# Big Data Analytics

## Session 5(b)

## Cross Validation

# So far

- Compute MSE/error rate on the training data
  - Easy!

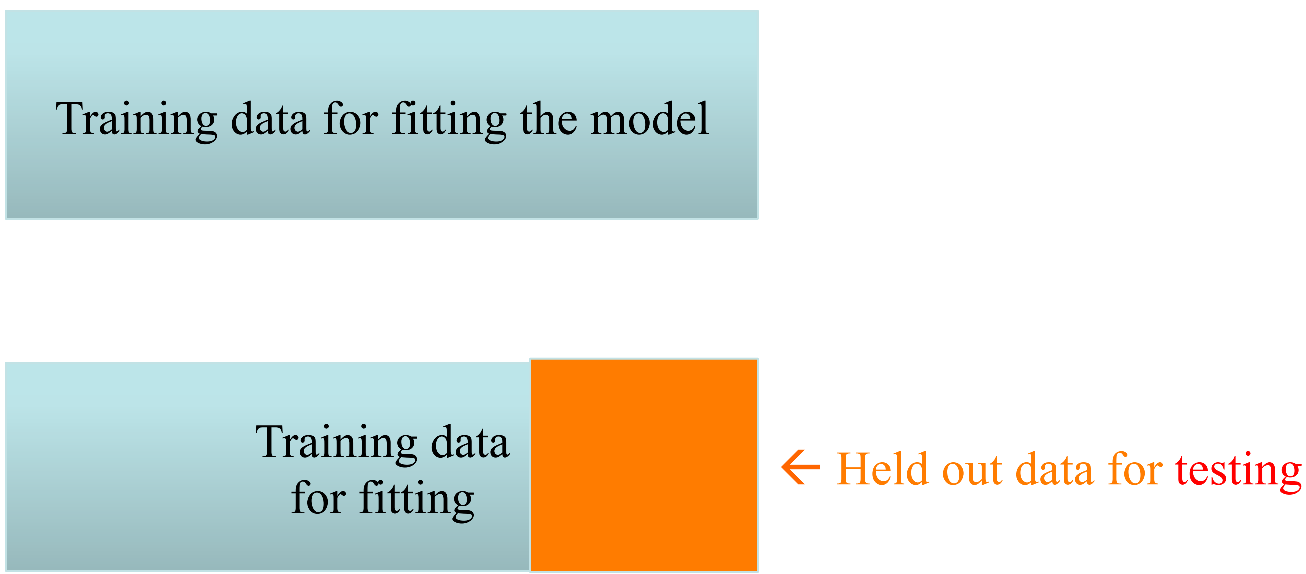
- Calculate MSE/error rate on the test data
  - Easy, if the designated test set is available
  - ➔ Unfortunately, this is usually not the case

- 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?

# 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.

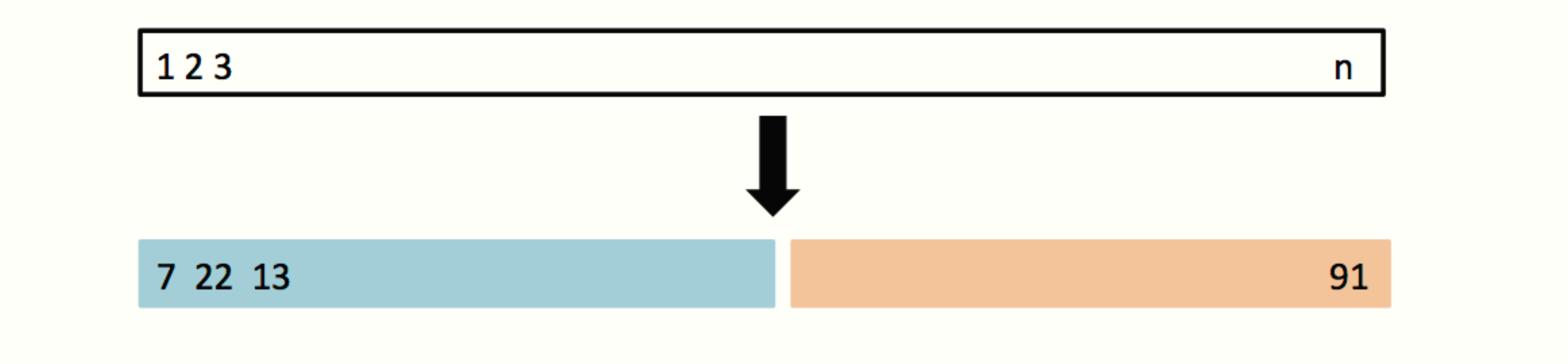


# Outline

- 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




Training Data Testing Data

# 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

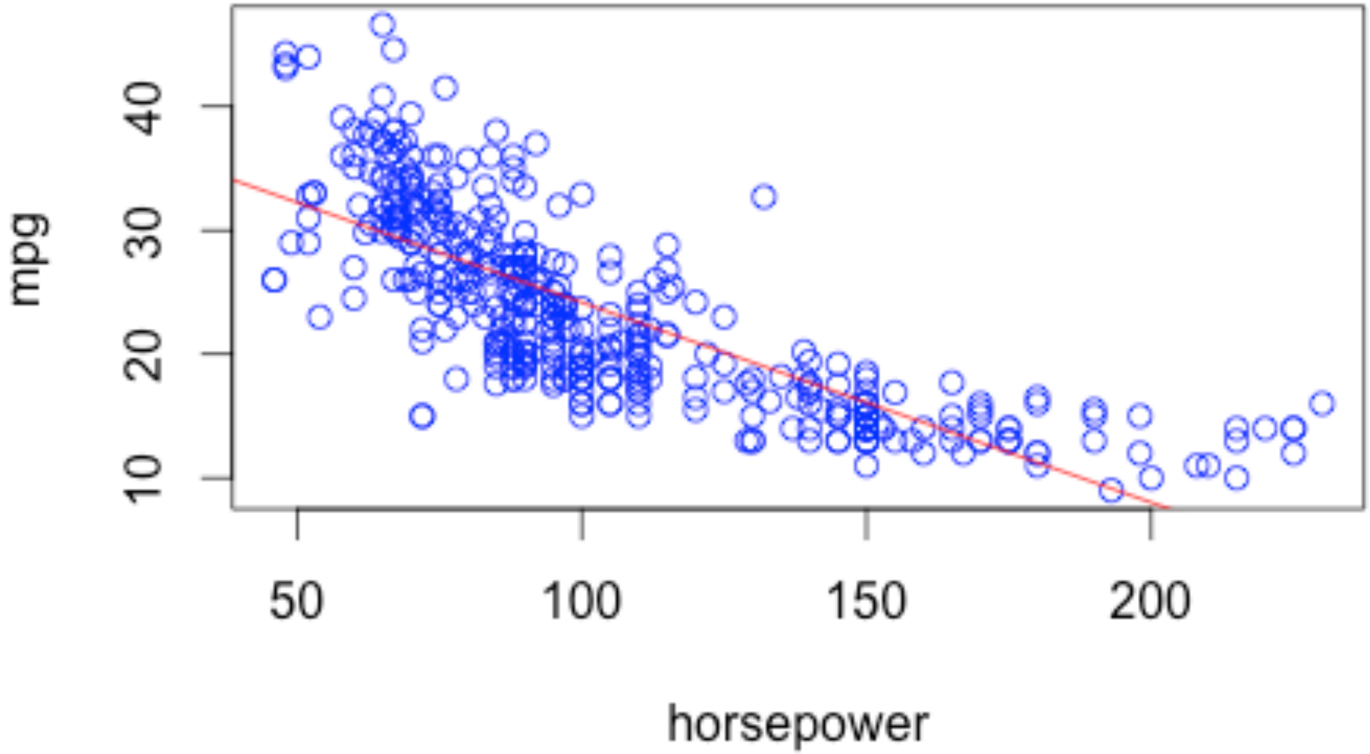
```
> lm.fit.train=lm(mpg~horsepower,data=Auto,subset=train)
```

  - 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

# Did you get this?



# 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 + horspower$^2$
- 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)
```

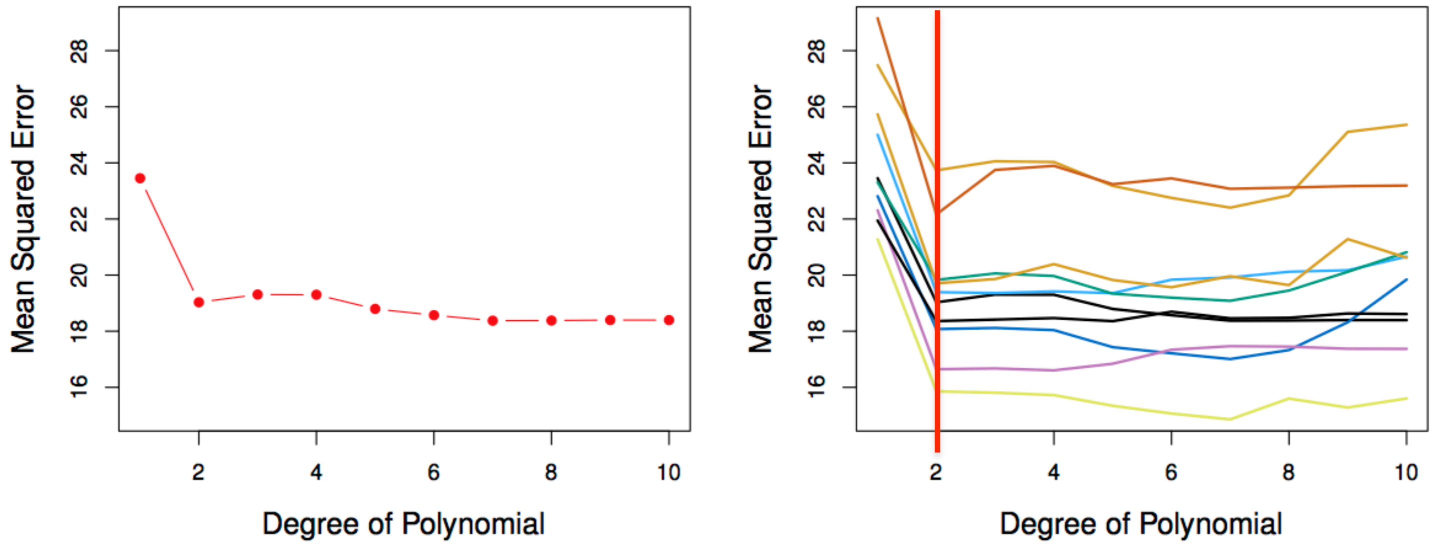
  - Then, evaluate the model using the validation data set

```
> 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!



# The Validation Set Approach

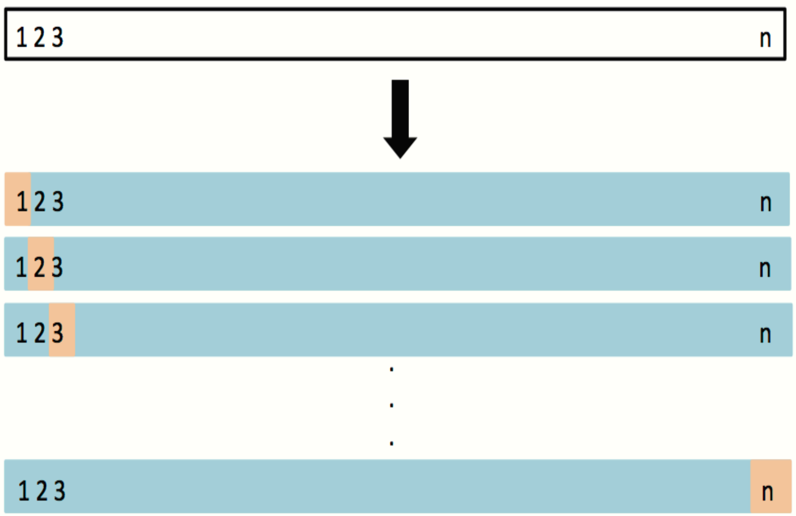
- Advantages:
  - Simple
  - Easy to implement
- Disadvantages:
  - The validation MSE can be highly variable
  - Only a subset of observations are used to fit the model (training data).
    Statistical methods tend to perform worse when trained on fewer observations.

# 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.fit(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
```

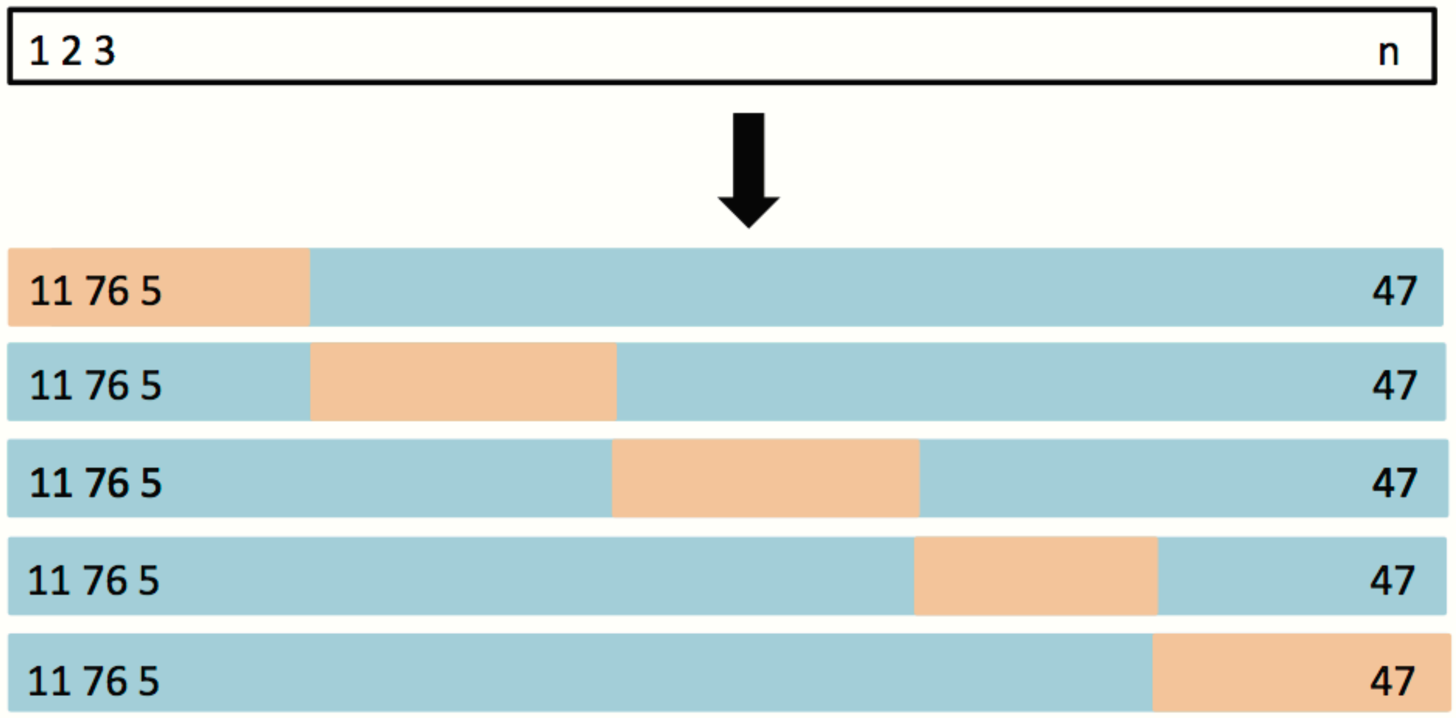
The MSE is 24.23151.

# 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



# Perform K-fold CV in R

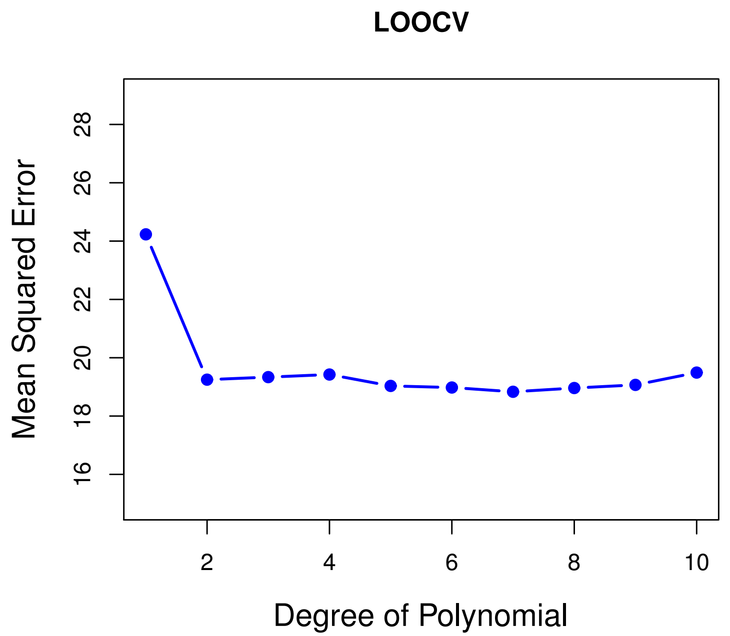
- Very easy!

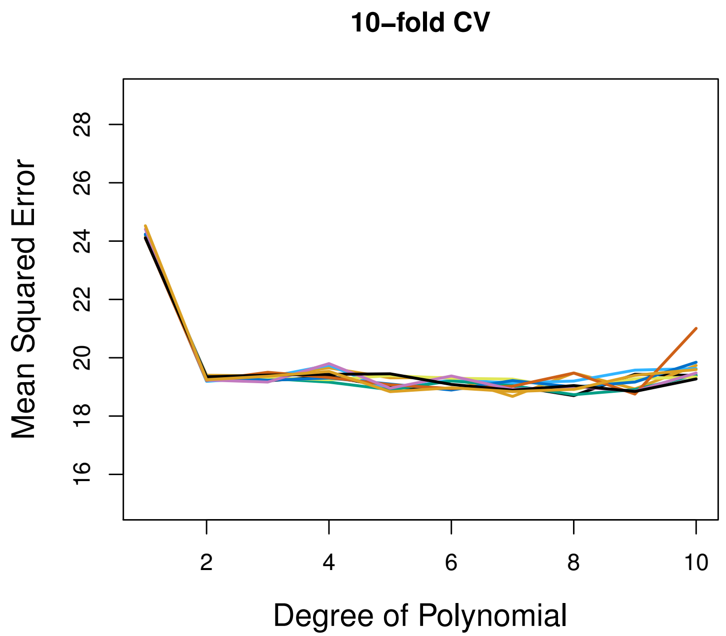
```
> glm.fit=glm(mpg~horsepower,data=Auto)
># This is the same as in LOOCV

> 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

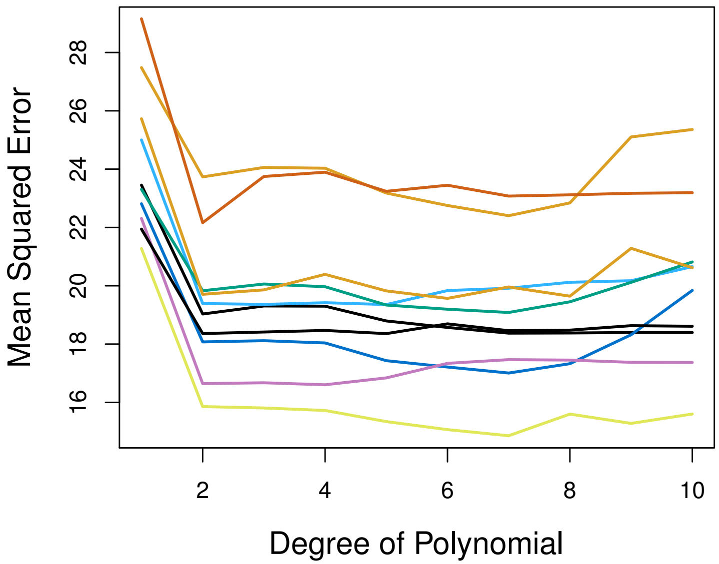
- Left: LOOCV error curve
- Right: 10-fold CV was run many times, and the figure shows the slightly different CV error rates
- LOOCV is a special case of *k*-fold, where $k = n$
- They are both stable, but LOOCV is more computationally intensive!

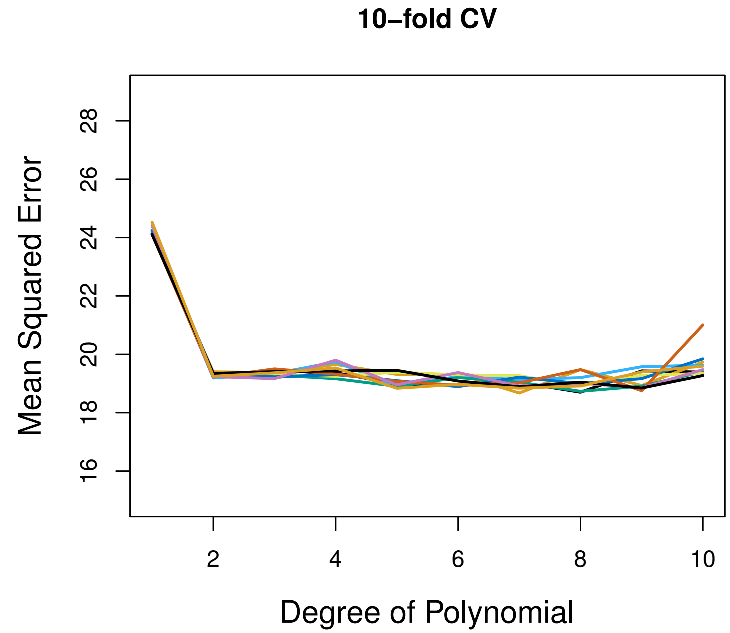


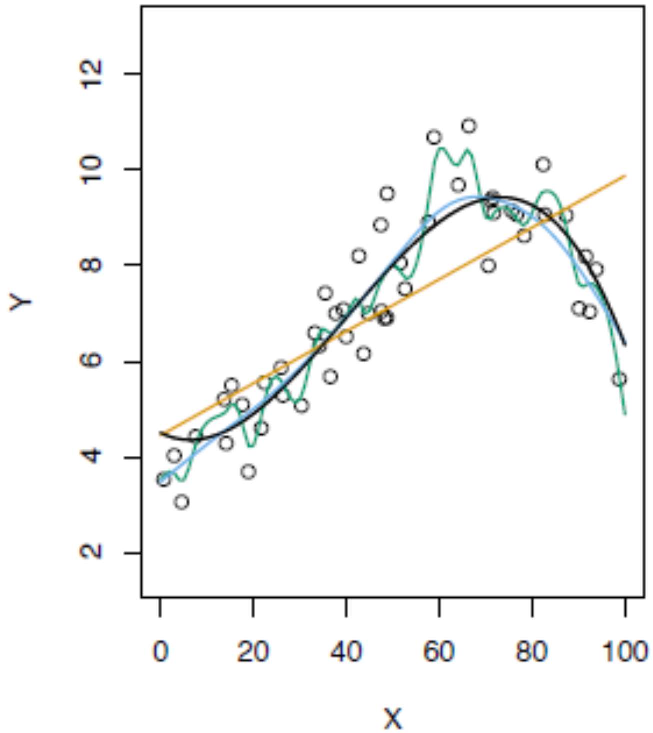
# Auto Data: Validation Set Approach vs. k-fold CV Approach

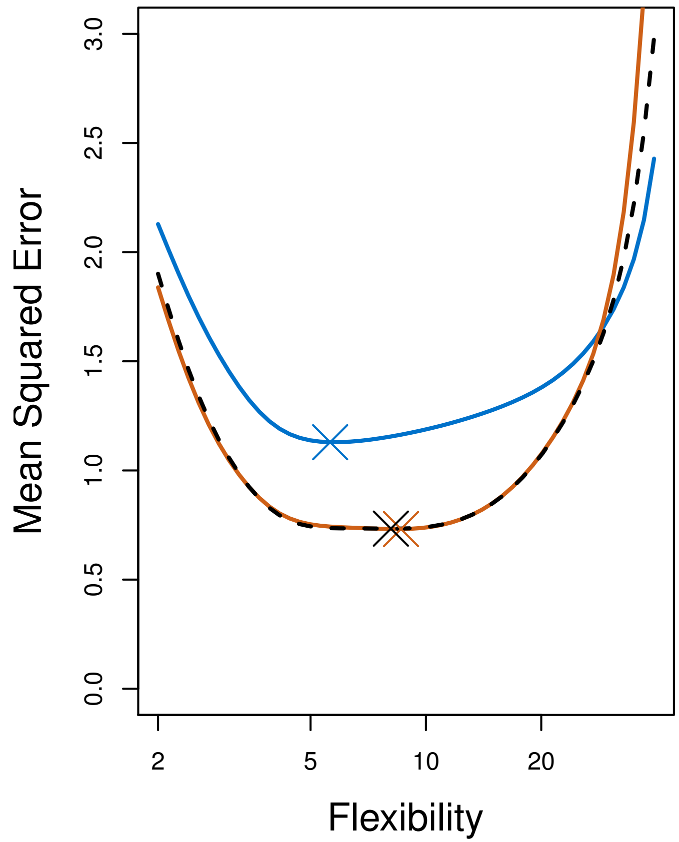
- Left: Validation Set Approach
- Right: 10-fold Cross Validation Approach
- Indeed, 10-fold CV is more stable!



# K-fold Cross Validation on the Simulated Data




- Blue: True Test MSE
- Black: LOOCV MSE
- Orange: 10-fold MSE
- Refer to chapter 2 for Fig 2.9. More example see Fig 5.6

Model: Smoothing spline

# Bias-Variance Trade-off for k-fold CV

- 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 hold out data
  - Repeat $k$ times
  - CV error rate is the average over the $k$ errors we have computed

# LAB

# The Validation Set Approach

- Goal: To estimate the test MSE
- Approach:
  - Randomly pick half of the data as the train data
    - Random (but repeatable): `set.seed(1)`
    - Pick half: `train = sample(392,196)`
  - Estimate the test MSE on the other half
    - Test MSE: $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$
    - Other half: set indices as `[-train]`
    - $y_i$: `mpg`
    - \hat{f}($x_i$):
      - `lm.fit=lm(mpg~horsepower,data=Auto,subset=train)`
      - `Predict(lm.fit,Auto)`
    - 1/n: `mean()`
    - `mean((mpg-predict(lm.fit,Auto))[-train]^2)`

# The Validation Set Approach

- Linear regression (Degree 1)
  - `lm.fit = lm(mpg~horsepower,data=Auto,subset=train)`
  - `mean((mpg-predict(lm.fit,Auto))[-train]^2)`
  - `26.14`
- Polynomial regression
  - Degree 2
    - `lm.fit2 = lm(mpg~poly(horsepower,2),data=Auto,subset=train)`
    - `mean((mpg-predict(lm.fit2,Auto))[-train]^2)`
    - `19.82`
  - Degree 3
    - `lm.fit3 = lm(mpg~poly(horsepower,3),data=Auto,subset=train)`
    - `mean((mpg-predict(lm.fit3,Auto))[-train]^2)`
    - `19.78`
- What can we conclude from the above results?

# The Validation Set Approach

- Choosing a different training set, then we will obtain different errors on the validation set.

  `set.seed(2)` or `set.seed(i) (i≠1)`

  `repeat the rest`

- Notice the variability on the results

# Leave-One-Out Cross Validation

- Function `glm()`
  - In logistic regression:

    `glm(y~x,family="binomial",data=..)`
  - In linear regression: `glm(y~x, data=..)`

```
> glm.fit=glm(mpg~horsepower,data=Auto)
> coef(glm.fit)
(Intercept)  horsepower
     39.936      -0.158
```

the same as `lm(y~x,data=..)`

```
> lm.fit=lm(mpg~horsepower,data=Auto)
> coef(lm.fit)
(Intercept)  horsepower
     39.936      -0.158
```

# Leave-One-Out Cross Validation

- Function `cv.glm()` in `boot` library
  - Produces a list with several components, including the cross-validation estimate for the test error: `delta`
  - `cv.glm(data, glmfit, cost, K)`

```
> library(boot)
> glm.fit=glm(mpg~horsepower,data=Auto)
> cv.err=cv.glm(Auto,glm.fit)
> cv.err$delta
    1     1
24.23 24.23
```

  - `Delta` is a vector of length two. For LOOCV, the two are the same.

# Leave-One-Out Cross Validation

- Experiment on the CV for increasingly complex polynomial fits
- Initialise a vector of length `len` all to be number `val`
  - `vec = rep(val,len)`
- Using for loop to repeat procedure

```
> cv.error=rep(0,5)
> for (i in 1:5){
+ glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
+ cv.error[i]=cv.glm(Auto,glm.fit)$delta[1]
+ }
> cv.error
[1] 24.23 19.25 19.33 19.42 19.03
```

# K-Fold Cross Validation

- Implement k-fold CV by passing the argument K

```
> set.seed(17)
> cv.error.10=rep(0,10)
> for (i in 1:10){
+ glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
+ cv.error.10[i]=cv.glm(Auto,glm.fit,K=10)$delta[1]
+ }
> cv.error.10
[1] 24.21 19.19 19.31 19.34 18.88 19.02 18.90 19.71 18.95 19.50
```

- The two numbers associated with delta
  - The first number is the raw/standard CV estimate of prediction error
  - The second number is the adjusted CV estimate. The adjustment is designed to compensate for the bias introduced by not using leave-one-out cross-validation