv.errors[j, i] <- mean((wine.data$quality[folds == j] - pred.forward)^2)  
 }  
}  
  
forward.cv.errors <- apply(f.cv.errors, 2, mean)  
forward.cv.errors

## 1 2 3 4 5 6 7 8   
## 0.6359855 0.5965686 0.5824032 0.5832526 0.5800027 0.5743192 0.5711655 0.5683492   
## 9 10 11   
## 0.5687762 0.5689857 0.5691309

par(mfrow = c(1,1))  
plot(forward.cv.errors, type = "b")



which.min(forward.cv.errors)

## 8   
## 8

forward.mse <- forward.cv.errors[["8"]]  
forward.mse

## [1] 0.5683492

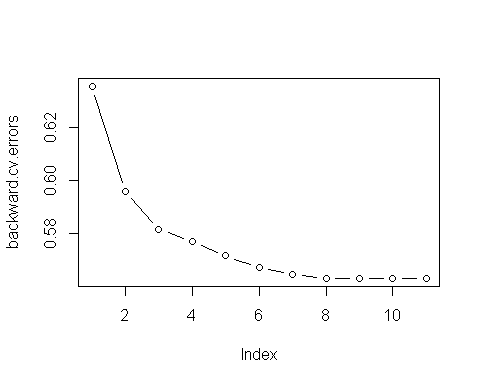
model.errors[model.errors$model == "Forward Stepwise", "test.error"] <- forward.mse

## Backward Stepwise Selection

# Backward Step-wise Subset Selection Using Cross Validation  
  
k <- 10  
n <- nrow(wine.data)  
set.seed(11)  
folds <- sample(rep(1:k, length = n))  
b.cv.errors <- matrix(NA, k, 11,  
 dimnames = list(NULL, paste(1:11)))  
  
for (j in 1:k) {  
 bstep.fit <- regsubsets(quality ~ .,  
 data = wine.data[folds != j, ],  
 nvmax = 11,  
 method = "backward")  
 for (i in 1:11) {  
 pred.backward <- predict.regsubsets(fstep.fit, wine.data[folds == j, ], id = i)  
 b.cv.errors[j, i] <-   
 mean((wine.data$quality[folds == j] - pred.backward)^2)  
 }  
}  
  
backward.cv.errors <- apply(b.cv.errors, 2, mean)  
backward.cv.errors

## 1 2 3 4 5 6 7 8   
## 0.6354239 0.5958132 0.5814734 0.5772214 0.5716449 0.5672763 0.5645731 0.5632897   
## 9 10 11   
## 0.5632278 0.5632074 0.5632012

par(mfrow = c(1,1))  
plot(backward.cv.errors, type = "b")



which.min(backward.cv.errors)

## 11   
## 11

backward.mse <- backward.cv.errors[["11"]]  
backward.mse

## [1] 0.5632012

model.errors[model.errors$model == "Backward Stepwise", "test.error"] <- backward.mse

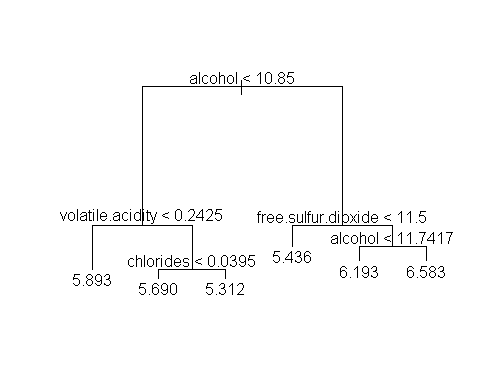
Both forward and backward step-wise subset selection using cross-validation determine that the best model uses 8 predictors. While forward step-wise yields the same test error as best subset selection, the backward step-wise selection model yields a slightly lower test MSE of 0.563.

## Regression Tree

# Create regression tree model for quality as response  
tree.wine <- tree(quality ~ ., data = wine.data, subset = train.wine)  
summary(tree.wine)

##   
## Regression tree:  
## tree(formula = quality ~ ., data = wine.data, subset = train.wine)  
## Variables actually used in tree construction:  
## [1] "alcohol" "volatile.acidity" "chlorides"   
## [4] "free.sulfur.dioxide"  
## Number of terminal nodes: 6   
## Residual mean deviance: 0.5665 = 1384 / 2443   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3.5830 -0.3120 0.1072 0.0000 0.4171 2.6880

# Plot the regression tree  
plot(tree.wine)  
text(tree.wine, pretty = 0)



The regression tree uses four predictors to construct the tree: alcohol, volatile acidity, free sulfur dioxide, and chlorides. The tree contains 6 terminal nodes.

tree.pred <- predict(tree.wine, newdata = wine.data[test.wine, ])  
tree.mse <- mean((tree.pred - y.test)^2)  
tree.mse

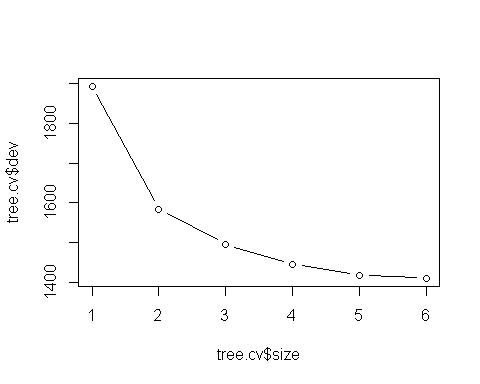
## [1] 0.5930002

model.errors[model.errors$model == "Tree", "test.error"] <- tree.mse

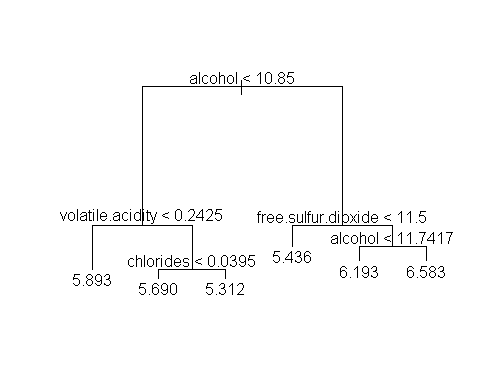
Using the regression tree to predict the quality of white wine, the test MSE is 0.593.

## Pruned Tree

# Use cross-validation to determine best tree size for pruning  
tree.cv <- cv.tree(tree.wine)  
plot(tree.cv$size, tree.cv$dev, type = "b")



# Prune tree and plot the tree  
prune.wine <- prune.tree(tree.wine, best = 6)  
plot(prune.wine)  
text(prune.wine, pretty = 0)



# Predict response with pruned tree  
prune.pred <- predict(prune.wine, newdata = wine.data[test.wine, ])  
prune.mse <- mean((prune.pred - y.test)^2)  
prune.mse

## [1] 0.5930002

model.errors[model.errors$model == "Pruned Tree", "test.error"] <- prune.mse

Using cross-validation to determine the ideal size of the tree does not give any new information: the ideal size of a regression tree for our training data is the same as that of the original tree we created. Predictably, the pruned tree with a size of 6 is gives the same tree and yields virtually the same test MSE.

## Bagging

set.seed(90)  
bag.wine <- randomForest(quality ~ ., data = wine.data, subset = train.wine,  
 mtry = (ncol(wine.data) - 1), importance = T)  
bag.wine

##   
## Call:  
## randomForest(formula = quality ~ ., data = wine.data, mtry = (ncol(wine.data) - 1), importance = T, subset = train.wine)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 11  
##   
## Mean of squared residuals: 0.4327978  
## % Var explained: 43.96

A bagging model is created to attempt to use a series of bootstrapped regression trees to better predict the quality of white wine. The bagging model considers all predictors as split candidates at each split in the trees. 500 trees are used in the model, the mean of squared residuals is 0.433, and 43.96% of the variance is explained with this model.

bag.pred <- predict(bag.wine, newdata = wine.data[test.wine, ])  
bag.mse <- mean((bag.pred - y.test)^2)  
bag.mse

## [1] 0.4079268

model.errors[model.errors$model == "Bagging", "test.error"] <- bag.mse

Using the bagging model to predict quality in the test data set yields a test MSE of 0.408, which is the lowest of any model thus far.

## Random Forest

set.seed(90)  
rf.wine <- randomForest(quality ~ ., data = wine.data[train.wine, ],  
 importance = T)  
rf.wine

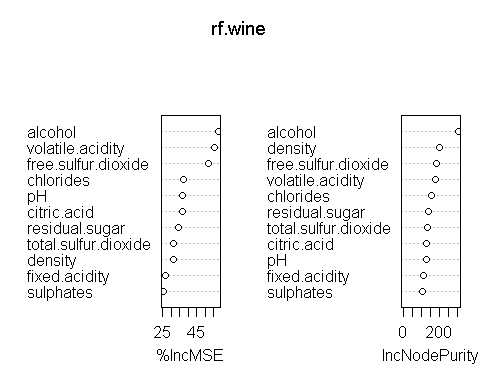
##   
## Call:  
## randomForest(formula = quality ~ ., data = wine.data[train.wine, ], importance = T)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 3  
##   
## Mean of squared residuals: 0.4229552  
## % Var explained: 45.23

Similar to bagging, the random forest tries to predict the response using 500 regression trees. The random forest model, however, uses considers only a random selection of 3 predictors at each split. The model has a slightly lower mean of squared residuals than the bagging model at 0.423, but explains 45.23% of the variance of the model.

importance(rf.wine)

## %IncMSE IncNodePurity  
## fixed.acidity 26.59123 114.8717  
## volatile.acidity 55.69239 180.5992  
## citric.acid 36.74146 127.3700  
## residual.sugar 34.04928 141.7358  
## chlorides 36.94861 158.7531  
## free.sulfur.dioxide 51.79472 183.1726  
## total.sulfur.dioxide 31.05734 134.0279  
## density 31.05210 200.2182  
## pH 36.88850 127.2465  
## sulphates 25.36442 107.2996  
## alcohol 57.86859 308.8811

varImpPlot(rf.wine)



Looking at the graphs indicating the importance of predictors in the random forest model, alcohol, volatile acidity, free sulfur dioxide, and chlorides are revealed to be four of the most important predictors to predict the quality of white wine. These are the same predictors featured in the pruned regression tree model.

rf.pred <- predict(rf.wine, newdata = wine.data[test.wine, ])  
rf.mse <- mean((rf.pred - y.test)^2)  
rf.mse

## [1] 0.4066441

model.errors[model.errors$model == "RF", "test.error"] <- rf.mse

The random forest model yields the lowest test MSE thus far at 0.407 when applied to the test data set.

## Boosting

tunings <- c(0.001, 0.005, 0.01, 0.02, 0.03)  
boost.errors <- data.frame(shrinkage = tunings,  
 training.error = rep(NA, length(tunings)),  
 test.error = rep(NA, length(tunings)))  
  
for (x in 1:length(tunings)) {  
 set.seed(91)  
 boost.wine <- gbm(quality ~ ., data = wine.data[train.wine, ],  
 distribution = "gaussian", n.trees = 1000,  
 interaction.depth = 4, shrinkage = tunings[x])  
 boost.errors[x, "training.error"] <- mean(boost.wine$train.error)  
}  
  
for (x in 1:length(tunings)) {  
 set.seed(91)  
 boost.wine <- gbm(quality ~ ., data = wine.data[train.wine, ],  
 distribution = "gaussian", n.trees = 1000,  
 interaction.depth = 4, shrinkage = tunings[x])  
 yhat.boost <- predict(boost.wine, newdata = wine.data[test.wine, ],  
 n.trees = 1000)  
 boost.errors[x, "test.error"] <- mean((yhat.boost - y.test)^2)  
}  
  
boost.errors

## shrinkage training.error test.error  
## 1 0.001 0.6506781 0.5982912  
## 2 0.005 0.5243904 0.4939699  
## 3 0.010 0.4735161 0.4782166  
## 4 0.020 0.4225135 0.4671453  
## 5 0.030 0.3887048 0.4672300

A boosting model, which attempts to improve on bagging by sequentially creating trees and using information from previous trees, was created using a set of different shrinkage parameter. Each model used 1000 trees and a tree size equal to 4. As the shrinkage parameter increases, the training error decreases; however, the minimum test error, 0.467, is seen when lambda equals 0.02.

model.errors[model.errors$model == "Boosting", "test.error"] <- min(boost.errors$test.error)

model.errors

## model test.error  
## 1 Least Squares 0.5698636  
## 2 Ridge Regression 0.5737655  
## 3 The LASSO 0.5751570  
## 4 PLS 0.5783051  
## 5 Best Subset 0.5683492  
## 6 Forward Stepwise 0.5683492  
## 7 Backward Stepwise 0.5632012  
## 8 Tree 0.5930002  
## 9 Pruned Tree 0.5930002  
## 10 Bagging 0.4079268  
## 11 RF 0.4066441  
## 12 Boosting 0.4671453

sqrt(min(model.errors$test.error))

## [1] 0.6376865

Least squares regression and the generalized linear models (GLMs) all yield a similar test mean squared error when trying to predict the quality of white wine using the test data set. The regression tree model performs worse on the test data than the GLMs; however, other tree based methods yield significantly lower mean squared errors than all other models. The best of these is the random forest model with a test MSE of 0.407, which means, on average, the random forest will predict a quality rating 0.638 units off the actual value.