Chapter 7 HW

Ryan Gallagher

2023-10-11

### 6. In this exercise, you will further analyze the Wage data set considered throughout this chapter.

#### (a) Perform polynomial regression to predict wage using age. Use cross-validation to select the optimal degree for the polynomial. What degree was chosen, and how does this compare to the results of the hypothesis testing using ANOVA?

data(Wage)  
  
range = 1:10  
  
x = Wage$age  
y = Wage$wage  
  
# MSE  
mean\_squared\_error = function(y\_true, y\_pred) {  
 return(mean((y\_true - y\_pred)^2))  
}  
  
k\_fold\_cv = function(x, y, degree, K = 10) {  
 n = length(y)  
 mse = numeric(K)  
   
 # 10 fold  
 folds = cut(seq(1, n), breaks = K, labels = FALSE)  
   
 for (k in 1:K) {  
 # Get the training and testing data  
 test\_indices = which(folds == k)  
 train\_indices = setdiff(seq(1, n), test\_indices)  
 x\_train = x[train\_indices]  
 y\_train = y[train\_indices]  
 x\_test = x[test\_indices]  
 y\_test = y[test\_indices]  
   
 poly\_x\_train = poly(x\_train, degree = degree, raw = TRUE)  
 model = lm(y\_train ~ poly\_x\_train)  
  
 poly\_x\_test = poly(x\_test, degree = degree, raw = TRUE)  
 y\_pred = predict(model, newdata = data.frame(poly\_x\_train = I(poly\_x\_test)))  
   
 # Calculate the MSE for this fold  
 mse[k] = mean\_squared\_error(y\_test, y\_pred)  
 }  
   
 # Return the mean MSE across all folds  
 return(mean(mse))  
}  
  
degree.range = 1:10 # for example, evaluate polynomials of degree 1 to 10  
cv.errors = sapply(degree.range, function(d) {  
 return(k\_fold\_cv(x, y, d))  
})  
  
optimal.degree = degree.range[which.min(cv.errors)]  
print(optimal.degree)

## [1] 4

optimal.poly.x = poly(x, degree = optimal.degree, raw = TRUE)  
optimal.model = lm(y ~ optimal.poly.x)  
summary(optimal.model)

##   
## Call:  
## lm(formula = y ~ optimal.poly.x)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -98.707 -24.626 -4.993 15.217 203.693   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -1.842e+02 6.004e+01 -3.067 0.002180 \*\*   
## optimal.poly.x1 2.125e+01 5.887e+00 3.609 0.000312 \*\*\*  
## optimal.poly.x2 -5.639e-01 2.061e-01 -2.736 0.006261 \*\*   
## optimal.poly.x3 6.811e-03 3.066e-03 2.221 0.026398 \*   
## optimal.poly.x4 -3.204e-05 1.641e-05 -1.952 0.051039 .   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 39.91 on 2995 degrees of freedom  
## Multiple R-squared: 0.08626, Adjusted R-squared: 0.08504   
## F-statistic: 70.69 on 4 and 2995 DF, p-value: < 2.2e-16

We find that degree 4 is optimal. This selection is different than the ANOVA fit:

fit.1 = lm(wage ~ age, data=Wage)  
fit.2 = lm(wage ~ poly(age, 2), data=Wage)  
fit.3 = lm(wage ~ poly(age, 3), data=Wage)  
fit.4 = lm(wage ~ poly(age, 4), data=Wage)  
fit.5 = lm(wage ~ poly(age, 5), data=Wage)  
anova(fit.1, fit.2, fit.3, fit.4, fit.5)

## Analysis of Variance Table  
##   
## Model 1: wage ~ age  
## Model 2: wage ~ poly(age, 2)  
## Model 3: wage ~ poly(age, 3)  
## Model 4: wage ~ poly(age, 4)  
## Model 5: wage ~ poly(age, 5)  
## Res.Df RSS Df Sum of Sq F Pr(>F)   
## 1 2998 5022216   
## 2 2997 4793430 1 228786 143.5931 < 2.2e-16 \*\*\*  
## 3 2996 4777674 1 15756 9.8888 0.001679 \*\*   
## 4 2995 4771604 1 6070 3.8098 0.051046 .   
## 5 2994 4770322 1 1283 0.8050 0.369682   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

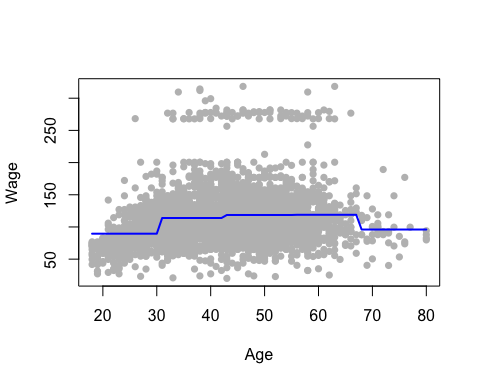
Anova agrees! We see that model 4 has the lowest RSS.

### (b) Fit a step function to predict using , and perform cross-validation to choose the optimal number of cuts. Make a plot of the fit obtained.

set.seed(1)  
k\_fold\_cv = function(x, y, cuts, K = 10) {  
 n = length(y)  
 mse = numeric(K)  
 folds = cut(seq(1, n), breaks = K, labels = FALSE)  
   
 # Create the cut variable for the entire dataset  
 cut\_var = cut(x, breaks = cuts, include.lowest = TRUE, right = TRUE)  
   
 for (k in 1:K) {  
 test\_indices = which(folds == k)  
 train\_indices = setdiff(seq(1, n), test\_indices)  
   
 cut\_var\_train = cut\_var[train\_indices]  
 y\_train = y[train\_indices]  
 cut\_var\_test = cut\_var[test\_indices]  
 y\_test = y[test\_indices]  
   
 model = lm(y\_train ~ cut\_var\_train)  
   
 y\_pred = predict(model, newdata = data.frame(cut\_var\_train = cut\_var\_test))  
   
 mse[k] = mean\_squared\_error(y\_test, y\_pred)  
 }  
   
 return(mean(mse))  
}  
  
cut.range = 2:30  
optimal.cuts = cut.range[which.min(cv.errors)]  
print(optimal.cuts)

## [1] 5

Wage$cut = cut(Wage$age, breaks = optimal.cuts)  
optimal.model = lm(wage ~ cut, data = Wage)  
  
# Create a new data frame for predictions  
newdata = data.frame(age = min(Wage$age):max(Wage$age))  
newdata$cut = cut(newdata$age, breaks = optimal.cuts)  
  
# Get the predictions  
newdata$pred = predict(optimal.model, newdata = newdata)  
  
# Plot the fit  
plot(Wage$age, Wage$wage, col = "gray", pch = 16, xlab = "Age", ylab = "Wage")  
lines(newdata$age, newdata$pred, col = "blue", lwd = 2)

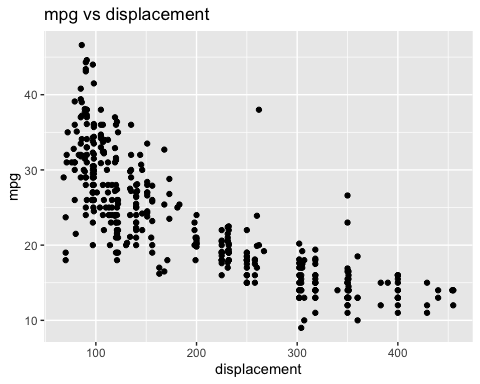


We find that 5 is the optimal degree according to the created step function.

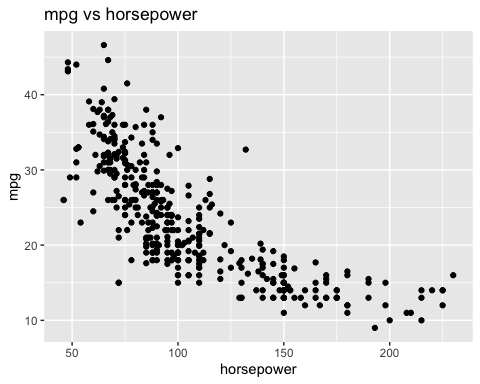
### 8. Fit some of the non-linear models investigated in this chapter to the dataset. Is there any evidence for non-linear relationships in this dataset? Create some informative plots to justify your answer

#Outcome is MPG  
#Continuous variable: displacement, horsepower, weight, acceleration  
  
  
#See variable relationships  
library(ggplot2)

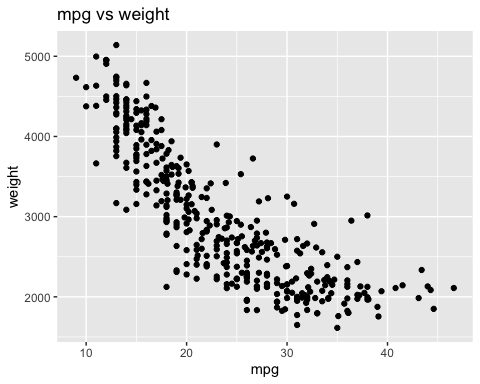
p = ggplot(Auto, aes(y=mpg, x=displacement)) + geom\_point() + labs(title="mpg vs displacement")  
p



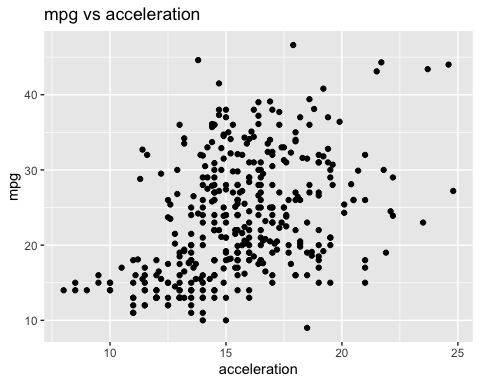
p = ggplot(Auto, aes(x=horsepower, y=mpg)) + geom\_point() + labs(title="mpg vs horsepower")  
p



p = ggplot(Auto, aes(x=mpg, y=weight)) + geom\_point() + labs(title="mpg vs weight")  
p



p = ggplot(Auto, aes(y=mpg, x=acceleration)) + geom\_point() + labs(title="mpg vs acceleration")  
p



There are clearly some non-linear relationships within this dataset. I’ll choose one and fit some nonlinear regressions.

From these available variables, I would look to fit to either horsepower or displacement (though these two are clearly associated). I’ll pick horsepower.

I’ll use the same cross validation method as was used earlier in this document to try polynomial regression

#### Polynomial Regression

range = 1:10  
  
x = Auto$horsepower  
y = Auto$mpg  
  
# MSE  
mean\_squared\_error = function(y\_true, y\_pred) {  
 return(mean((y\_true - y\_pred)^2))  
}  
  
k\_fold\_cv = function(x, y, degree, K = 10) {  
 n = length(y)  
 mse = numeric(K)  
   
 # 10 fold  
 folds = cut(seq(1, n), breaks = K, labels = FALSE)  
   
 for (k in 1:K) {  
 # Get the training and testing data  
 test\_indices = which(folds == k)  
 train\_indices = setdiff(seq(1, n), test\_indices)  
 x\_train = x[train\_indices]  
 y\_train = y[train\_indices]  
 x\_test = x[test\_indices]  
 y\_test = y[test\_indices]  
   
 poly\_x\_train = poly(x\_train, degree = degree, raw = TRUE)  
 model = lm(y\_train ~ poly\_x\_train)  
  
 poly\_x\_test = poly(x\_test, degree = degree, raw = TRUE)  
 y\_pred = predict(model, newdata = data.frame(poly\_x\_train = I(poly\_x\_test)))  
   
 # Calculate the MSE for this fold  
 mse[k] = mean\_squared\_error(y\_test, y\_pred)  
 }  
   
 # Return the mean MSE across all folds  
 return(mean(mse))  
}  
  
degree.range = 1:10 # for example, evaluate polynomials of degree 1 to 10  
cv.errors = sapply(degree.range, function(d) {  
 return(k\_fold\_cv(x, y, d))  
})  
  
optimal.degree = degree.range[which.min(cv.errors)]  
print(optimal.degree)

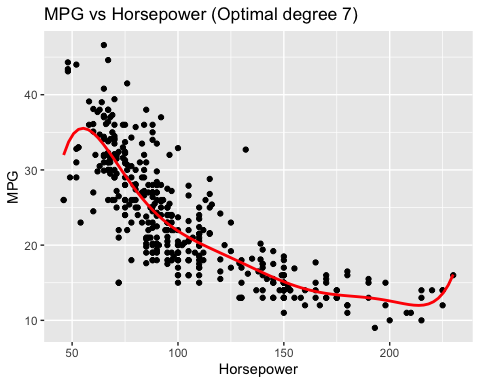
## [1] 7

optimal.poly.x = poly(x, degree = optimal.degree, raw = TRUE)  
optimal.model = lm(y ~ optimal.poly.x)  
summary(optimal.model)

##   
## Call:  
## lm(formula = y ~ optimal.poly.x)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.5171 -2.5682 -0.2082 2.1757 15.2986   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -4.891e+02 1.898e+02 -2.576 0.01036 \*   
## optimal.poly.x1 3.325e+01 1.251e+01 2.658 0.00818 \*\*  
## optimal.poly.x2 -8.476e-01 3.372e-01 -2.514 0.01235 \*   
## optimal.poly.x3 1.135e-02 4.829e-03 2.350 0.01926 \*   
## optimal.poly.x4 -8.755e-05 3.979e-05 -2.200 0.02839 \*   
## optimal.poly.x5 3.914e-07 1.892e-07 2.069 0.03926 \*   
## optimal.poly.x6 -9.429e-10 4.823e-10 -1.955 0.05131 .   
## optimal.poly.x7 9.472e-13 5.099e-13 1.858 0.06397 .   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.296 on 384 degrees of freedom  
## Multiple R-squared: 0.7025, Adjusted R-squared: 0.6971   
## F-statistic: 129.5 on 7 and 384 DF, p-value: < 2.2e-16

Here, we find that the optimal polynomial regression degree is 7.

p = ggplot(Auto, aes(x=horsepower, y=mpg)) + geom\_point() + labs(title="MPG vs Horsepower (Optimal degree 7)", x='Horsepower', y='MPG') +  
 geom\_smooth(formula = y ~ poly(x, 7), method='lm', se=FALSE, color='red')  
p



residuals = resid(optimal.model)  
print(sum(residuals^2))

## [1] 7086.644

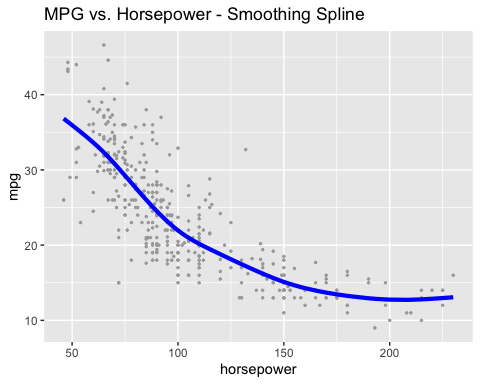
#### Spline

I’ll simply fit a smoothing spline with the cross validation option.

library(ggplot2)  
  
fit2 = smooth.spline(Auto$horsepower, Auto$mpg, cv=TRUE)

## Warning in smooth.spline(Auto$horsepower, Auto$mpg, cv = TRUE):  
## cross-validation with non-unique 'x' values seems doubtful

predicted = data.frame(horsepower = fit2$x, mpg = fit2$y)  
  
# Plot using ggplot2  
ggplot(Auto, aes(x=horsepower, y=mpg)) +  
 geom\_point(aes(color="data"), size=0.5) +   
 geom\_line(data=predicted, aes(color="spline"), linewidth=1.5) +  
 labs(title="MPG vs. Horsepower - Smoothing Spline", color="Legend") +  
 scale\_color\_manual(values=c(data="darkgray", spline="blue")) +  
 theme(legend.position="none")



predicted\_values = fitted(fit2)  
residuals = Auto$mpg - predicted\_values  
  
# RSS  
RSS = sum(residuals^2)  
print(RSS)

## [1] 7208.845

So far, the degree 7 polynomial has a smaller RSS than the smoothed spine.

### GAM

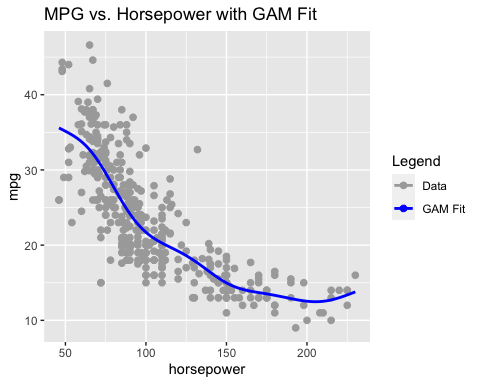
library(mgcv)

## Loading required package: nlme

## This is mgcv 1.9-0. For overview type 'help("mgcv-package")'.

# Fit a GAM  
gam\_fit <- gam(mpg ~ s(horsepower), data=Auto)  
  
# Create a new data frame with a sequence of horsepower values  
newdata = data.frame(horsepower = seq(min(Auto$horsepower), max(Auto$horsepower), length.out=100))  
  
newdata$predicted = predict(gam\_fit, newdata=newdata)  
  
ggplot(Auto, aes(x=horsepower, y=mpg)) +  
 geom\_point(aes(color="Data"), size=2) +   
 geom\_line(data=newdata, aes(x=horsepower, y=predicted, color="GAM Fit"), size=1) +  
 labs(title="MPG vs. Horsepower with GAM Fit", color="Legend") +  
 scale\_color\_manual(values=c(Data="darkgray", `GAM Fit`="blue"))

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



residuals\_gam = resid(gam\_fit)  
RSS = sum(residuals\_gam^2)  
print(RSS)

## [1] 7111.894

This GAM has a comparable, but still lowest RSS yet.