CSCE 633: Machine Learning

Lecture 13: Logistic Regression and Gradient Descent

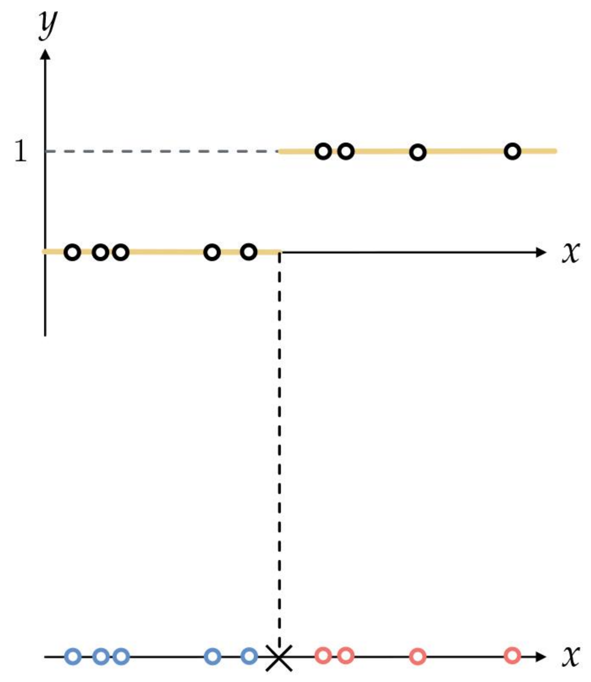
Texas A&M University

Bobak Mortazavi

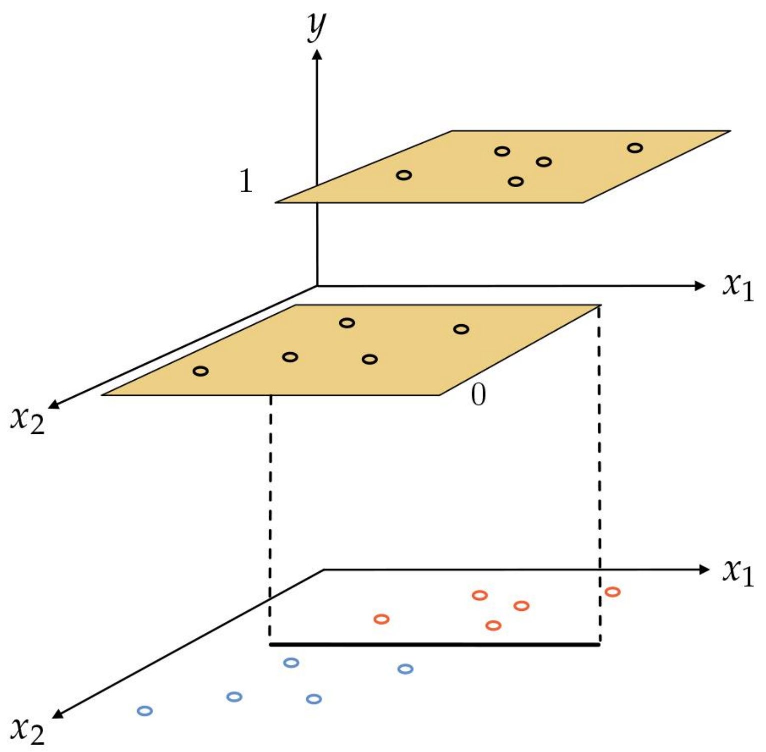
Goals

- Re-motivate Logistic Regression
- Understand why the shift from regression to classification results in needing Gradient Descent
- Learn how to use Cross-Validation

Classification – Step Functions

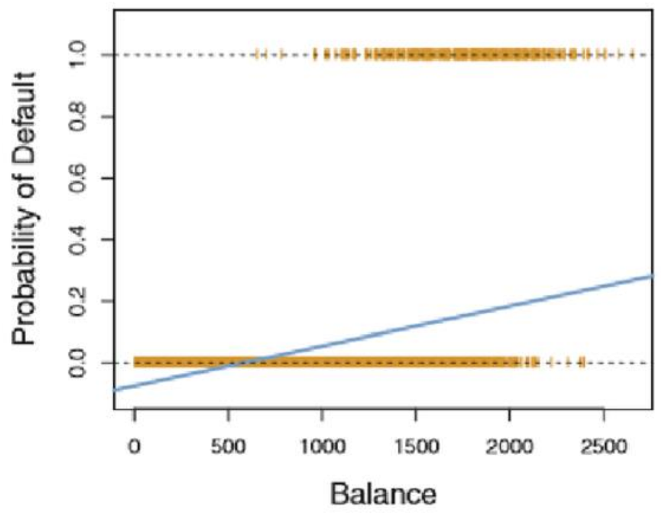


one-dimensional input:
decision boundary is a single point



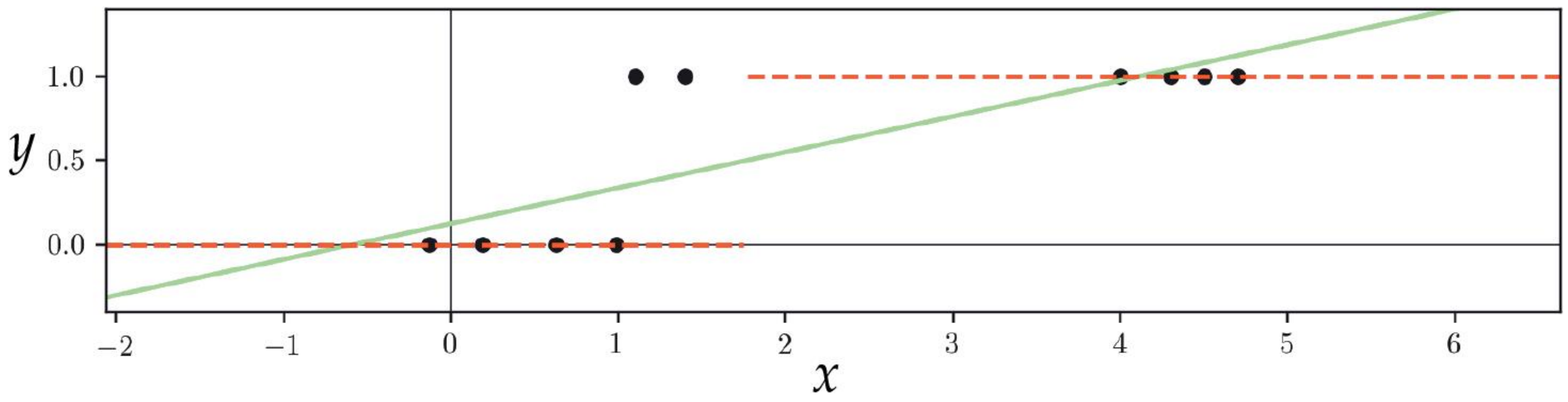
two-dimensional input:
decision boundary is a line

Linear Regression for Classification



If you set y > 0.5 as a decision rule – not clear it works
Hard to interpret – results are not contained in [0,1]

Step function from Linear Regression Boundary



Bernouli Distribution

- We can create a probability density function that represents a single experiment asking yes/no

$$Y \sim Bernouli\ (\theta), Y \in \{0,1\}$$

$$p(y|\theta) = \theta^{I(y=1)}(1-\theta)^{I(y=0)} = \begin{cases} \theta\ , y = 1 \\ 1-\theta, y = 0 \end{cases}$$

- Can think of this as a coin toss experiment and the likelihood of heads vs. tails

Logistic Regression

- Parametric classification method (not regression), which is sometimes referred to as a "generalization" of linear regression because
  - We still compute a linear combination of feature inputs, $\boldsymbol{x}^{\boldsymbol{T}}\ \boldsymbol{w}$ (sometimes written $\boldsymbol{w}^{\boldsymbol{T}}\ \boldsymbol{x}$)
  - However, instead of estimating the continuous output variable, we pass this into a function

$$\mu(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})}}$$

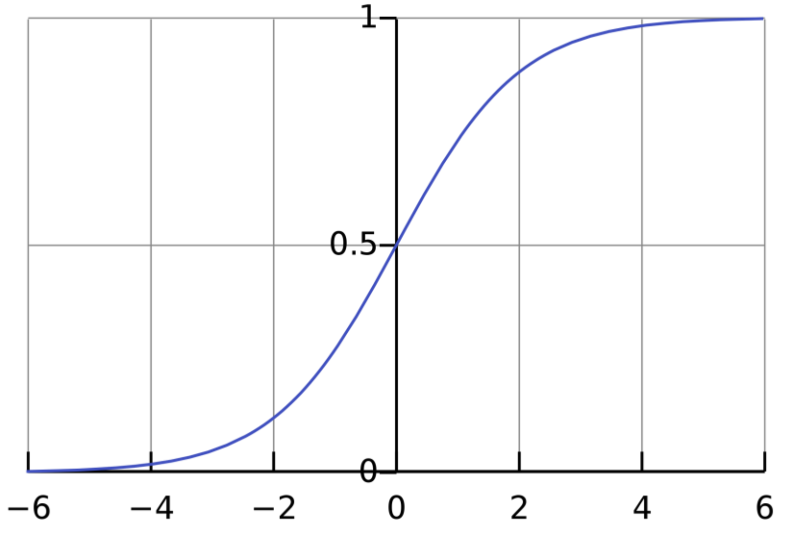
  - Where $0 \leq \mu(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}) \leq 1$, and where the Gaussian noise of linear regression is replaced by the Bernoulli Distribution so that

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = Ber(y|\mu(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}))$$

  - Therefore, the output belongs to a class 1 (y = 1) with probability $\mu(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})$, and class 0 (y = 0) with probability $1 - \mu(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x})$

Why use a Sigmoid Function?

$$\sigma(\boldsymbol{\eta}) = \frac{1}{1 + e^{-(\boldsymbol{\eta})}} = \frac{e^{\eta}}{1 + e^{\eta}}$$



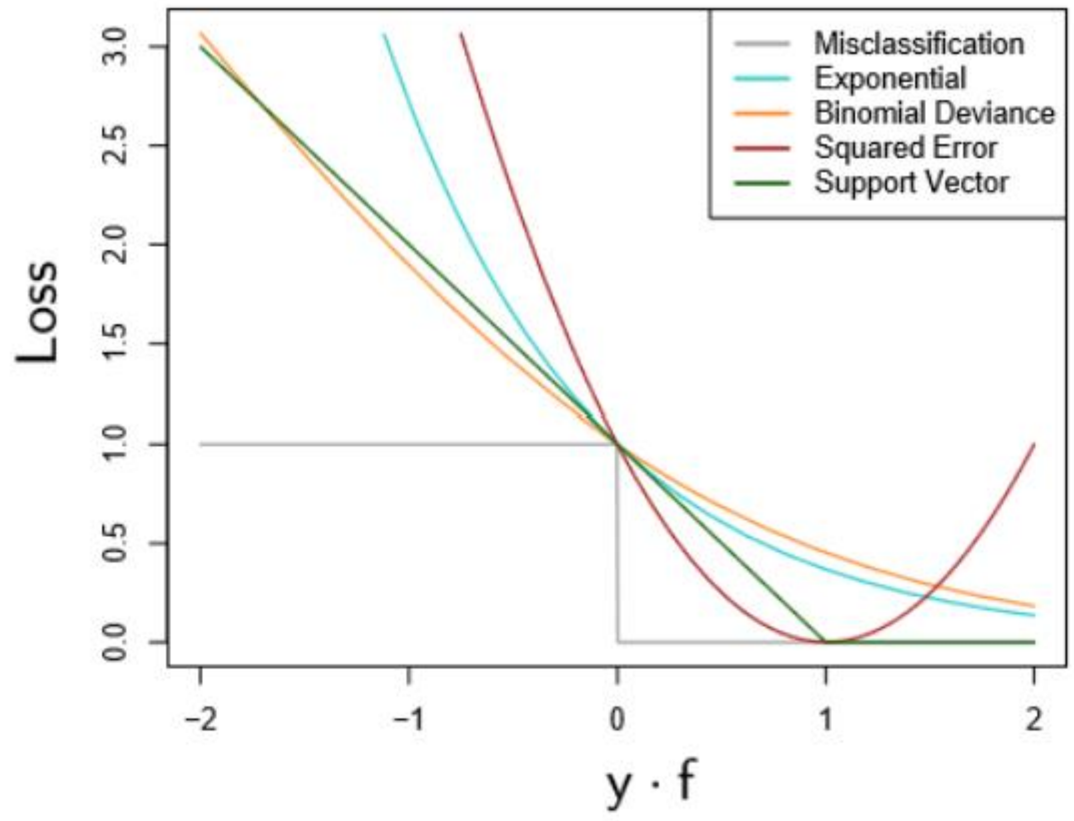
- Has very nice properties for classification
  - Bounded between 0 and 1 <- thus interpretable as a probability
  - Monotonically increasing <- thus can be used for classification rules

$\sigma(\boldsymbol{\eta}) > 0.5$, positive class (y = 1)
$\sigma(\boldsymbol{\eta}) \leq 0.5$, negative class (y = 0)

  - Nice computational properties for optimizing criterion function

Decision Boundaries (Margin Classification)

Types of Loss



Logistic Regression: Representation

- Setup classification problem for two classes
  - Input: $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{p}}$
  - Output: $y \in \{0, 1\}$
  - Training Data: $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - Model parameters: **w** (weights)
  - Model:

$$p(y\,|\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}\right), \sigma(\boldsymbol{\eta}) = \frac{1}{1 + e^{-(\eta)}} = \frac{e^{\eta}}{1 + e^{\eta}}$$

$$y = \begin{cases} 1, p(y\,|\boldsymbol{x}, \boldsymbol{w}) > 0.5 \\ 0, otherwise \end{cases}$$

Maximum Likelihood Estimation

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta; x)$$

- Likelihood is $\prod_{p:y_p=1} p(x_p) \prod_{p':y_{p'}=0} (1 - p(x_{p'}))$
- With one dimension this means optimizing intercept w_0 and slope w_1
- So maximizing the parameters means finding the optimal w that maximize the product

Multiple Logistic Regression

$$\log\left(\frac{p(x)}{1-p(x)}\right) = w_0 + w_1 x_1 + \cdots + w_p x_p$$

- $p(y|x,w) = Ber(y|\sigma(w^T x))$, with a linear decision boundary at p(x) > 0.5
- We can then calculate the negative of the log of the likelihood (negative log likelihood)

$$NLL(w) = -\sum_{i=1}^{n}\left(y_i \log \sigma(w^T x_i) + (1 - y_i)\log\big(1 - \sigma(w^T x_i)\big)\right)$$

Multiple Logistic Regression

$$\log\left(\frac{p(x)}{1-p(x)}\right) = w_0 + w_1 x_1 + \cdots + w_p x_p$$

- $p(y|x,w) = Ber(y|\sigma(w^T x))$, with a linear decision boundary at p(x) > 0.5
- We can then calculate the negative of the log of the likelihood (negative log likelihood)

$$NLL(w) = -\sum_{i=1}^{n}\left(y_i \log \sigma(w^T x_i) + (1 - y_i)\log\left(1 - \sigma(w^T x_i)\right)\right)$$

- Now suppose we recast $\tilde{y} = \{-1, 1\}$, then

$$NLL(w) = -\sum_{i=1}^{n}\left(\log\left(1 + e^{-\tilde{y}_i w^T x_i}\right)\right)$$

- Which has no closed form solution

Logistic Regression: Evaluation

- Data likelihood (1 training sample)

$$p(y|x) = \begin{cases} \sigma(w^T x) & y = 1 \\ 1 - \sigma(w^T x) & otherwise \end{cases} = \sigma(w^T x)^y (1 - \sigma(w^T x))^{1-y}$$

- Data likelihood (all training samples)

$$L(D, w) = \prod_{i=1}^{n} p(y_i|x_i, w) = \prod_{i=1}^{n} \sigma(w^T x_i)^{y_i} (1 - \sigma(w^T x_i))^{1-y_i}$$

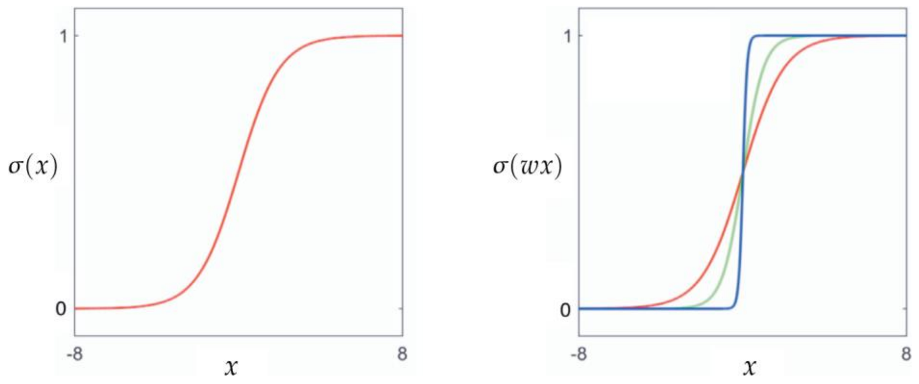
- Log Likelihood

$$l(D, w) = \sum_{i=1}^{n} \left(y_i \log \sigma(w^T x_i) + (1 - y_i) \log\left(1 - \sigma(w^T x_i)\right)\right)$$

- Negative Log Likelihood (the cross-entropy error)

$$nll(w) = -\sum_{i=1}^{n} \left(y_i \log \sigma(w^T x_i) + (1 - y_i) \log\left(1 - \sigma(w^T x_i)\right)\right)$$

Training Logistic Regression



Where training in linear regression shifted line (slope/intercept)
training in logistic regression alters the “step”-like nature of the sigmoid

Training Logistic Regression with Gradient Descent

- Gradient Descent Methods
- How to optimize the coefficients

Logistic Regression: Optimization

- The cross-entropy error

$$\mathcal{E}(w) = -\sum_{i=1}^{n}\left(y_i \log \sigma(w^T x_i) + (1 - y_i) \log\left(1 - \sigma(w^T x_i)\right)\right)$$

- Must find weights w of logistic regression that minimize the error
- Because there is no closed form solution, unlike linear regression, we need gradient descent

$$\nabla\mathcal{E}(w) = \sum_{i=1}^{n}(\sigma(w^T x_i) - y_i)\ x_i$$

- Is this convex?

Cross Entropy Error: Convex

$$\nabla \mathcal{E}(w) = \frac{\partial \mathcal{E}(w)}{\partial w} = \sum_{i=1}^{n} (\sigma(w^T x_i) - \ y_i) \ x_i$$

$$H = \frac{\partial^2 \mathcal{E}(w)}{\partial w w^T} = \sum_{i=1}^{n} \sigma(w^T x_i) \left(1 \ - \sigma(w^T x_i)\right)\left(x_i x_i^T\right)$$

For all **v,** substituting $\mu = \sigma(w^T x_i)\left(1 \ - \sigma(w^T x_i)\right) \ \geq 0$

$$\boldsymbol{v}^T H \boldsymbol{v} = \mu \boldsymbol{v}^T \left(\sum_{i=1}^{n} x_i x_i^T\right) \boldsymbol{v} = \mu \sum_{i=1}^{n} \left(x_i^T \boldsymbol{v}\right)^T \left(x_i^T \boldsymbol{v}\right) = \mu \sum_{i=1}^{n} \left\| x_i^T \boldsymbol{v} \right\|_2^2 \ \geq 0$$

Gradient Descent in Logistic Regression

- The cross-entropy error

$$\mathcal{E}(w) = -\sum_{i=1}^{n}\left(y_i \log \sigma(w^T x_i) + (1 - y_i) \log\left(1 - \sigma(w^T x_i)\right)\right)$$

- Gradient Descent Optimization Expression

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha(k)\nabla\mathcal{E}(w)$$

$$\nabla\mathcal{E}(w) = \sum_{i=1}^{n}(\sigma(w^T x_i) - y_i)\; x_i$$

Batch Gradient Descent

```
1. Initialize $w, \epsilon, \alpha, k = 0$
2. While $\|\nabla \mathcal{E}(w)\|_2 > \epsilon$
    2a. k = k + 1
    2b. $w = w - \alpha(k)(\sum_{i=1}^{n}(\sigma(w^T x_i) - y_i)\ x_i)$
```

Stochastic Gradient Descent

Update weights one sample at a time

1. Initialize $w, \epsilon, \alpha, k = 0$
2. Loop until convergence

    2a. k = k + 1

    2b. Randomly choose a single sample (x_i, y_i)

    2c. Compute its contribution to the gradient $g_i = (w^T x_i - y_i) x_i$

    2d. Update weights $w = w - \alpha(k) g_i$

Mini Batch Gradient Descent

Update weights with a subset of samples at a time

1. Initialize $w, \epsilon, \alpha, k = 0$
2. Loop until convergence

    2a. k = k + 1

    2b. Randomly choose a set of samples (x_i, y_i)

    2c. Compute their contribution to the gradient $g_i = (w^T x_i - y_i)x_i$

    2d. Update weights $w = w - \alpha(k) \sum_{i=1}^{M} g_i$

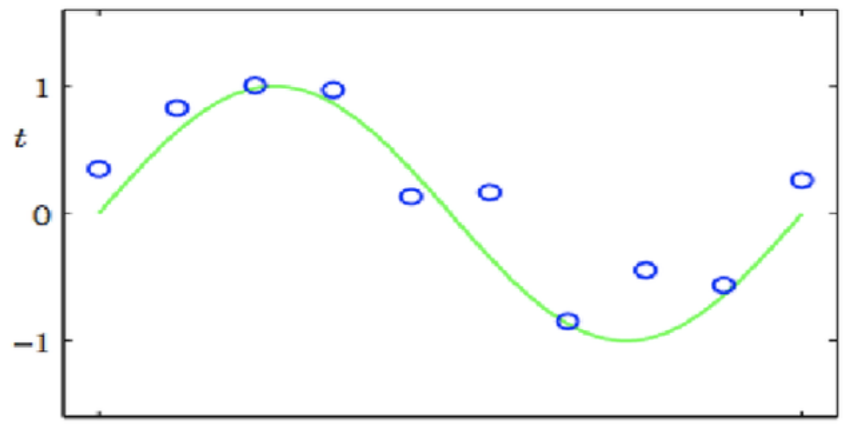
This is a good compromise between batch gradient descent (complete gradient) and stochastic gradient descent (gradient of one sample at a time)

Common mini-batch is a set M between 50 and 250 samples

This is typically used for training neural networks

What if data does not fit a line?

Example: Samples of a sine function



We can use a non-linear basis function

$$\phi(x): x \in \mathbb{R}^p \rightarrow z \in \mathbb{R}^M$$

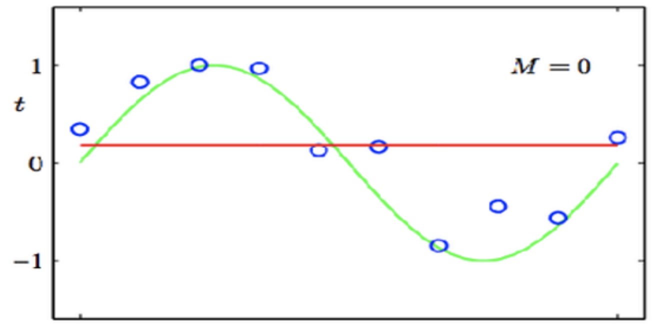
So that $y = w^T z = w^T \phi(x)$

More non-linear basis functions

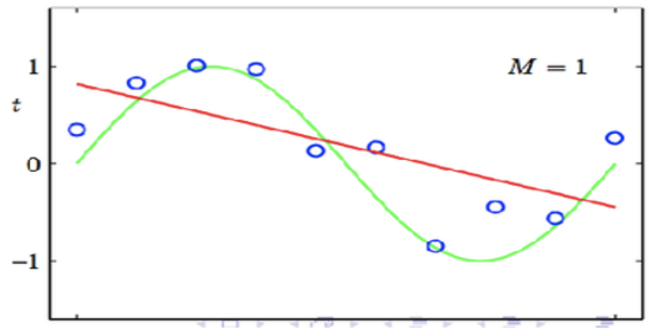
Example: Samples from a sine function

Polynomial basis function $\phi(\mathbf{x}) = [1\ x\ \ldots\ x^M]^T$

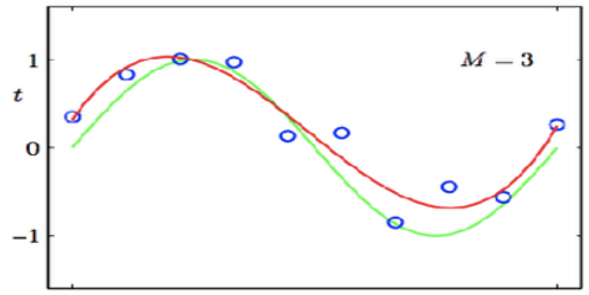
$M = 0$ underfitting



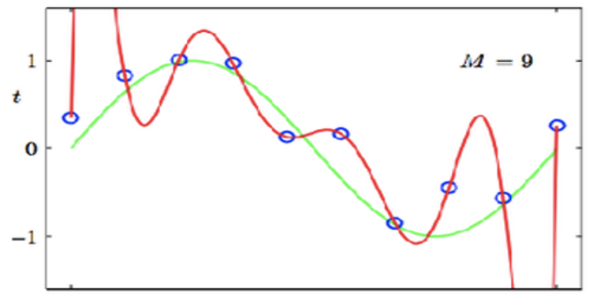
$M = 1$ underfitting



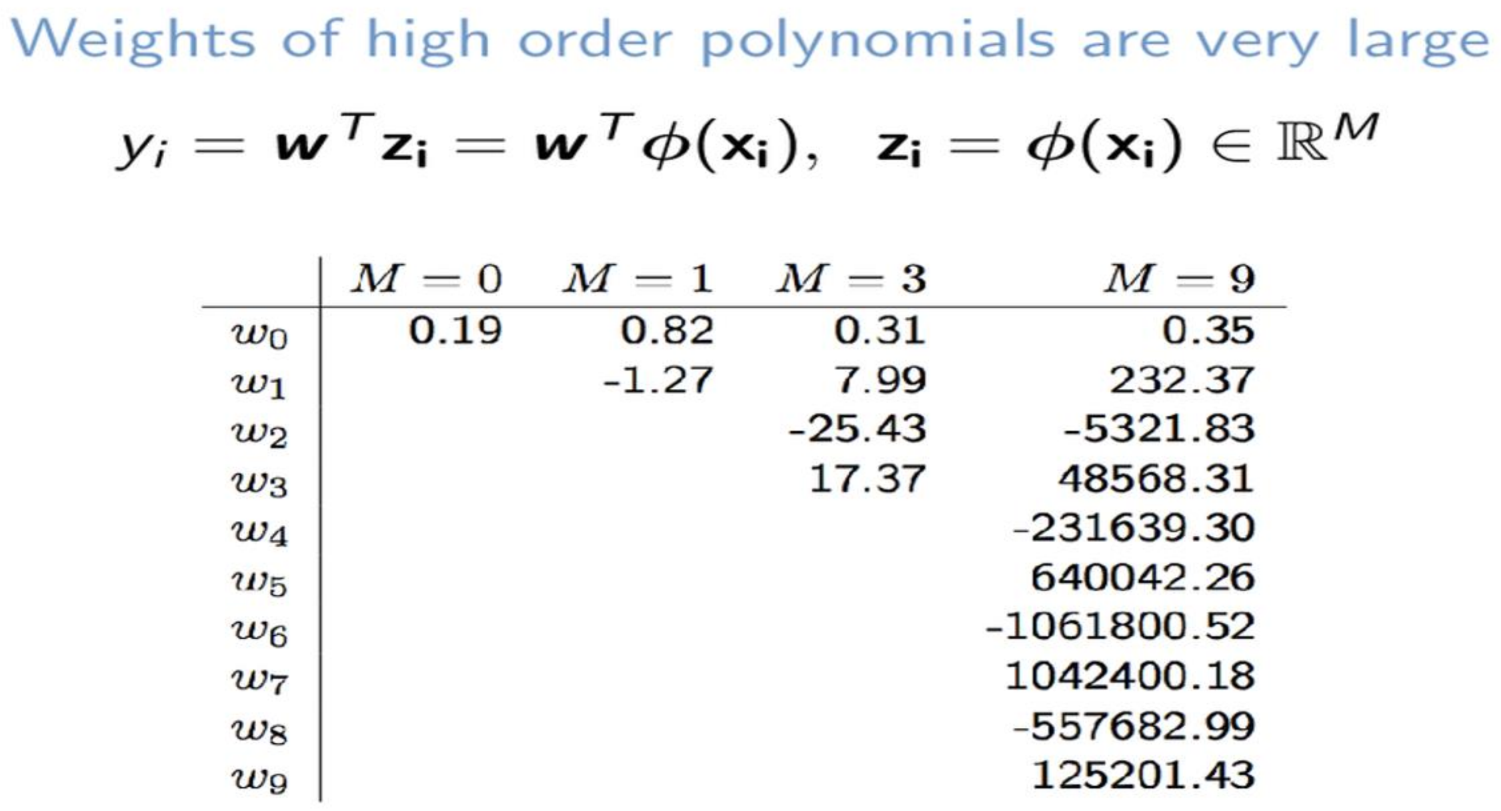
$M = 3$



$M = 9$ overfitting



But higher-order non-linearity tends to overfit



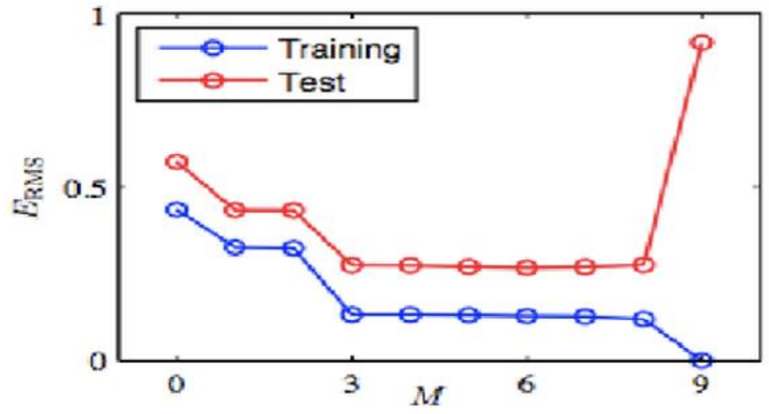
Weights of high order polynomials are very large

$$y_i = \mathbf{w}^T \mathbf{z_i} = \mathbf{w}^T \phi(\mathbf{x_i}), \quad \mathbf{z_i} = \phi(\mathbf{x_i}) \in \mathbb{R}^M$$

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Overfitting

- The risk of using highly flexible (complicated) models without enough data is that the model overfits
- Leads to poor generalization
- How do you detect overfitting?
  - Can plot model complexity vs. the objective function
  - As complexity improves, will see training improves while testing improves then deteriorates



- How do you prevent this?
  - Feature selection, more data, or regularization

Linear Regression: Summarization

- Representation

$$y = w^T x$$

- Evaluation

$$\min_w RSS(w) = \sum_{\mathrm{i}=1}^{N} (y_i - w^T x_i)^2$$

- Analytic Optimization

$$\boldsymbol{w}^* = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

- Approximate Optimization

via gradient descent

- In linear regression the maximum likelihood estimation = the ordinary least squares solution

Logistic Regression: Summarization

- Generalization of linear regression

$$\sigma(w^T x)$$

is interpretable as a probability

- Evaluation through cross-entropy error

$$\mathcal{E}(w) = -\sum_{i=1}^{n}\left(y_i \log \sigma(w^T x_i) + (1 - y_i) \log\left(1 - \sigma(w^T x_i)\right)\right)$$

- Optimization through gradient descent

Practical Use of Logistic Regression

- Understanding re-sampling
- Measures of Accuracy when repeating modeling
- Python/Discussion!

Resampling Methods. Why?

- Resampling is a method by which we re-draw subsets of training data to see if model fit changes
- This allows us to see how stable our model is by how much the coefficient weights change
- We want to reduce model error rates
- How does one guarantee performance on a test set when only evaluating a training set?

Resampling Methods

Cross-validation

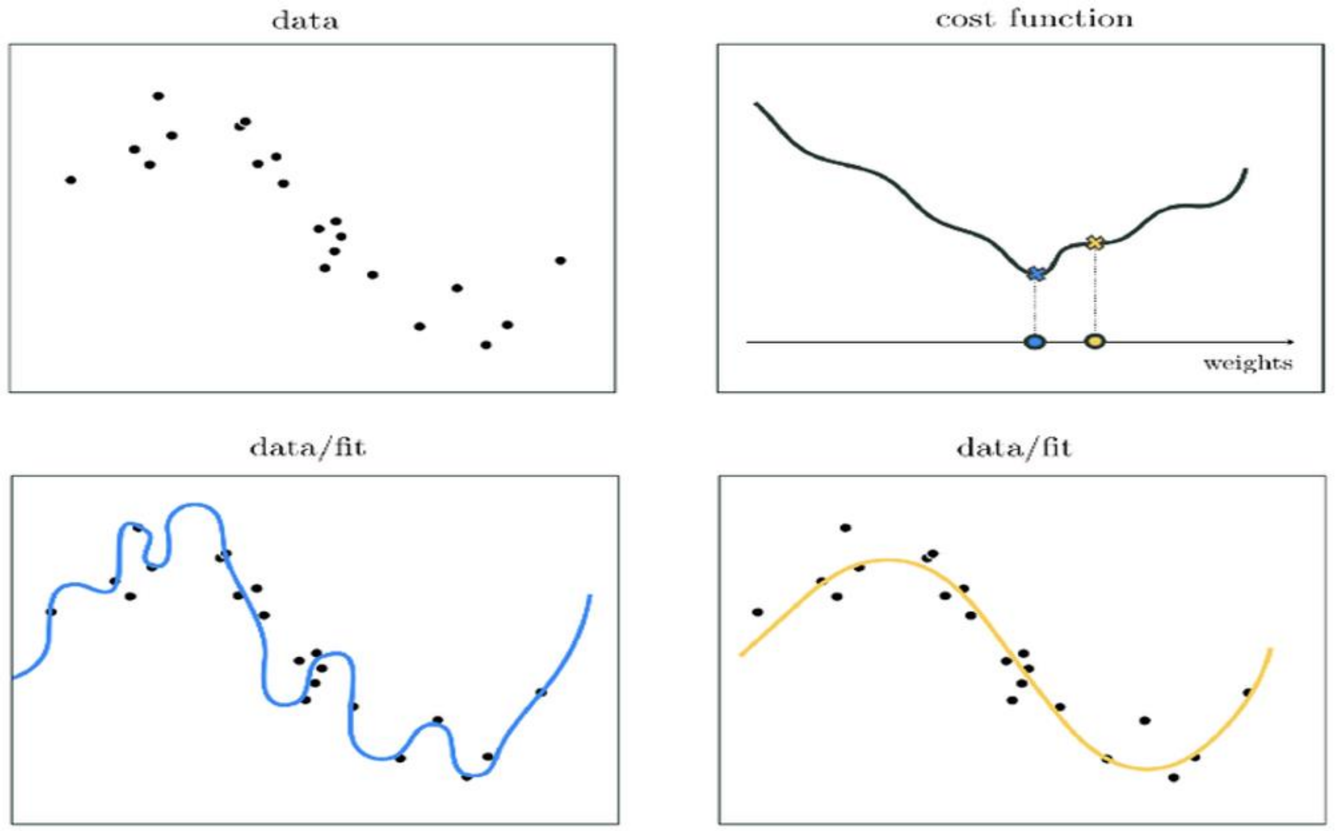
- Helps estimate measures of performance
- Allows for a choice in levels of model flexibility
- Used for hyperparameter tuning
- (You tend to see this more in computer science work)

Bootstrapping

- Also helps give estimates of model performance
- Measures of accuracy surrounding parameter estimates
- Repeated trials help simulate different event rates
- (you tend to see this more in biostatistics work)

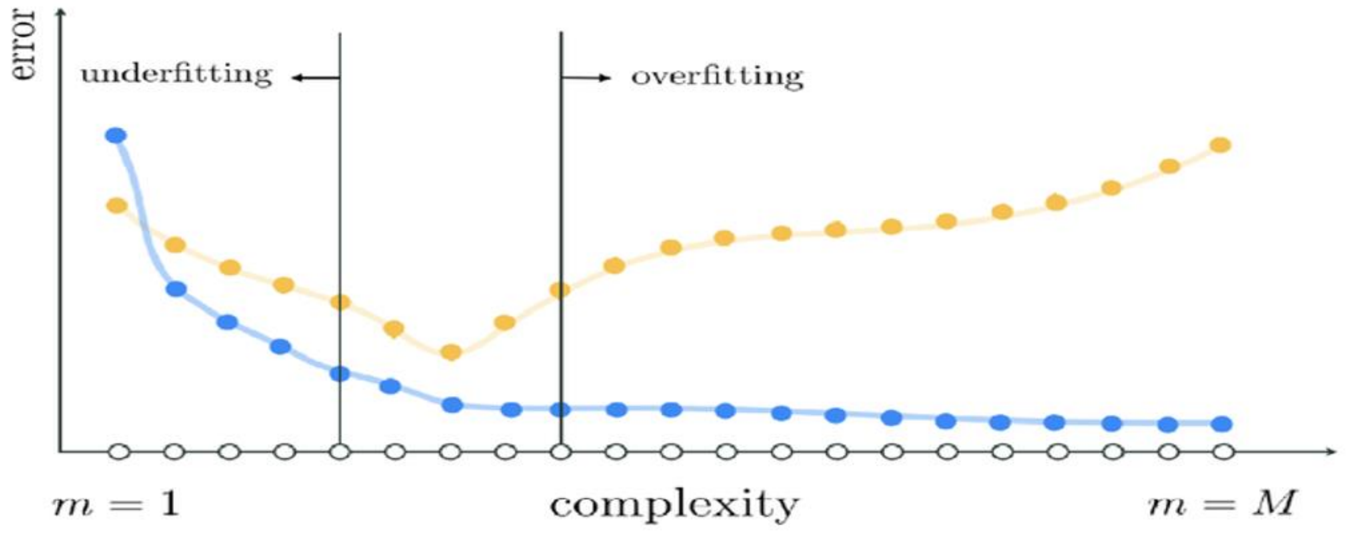
Overfitting

- Consider fitting a polynomial model to data, how does training error change with increasing degree?



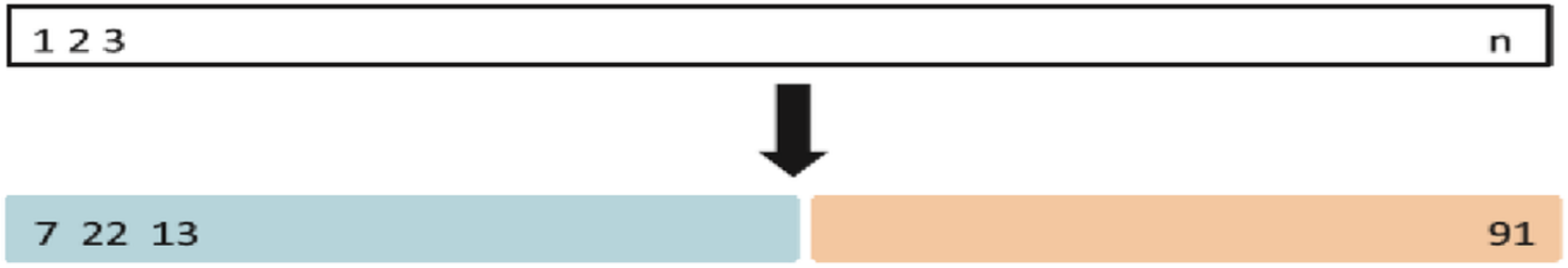
Overfitting

- As a model gets more complex, we become prone to reducing training error at the expense of testing error
- We also see sensitivity increase to specific inclusion of variables



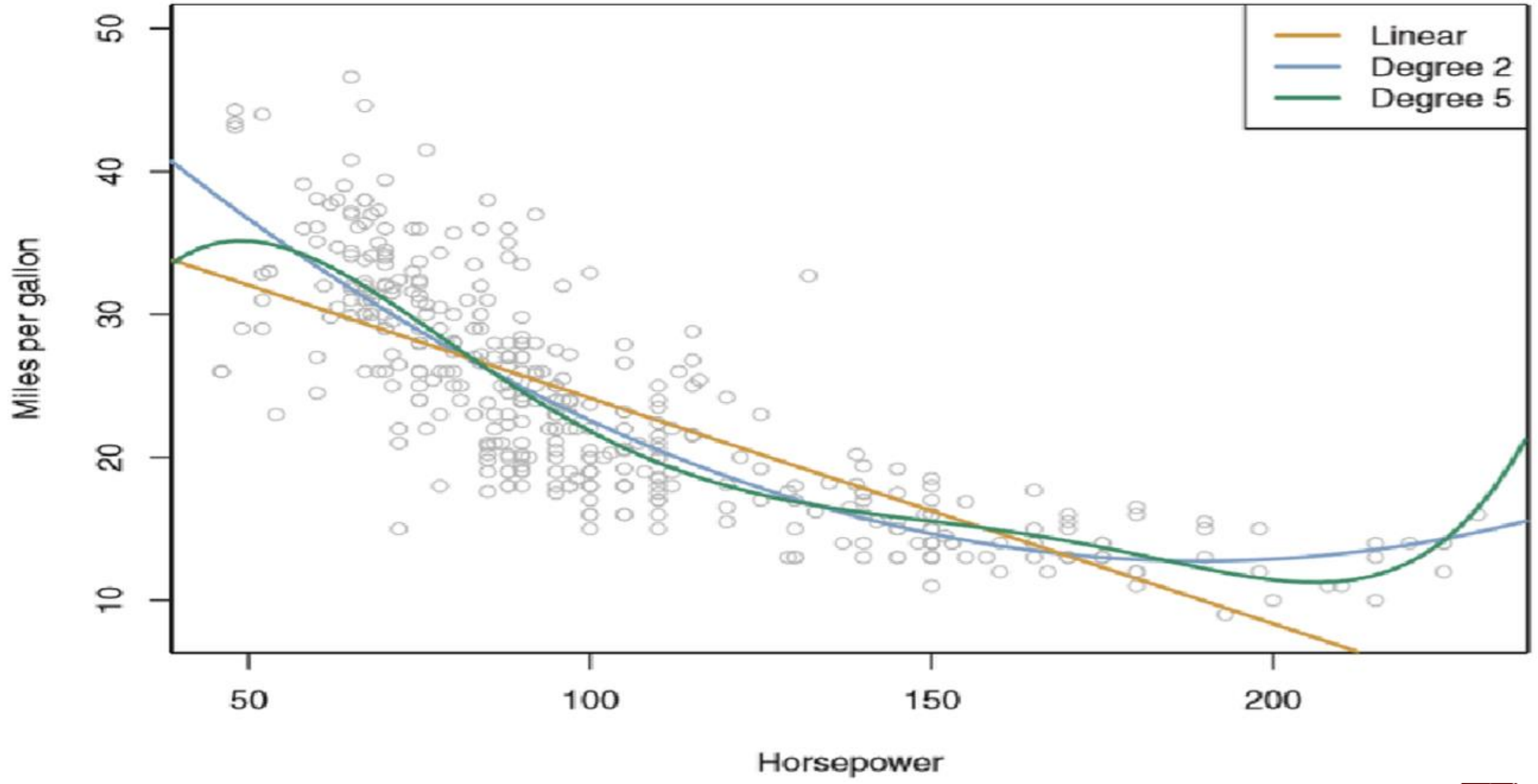
Cross-Validation: Test Set Approach

- We can randomly split the training data (after having removed our “hold out” data) to train and test (or validate).
- These splits can be 50/50, 80/20, 90/10



Let’s compare polynomial fit for horsepower and mpg

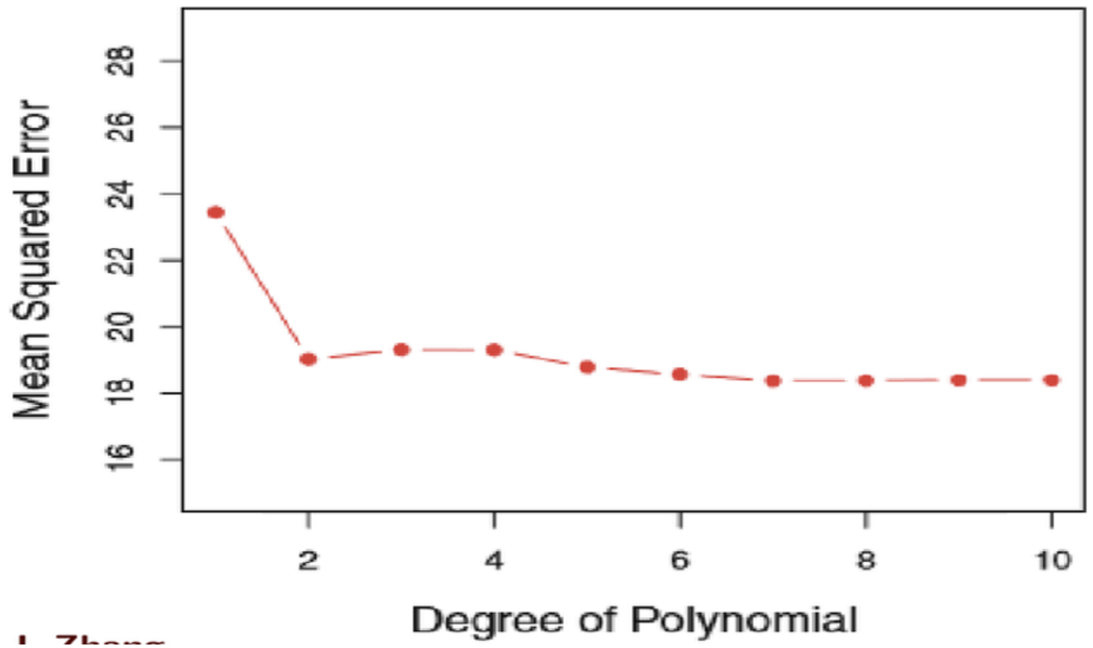
- Model for estimating relationship between engine horsepower and the miles per gallon achieved
- Best fit is $mpg = w_0 + w_1 horsepower + w_2 horsepower^2 + \epsilon$
- So how do we determine the best degree of polynomial?



TEXAS A&M UNIVERSITY

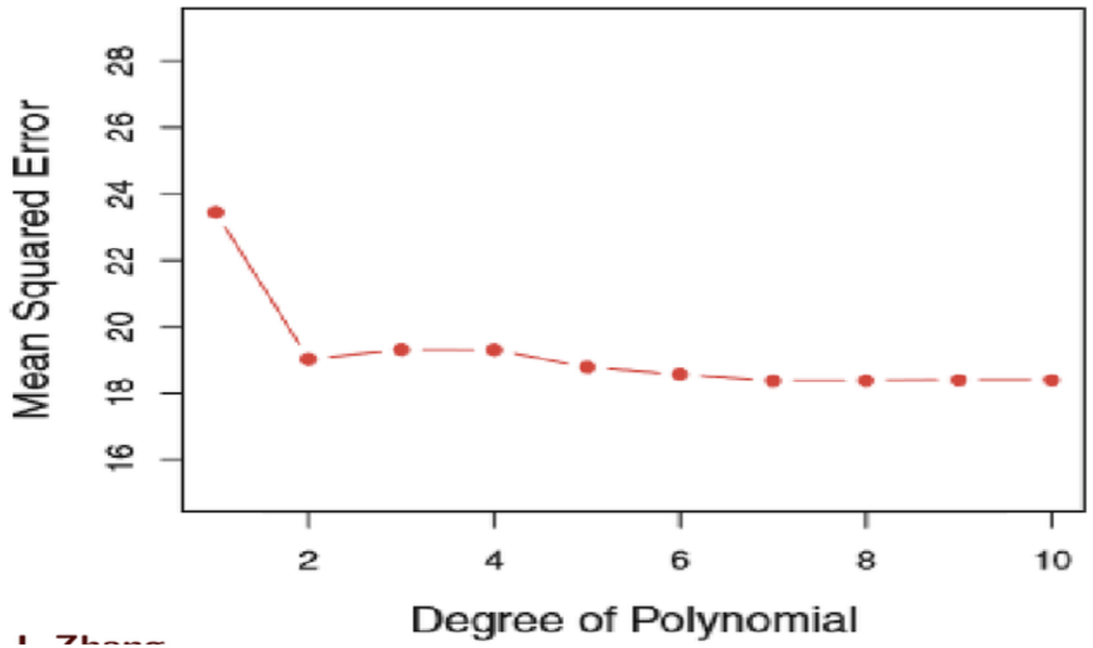
Cross-validation: Test set approach

- Best fit is $mpg = w_0 + w_1 horsepower + w_2 horsepower^2 + \epsilon$
- Split the data 50/50
- See that the cubic term actually increases error



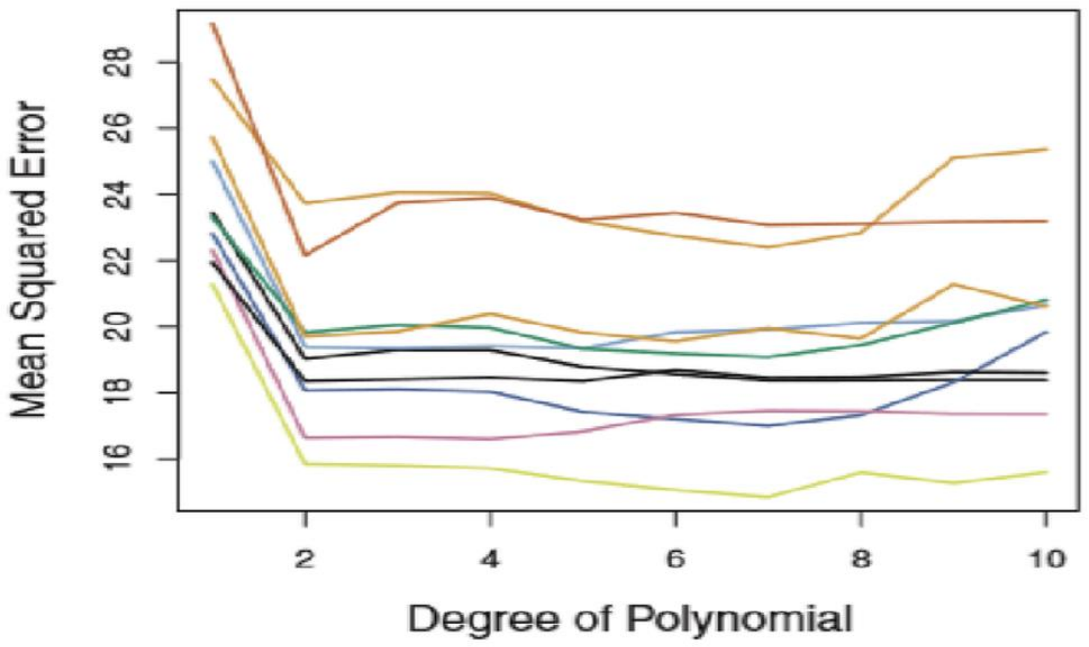
Cross-validation: Test set approach

- Best fit is $mpg = w_0 + w_1 horsepower + w_2 horsepower^2 + \epsilon$
- Split the data 50/50
- See that the cubic term actually increases error
- What if we repeat this experiment multiple times?



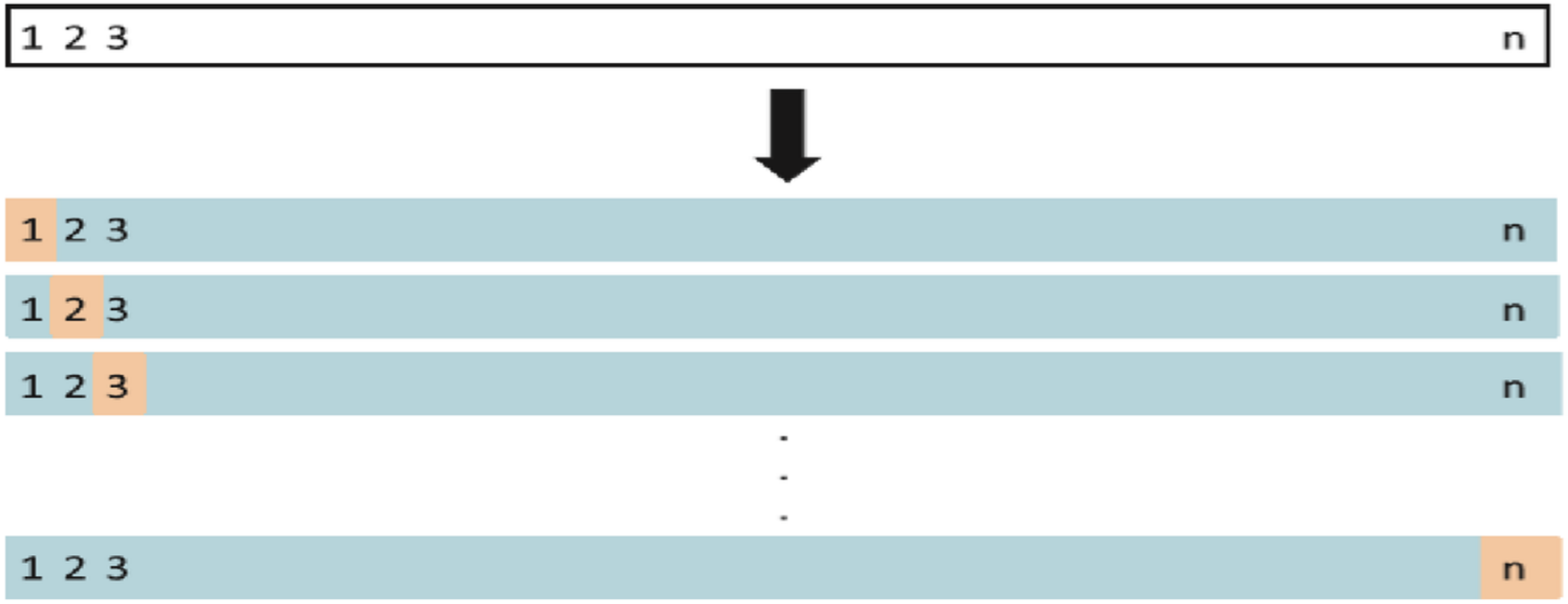
Cross-validation: Test set approach

- Best fit is $mpg = w_0 + w_1 horsepower + w_2 horsepower^2 + \epsilon$
- Split the data 50/50
- See that the cubic term actually increases error
- What if we repeat this experiment multiple times?



TEXAS A&M UNIVERSITY

Leave-one-out Cross-validation

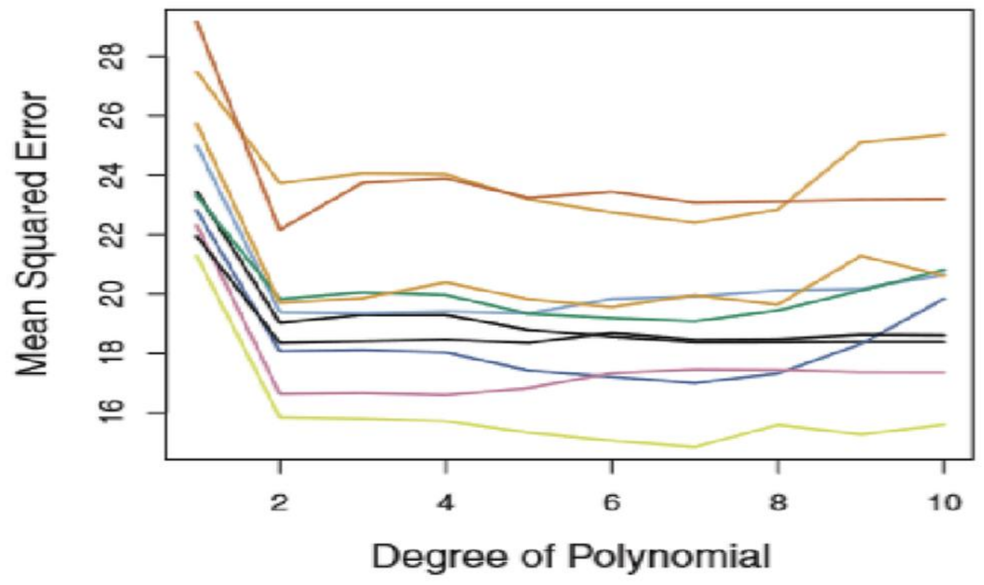


Advantages of LosoCV

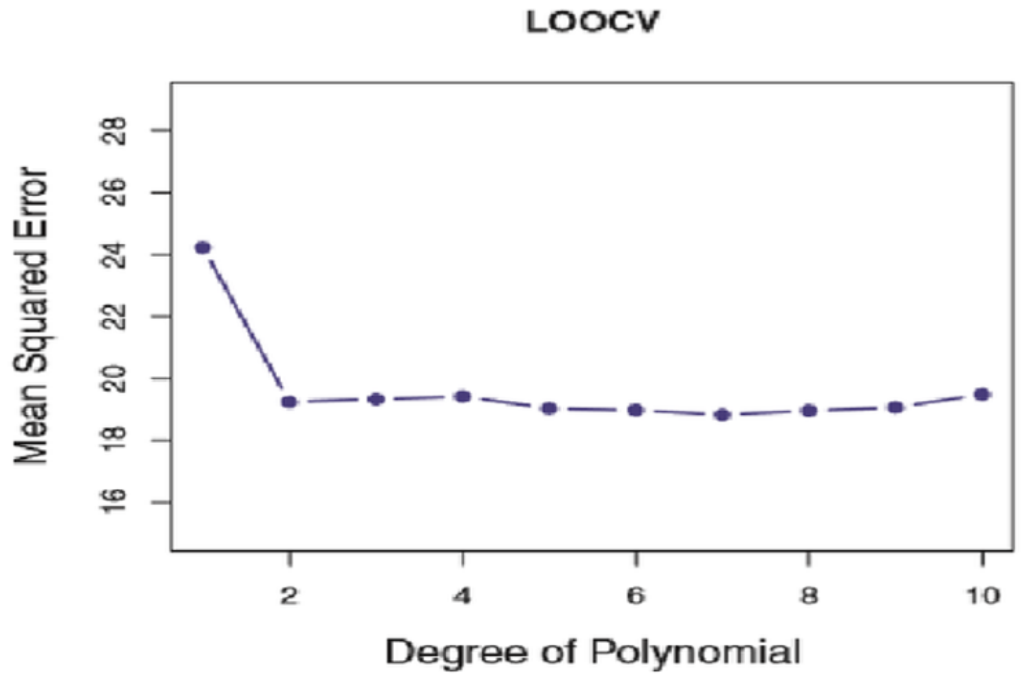
- In each iteration, train with as much data as possible
- Consider error as the average across all iterations
- This presents far less bias
- It won’t overestimate/underestimate test error as much as a single train/test split can

Comparison with MPG and Horsepower

Train-test split

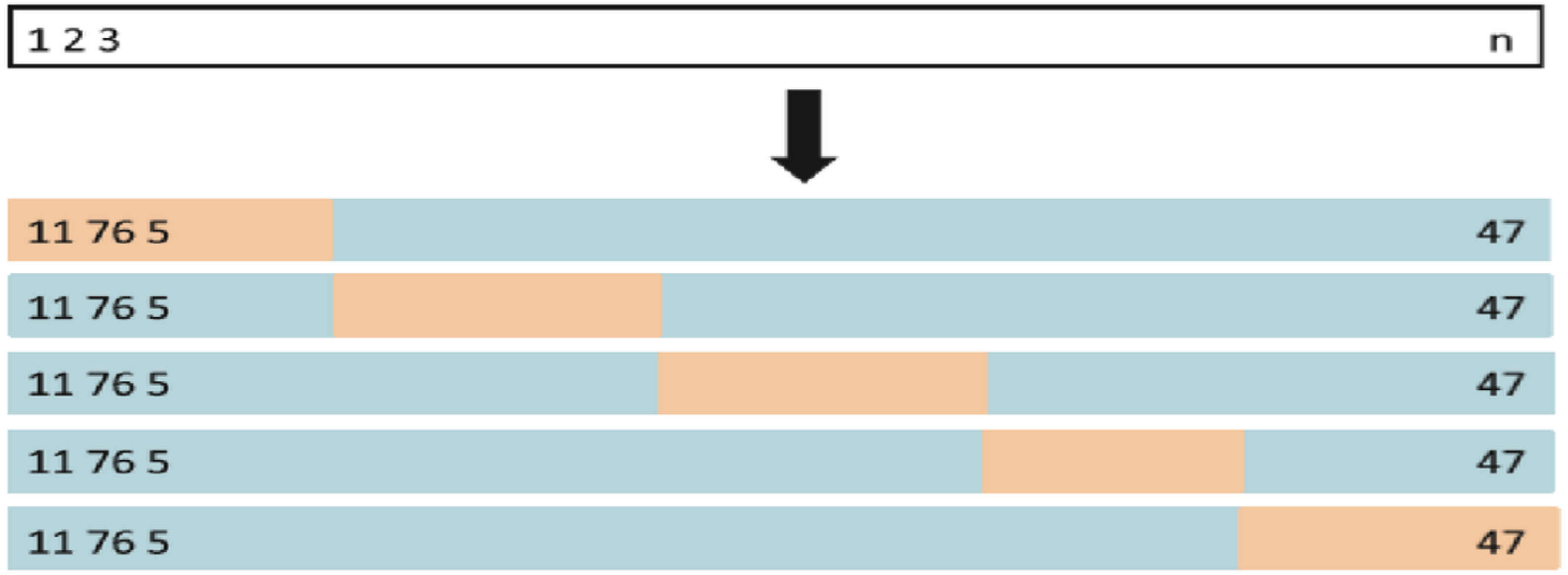


Loso cv



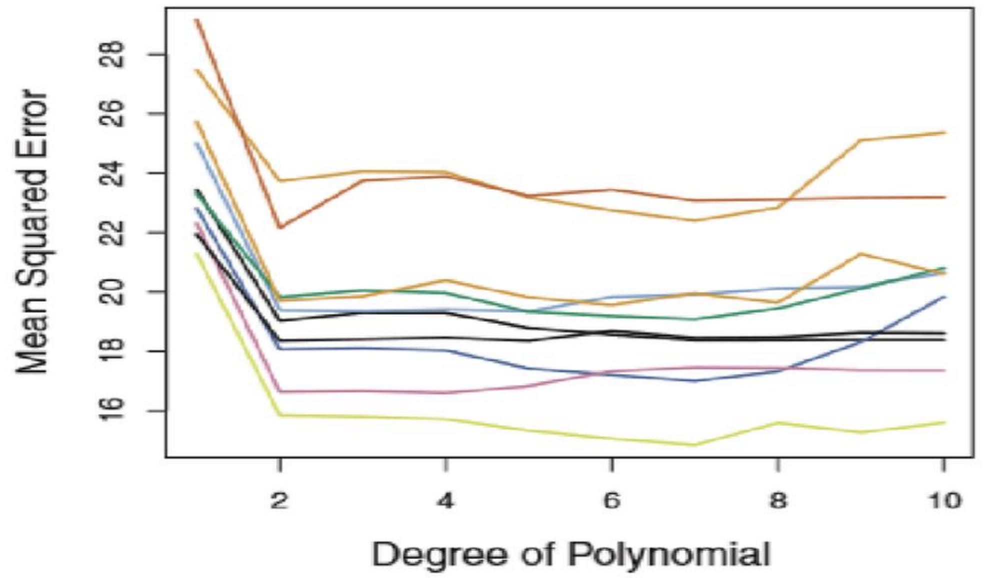
K-fold Cross-Validation

- LOSO CV can be computationally expensive. A single train/test split not robust enough.
- Iterate subsets instead to find a balance

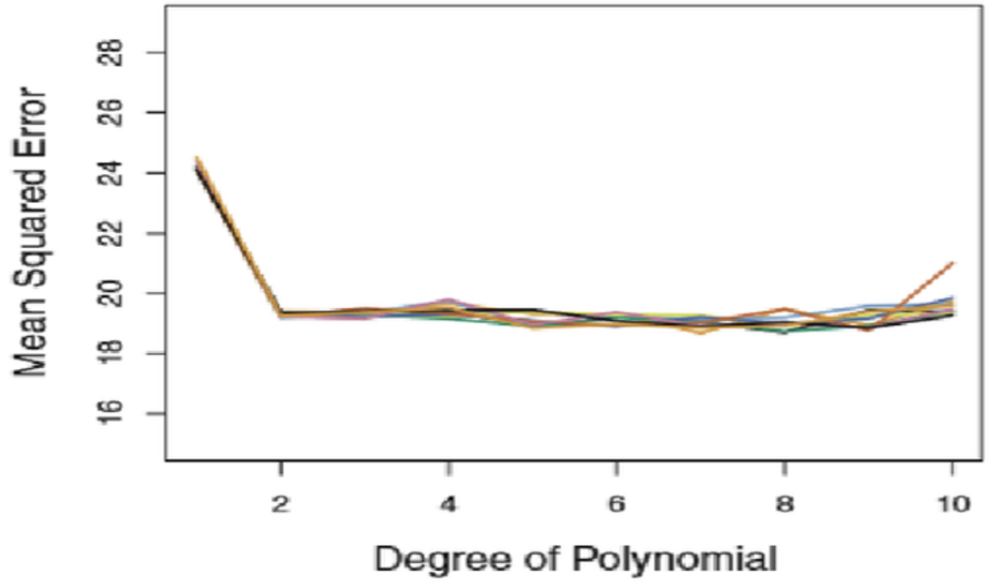


Comparison with MPG and Horsepower

Train-test split



K=10 fold cross-validation



Takeaways

- Why do we need Logistic Regression?
- Understand why the shift from regression to classification results in needing Gradient Descent
- Learn how to use Cross-Validation
- **Next time: Regularization**