Lab3: Cross-Validation

Dr. Purna Gamage

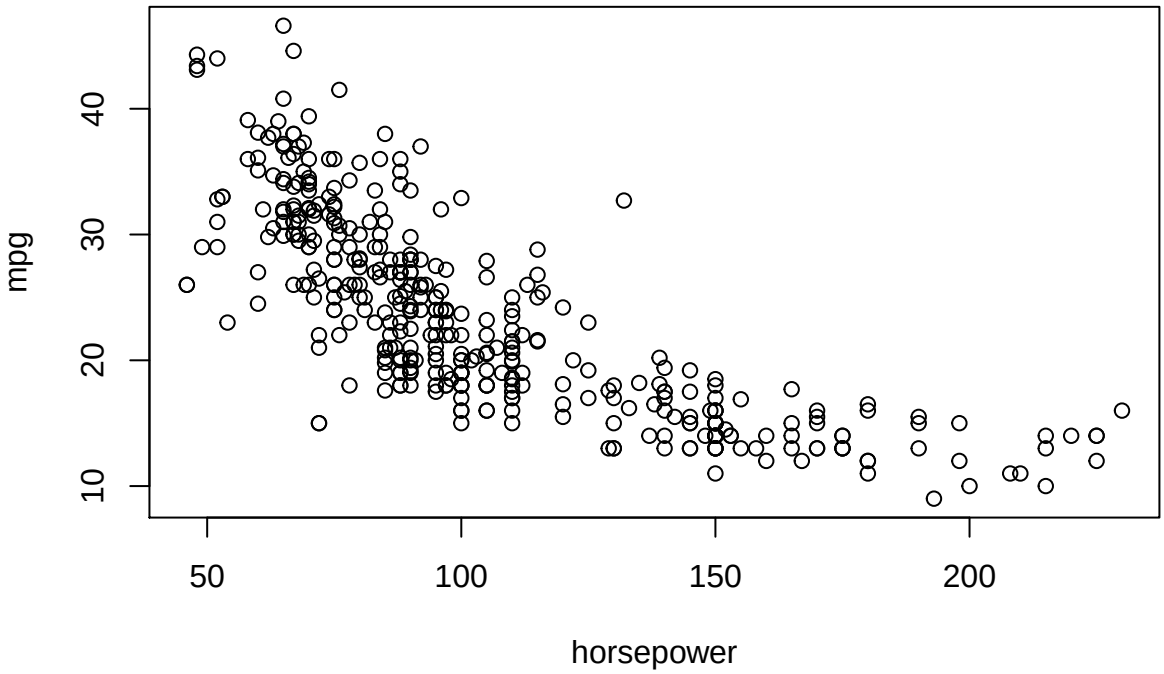
2/9/2021

```
library(ISLR)
require(boot)
```

```
## Loading required package: boot
```

```
?cv.glm

plot(mpg~horsepower,data = Auto)
```



Leave-One-Out Cross-Validation

If we use glm() to fit a model without passing in the family argument, then it performs linear regression, just like the lm() function.

```
glm.fit=glm(mpg~horsepower,data=Auto)
coef(glm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```
lm.fit=lm(mpg~horsepower,data=Auto)
coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

identical linear regression models.

We will perform linear regression using the glm() function rather than the lm() function because the former can be used together with cv.glm().

The cv.glm() function is part of the boot library.

```
cv=cv.glm(Auto,glm.fit) #prettey slow (doesn't use  formula (5.2))
summary(cv)
```

```
##       Length Class  Mode
## call    3    -none- call
## K       1    -none- numeric
## delta   2    -none- numeric
## seed  626    -none- numeric
```

```
cv$delta
```

```
## [1] 24.23151 24.23114
```

The cv.glm() function produces a list with several components. The two numbers in the delta vector contain the cross-validation results.

First number we see above is raw LOOCV result and the second number is a bias corrected version of it. The Bias correction is because data set that we train on is slighlty smaller than the one that we actually would like to get error for, which is the full data set of size n. it has more of an effect for K-fold CV.

On this data set, the two estimates are very similar to each other.

This command will likely take a couple of minutes to run.

Let's write a simple function to use formula(5.2)

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2, \qquad (5.2)$$

```
loocv=function(fit){ #function loocv takes the fit as the argument
  h =lm.influence(fit)$h #lm.influence is a post-processor for lm.fit
  #will extract the element h from that which gives you the diagonal elements
  #of the hat matrix =levarage => put that in a vector h
  mean((residuals(fit)/(1-h))^2) #formula 5.2
  }

#Now lets try it

loocv(glm.fit)
```

```
## [1] 24.23151
```

very quick, same as the result from the previous glm.fit. so our function works.

Now let's fit polynomials of different degrees(1-5). Because data looks very non linear.

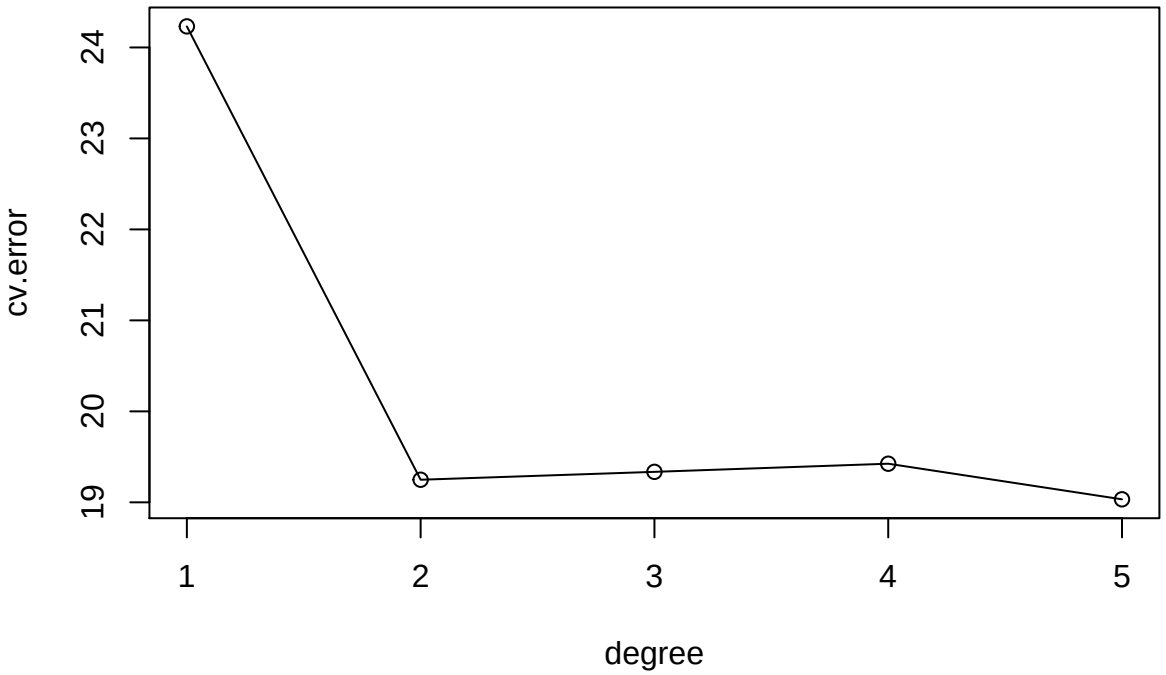
```r
cv.error=rep(0,5)

degree =1:5

for (d in degree){
  glm.fit=glm(mpg~poly(horsepower,d),data=Auto) #fit the polynomial of the degree d
  cv.error[d]=loocv(glm.fit)#a vector to collect errors
}
cv.error
```

```
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321
```

```r
plot(degree,cv.error, type='o') #plot the error against the degree
```



Degree 1 does poorly. Degree 2 error jumps down from 24 down to just above 19. Higher degrees really don't make much difference.

By looking at the graph we can conclude that a quadratic model will be a good fit.

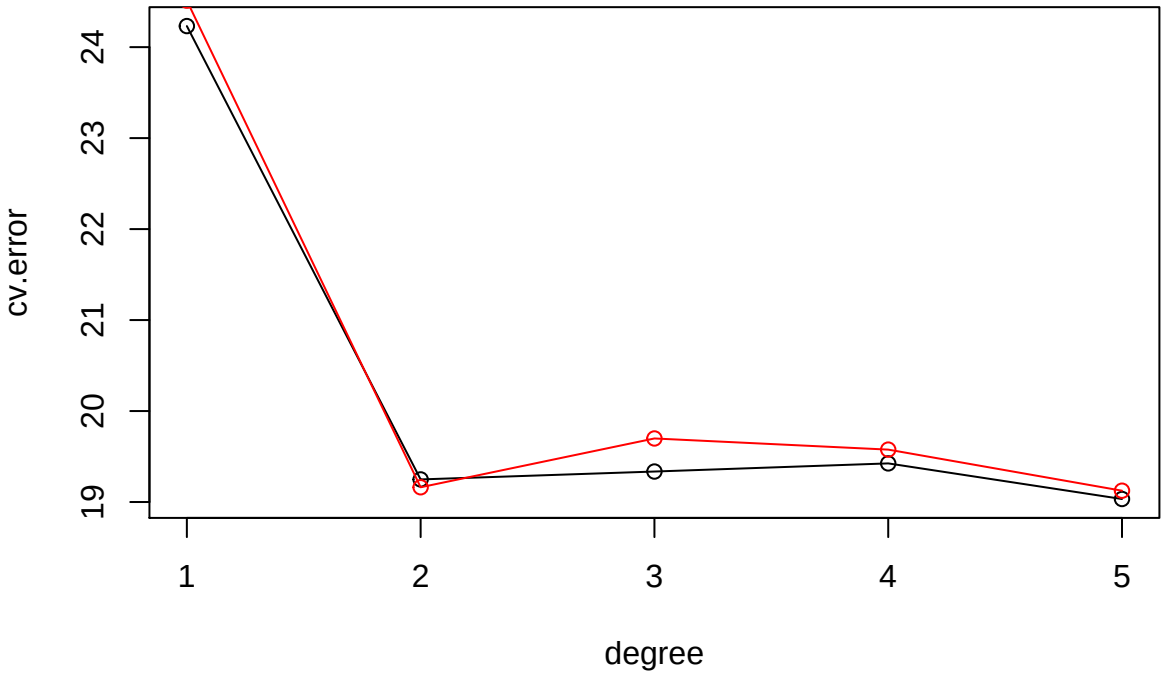
10-Fold Cross-Validation

```r
set.seed(17)

cv.error.5=rep(0,5)
for (d in degree){
  glm.fit=glm(mpg~poly(horsepower,d),data=Auto)
  cv.error.5[d]=cv.glm(Auto,glm.fit,K=5)$delta[1]
} #K=10 is the number of folds.
cv.error.5
```

```
## [1] 24.51158 19.16205 19.69842 19.57611 19.12434
```

```
plot(degree,cv.error, type='o')
lines(degree,cv.error.5, type = "o",col="red")
```



It's not much different but more stable (less variation) than LOOCV.

Notice that the computation time is much shorter than that of LOOCV.

Auto example

(a). What is the best model?

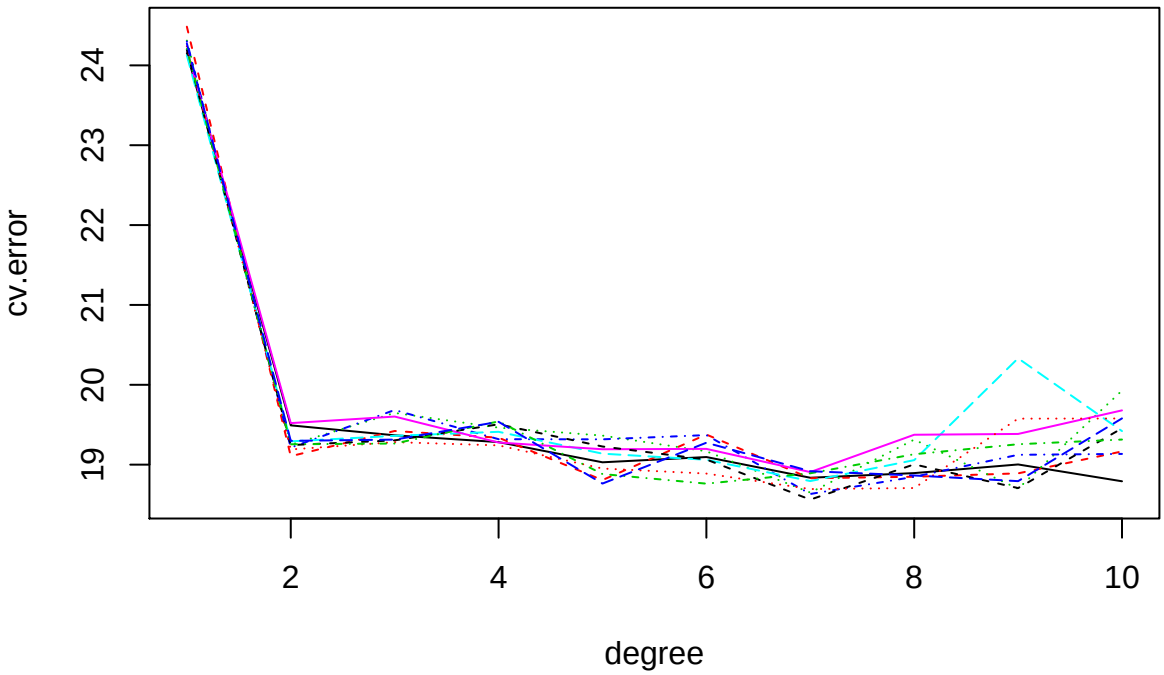
Best model: $mpg = -120.14 * horsepwer + 44.09 * (horsepower)^2$

(b) Compare the cross validation error rate for different splits.

```
cv.error = matrix(data = NA,nrow=10, ncol = 10)

for (j in 1:10){
for (i in 1:10){
  glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
  cv.error[i,j]=cv.glm(Auto,glm.fit,K=10)$delta[1]
}
}

degree=1:10
matplot(degree,cv.error,type="l")
```




```
#best model
glm.fit=glm(mpg~poly(horsepower,2),data=Auto)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = mpg ~ poly(horsepower, 2), data = Auto)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.7135   -2.5943   -0.0859    2.2868   15.8961
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              23.4459     0.2209  106.13   <2e-16 ***
## poly(horsepower, 2)1 -120.1377     4.3739  -27.47   <2e-16 ***
## poly(horsepower, 2)2   44.0895     4.3739   10.08   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 19.13118)
##
##     Null deviance: 23819  on 391  degrees of freedom
## Residual deviance:  7442  on 389  degrees of freedom
## AIC: 2274.4
##
## Number of Fisher Scoring iterations: 2
```

Example

Consider the built-in data set **cars** (see problem 1). We wish to predict the braking distance **dist** from **speed**. Use leave-one-out cross validation to find the best polynomial regression model. Repeat with 10-fold cross validation. Compare the two answers.

Solution.

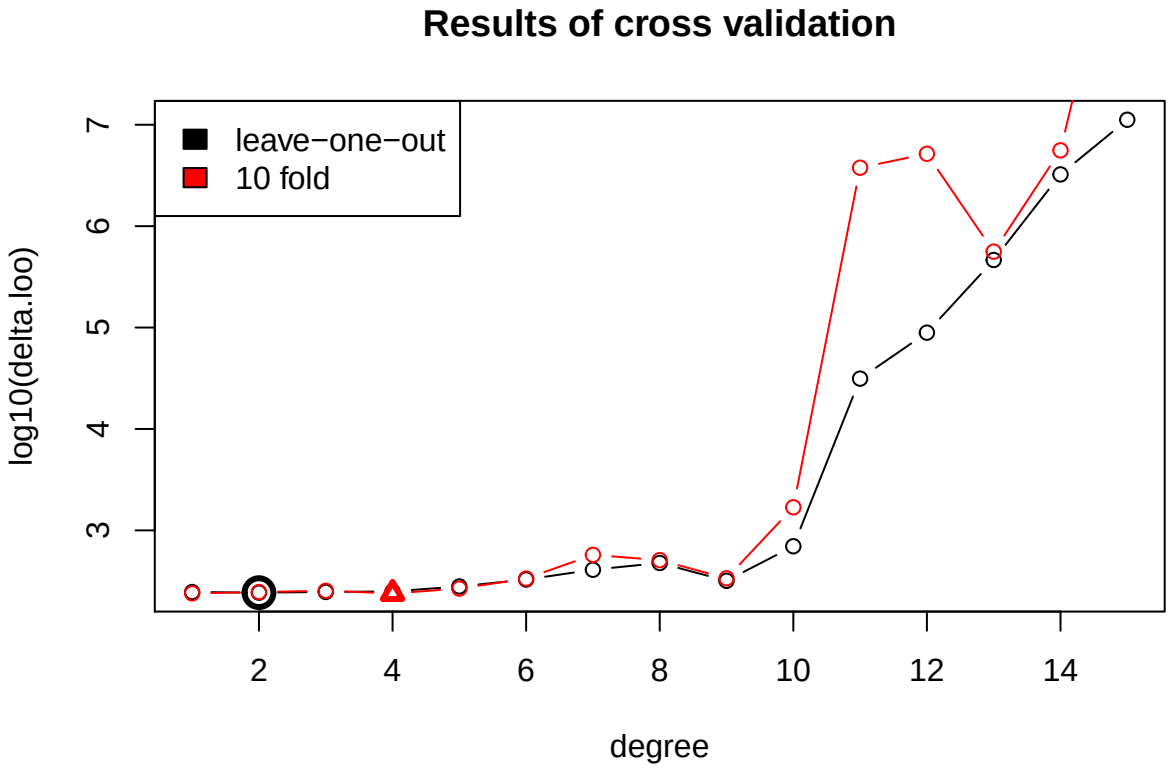
First, use leave-one-out cross validation. We can use degrees 1 to 15 to fit polynomial models.

Next Plot the results. It turns out that the delta values of very large if the degree of the polynomial is large., therefore it is better to plot the logarithms (done below).

The model with minimal delta for leave-one-out cross validation is 2 (black circle).

The model with minimal delta for 10-fold cross validation is also 2 (red triangle). *This depends on the random seed with which the cross validation is done.*

```
set.seed(101)
data(cars)
results.41 <- data.frame(degree = 1:15, delta.loo = NA, delta.k10 = NA)
for (j in 1:15){
  fit.41 = glm(dist ~ poly(speed,j), data = cars)
  fit.loo <- cv.glm(cars, fit.41, K = 50)
  fit.k10 <- cv.glm(cars, fit.41, K = 10)
  results.41$delta.loo[j] <- fit.loo$delta[1]
  results.41$delta.k10[j] <- fit.k10$delta[1]
}
plot(log10(delta.loo) ~ degree, data = results.41, type = 'b',
     main = "Results of cross validation")
legend(x = "topleft", legend = c("leave-one-out", "10 fold"), fill = 1:2)
n.1 <- which.min(results.41$delta.loo)
points(n.1, log10(results.41$delta.loo[n.1]), lwd = 3, col = 1, cex = 2)
n.2 <- which.min(results.41$delta.k10)
points(n.2, log10(results.41$delta.k10[n.2]), lwd = 3, pch = 2, col = 2)
lines(log10(delta.k10) ~ degree, data = results.41, type = 'b', col = 2)
```



Lab Assignment Question

1 Cross Validation Approaches

We'll consider a regression problem with the Carseats dataset.

1. Fit a linear model to predict Sales from all other attributes. Looking at the model summary, what is the residual error?
2. Fit another linear model, but now do not include the predictors Population, Education, Urban, and US. What is the residual error?
3. How do determine which model is better? Create a train/test split of the data and retrain your model.

Use the fitted model to make predictions on the test set, and check the model performance by calculating the root mean squared error and R squared values.

(Conduct this for both the model from part (1) and part (2)).

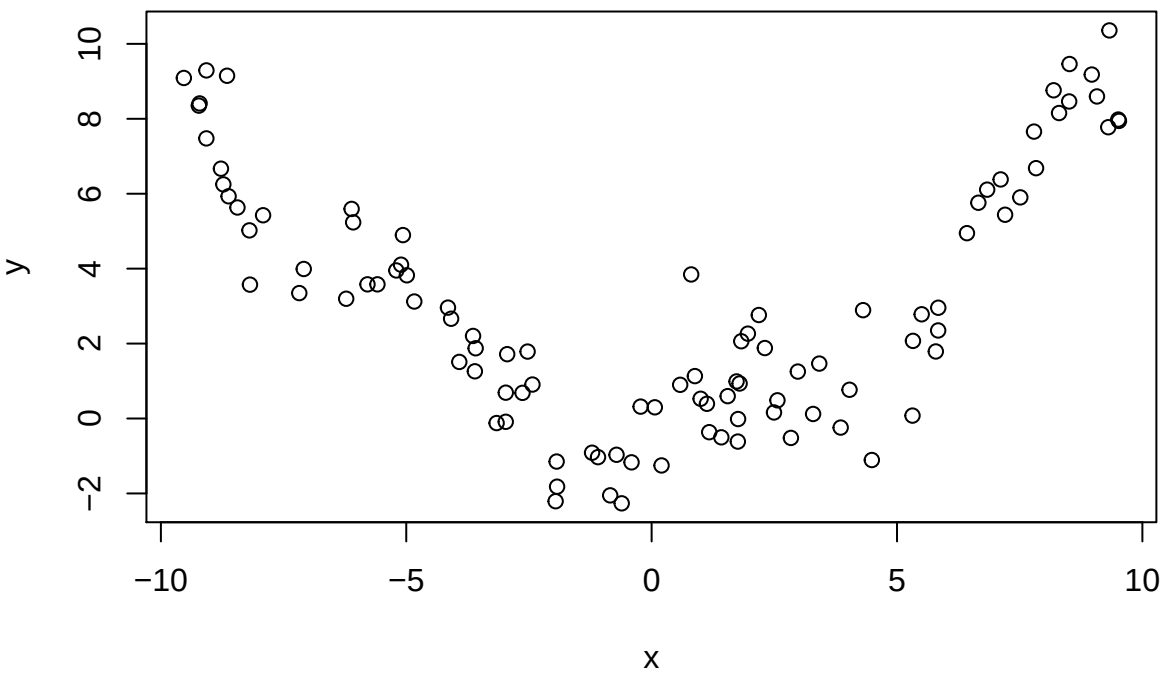
Which model is the best?

4. Use the cv.glm() function from the 'boot' library to conduct a cross validation analysis. Run 5-fold, 10-fold, and Leave One Out cross validation for part 1 and 2. Which model has a lower error?

2 CV for Model Selection

Here is a synthetic (noisy) dataset.

```
mydf = data.frame(x = runif(100,min = -10, max = 10))
mydf$y = sin(mydf$x) + .1*mydf$x^2 + rnorm(100)
plot(y ~ x, data = mydf)
```



1. Fit a linear model to this data. Use 10-fold CV to estimate the generalization (calculated from CV) error.
2. Use the poly() function to fit a quadratic model to this data. Use 10-fold CV to estimate the error.
3. Repeat this for polynomials up to order 15. Make a plot of the generalization error (calculated from CV) versus the order of the polynomial.