Birkbeck UNIVERSITY OF LONDON

Big Data Analytics

Session 5(b)

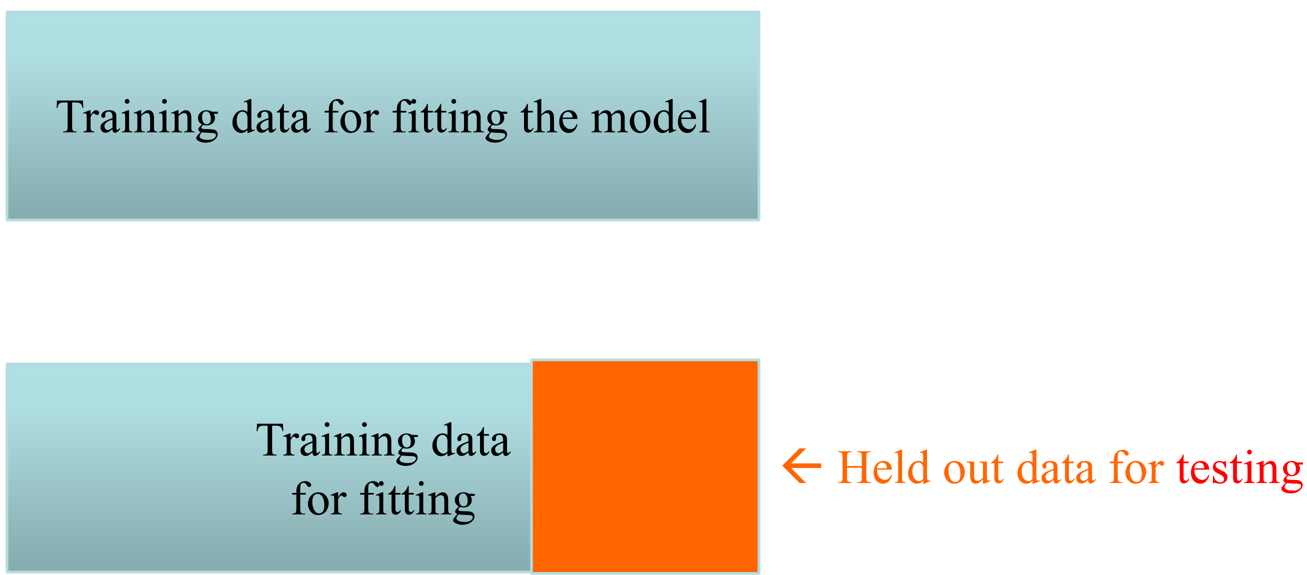
Cross Validation

So far

- Compute MSE/error rate on the training data
  - Easy!
- Calculate MSE/error rate on the test data
  - Easy, if the designated test set is available
  - ➔ Unfortunately, this is usually not the case
- Training MSE/error rate can dramatically underestimate the test MSE/error rate.
- Main question: How to estimate the test MSE/error rate in the absence of the designated test data? (Based on Ch. 5.1)

Cross Validation

- Solution:
  - Estimate the test error rate by
    - holding out a subset of the training observations from the fitting process, and then
    - applying the statistical learning method to those held out observations.

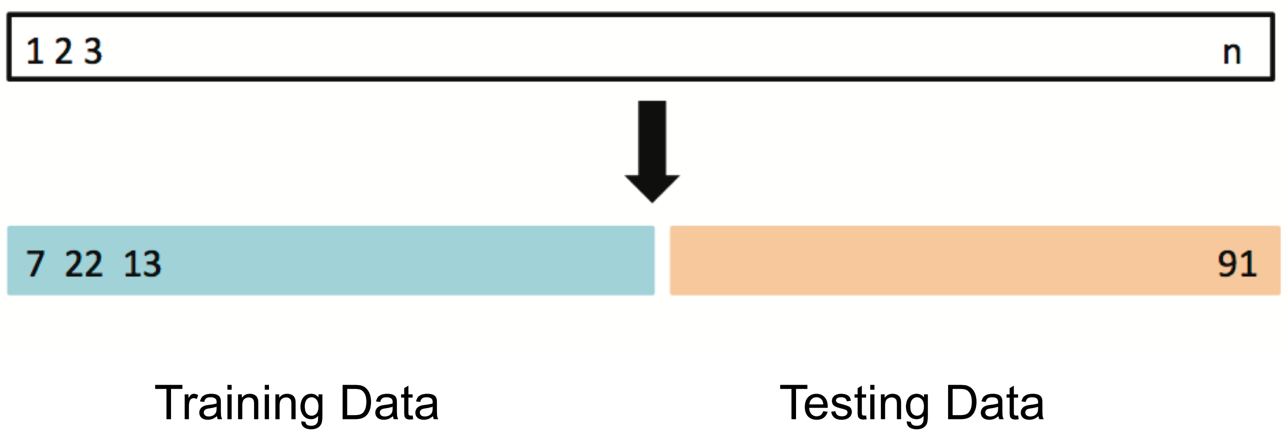


Outline

- Cross Validation on Regression Problems
    1. The Validation Set Approach
    2. Leave-One-Out Cross Validation
    3. K-fold Cross Validation
        - Bias-Variance Trade-off for k-fold Cross Validation
- Cross Validation on Classification Problems

1. The Validation Set Approach

- Suppose that we would like estimate the test error associated with fitting a particular statistical learning method
- We can achieve this goal by randomly splitting the data into
  - training part and
  - validation (testing, or hold-out) part



Example: Auto Data

- Suppose that we want to predict mpg from horsepower
- Linear model:
  - mpg ~ horsepower
- How to do it?
  - Randomly split Auto data set (392 obs.) into training (196 obs.) and validation data (196 obs.)

```
set.seed(1)
train <- sample(392,196)
```

  - Fit the model using the training data set

```
lm.fit.train <- lm(mpg~horsepower,data=Auto,subset=train)
```

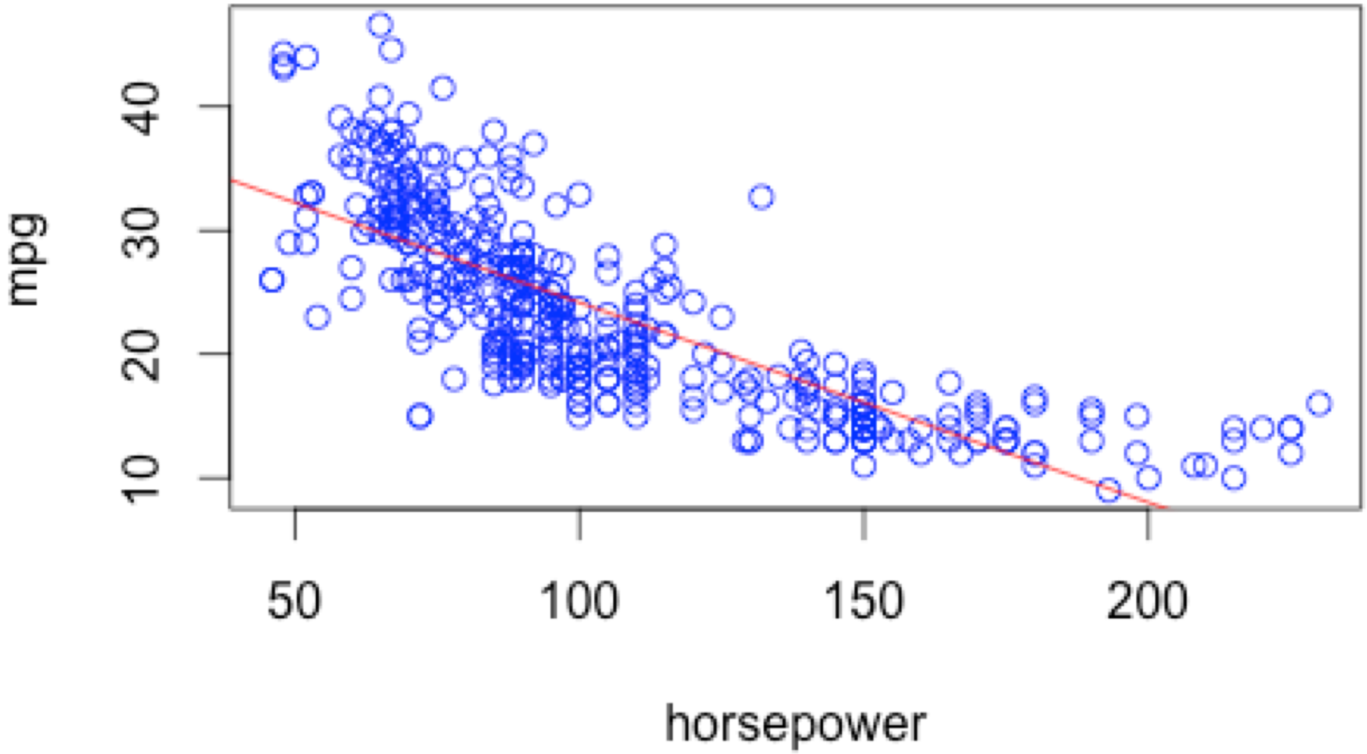
  - Then, evaluate the model using the validation data set

```
mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
[1] 26.14142
```

Plot the observations and linear relationship between mpg and horsepower

horsepower vs mpg

```
plot(Auto$horsepower, Auto$mpg,
          xlab="horsepower",
          ylab="mpg",
          col="blue")
abline(lm.fit.train,col="red")
```



A way to improve

- From the plot, there appears to be a non-linear relationship between mpg and horsepower.
- Try the quadratic model: mpg ~ horsepower + horsepower^2
- Repeat the procedure
  - Randomly split Auto data set (392 obs.) into training (196 obs.) and validation data (196 obs.) – the same as before
  - Fit the model using the training data set

```
lm.fit2.train <- lm(mpg~poly(horsepower,2),data=Auto, subset=train)
```

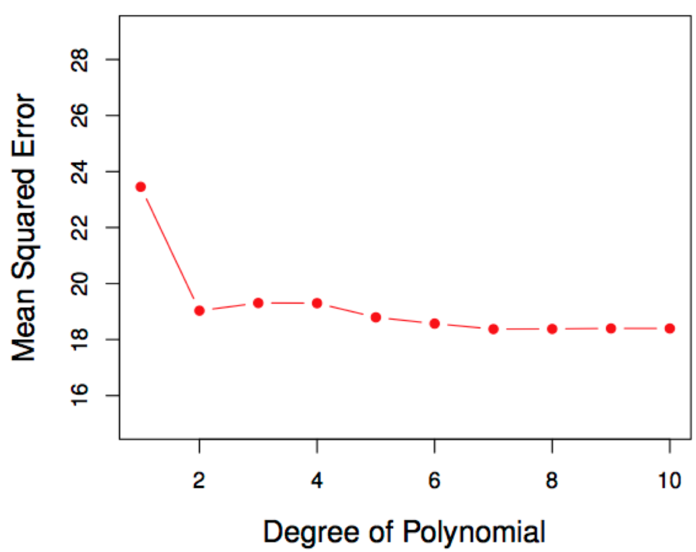
  - Then, evaluate the model using the validation data set

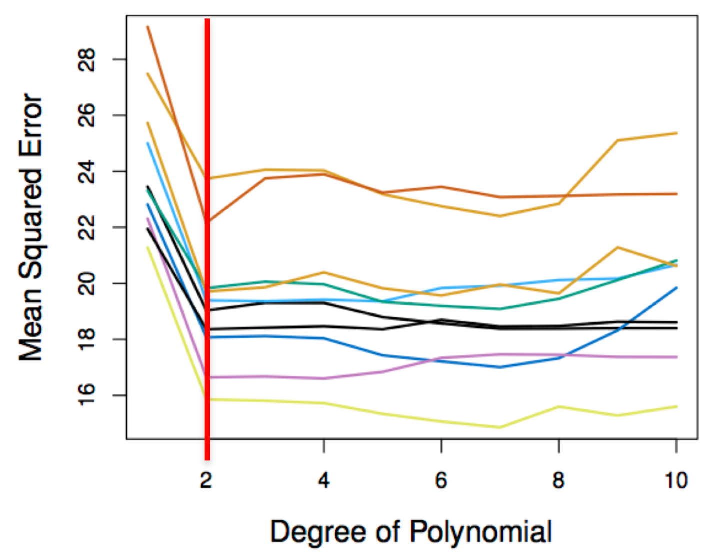
```
mean((Auto$mpg-predict(lm.fit2.train,Auto))[-train]^2)
[1] 19.82259                         #linear model: 26.14142
```

- Compare the two test errors
  - The quadratic model has a smaller test error, thus is better!

Results: Auto Data

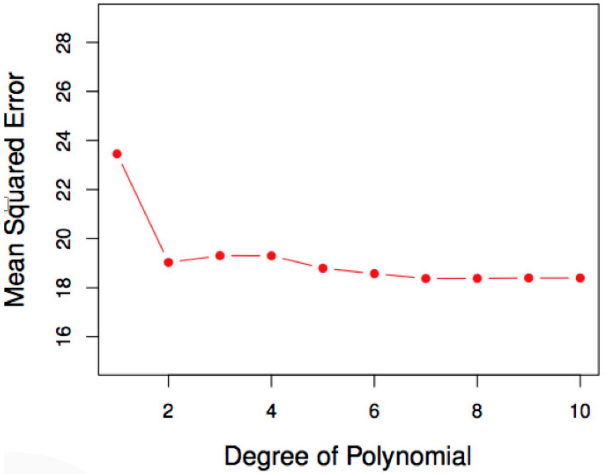
- Left: Validation error rate for a single split
- Right: Validation method repeated 10 times, each time the split is done randomly!
- There is a lot of variability among the MSE's… Not good! We need more stable methods!





Code to Plot Slide 9's Left Figure

```
## Slide 9 Left:
errors <- rep(0,10)      ## errors is a vector of size 10, initially all 0. It records the
                         ## MSE for models that varying in degrees.

set.seed(1)
train <- sample(392,196)

for(i in 1:10){
 lm.fit.train <- lm(mpg~poly(horsepower,i),data=Auto,subset=train)
 errors[i] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
}

#Plot left figure on Slide 9:
plot(errors,
    col="red",
    pch=16,    ## 16 means solid round dots
    type="b",   ## "b" stands for broken lines, use "l" if you need continuous lines
    xlab="Degrees of Polynomial", ylab="Mean Squared Error",
    main="10 times random split")
```

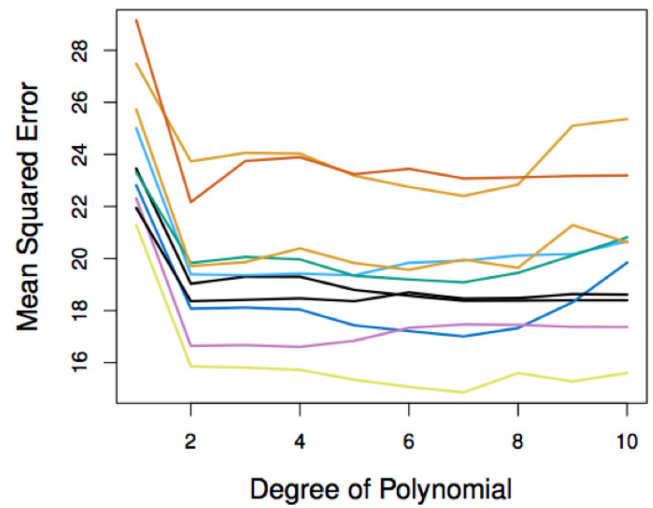


Code to Plot Slide 9's Right Figure (Base Graph)

```
## Next, to plot the figure on the right on Slide 9:
## There are two approaches.
## The first approach is to keep only one vector for errors. It is easier to understand, but all
## the previous data will be lost.
## The second approach is to keep a two-dimensional matrix. It will store all the errors
calculated so far. But it may take a while to understand.

## No matter which approach you choose, you need to plot a base graph. All the lines will
be added on this base graph.

## Base graph: just a 0, it is to create an empty graph
plot(0,
	xlab="Degrees of Polynomial",ylab="Mean Squared Error",
	main="10 times random split",
	ylim = c(14,27), xlim=c(0,10),
	type="l")
```

Code to Plot Slide 9's Right Figure (1st Approach)

```
#The first approach:
errors <- rep(0,10)   # errors is a vector of size 10, initially evaluated to all 0's
for(i in 1:10){
  set.seed(i)  #can try 100+i, or 874+i  #to change a seed before a new error is calculated
  train <- sample(392,196) #this is to get a new training set
 #for each training set we draw one line as follows:
for(j in 1:10){
    lm.fit.train <- lm(mpg~poly(horsepower,j),data=Auto,subset=train)
    errors[j] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
  }
  lines(errors,col=i)
}
```



Code to Plot Slide 9's Right Figure (2nd Approach)

```
##The second approach:
errorMatrix <- matrix(nrow=10, ncol=10) # the matrix to record errors
# the number errorMatrix[i,j] records the error using i's training set with degree j
for(i in 1 : 10){
  set.seed(i)
  train <- sample(392,196)
  for(j in 1:10){
    lm.fit.train <- lm(mpg~poly(horsepower,j),data=Auto,subset=train)
    errorMatrix[i,j] <- mean((Auto$mpg-predict(lm.fit.train,Auto))[-train]^2)
  }
  lines(errorMatrix[i,],col=i)
}

##The legend function will draw a list of legends on the plot on the position you determined.
legend("topleft",c("1","2","3","4","5","6","7","8","9","10"),
         lty=rep(1,10),  col=1:10,  lwd=rep(2.5,10),  cex=0.6)
#lty is line type, lwd is line width, cex is the size
```

The Validation Set Approach

- Advantages:
  - Simple
  - Easy to implement
- Disadvantages:
  - The validation MSE can be highly variable
  - Only a subset of observations are used to fit the model (training data).

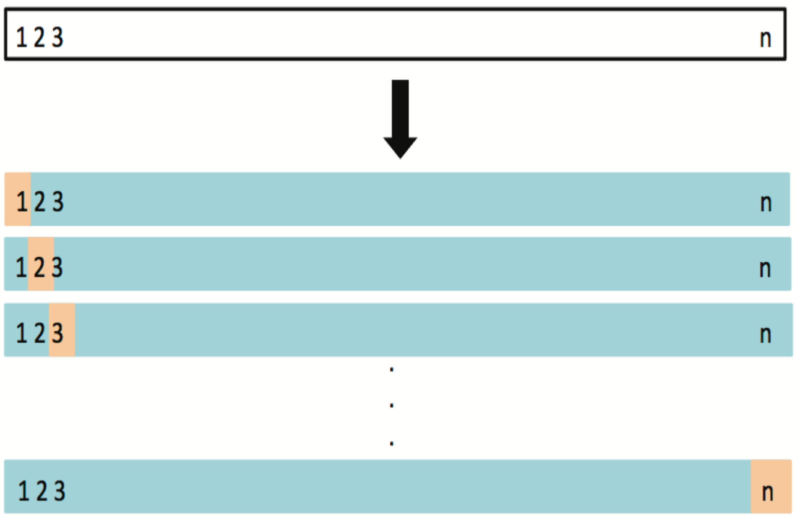
    Statistical methods tend to perform worse when trained on fewer observations.

2. Leave-One-Out Cross Validation (LOOCV)

- This method is similar to the Validation Set Approach, but it tries to address the latter's disadvantages.

- For each suggested model, do:
  - Split the data set of size n into
    - Training data set (blue) size: n -1
    - Validation data set (beige) size: 1
  - Fit the model using the training data
  - Validate model using the validation data, and compute the corresponding MSE
  - Repeat this process n times
  - The MSE for the model is computed as follows:

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MSE}_i .$$



LOOCV vs. Validation Set Approach

- LOOCV has less bias
  - We repeatedly fit the statistical learning method using training data that contains n - 1 obs., i.e. almost all the data set is used
- LOOCV produces a less variable MSE
  - The validation set approach produces different MSE when applied repeatedly due to randomness in the splitting process
  - Performing LOOCV multiple times will always yield the same results, because we split based on 1 obs. each time
- LOOCV is computationally intensive (disadvantage)
  - We fit a model n times!

Perform LOOCV in R

- Using the Auto data set again, building a linear model

```
glm.fit <- glm(mpg~horsepower,data=Auto)
# This is the same as lm(mpg~horsepower,data=Auto)

library(boot) #cv.glm() is in the boot library
cv.err <- cv.glm(Auto,glm.fit)
# cv.glm() does the LOOCV

cv.err$delta
[1] 24.23151 24.23114  (raw CV est., adjusted CV est.)
```

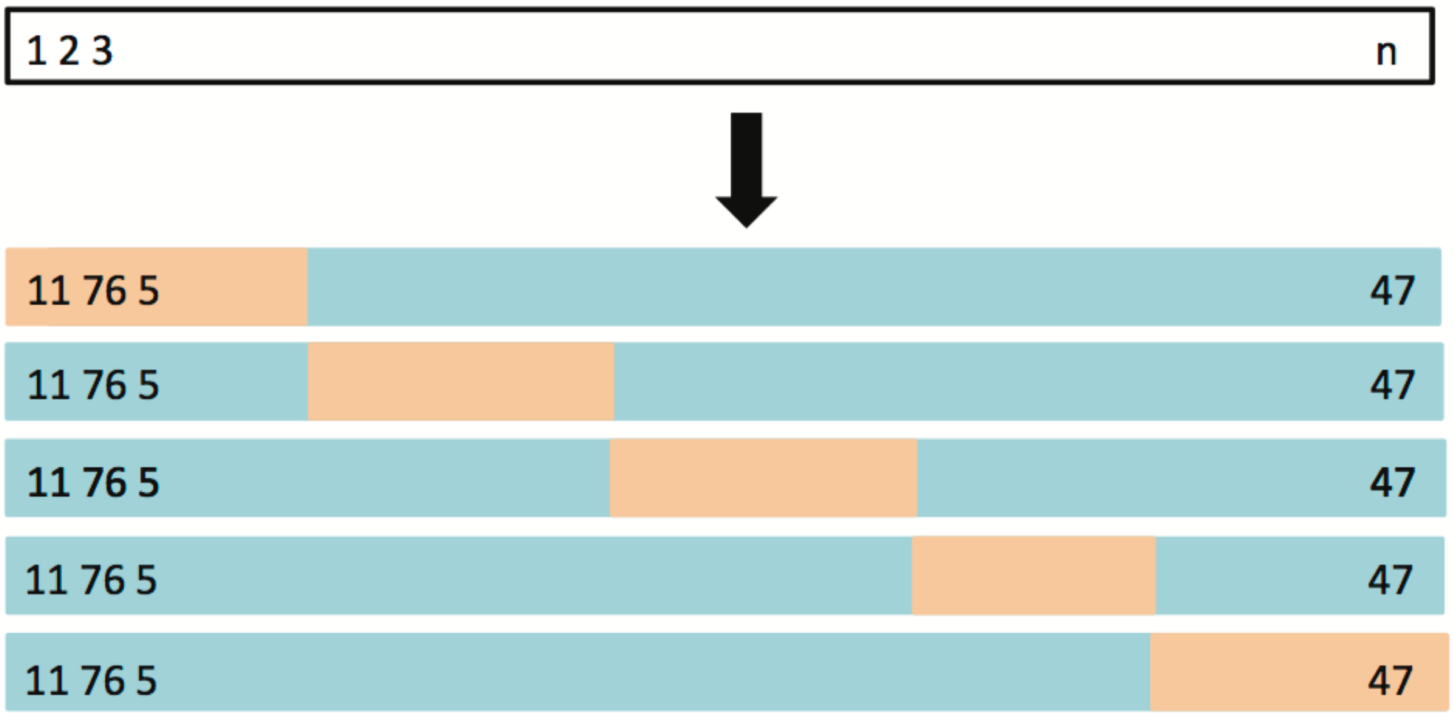
The MSE is 24.23151.

3. k-fold Cross Validation

- LOOCV is computationally intensive, so we can run *k*-fold Cross Validation instead
- With *k*-fold CV, we divide the data set into *k* different parts (e.g. $k = 5$, or $k = 10$, etc.)
- We then remove the first part, fit the model on the remaining *k*-1 parts, and see how good the predictions are on the left out part (i.e. compute the MSE on the first part)
- We then repeat this *k* different times taking out a different part each time
- By averaging the *k* different MSE's we get an estimated validation (test) error rate for new observations

$$\mathrm{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{MSE}_i .$$

K-fold Cross Validation



$k = ?$

Perform K-fold CV in R

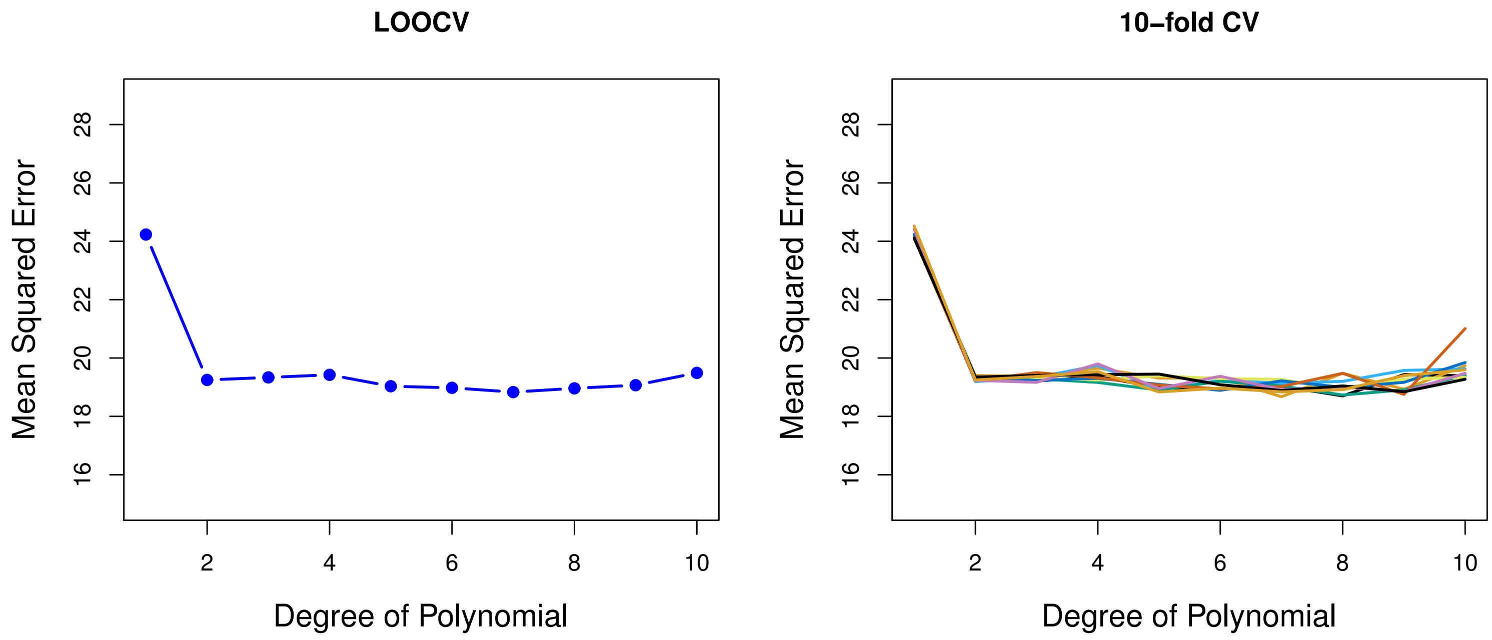
- Very easy!

```
> glm.fit <- glm(mpg~horsepower,data=Auto)
># This is the same as in LOOCV

> library(boot)  # This is the same as in LOOCV
> cv.err <- cv.glm(Auto,glm.fit, K=10)
#K means K-fold, can be 5, 10 or other numbers

> cv.err$delta
[1] 24.3120 24.2926
```

The MSE is 24.3120.

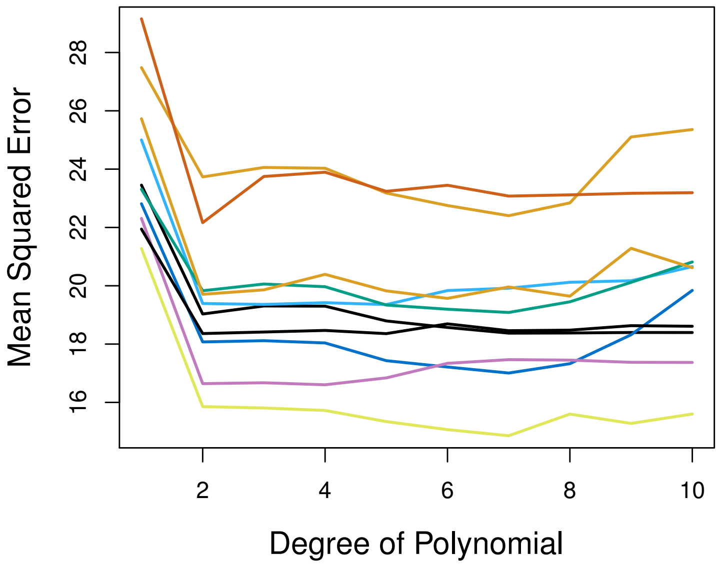
Auto Data: LOOCV vs. k-fold CV

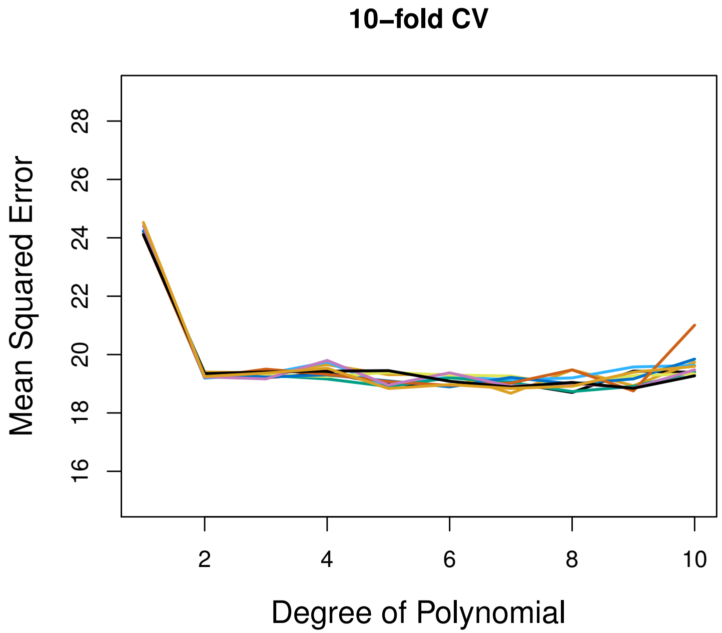
- Left: LOOCV error curve
- Right: 10-fold CV was run many times, and the figure shows the slightly different CV error rates
- LOOCV is a special case of k-fold, where $k = n$
- They are both stable, but LOOCV is more computationally intensive!



Auto Data: Validation Set Approach vs. k-fold CV Approach

- Left: Validation Set Approach
- Right: 10-fold Cross Validation Approach
- Indeed, 10-fold CV is more stable!





Bias-Variance Trade-off for k-fold CV

Birkbeck UNIVERSITY OF LONDON

- Putting aside that LOOCV is more computationally intensive than k-fold CV… Which is better LOOCV or *k*-fold CV?
  - LOOCV is less bias than *k*-fold CV (when $k < n$)
    - LOOCV: uses n-1 observations
    - K-fold CV: uses (k-1)n/k observations
  - But, LOOCV has higher variance than *k*-fold CV (when $k < n$)
    - The mean of many highly correlated quantities has higher variance
  - Thus, there is a trade-off between what to use
- Conclusion:
  - We tend to use k-fold CV with ($k = 5$ and $k = 10$)
  - These are the magical *k*'s ☺
  - It has been empirically shown that they yield test error rate estimates that suffer neither from excessively high bias, nor from very high variance

Cross Validation on Classification Problems

- So far, we have been dealing with CV on regression problems
- We can use cross validation in a classification situation in a similar manner
  - Divide data into k parts
  - Hold out one part, fit using the remaining data and compute the error rate on the held out data
  - Repeat k times
  - CV error rate is the average over the k errors we have computed