CSCE 633: Machine Learning

Lecture 2: Model Selection

Texas A&M University

8-28-19

Last Time

- Machine learning tasks
- Basics of learning
- Bayes error
- Brief discussion on KNN

Goals for Today

- KNN: a closer look
- Understanding re-sampling
- Understanding measures of accuracy in re-sampling

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)

setosa

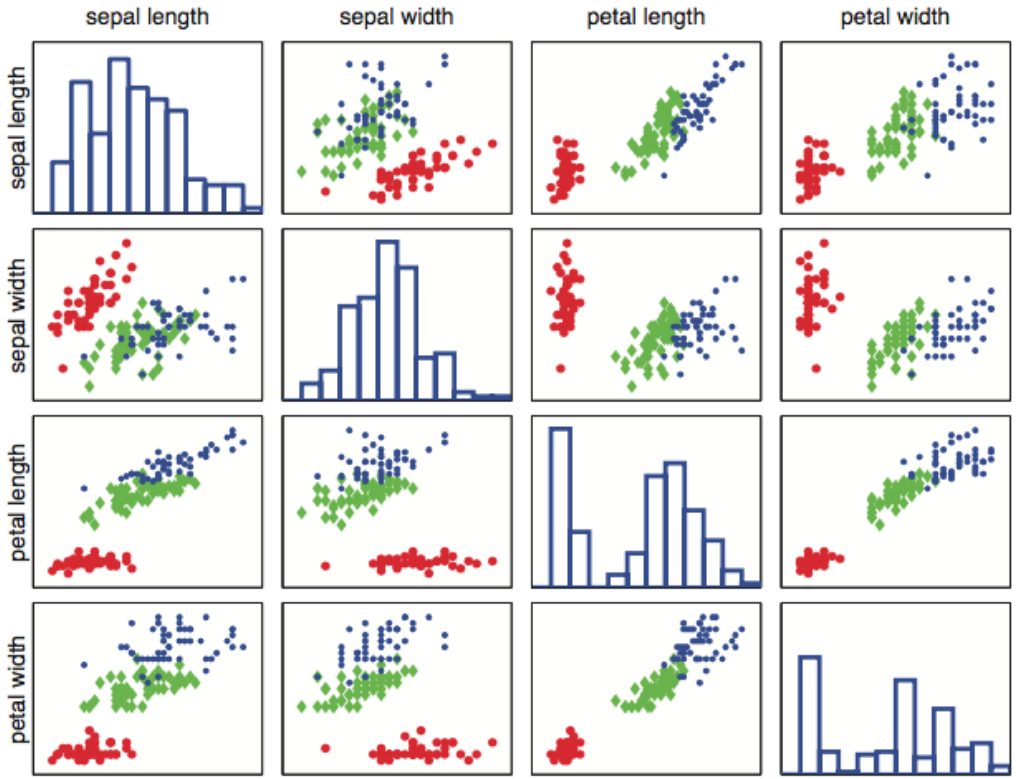
versicolor

virginica

Features: the widths and lengths of sepal and petal

K-Nearest Neighbor: Example

Visualizing features to get better intuition about our data

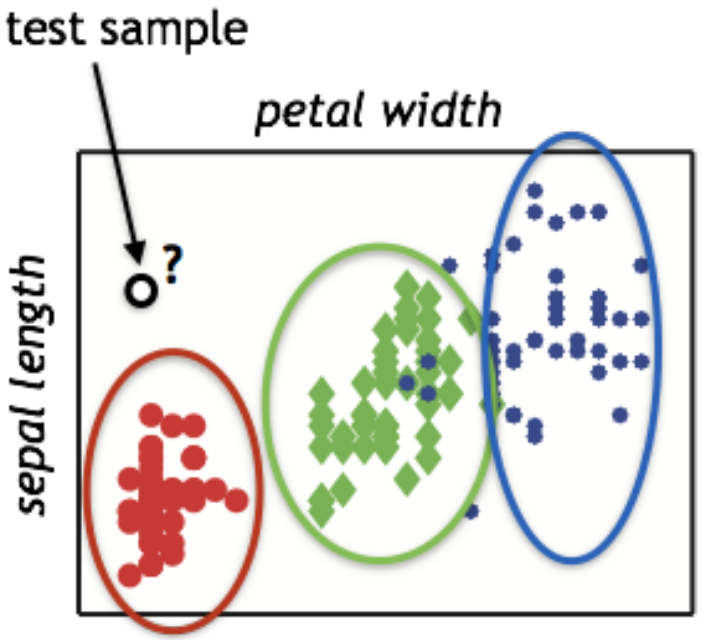


Each colored datapoint is one sample

setosa, versicolor, virginica

K-Nearest Neighbor: Example

Using two features: sepal length & petal width



setosa, versicolor, virginica

Test sample is closer to red cluster → label it as setosa

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)

Often data is organized in a table

Each row is one sample with 4 features and 1 label

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
```

```
Attribute Information:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica
```

[Source: `https://archive.ics.uci.edu/ml/datasets/iris`]

K-Nearest Neighbor: Representation

Training Data

- N samples/datapoints/instances: $\mathcal{S}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Used for learning representation $f : \mathbf{x} \rightarrow y$

Testing Data

- M samples/datapoints/instances: $\mathcal{S}^{test} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_M}, y_M)\}$
- Used to assess how well $f(\cdot)$ will do in predicting an unseen sample

Train and test data should not overlap: $\mathcal{S}^{train} \cap \mathcal{S}^{test} = \emptyset$

K-Nearest Neighbor: Representation

Classify data into one out of multiple classes

- Input: $\mathbf{x} \in \mathbb{R}^n$ (features, attributes, etc.)
- Output: $y \in \{1, 2, \ldots, C\}$ (labels)
- Model: $f : \mathbf{x} \rightarrow y$

Special case: binary classification (C=2)

- Output: $y \in \{1, 2\}$ or $\{0, 1\}$ or $\{-1, 1\}$, etc.

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN)

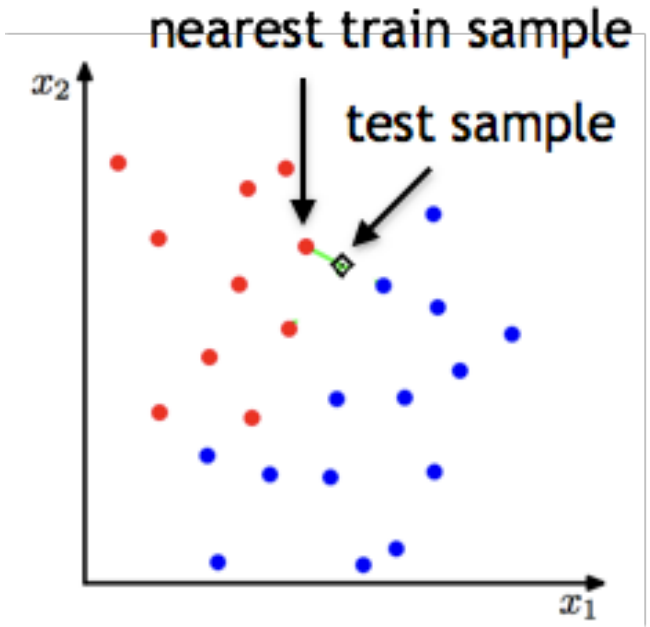
- Assigns test sample $\mathbf{x}$ to the closest training sample
- Model

$$y = f(\mathbf{x}) = y_{nn(\mathbf{x})}$$

$$nn(\mathbf{x}) = arg \min_{i=1,\ldots,m} \|\mathbf{x} - \mathbf{x_n}\|^2 = arg \min_{i=1,\ldots,m} \sum_{d=1}^{n} (x_d - x_{id})^2$$

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example

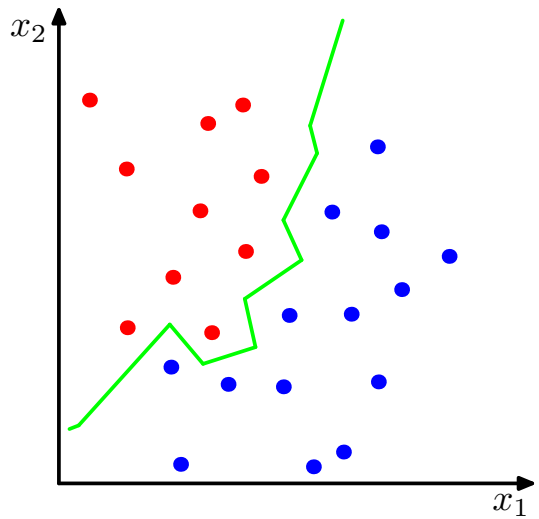


The nearest point to test sample **x** is a red training instance, therefore **x** will be labeled as red.

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example

Decision boundary: For every point in the space, we can determine its label using the nearest neighbor rule. This gives us a decision boundary that partitions the space into different regions.



The above decision boundary is very sensitive to noise

What would be the solution for this?

K-Nearest Neighbor: Representation

Increase number of nearest neighbors to use

- 1-nearest neighbor: $nn_1(\mathbf{x}) = arg\,\min_{i \in \{1,\ldots,m\}} \|\mathbf{x} - \mathbf{x_i}\|_2^2$
- 2-nearest neighbor: $nn_2(\mathbf{x}) = arg\,\min_{i \in \{1,\ldots,m\} \setminus nn_1(\mathbf{x})} \|\mathbf{x} - \mathbf{x_i}\|_2^2$
- 3-nearest neighbor:
  $nn_3(\mathbf{x}) = arg\,\min_{i \in \{1,\ldots,m\} \setminus \{nn_1(\mathbf{x}), nn_2(\mathbf{x})\}} \|\mathbf{x} - \mathbf{x_i}\|_2^2$

The set of K-nearest neighbors is

$$knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \ldots, nn_K(\mathbf{x})\}$$

Neighbors $nn_1, \ldots, nn_K$ in order of increasing distance from sample $\mathbf{x}$

K-Nearest Neighbor: Representation

K-NN Model

- Each neighbor in $knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \ldots, nn_K(\mathbf{x})\}$ votes one class
- Count the number of neighbors that have voted each class

$$v_c = \sum_{k \in knn(\mathbf{x})} \mathbb{I}(y_k = c), \quad c = 1, \ldots, C$$

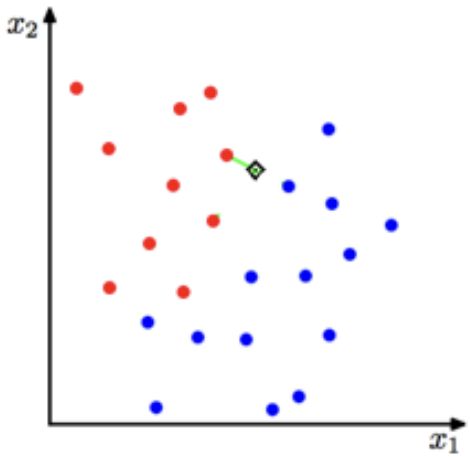
- Assign test sample $\mathbf{x}$ to to the majority class membership of the K neighbors

$$y = f(\mathbf{x}) = arg \max_{c=1,\ldots,C} v_c$$

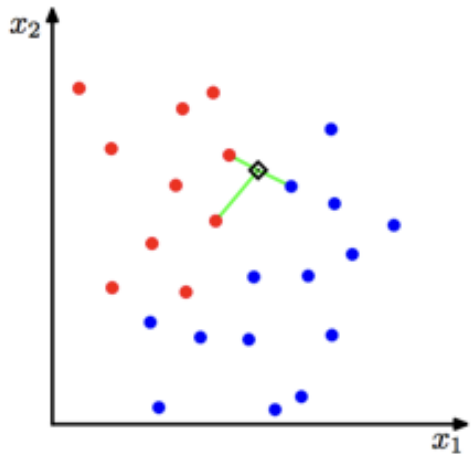
K-Nearest Neighbor: Representation

K-NN Example

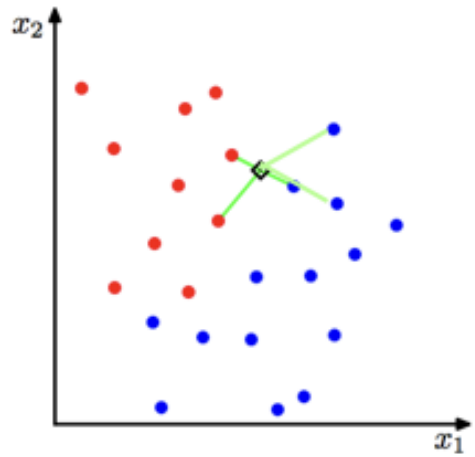
K=1, label=red



K=3 label=red

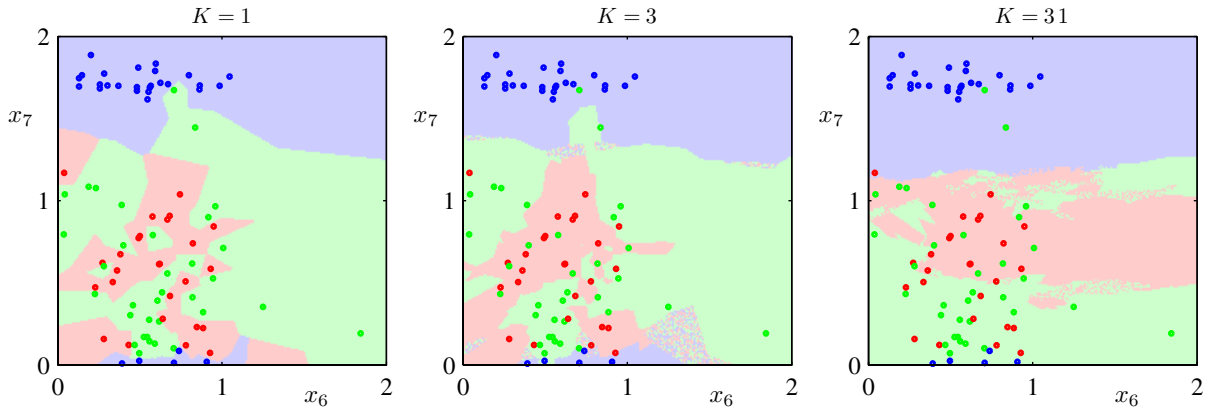


K=5, label=blue



K-Nearest Neighbor: Representation

K-NN Decision Boundary



Number of neighbors K controls the degree of smoothing

$K \downarrow$: many small regions of each class

$K \uparrow$: fewer larger regions of each class

K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^n$ given that we have m training data (assuming $n > K$)?

A) $O(mn)$
B) $O(Kn)$
C) $O(Km)$
D) $O(Kmn)$

K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^n$ given that we have m training data (assuming $n > K$)?

A) $O(mn)$
B) $O(Kn)$
C) $O(Km)$
D) $O(Kmn)$

The correct answer is A.

The cost of measuring the distance between the test sample and every sample in the training data is $O(n)$
The cost of computing distances for all m train samples is $O(mn)$
The cost of finding the K closets samples is $O(Km)$ (can be optimized)
So the total cost is $O(mn + Km)$, assuming $n > K$ it becomes $O(mn)$

[Nice video source: `https://www.youtube.com/watch?v=UPAnUE_g5SQ`]

Resampling Methods: Why?

- Resampling is a method by which we re-draw subsets of the training data to see if the model we fit changes
- This allows us to determine how well our regression (for example) fits or how often the coefficients change

Resampling Methods: Two Techniques

Cross-Validation

- Helps give estimate of performance (measures test error)
- Allows for a choice in level of flexibility in model selection
- Helps drive hyperparameter tuning
- (you tend to see this more in computer science work)

Bootstrapping

- Helps give estimate of performance (measures test error)
- Provides measures of accuracy for parameter estimates
- (you tend to see this more in statistical learning and biostatistics work)

Cross-Validation: Why

- Recall in model fitting we want to minimize error rates
- However, does a good performance on the training set guarantee a good performance on the test set? (what is overfitting?) changed from:if we overfit to minimizing training error rate what happens to test error?

Cross-Validation: Why

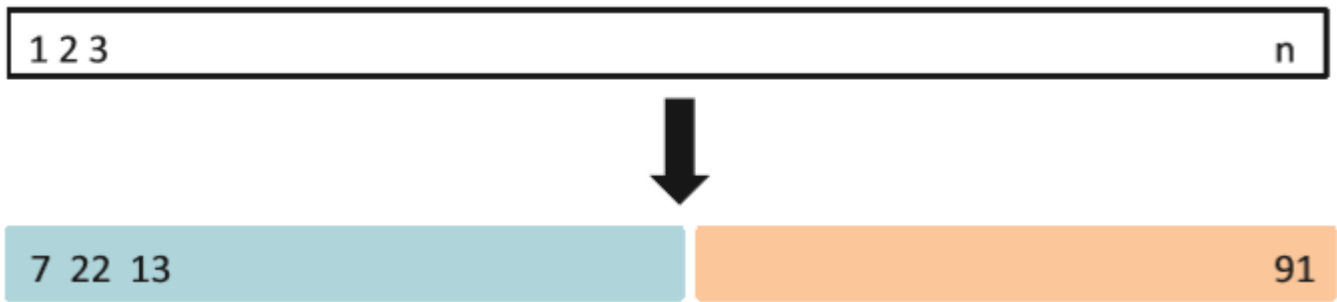
- Recall in model fitting we want to minimize error rates
- However, does a good performance on the training set guarantee a good performance on the test set? (what is overfitting?) changed from:if we overfit to minimizing training error rate what happens to test error?
- U-shape error comes from training models based upon training error minimization

Cross-Validation: Why

- Recall in model fitting we want to minimize error rates
- However, does a good performance on the training set guarantee a good performance on the test set? (what is overfitting?) changed from:if we overfit to minimizing training error rate what happens to test error?
- U-shape error comes from training models based upon training error minimization
- Hold Out Set (sometimes called validation set, sometimes called test set): randomly select a portion of the training set and set aside to evaluate a "test error"

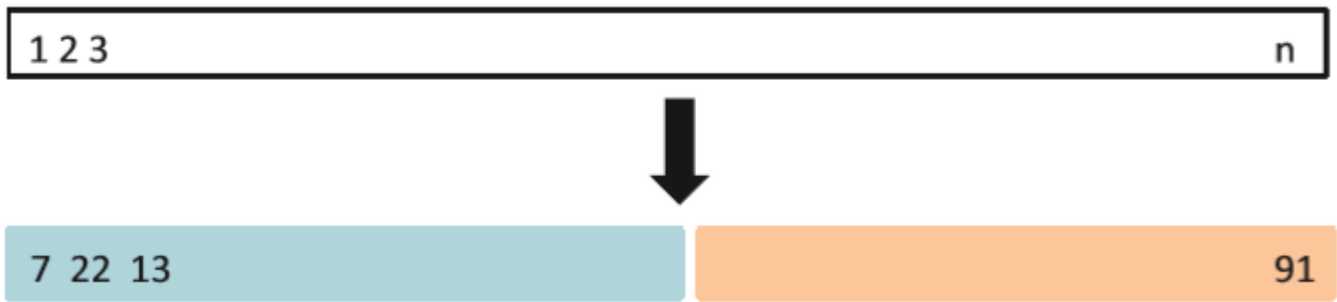
Cross-Validation: Test Set Approach

- We can vary the split point (50/50, 80/20, 90/10 are common divisions)
- It is important to do this step prior to ANY data cleaning, feature engineering, model selection



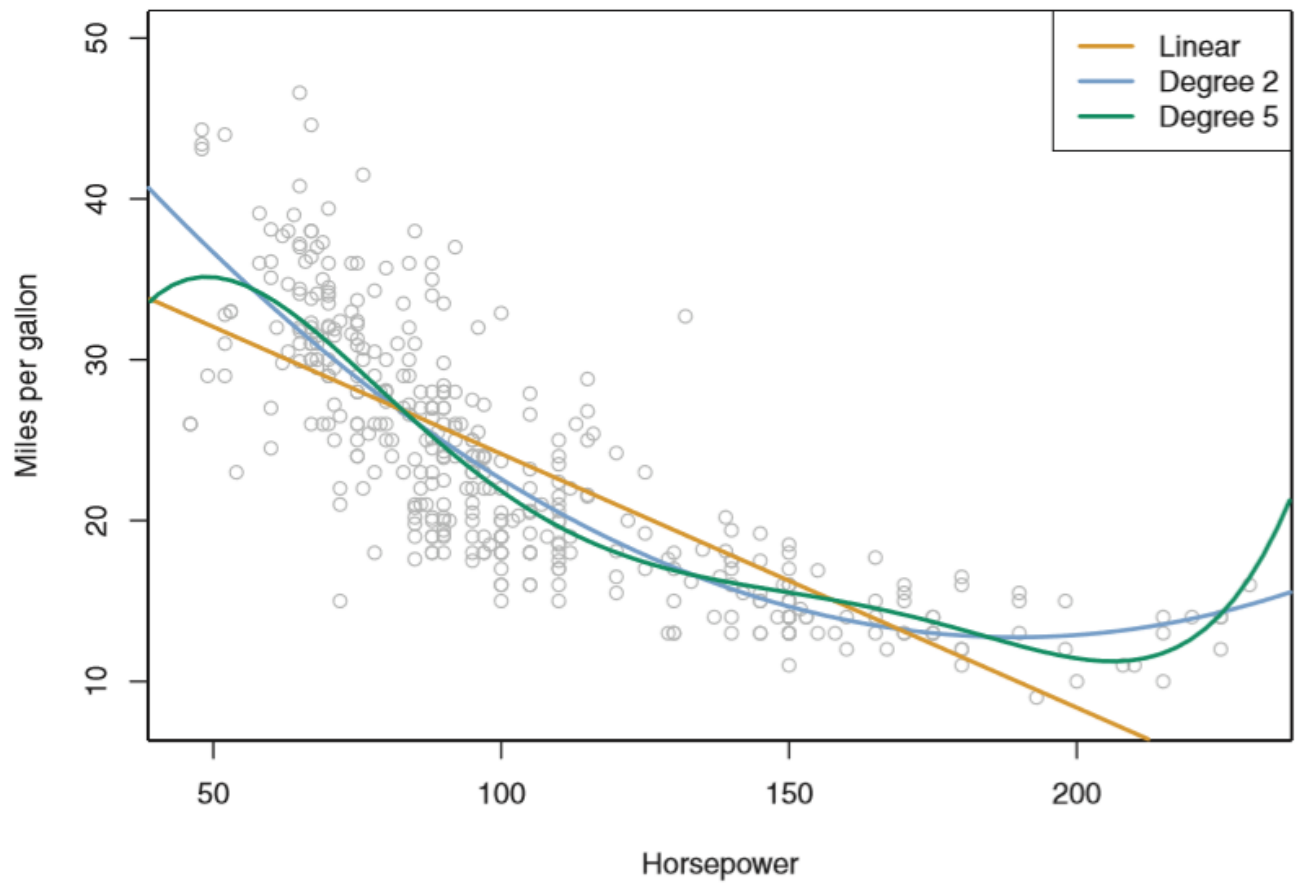
Cross-Validation: Test Set Approach

- We can vary the split point (50/50, 80/20, 90/10 are common divisions)
- It is important to do this step prior to ANY data cleaning, feature engineering, model selection



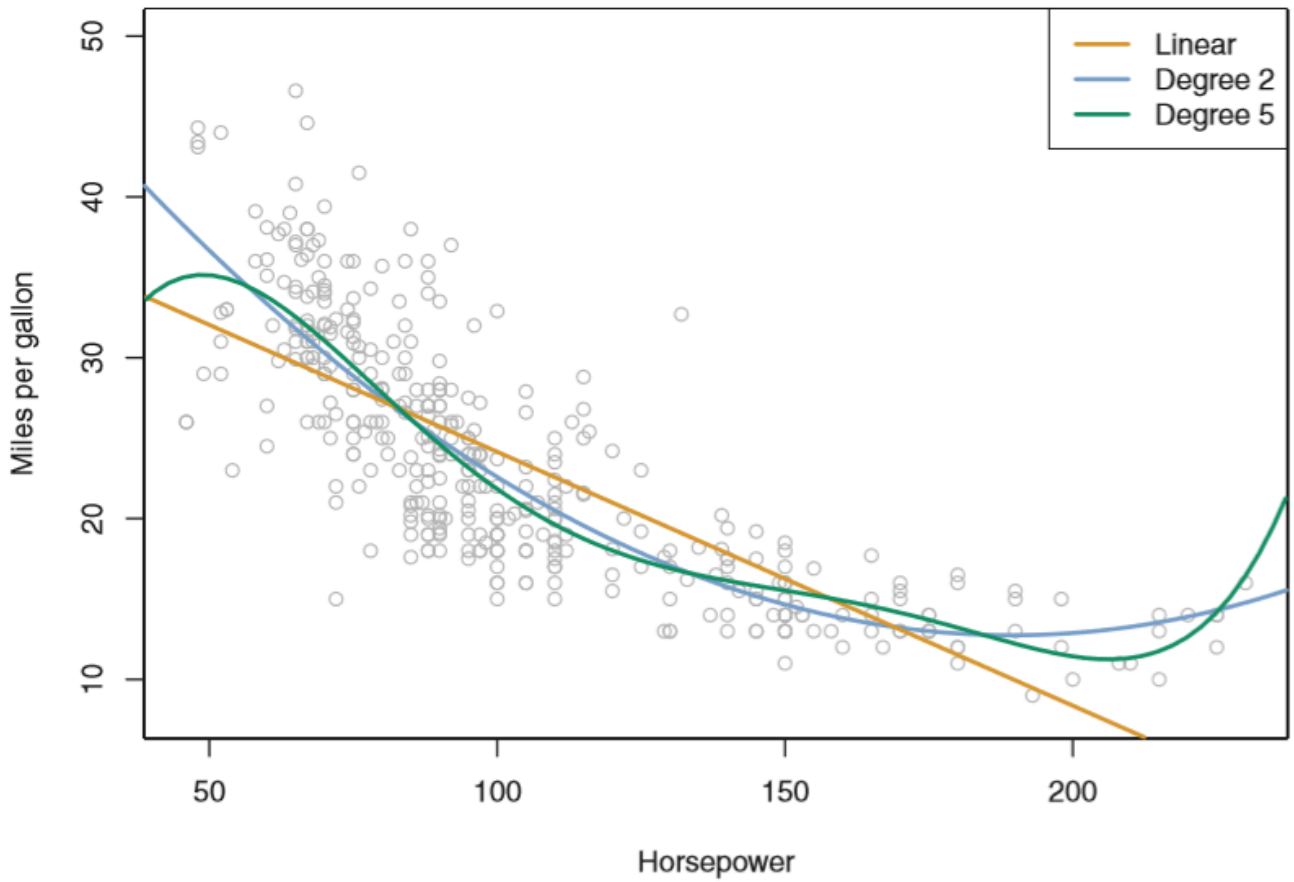
Cross-Validation: MPG from Horsepower

- Recall the model of miles per gallon from horsepower
- Best fit was $mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \epsilon$
- How should we determine what the right degree of polynomial is?



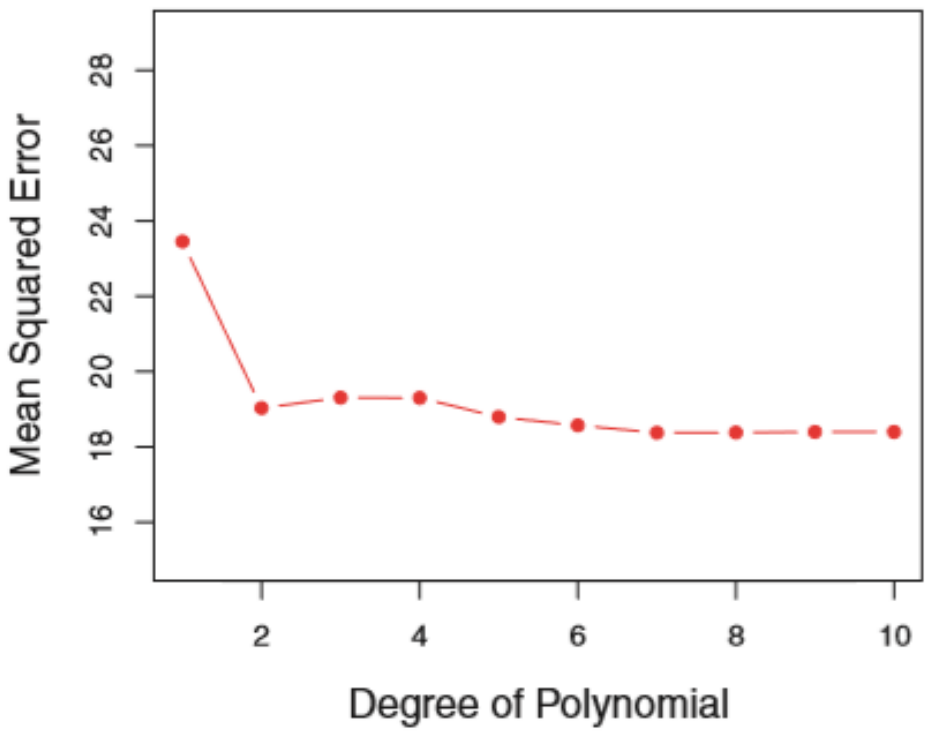
Cross-Validation: MPG from Horsepower

- Recall the model of miles per gallon from horsepower
- Best fit was $mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \epsilon$
- Can we determine that using cross-validation?



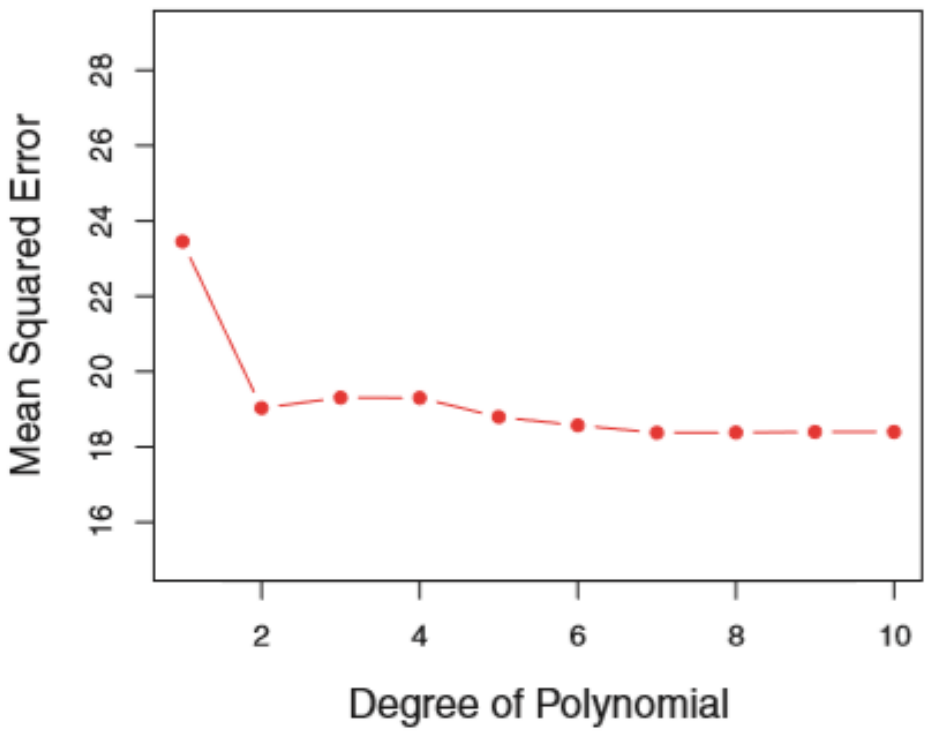
Cross-Validation: MPG from Horsepower

- Best fit was $mpg = \beta_0 + \beta_1 horsepower + \beta_2 horsepower^2 + \epsilon$
- We randomly split this data into a training set with 50% of the data and testing set with 50% of the data and plot the errors from the testing set when using different polynomials
- We see adding the cubic term actually increases the error, and find degree 2 is the best choice



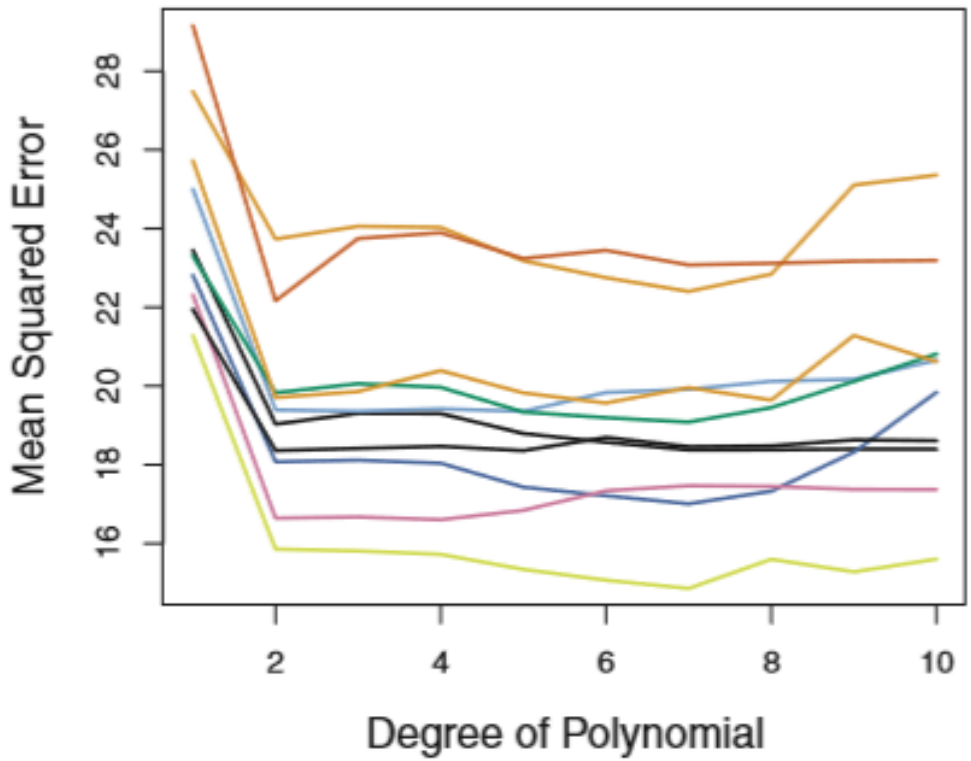
Cross-Validation: Train/Test Drawbacks

- We are only using a subset of the data - but we have discussed how more training data allows for more stable solutions
- The test set error may over estimate the final testing error
- Variability in the train/test split may result in inaccurate readings in noisier regions (high degree polynomials)
- What if we repeat this train/test split multiple times?



Cross-Validation: Train/Test Variability

- We repeat this process and see that we get different estimates
- On the average we now find that degree 2 polynomials fit the data best (as expected)
- Still, how many repetitions do we need to solve all of our drawback questions? Answer is unclear



Leave-One-Out Cross-Validation

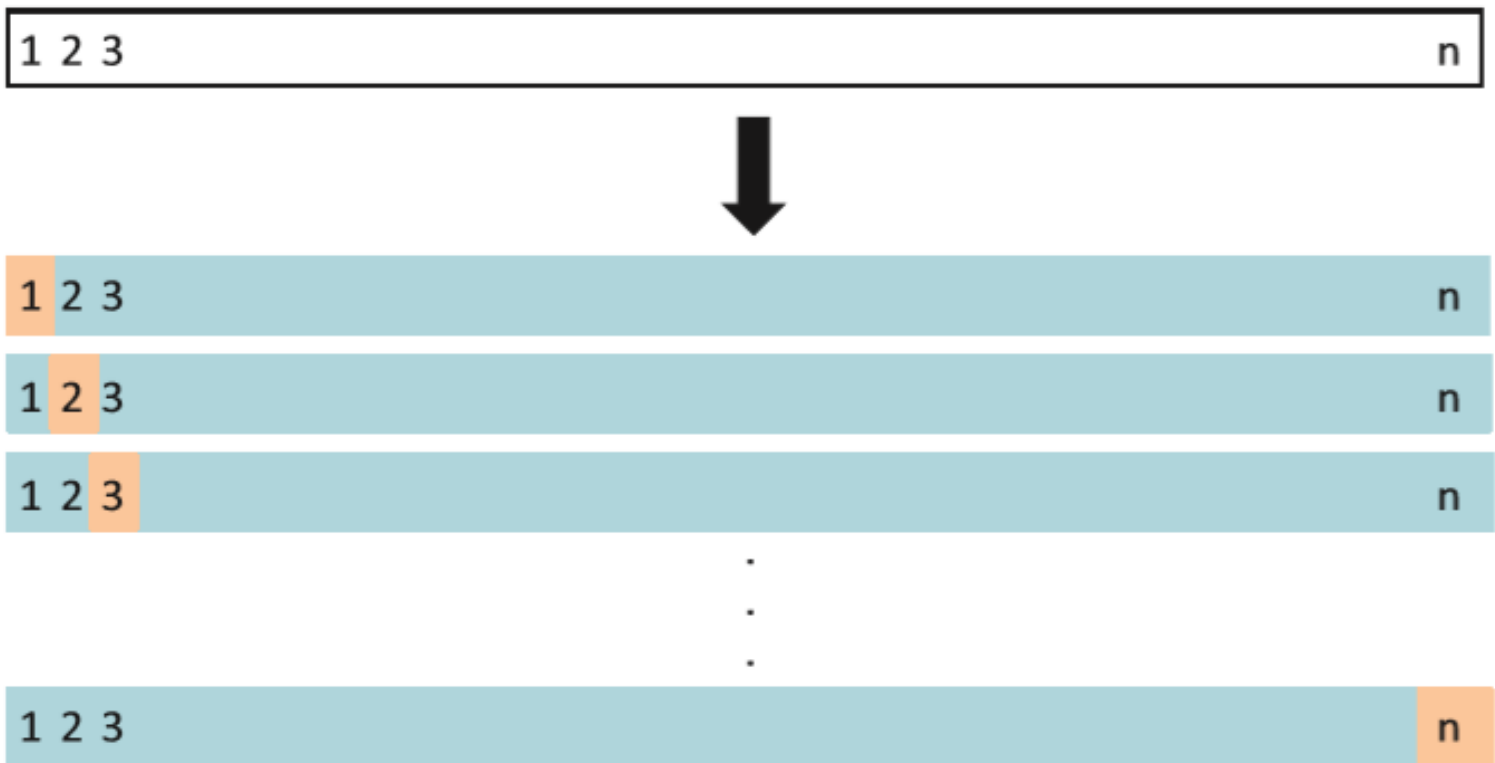
- We can solve the drawbacks by creating a fixed, repetitive procedure
- If our data samples $S = \{(x_i, y_i)\}_{i=1}^{m}$ we create a LOOCV by:

$$\text{Test samples: } S_{test} = \{(x_1, y_1)\}$$
$$\text{Train samples: } S_{train} = \{(x_i, y_i)\}_{i=2}^{m}$$

We then repeat this process n times, where each subject i serves as the test subject exactly once

Leave-One-Out Cross-Validation

- We can solve the drawbacks by creating a fixed, repetitive procedure
- If our data samples $S = \{(x_i, y_i)\}_{i=1}^{m}$ we create a LOOCV by:



LOOCV: Advantages

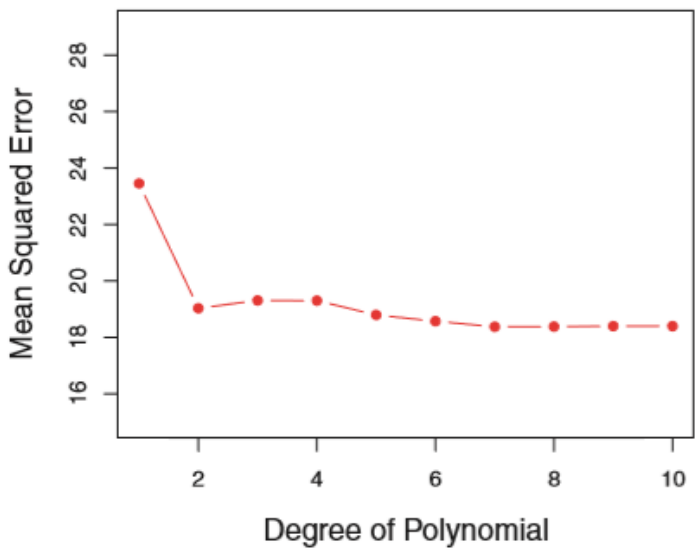
- In each iteration, our training set is using as much data as possible.
- We take the total error to be the average of our cross-validation error

$$MSE_{CV_m} = \frac{1}{m}\sum_{i=1}^{m} MSE_i$$

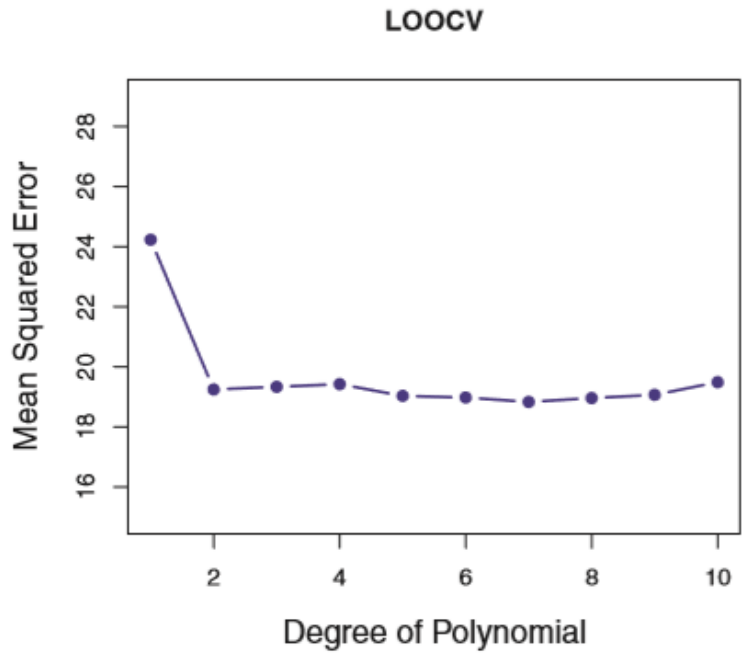
- This presents far less bias for each model by training on most of the data
- As a result, this doesn't overestimate test error as much as a single split
- Not as impacted by randomness of the train/test split approach
- Let's revisit our MPG/Horsepower example

LOOCV: Advantages

Train/Test Split



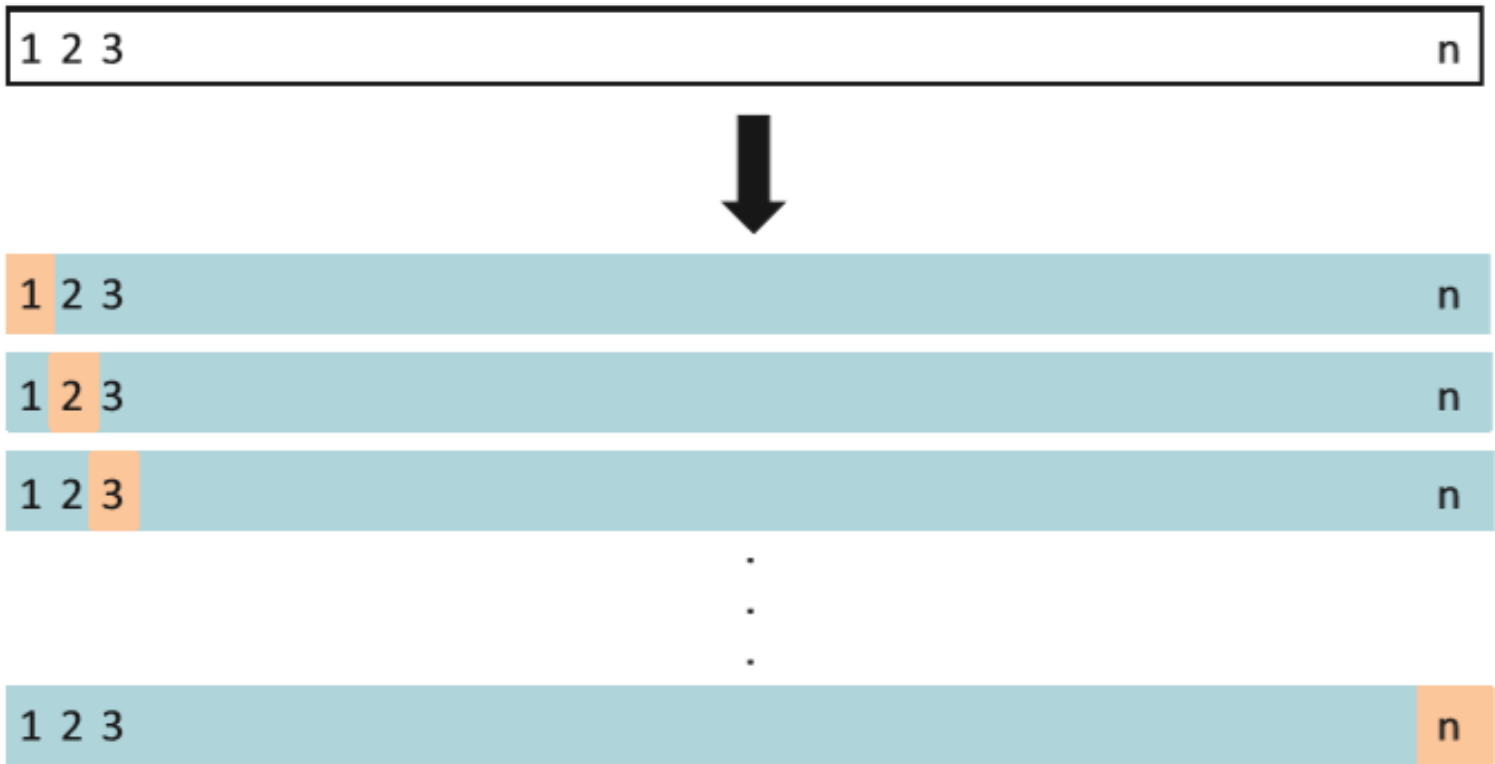
LOOCV



- LOOCV is more representative of the average from train/test splits - including rise in error as degree of polynomial overfits model

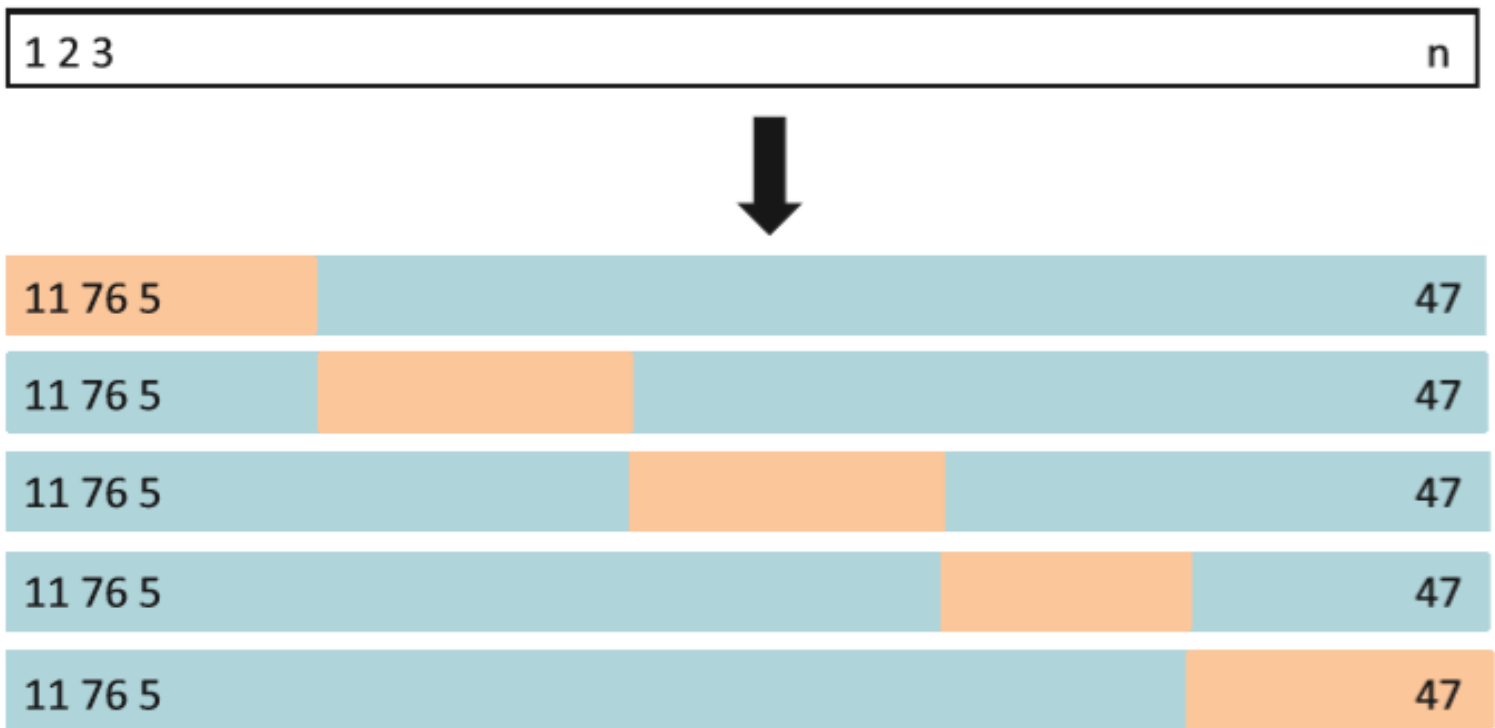
LOOCV: Drawbacks

- If m is large, (e.g. model of $m-1$ takes 4 days to train), what happens?
- LOOCV is computationally expensive
- Can we meet in the middle of LOOCV and train/test split?



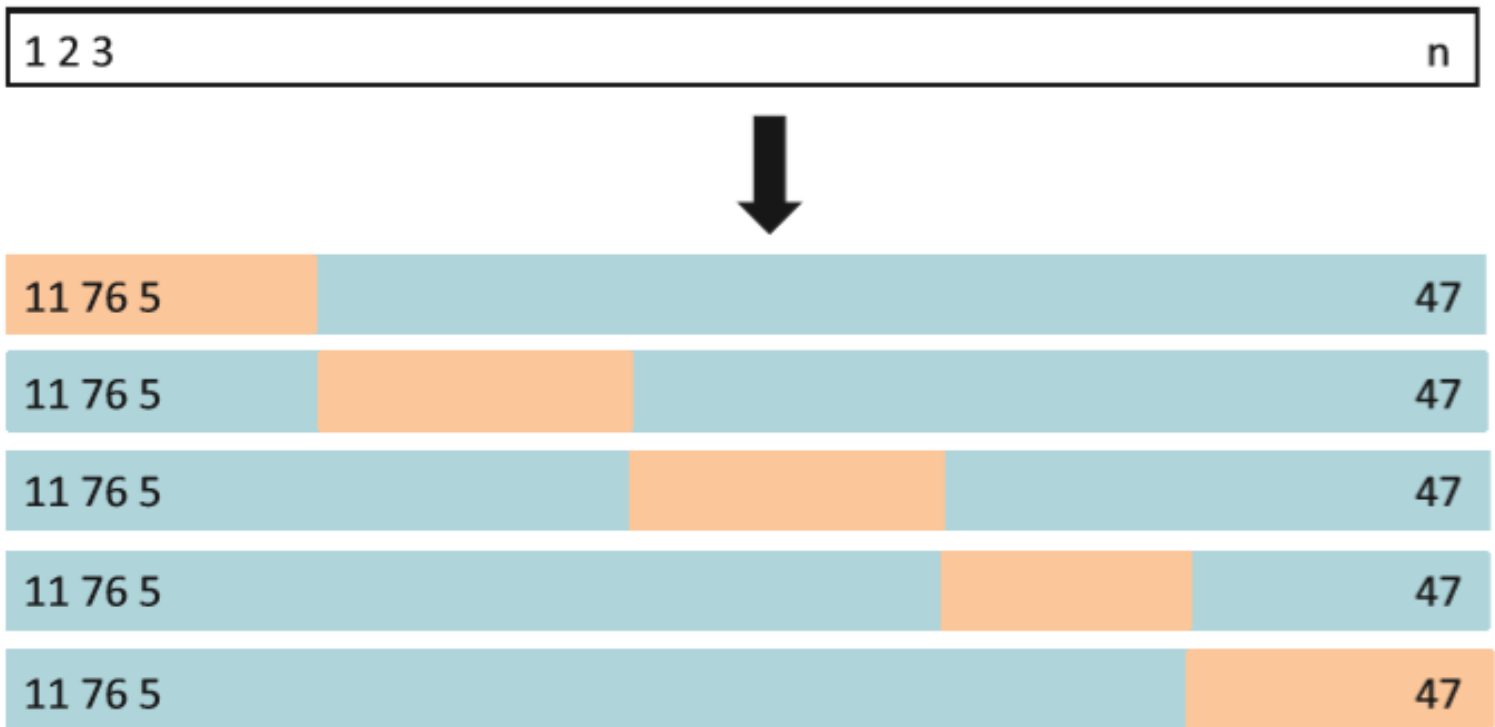
k-fold Cross-Validation

- Randomly assign subsets of data like train/test split but in k equal groups
- iterate which fold is the test set, like LOOCV
- Error is now $MSE_{CV_k} = \frac{1}{k} \sum_{i=1}^{k} MSE_i$



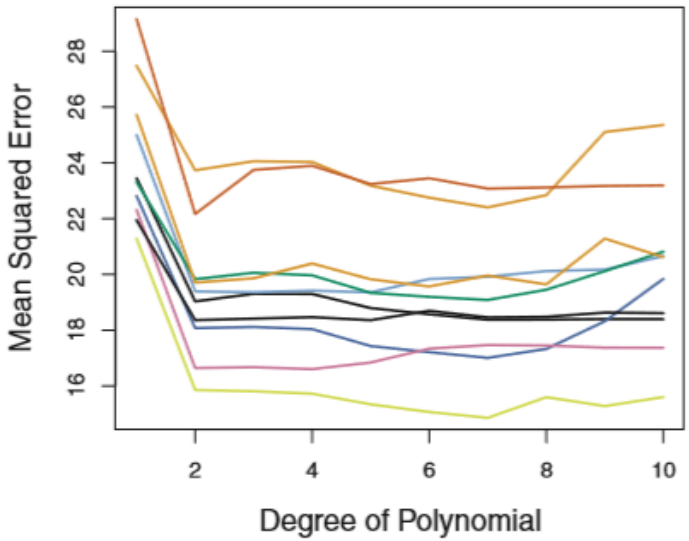
k-fold Cross-Validation

- What is the train/test split size if $k = 2$? (in terms of % of the total data?)
- What is the train/test split size if $k = 5$?
- What is the train/test split size if $k = 10$?
- Can we have other k? Yes but those are the most common.

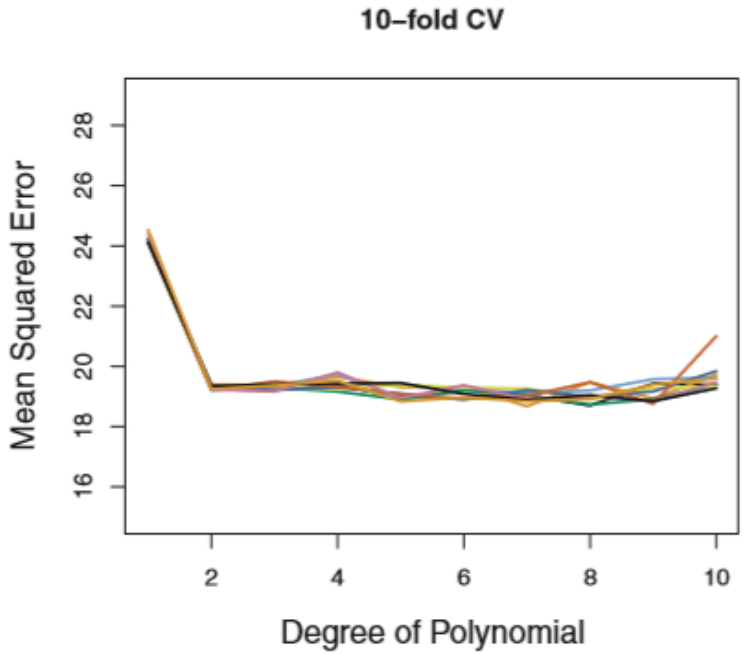


k-fold Cross-Validation: MPG Example

Train/Test Split



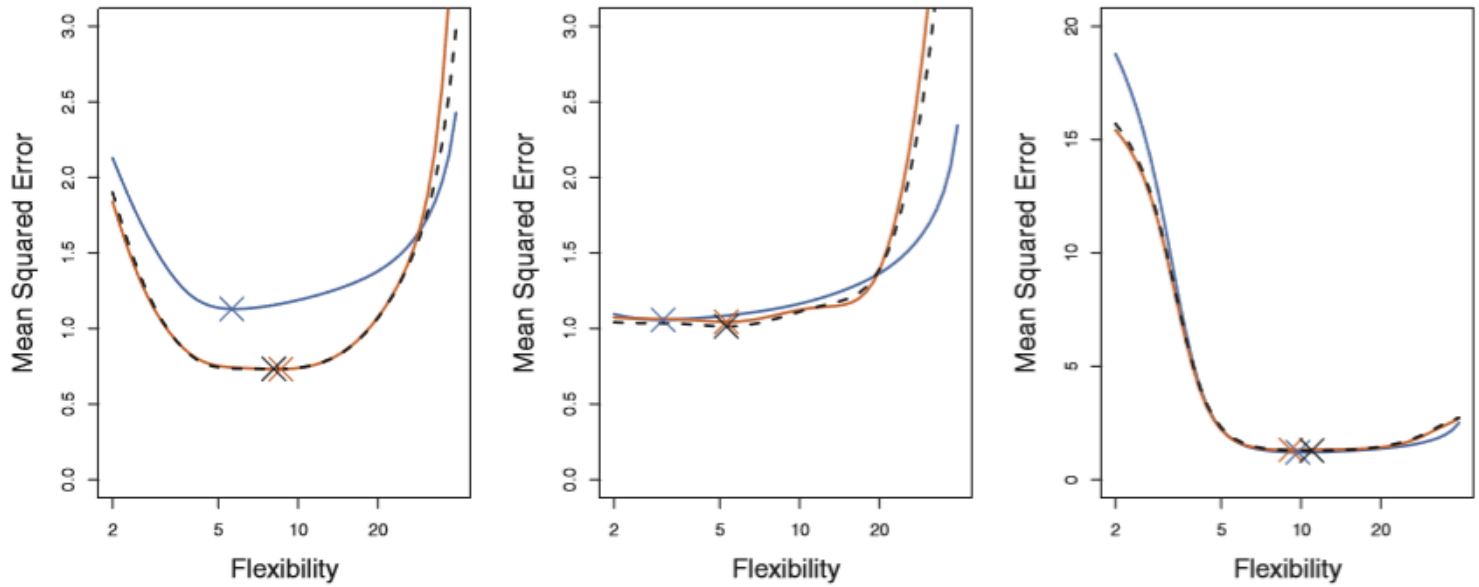
$k = 10$ fold Cross-Validation



- $10-$ *fold* cross-validation much more stable (training set size) but computationally more efficient than LOOCV

Bias-Variance Trade Off

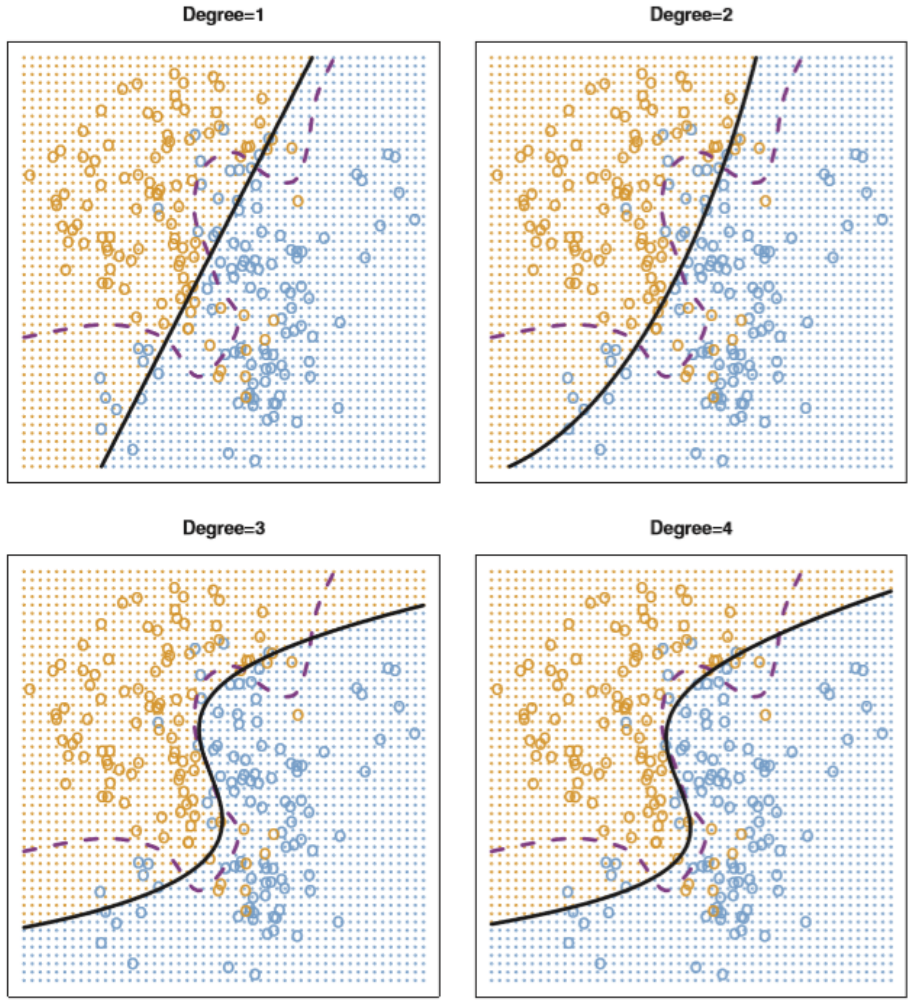
- Important to realize the flexibility, computational costs, and error estimates in each cross-validation type
- Trade offs, like everything else, no great way of knowing correct answer up front.
- Example of bias-variance trade-offs in different datasets with Blue being the true MSE, black being LOOCV, and orange being 10-fold CV



CV for Classification

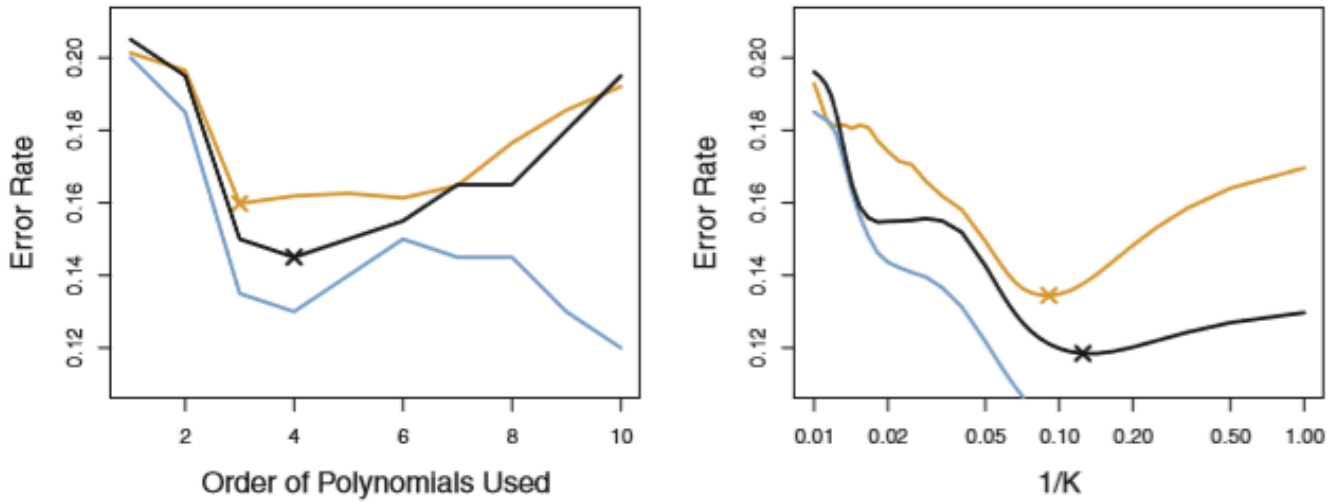
- $Err_{CV_m} = \frac{1}{m} \sum_{i=1}^{m} Err_i$ where $Err_i = \mathbb{I}(y_i \neq \hat{y}_i)$

CV for Classification



- Dashed purple line is is the Bayes decision boundary, solid black line is the logistic regression decision boundary

CV helps predict test error



- Polynomial regression (left) and KNN (right)
- Blue line is training error, orange line is testing error - and we once again try to address this discrepancy
- Black line shows the cross-validation error, which is representative of the testing error
- One additional option: Stratified k-fold Cross-Validation - k-fold where we keep the event rate the same (important when our dataset is imbalanced between 0 and 1 response samples)

Bootstrapping

- Bootstrapping is a statistical tool that let's us quantify uncertainty for methods
- Allows for estimates of coefficient fit.
- Illustrative example, you have two financial assets X and Y, want to know how much α to invest in X and $(1 - \alpha)$ to invest in Y

Bootstrapping

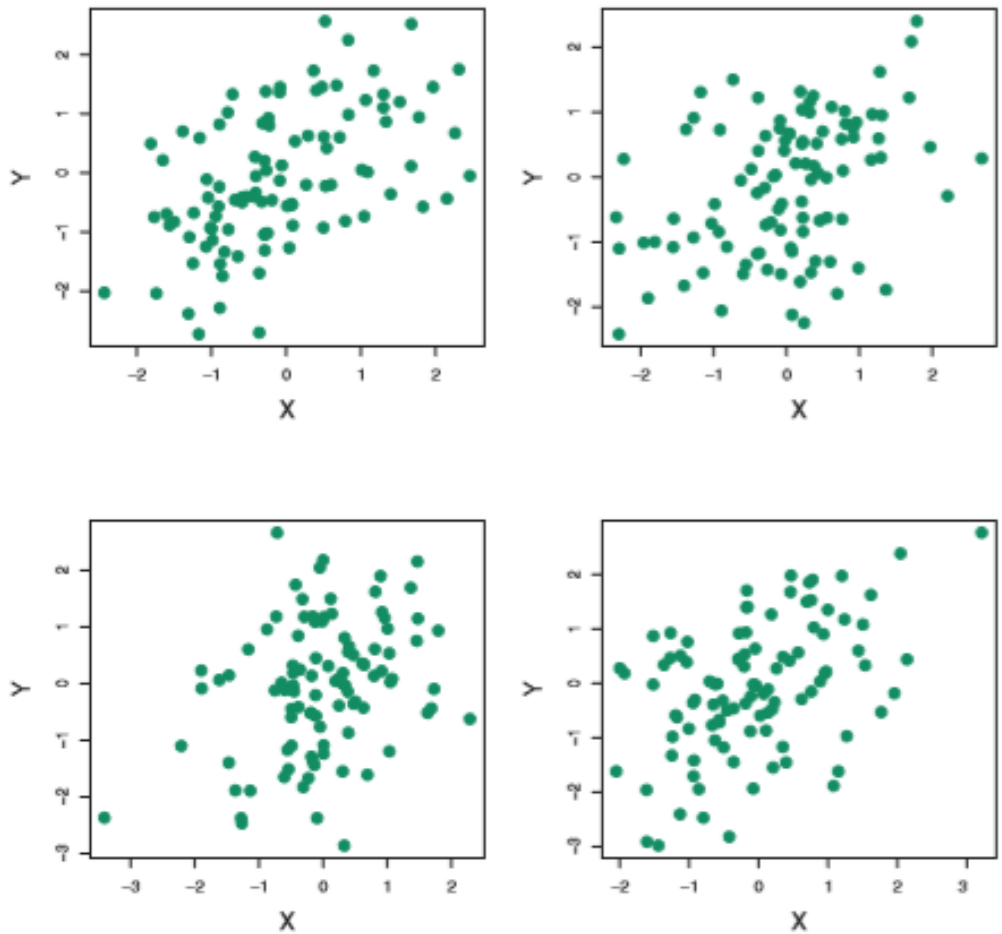
- Use boostrapping to help minimize $Var(\alpha X + (1-\alpha)Y)$

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

- Where $\sigma_X^2 = Var(X)$, $\sigma_Y^2 = Var(Y)$, and $\sigma_{XY} = Cov(X, Y)$ - which are all unknown on real data, so we need to compute estimates $\hat{\alpha}$

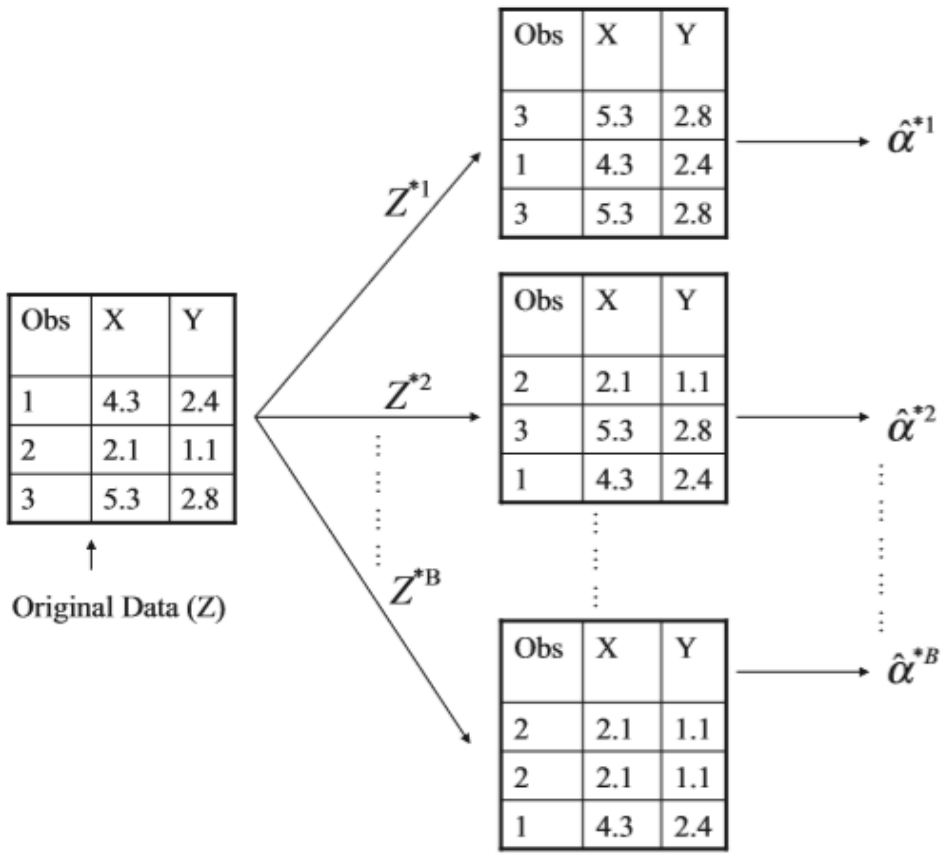
Bootstrapping: Estimating $\hat{\alpha}$

- We randomly sample X and Y in 100 runs. Then repeat that 4 times.
- In reality - can we really just Generate new data? Probably not!



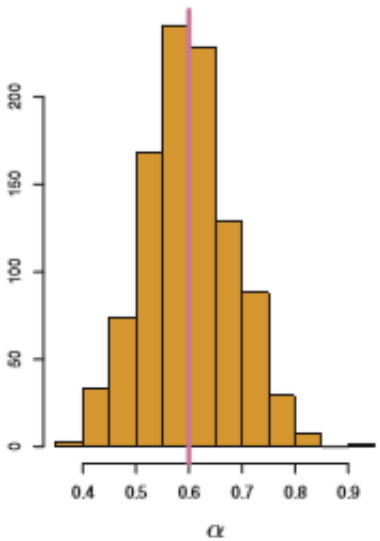
Bootstrapping: Estimating $\hat{\alpha}$

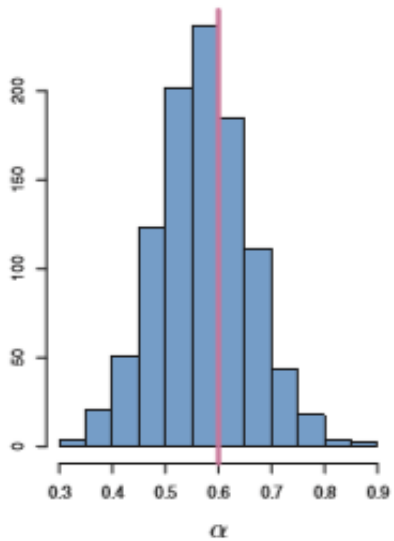
- In Bootstrapping, we take our training data and randomly sub-sample it!

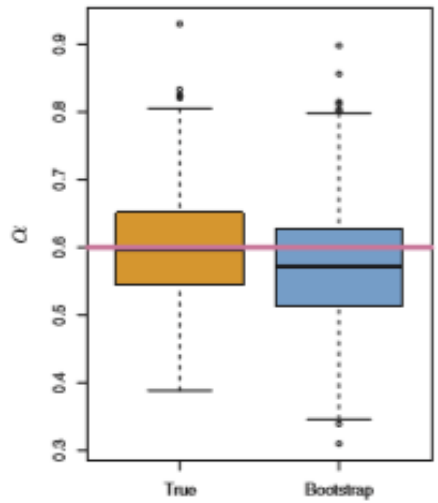


Bootstrapping: Estimating $\hat{\alpha}$

- In Bootstrapping, we take our training data and randomly sub-sample it!
- We repeat this process 1000 times and compare estimates against true population (assuming our training data is representative)







Bootstrapping: Estimating $\hat{\alpha}$

$$SE_B(\alpha) = \sqrt{\frac{1}{B}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'})^2}$$

- We see calculating the standard error from bootstrapping gives us a reasonable approximation of mean and standard deviation from true population