EE2211 Lecture 7: Overfitting, Model Complexity, Feature Selection & Regularization, Bias-Variance Tradeoff

Vincent Y. F. Tan

Department of Electrical and Computer Engineering, NUS

EE2211 Spring 2023

Acknowledgements to

Xinchao, Helen, Thomas, Kar Ann, Chen Khong, Robby, and Haizhou

Outline

1 Review of Regression

Review of Regression

- Goal: Given feature(s) $\mathbf{x} \in \mathbb{R}^d$, we want to predict target $y \in \mathbb{R}$.
  1. $\mathbf{x}$ can be one-dimensional (in which case we may write x);
  2. Or $\mathbf{x}$ may be high dimensional;
  3. y is often one-dimensional (trying to predict one output).

- Two types of input data
  1. Training Set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$;
  2. Test Set $\{\mathbf{x}_j\}_{j=m+1}^n$ (no targets) [We called one of these samples $\mathbf{x}_{\text{new}}$ previously];

- m training samples, n test samples, all in $\mathbb{R}^d$

- Learning/Training
  Training set used to estimate regression coefficients $\bar{\mathbf{w}}^*$.

- Testing/Prediction
  Prediction performed on test set to evaluate performance.

Review of Regression: Linear Case

- Suppose the feature is one-dimensional $x \in \mathbb{R}$.
- Want to learn the affine relationship between x and $y \in \mathbb{R}$.
- With $m = 4$ training samples, the design matrix and target vector are

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

- Learning/Training

$$\bar{\mathbf{w}}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

- Testing/Prediction on x_{new}

$$y_{\text{test}} = \begin{bmatrix} 1 \\ x_{\text{new}} \end{bmatrix}^\top \bar{\mathbf{w}}^*.$$

Regression Review: Polynomial

- Features $\mathbf{x} \in \mathbb{R}^d$ may be one-dimensional or high dimensional; $y \in \mathbb{R}$ is a one-dimensional target as usual.
- Want to learn a polynomial relationship between x and y
- Quadratic illustration ($m = 4$ training samples and x is one-dim) of design matrix and target vector

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

- Learning/Training

$$\bar{\mathbf{w}}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \in \mathbb{R}^3.$$

- Testing/Prediction on x_{new}

$$y_{\text{test}} = \begin{bmatrix} 1 \\ x_{\text{new}} \\ x_{\text{new}}^2 \end{bmatrix}^\top \bar{\mathbf{w}}^*.$$

Polynomial Regression Review: Notation

- Sometimes, we use the notation $\mathbf{P}$ in place of $\mathbf{X}$ to emphasize that the features $\mathbf{x}$ are encoded in a polynomial way in the design matrix $\mathbf{P} = \mathbf{X}$.
- Learning/Training

$$\bar{\mathbf{w}}^* = (\mathbf{P}^\top \mathbf{P})^{-1} \mathbf{P}^\top \mathbf{y} \in \mathbb{R}^3.$$

- Testing/Prediction on x_{new}

$$y_{\mathrm{new}} = \underbrace{\begin{bmatrix} 1 \\ x_{\mathrm{new}} \\ \color{red}{x_{\mathrm{new}}^2} \end{bmatrix}^\top}_{= \mathbf{p}_{\mathrm{new}}^\top} \bar{\mathbf{w}}^*.$$

Note on Training & Test Sets

- Affine is special case of polynomial
- Use $\mathbf{P}$ instead of $\mathbf{X}$ from now on.
- Training/Learning (primal) on training set

$$\bar{\mathbf{w}}^* = (\mathbf{P}^\top \mathbf{P})^{-1} \mathbf{P}^\top \mathbf{y} \in \mathbb{R}^{d'}.$$

- Prediction/Testing on the entire test set

$$\mathbf{y}_{\mathrm{new}} = \mathbf{P}_{\mathrm{new}} \bar{\mathbf{w}}^* \in \mathbb{R}^n$$

where

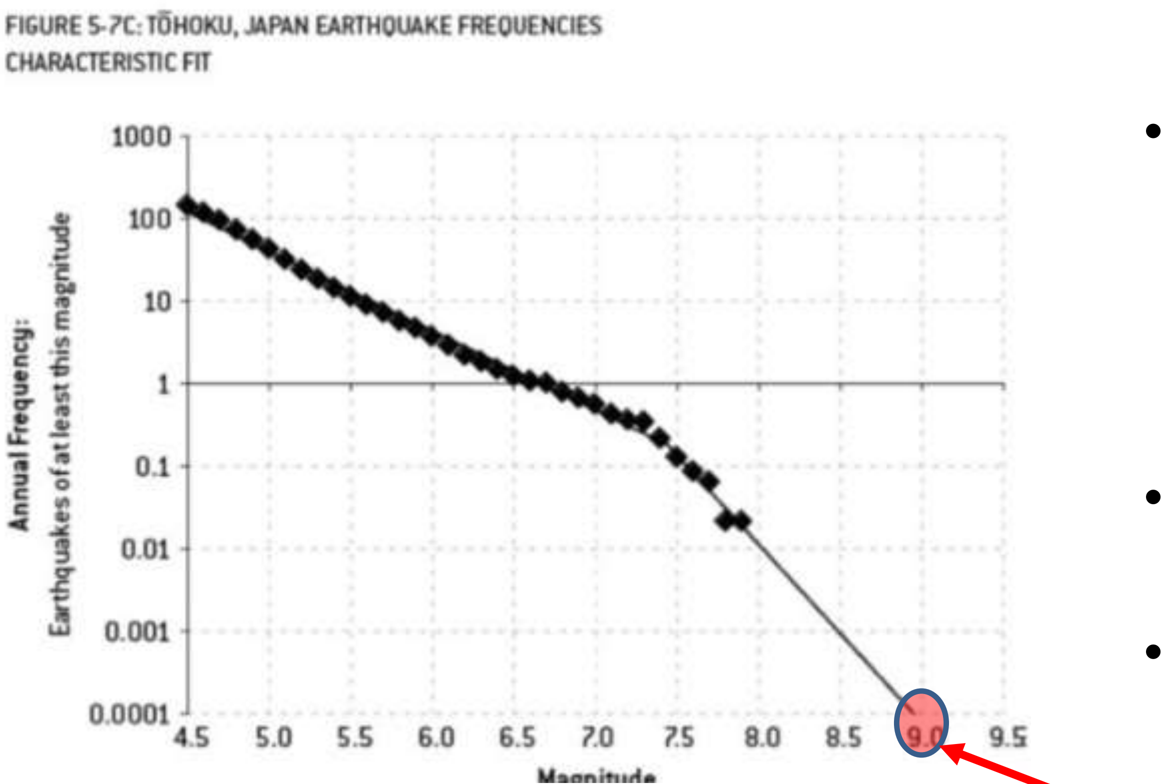
$$\mathbf{P}_{\mathrm{new}} = \begin{bmatrix} \mathbf{p}_{m+1}^\top \\ \mathbf{p}_{m+2}^\top \\ \vdots \\ \mathbf{p}_{m+n}^\top \end{bmatrix} \in \mathbb{R}^{n \times d'}$$

Note on Training & Test Sets

- There should be zero overlap between training & test sets.
- Important goal of regression: prediction on new unseen data, i.e., the test set.
- Question: Why is test set important for evaluation?

Questions?

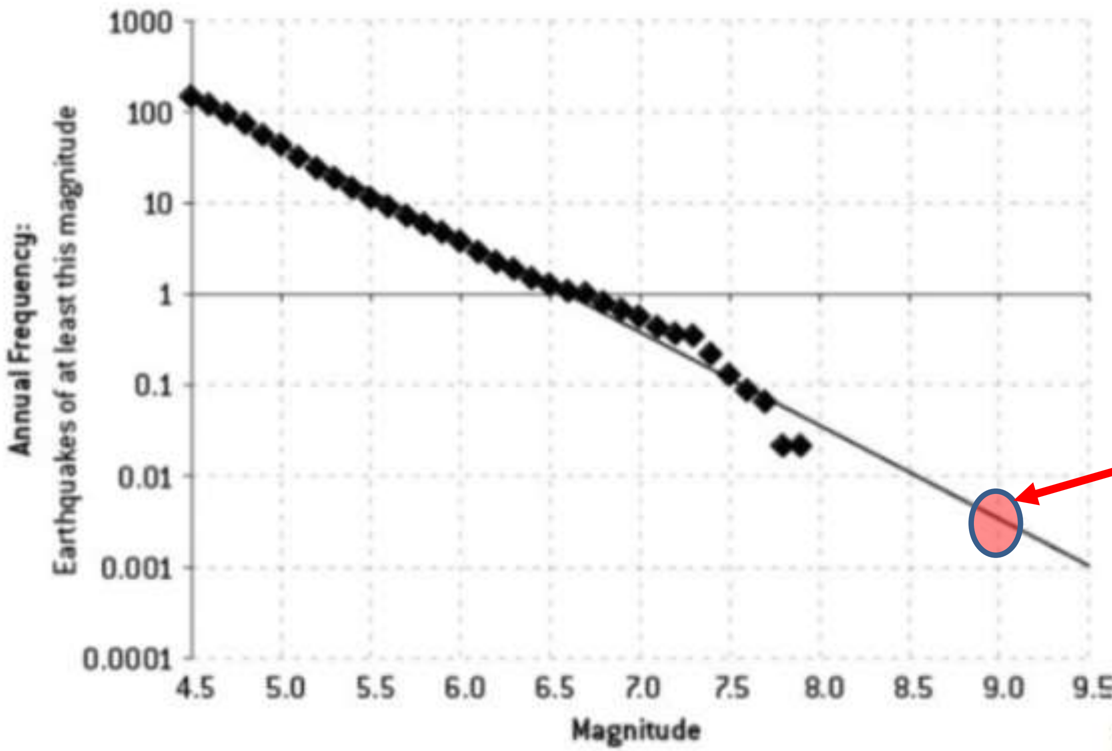
Overfitting Motivation: Fukushima Disaster



- In March 2011, there was a large earthquake in Japan.
- Small earthquakes occur frequently while massive earthquakes occur rarely.
- Engineers fitted a polynomial model
- Earthquake of magnitude 9.0 occurs once every 10,000 years

Overfitting Motivation: Fukushima Disaster

FIGURE 5-7B: TŌHOKU, JAPAN EARTHQUAKE FREQUENCIES
GUTENBERG-RICHTER FIT



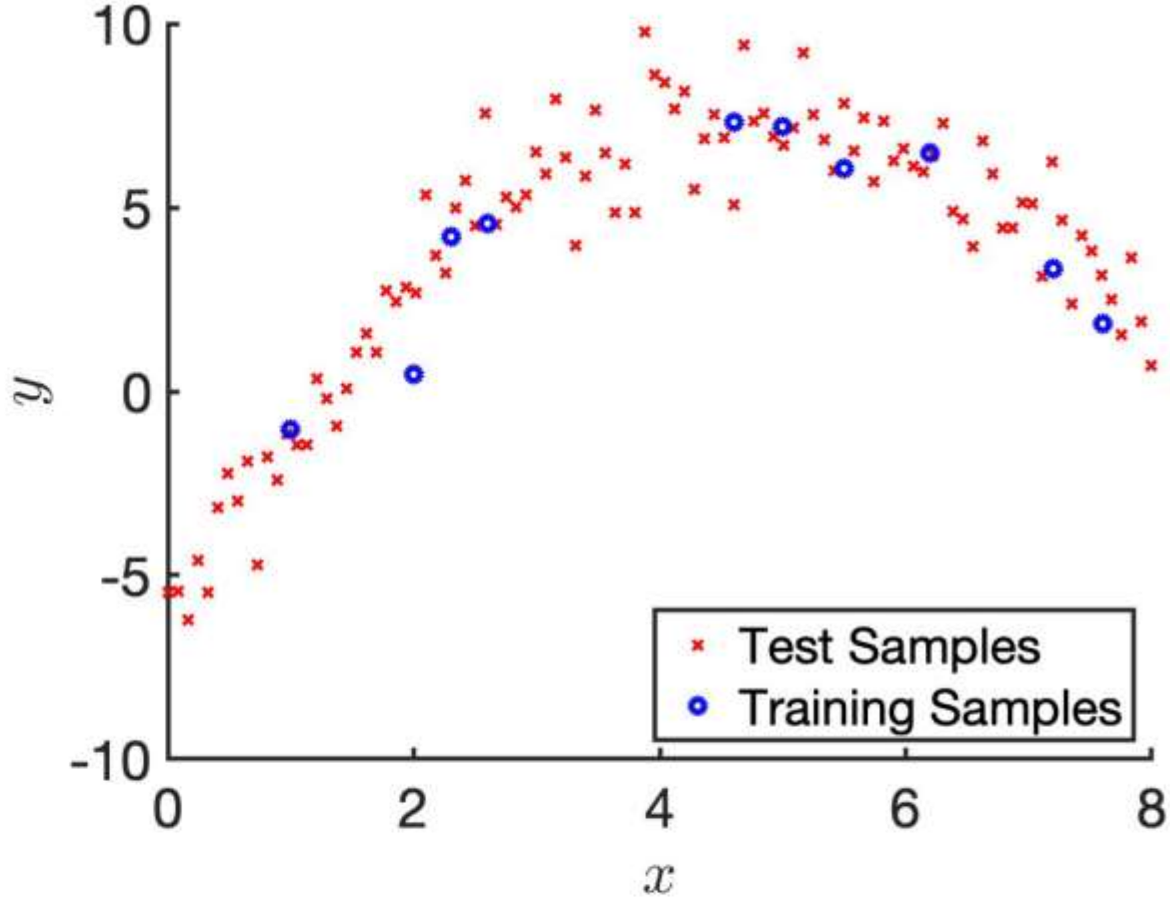
(Si

Might want to fit an affine model instead

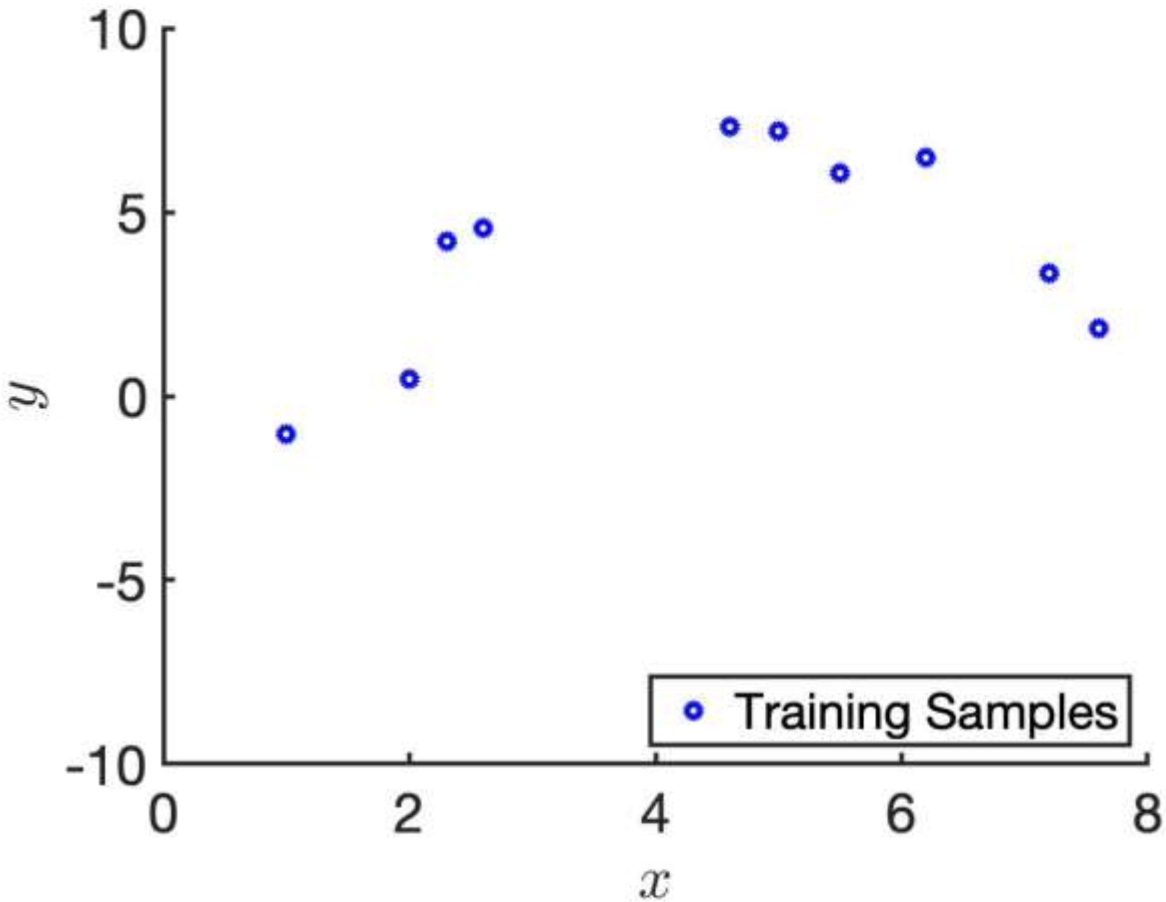
Predicts that an earthquake of magnitude 9.0 occurs once every 500 years

Could have reinforced the buildings

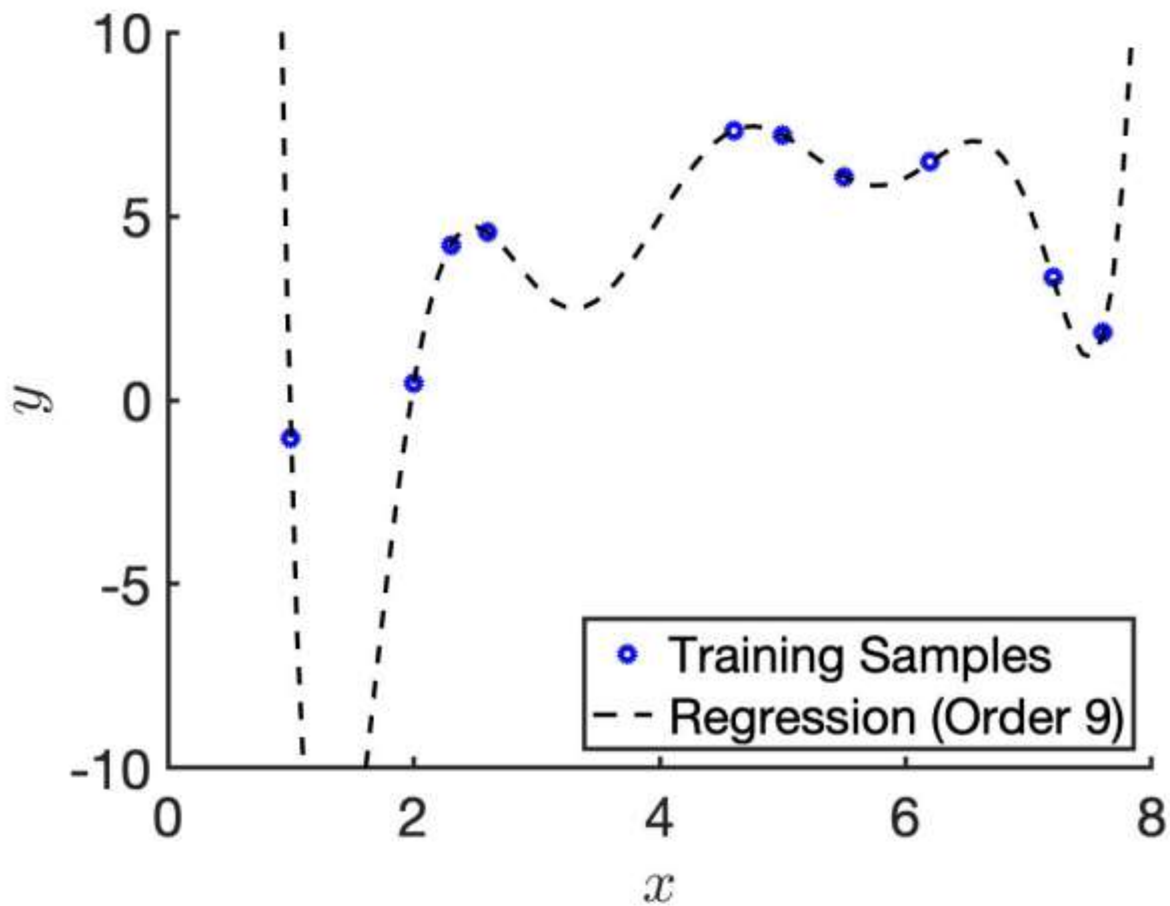
Overfitting Example



Overfitting Example

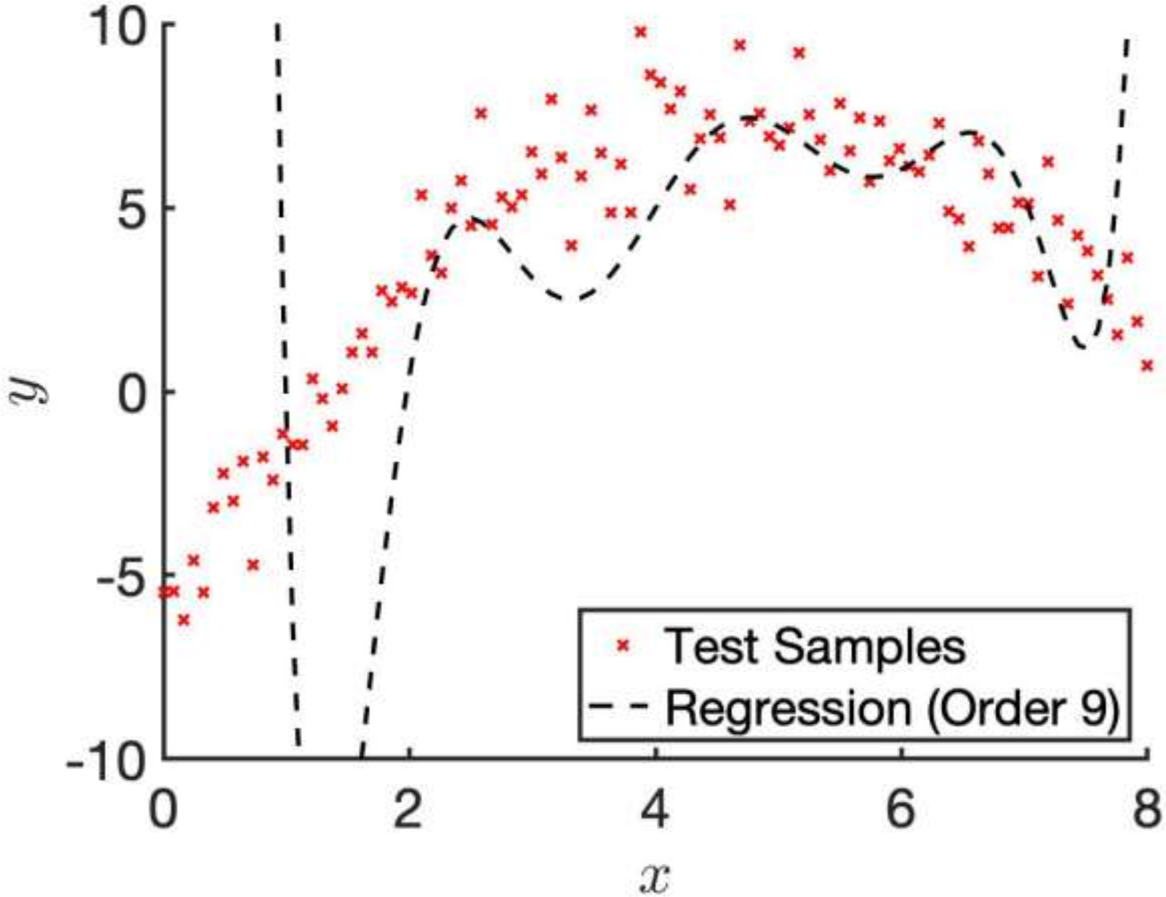


Overfitting Example



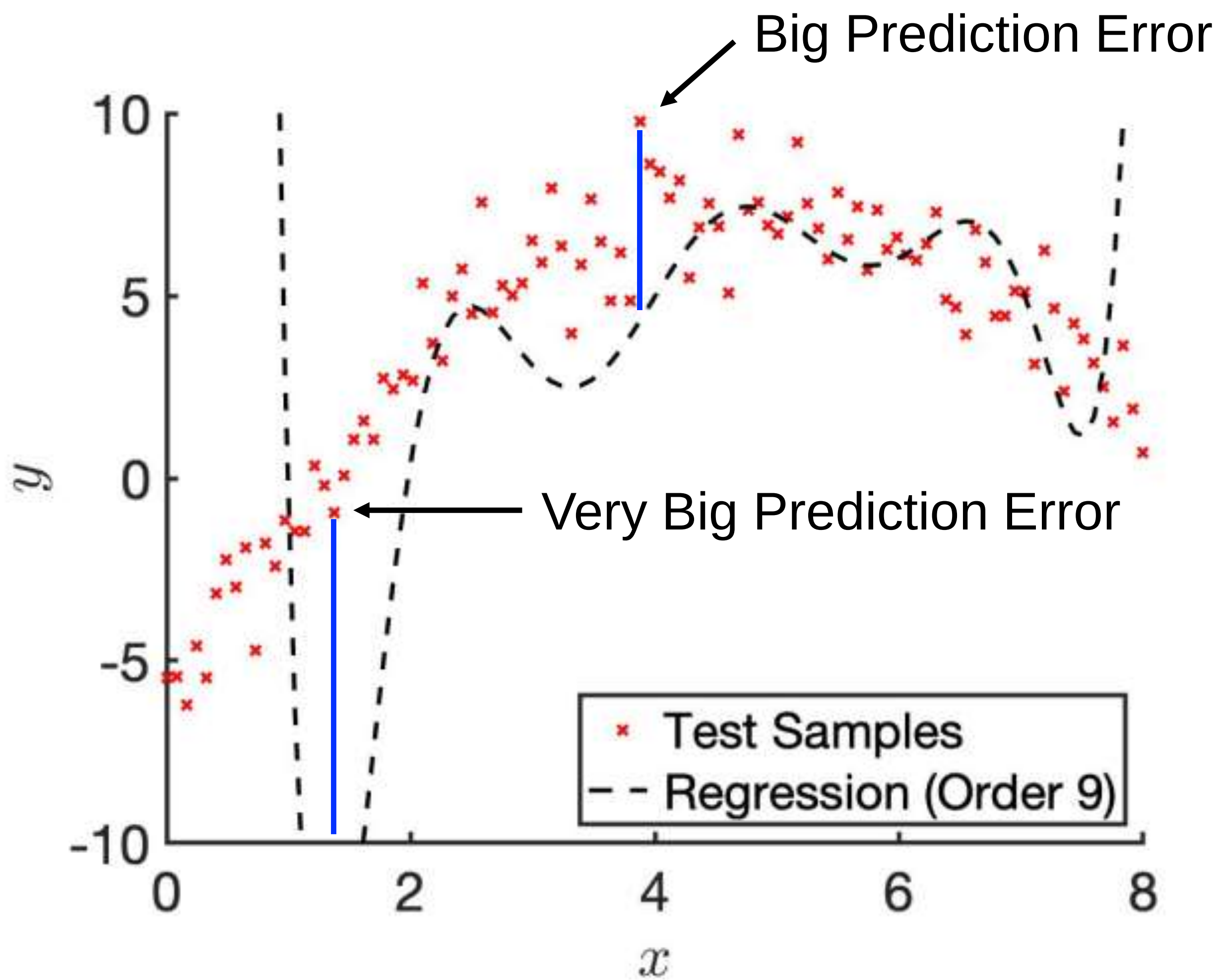
	Training Set Fit	
Order 9	Good	

Overfitting Example



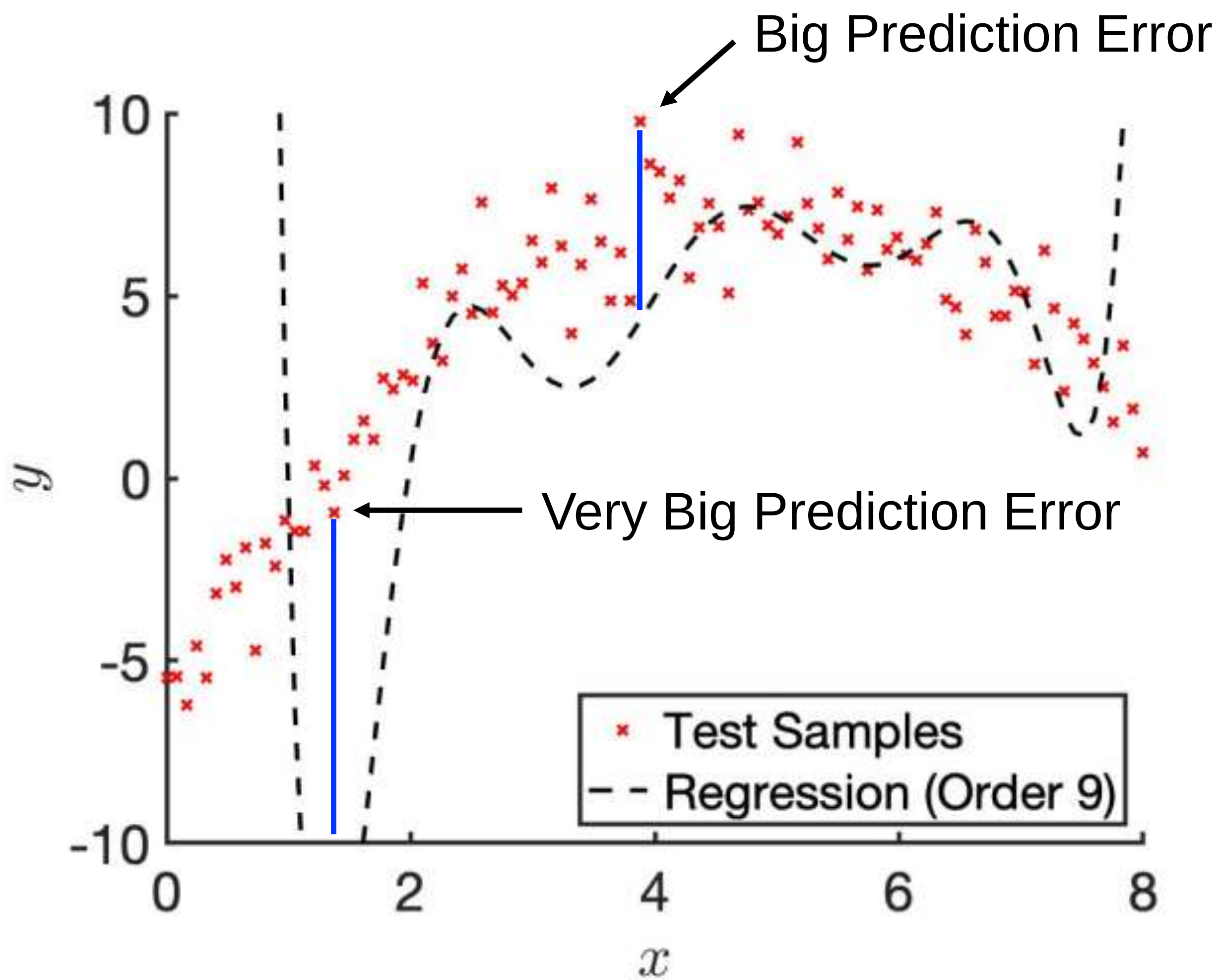
	Training Set Fit	Test Set Fit
Order 9	Good	

Overfitting Example



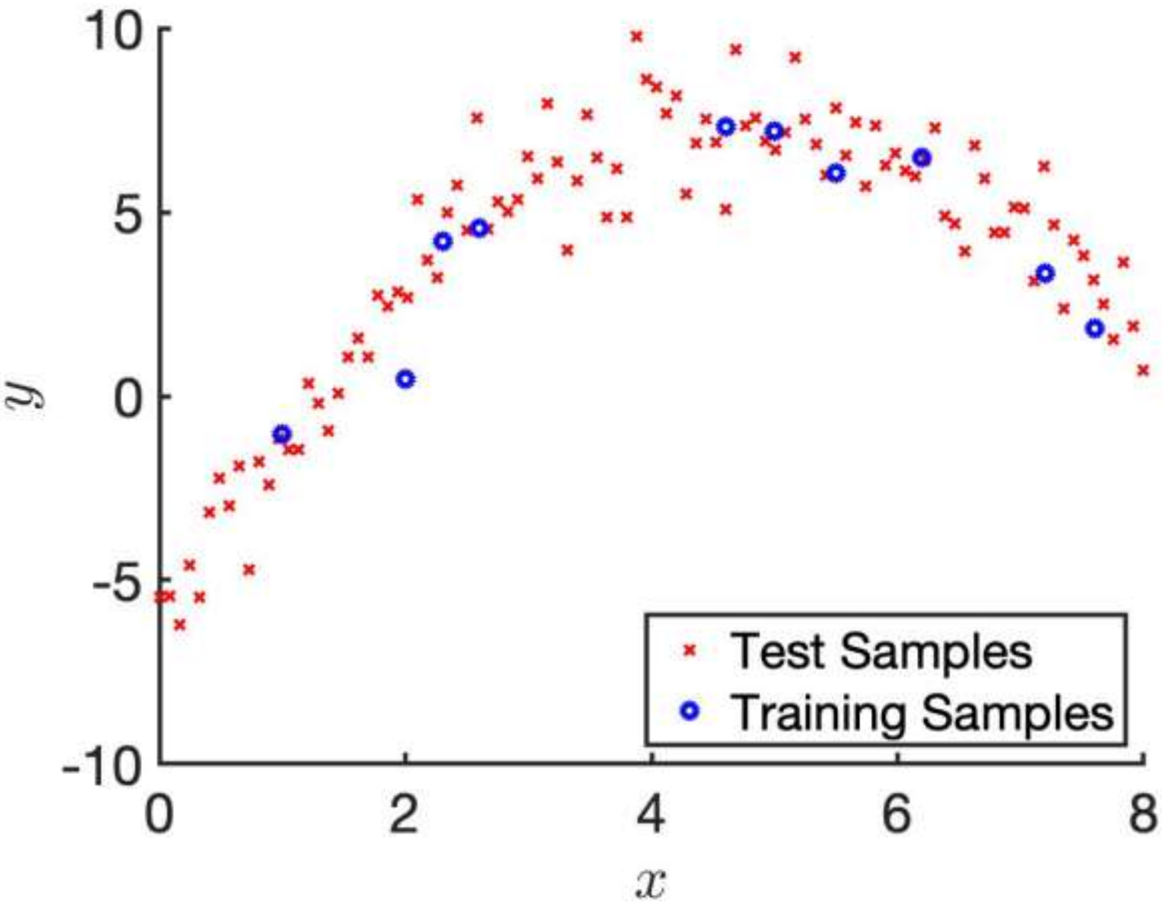
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Overfitting Example



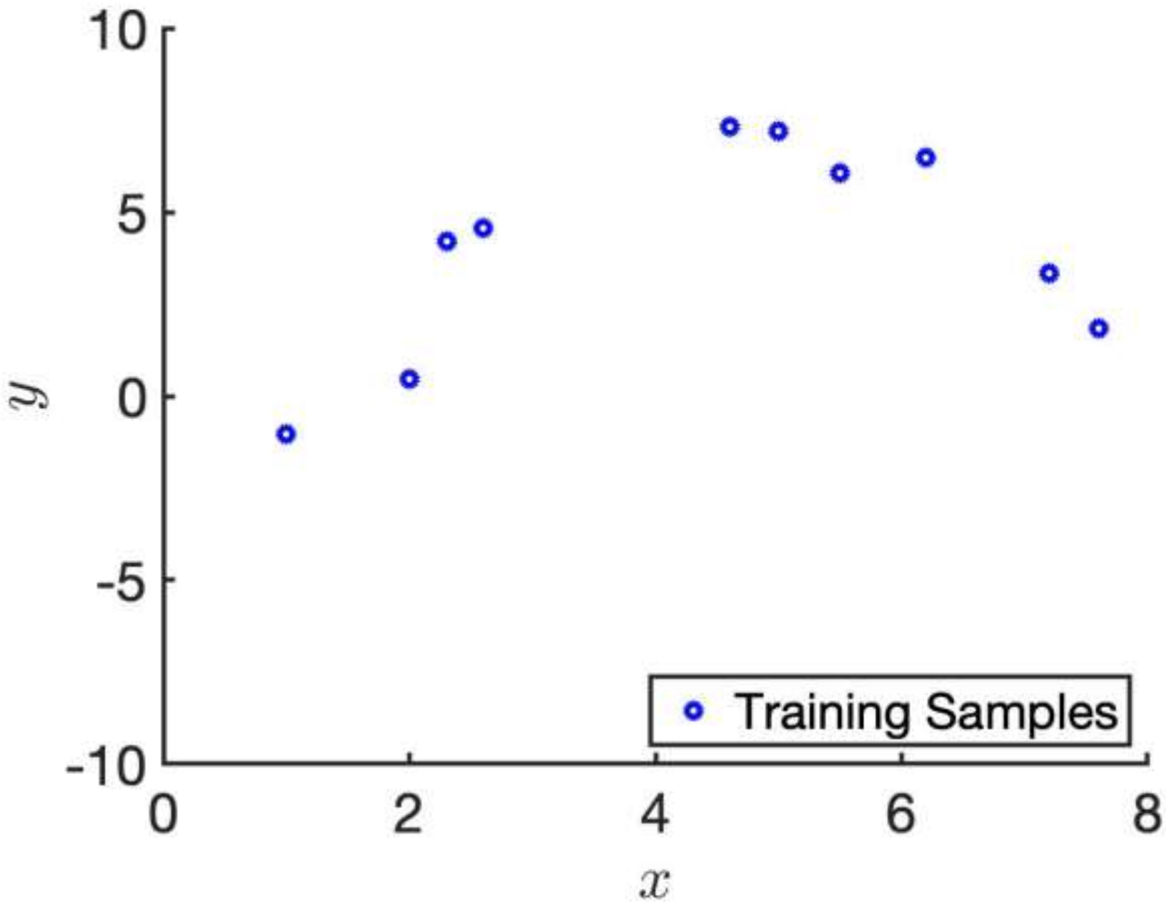
- If we take one of the blue lines and compute the square of its length, this is called **"squared error" for that particular data point**
- If we average squared errors across all the red crosses, it's called **mean squared error (MSE)** in the test set

Underfitting Example



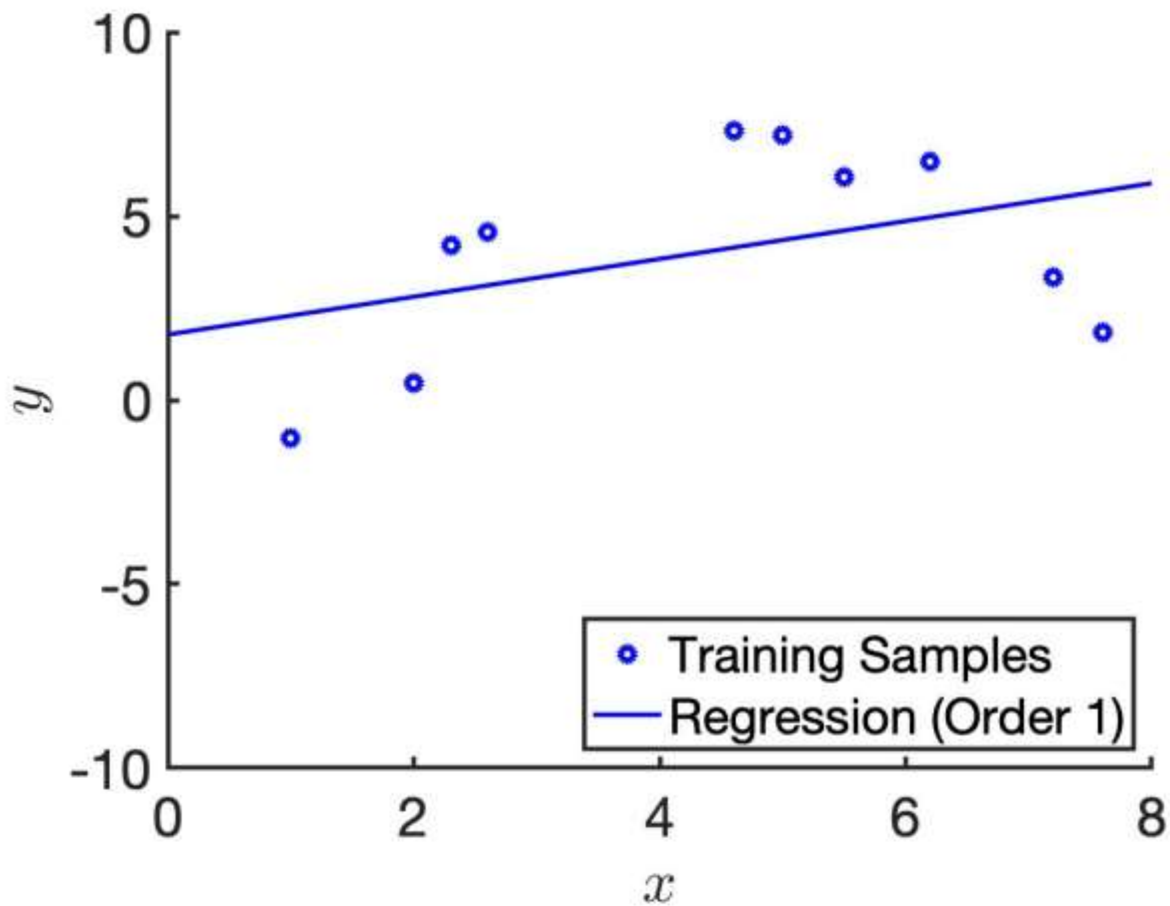
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Underfitting Example



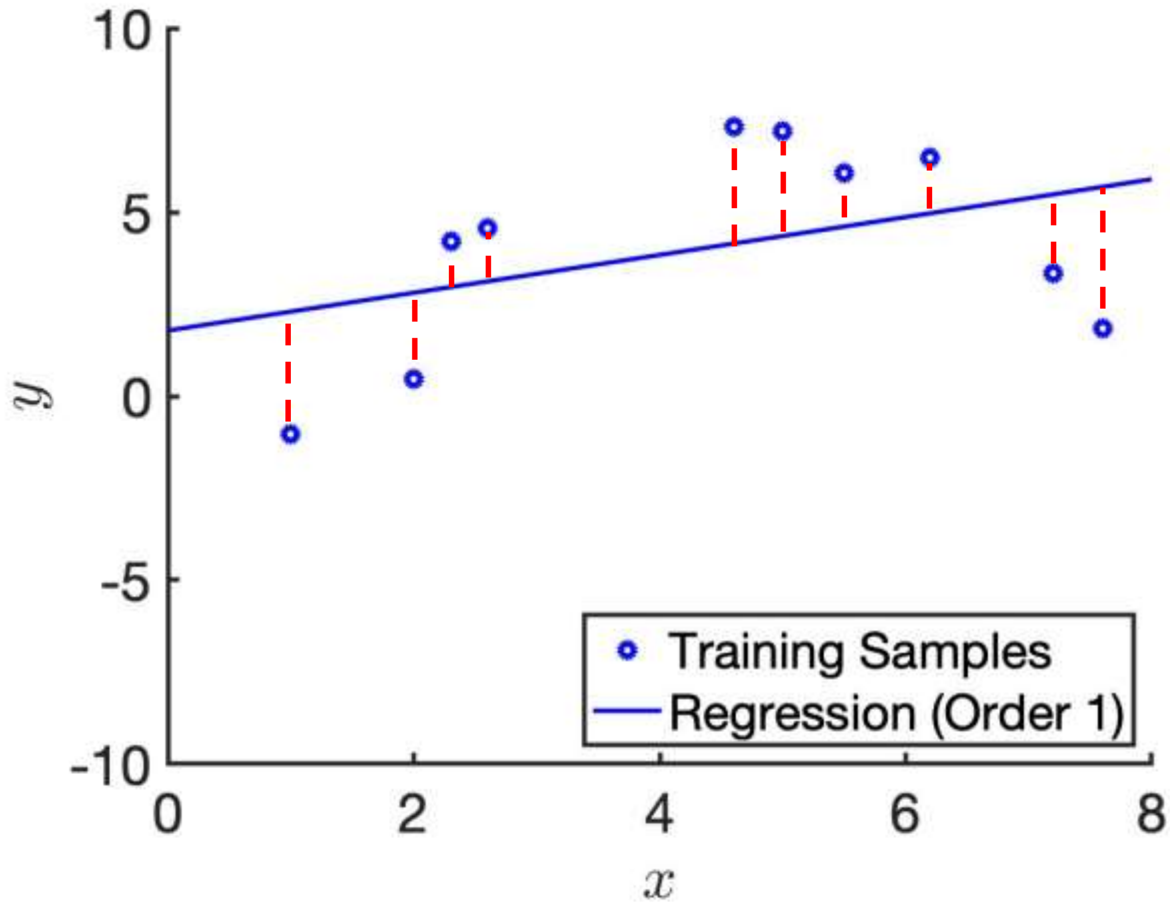
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Underfitting Example



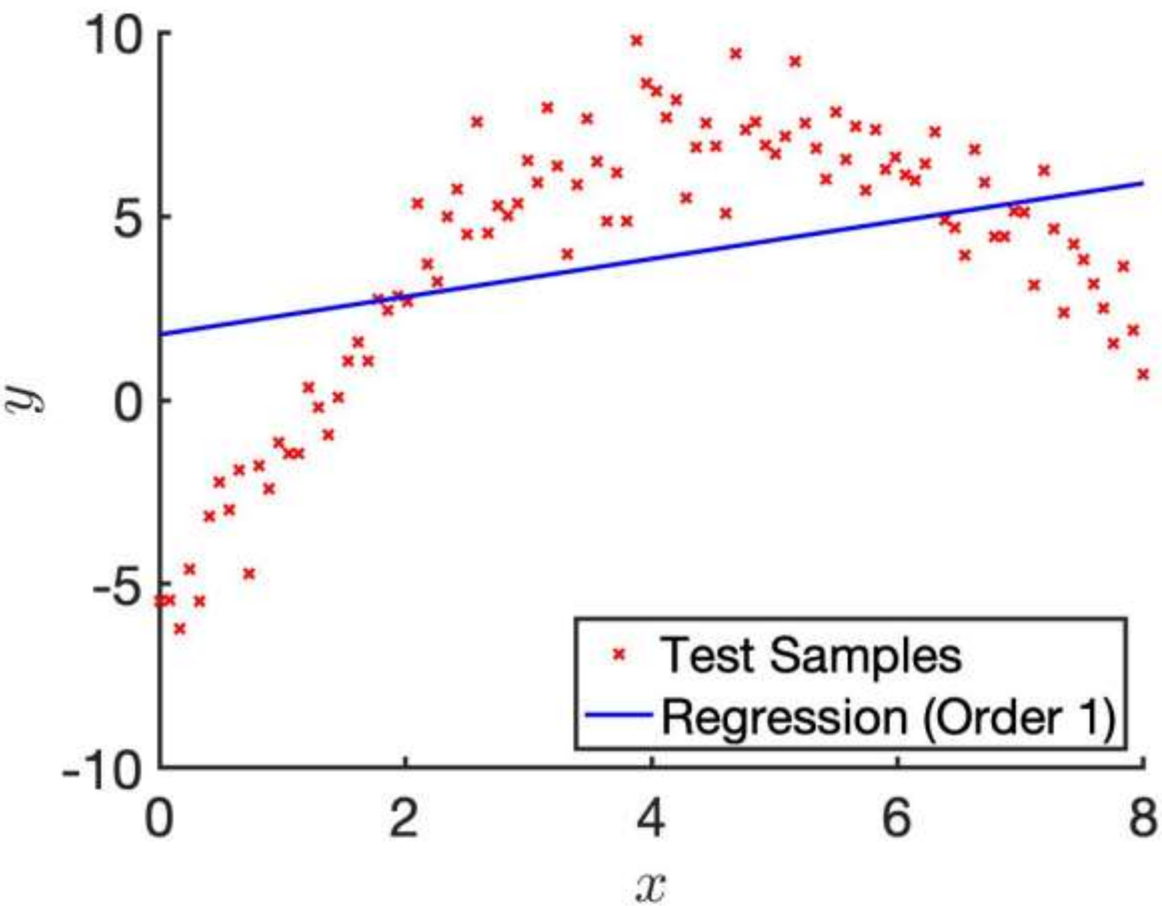
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1		

Underfitting Example



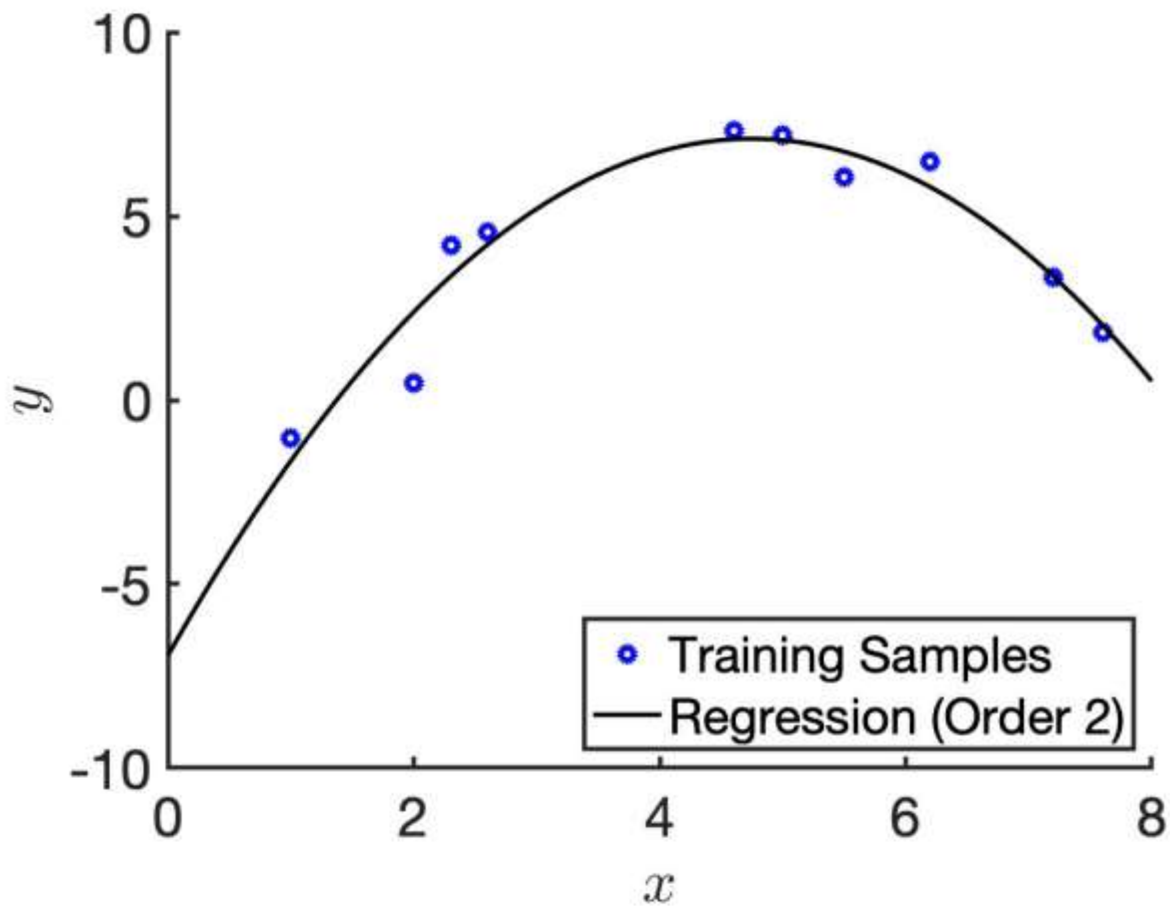
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	

Underfitting Example



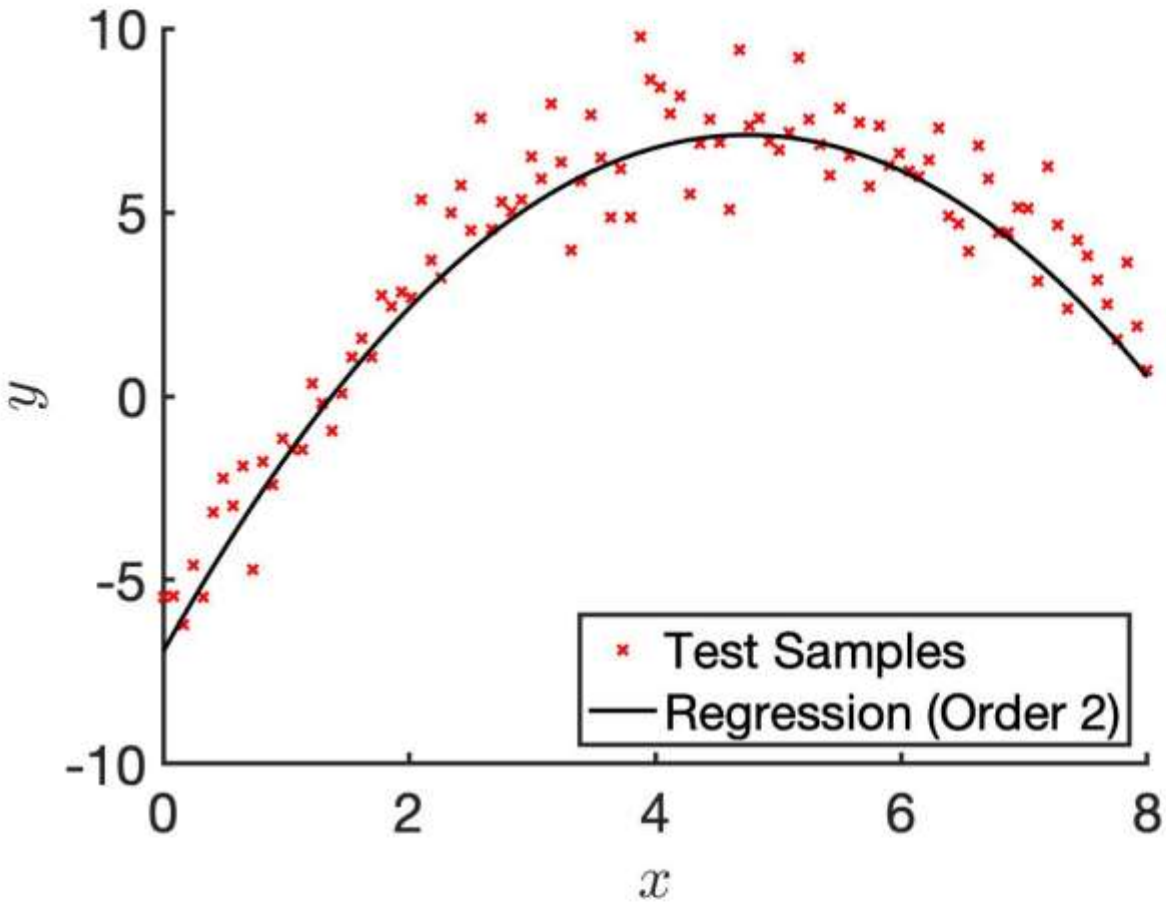
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad

“Just Nice”



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	

"Just Nice"



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting & Underfitting

	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting → Order 9

Underfitting → Order 1

Overfitting & Underfitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly

Overfitting & Underfitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- **Reason 1**
  – Model is too complex for the data
  – Previous example: Fit order 9 polynomial to 10 data points

Overfitting & Underfitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- **Reason 1**
  - Model is too complex for the data
  - Previous example: Fit order 9 polynomial to 10 data points
- **Reason 2**
  - Too many features but number of training samples too small
  - Even linear model can overfit, e.g., linear model with 9 input features (i.e., w is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns <u>well</u>

Overfitting & Underfitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- **Reason 1**
  - – Model is too complex for the data
  - – Previous example: Fit order 9 polynomial to 10 data points
- **Reason 2**
  - – Too many features but number of training samples too small
  - – Even linear model can overfit, e.g., linear model with 9 input features (i.e., w is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns <u>well</u>
- **Solutions**
  - – Use **simpler models** (e.g., lower order polynomial)
  - – Use **regularization** (see next part of lecture)

Overfitting & Underfitting

- **Underfitting** is the inability of trained model to predict the targets in the training set

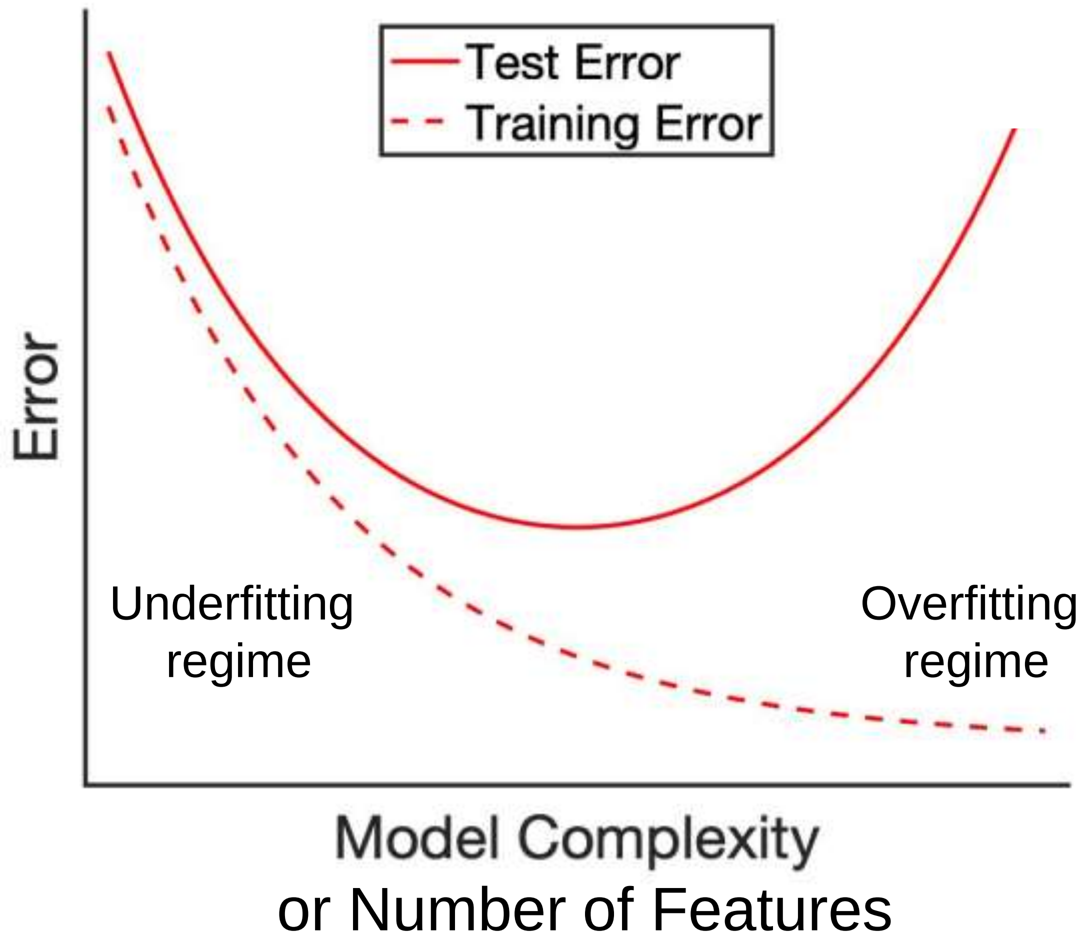
Overfitting & Underfitting

- **Underfitting** is the inability of trained model to predict the targets in the training set
- **Reason 1**
  – Model is too simple for the data
  – Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
  – **Solution**: Try more complex model

Overfitting & Underfitting

- **Underfitting** is the inability of trained model to predict the targets in the training set
- **Reason 1**
  - Model is too simple for the data
  - Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
  - **Solution**: Try more complex model
- **Reason 2**
  - Features are not informative enough
  - **Solution**: Try to develop more informative features

Overfitting / Underfitting Schematic



Questions?

Feature Selection

- Less features might reduce overfitting
  - Want to discard useless features & keep good features, so perform feature selection

Feature Selection

- Less features might reduce overfitting
  - Want to discard useless features & keep good features, so perform feature selection
- Feature selection procedure
  - Step 1: feature selection in training set
  - Step 2: fit model using selected features in training set
  - Step 3: evaluate trained model using test set

Feature Selection

- Less features might reduce overfitting
  - Want to discard useless features & keep good features, so perform feature selection
- Feature selection procedure
  - Step 1: feature selection in training set
  - Step 2: fit model using selected features in training set
  - Step 3: evaluate trained model using test set
- Very common mistake
  - Feature selection with test set (or full dataset) leads to inflated performance
  - Do not perform feature selection with test data

Selecting Features With Pearson's r

- Given features x, we want to predict target y

Selecting Features With Pearson's r

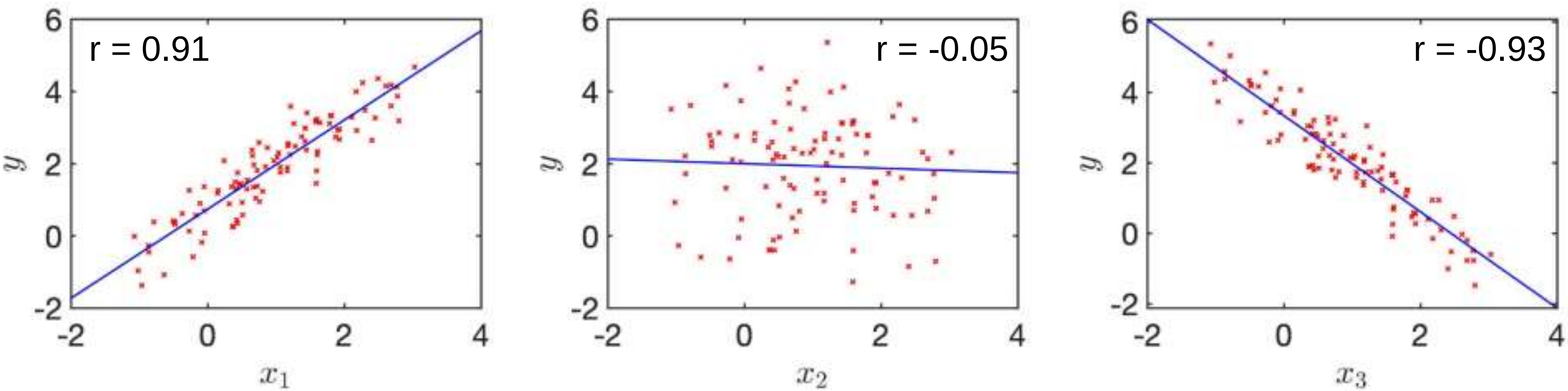
- Given features x, we want to predict target y
- Assume x & y both continuous

Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  – Pearson's correlation r measures linear relationship between two variables

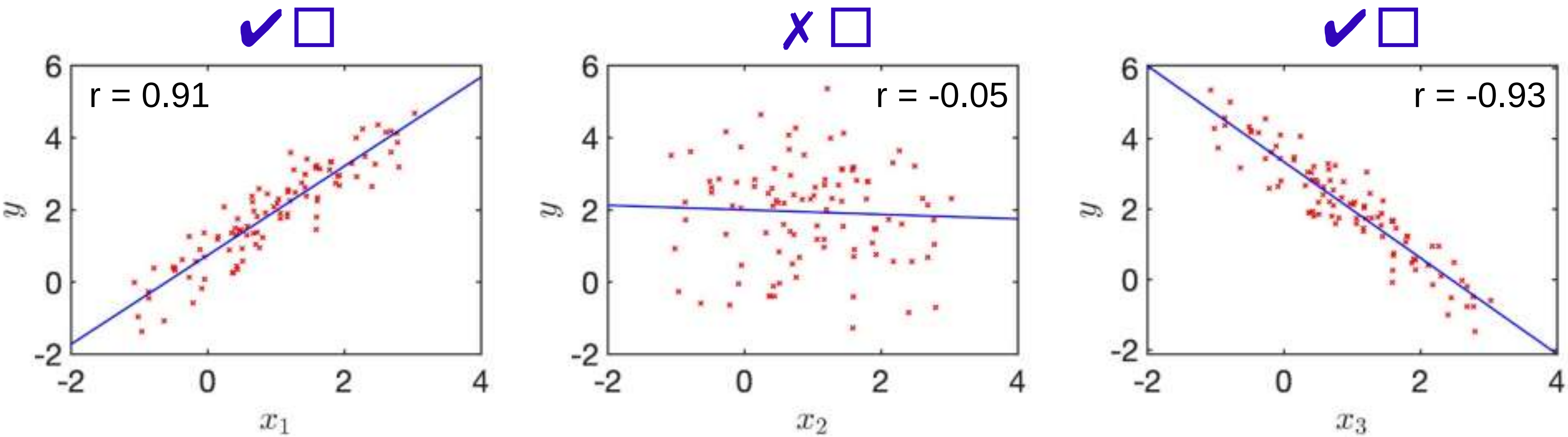
Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  – Pearson's correlation r measures linear relationship between two variables



Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  - Pearson's correlation r measures linear relationship between two variables



Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  - Pearson's correlation r measures linear relationship between two variables
- Two options
  - Option 1: Pick K features with largest absolute correlations
  - Option 2: Pick all features with absolute correlations > C
  - K & C are "magic" numbers set by the ML practitioner

Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  - Pearson's correlation r measures linear relationship between two variables
- Two options
  - Option 1: Pick K features with largest absolute correlations
  - Option 2: Pick all features with absolute correlations > C
  - K & C are "magic" numbers set by the ML practitioner
- Other metrics besides Pearson's correlation are possible

Questions?

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  – For example, estimate 10th order polynomial with just 5 datapoints

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  – For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  – For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  – For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

- For $\lambda > 0$, matrix becomes invertible (Motivation 1)

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

- $\hat{\mathbf{w}}$ might also perform better in test set, i.e., reduces overfitting (Motivation 2) – will show example later

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Cost function quantifying data fitting error in training set

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Cost function quantifying data fitting error in training set

Regularization

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay)

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}} \, \textbf{Data-Loss}(\mathbf{w}) + \lambda \textbf{Regularization}(\mathbf{w})$$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,\mathbf{Data\text{-}Loss}(\mathbf{w})+\lambda\mathbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data

Regularization

- Consider minimization from previous slide

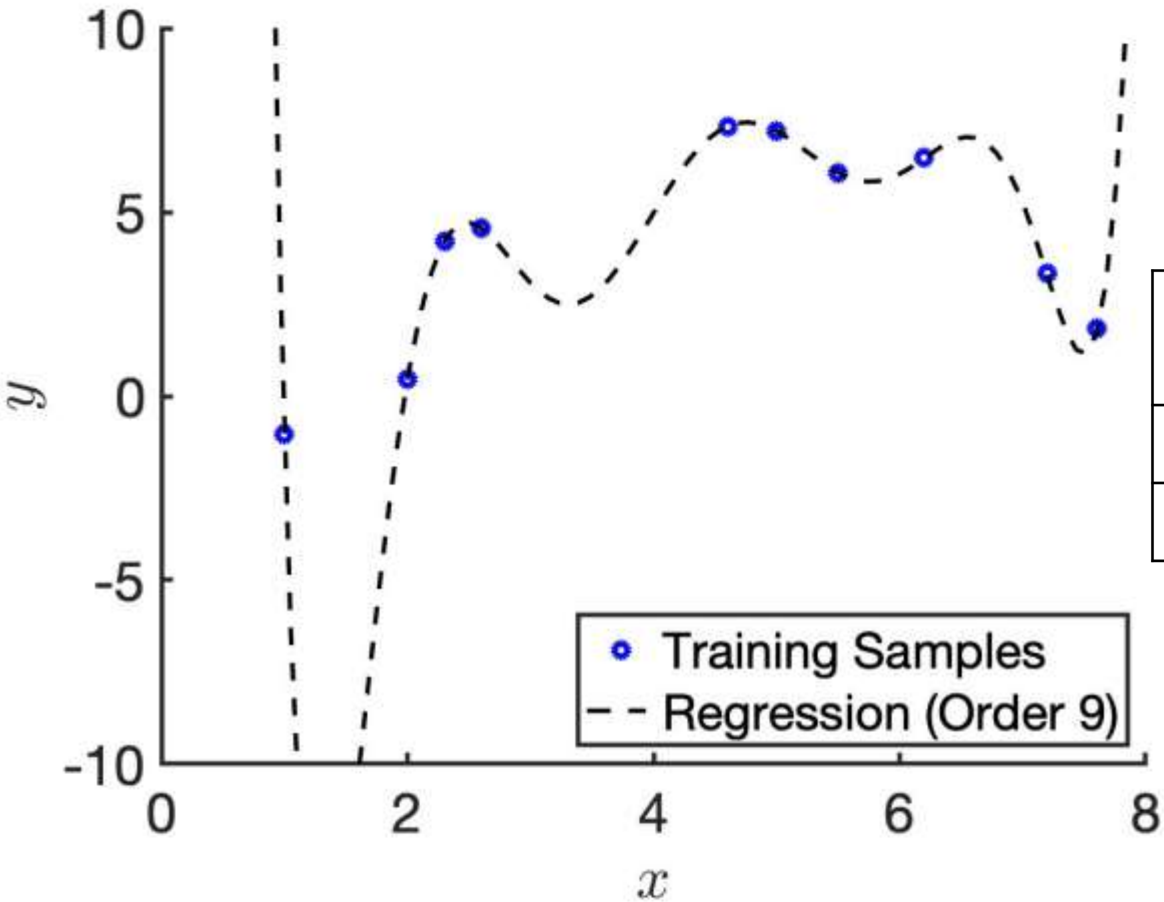
$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{Pw} - \mathbf{y})^T (\mathbf{Pw} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}} \, \mathbf{Data\text{-}Loss}(\mathbf{w}) + \lambda \mathbf{Regularization}(\mathbf{w})$$

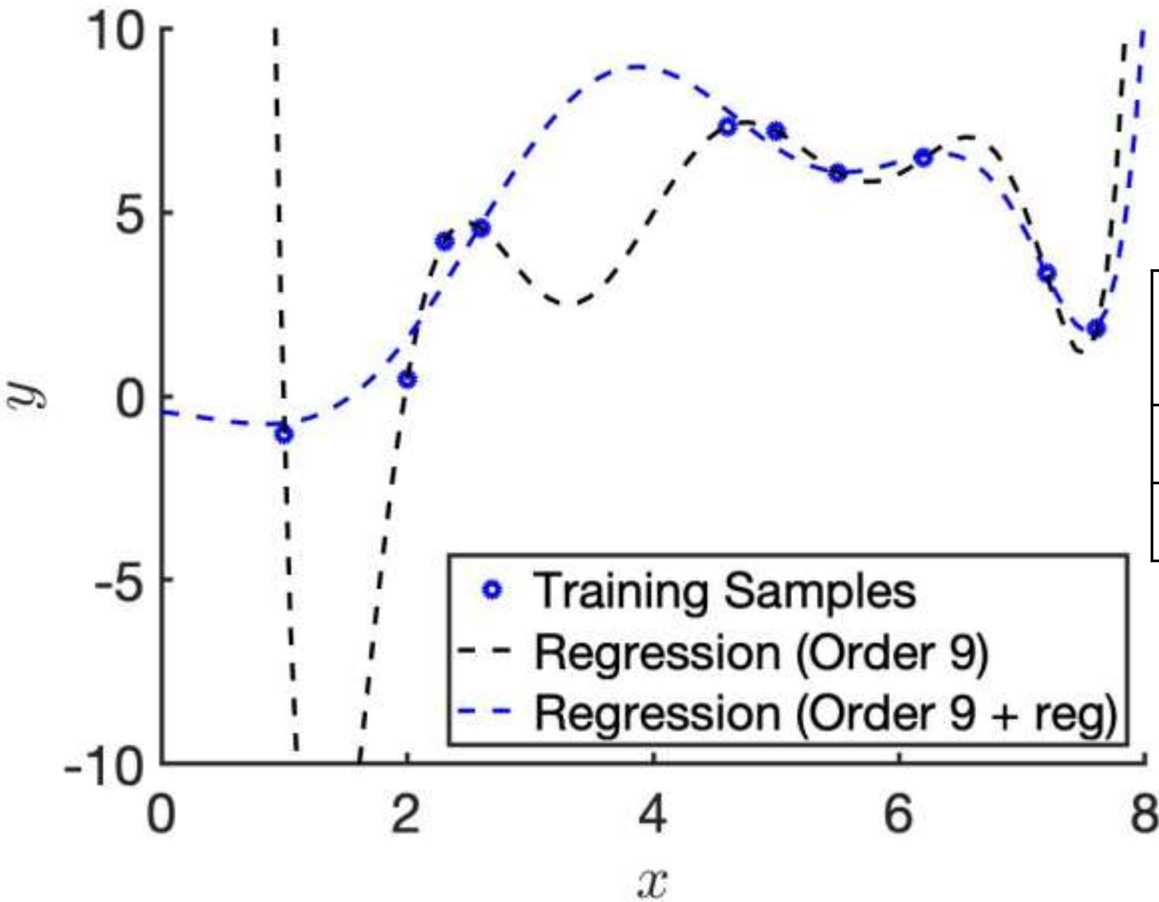
- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models

Regularization Example



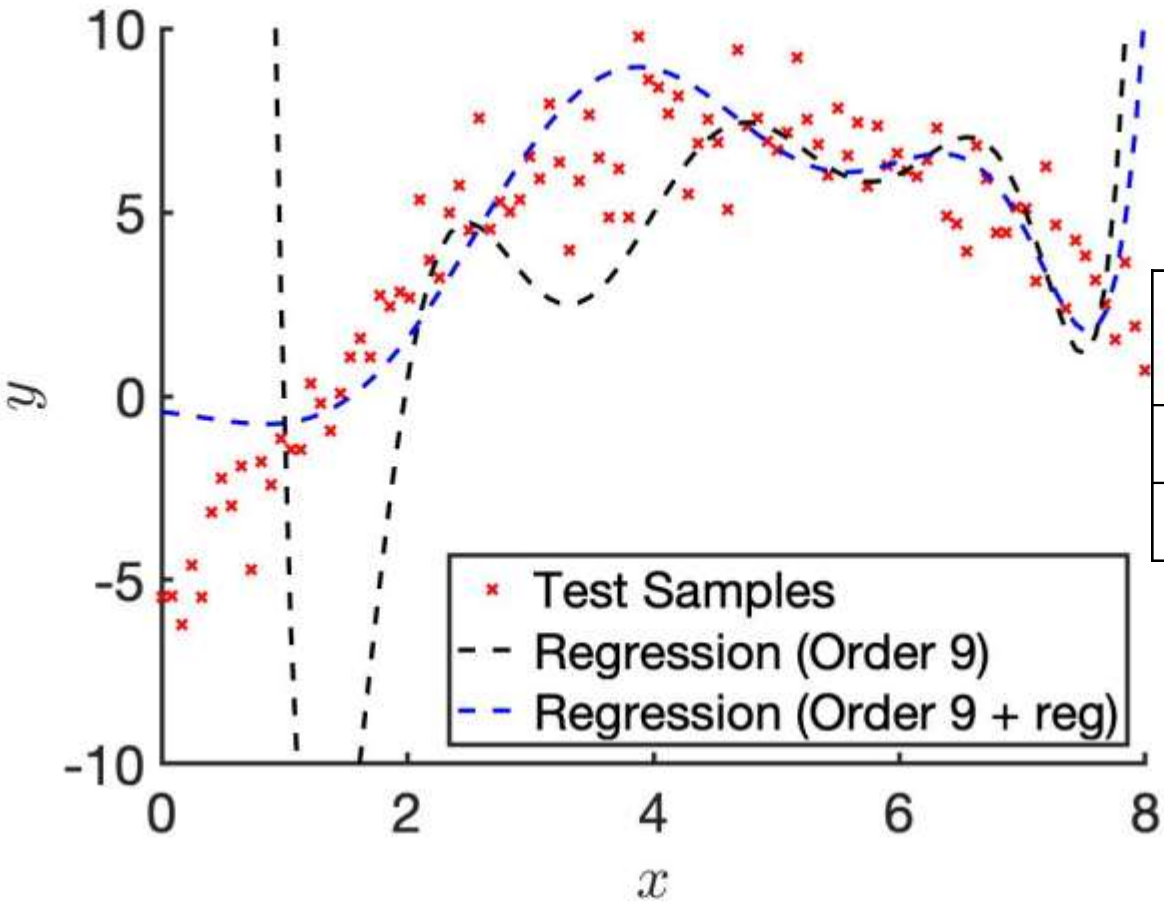
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Regularization Example



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9, $\lambda = 1$	Good	

Regularization Example

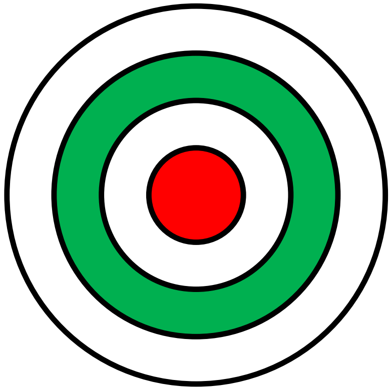


	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9, λ = 1	Good	Good

Questions?

Bias versus Variance

- Suppose we are trying to predict red target below:

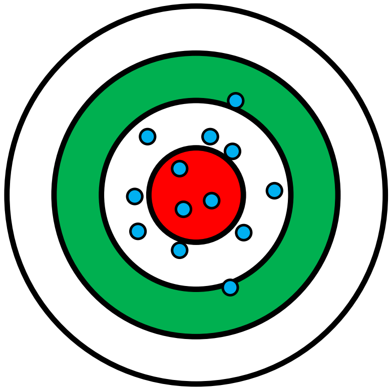


Bias versus Variance

- Suppose we are trying to predict red target below:

High Variance

Low Bias

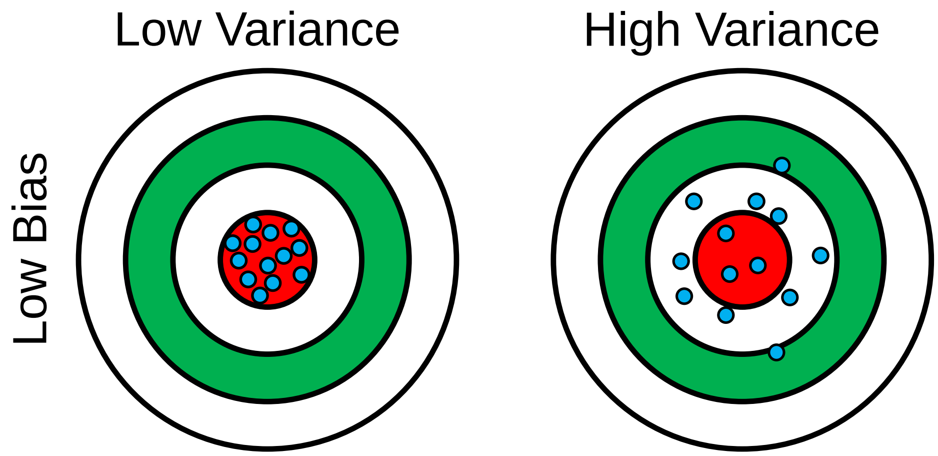


Low Bias: blue predictions on average close to red target
High Variance: large variability among predictions

Bias versus Variance

- Suppose we are trying to predict red target below:

Low Bias: blue predictions on average close to red target
Low Variance: low variability among predictions



Low Bias: blue predictions on average close to red target
High Variance: large variability among predictions

Bias versus Variance

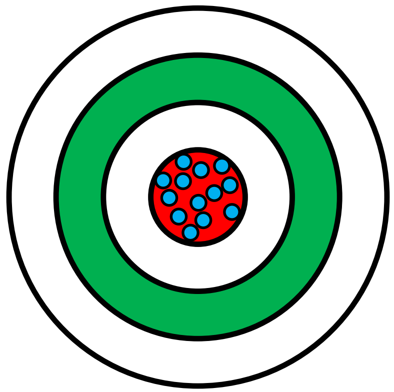
- Suppose we are trying to predict red target below:

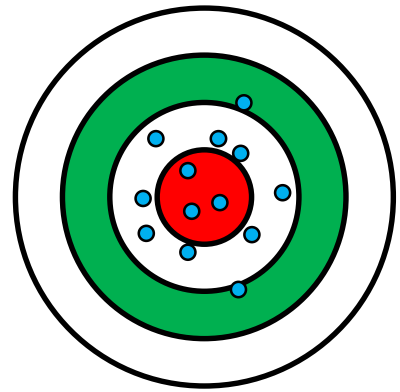
Low Variance

High Variance

Low Bias: blue predictions on average close to red target
Low Variance: low variability among predictions

Low Bias

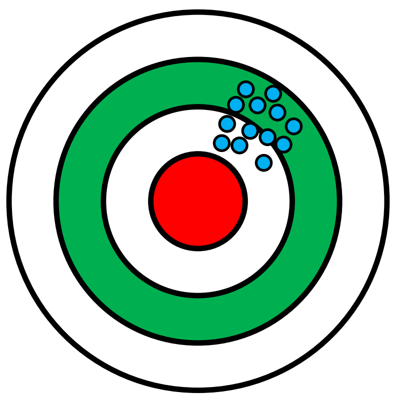




Low Bias: blue predictions on average close to red target
High Variance: large variability among predictions

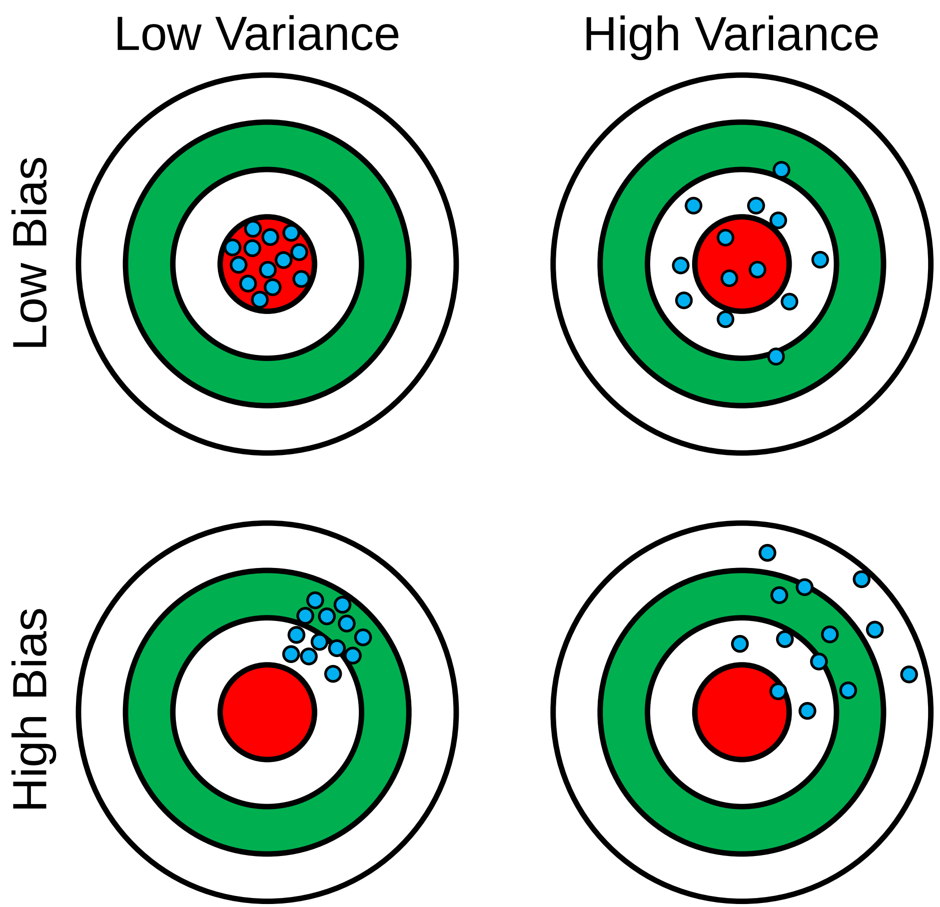
High Bias: blue predictions on average not close to red target
Low Variance: Low variability among predictions

High Bias



Bias versus Variance

- Suppose we are trying to predict red target below:



Low Bias: blue predictions on average close to red target
Low Variance: low variability among predictions

Low Bias: blue predictions on average close to red target
High Variance: large variability among predictions

High Bias: blue predictions on average not close to red target
Low Variance: Low variability among predictions

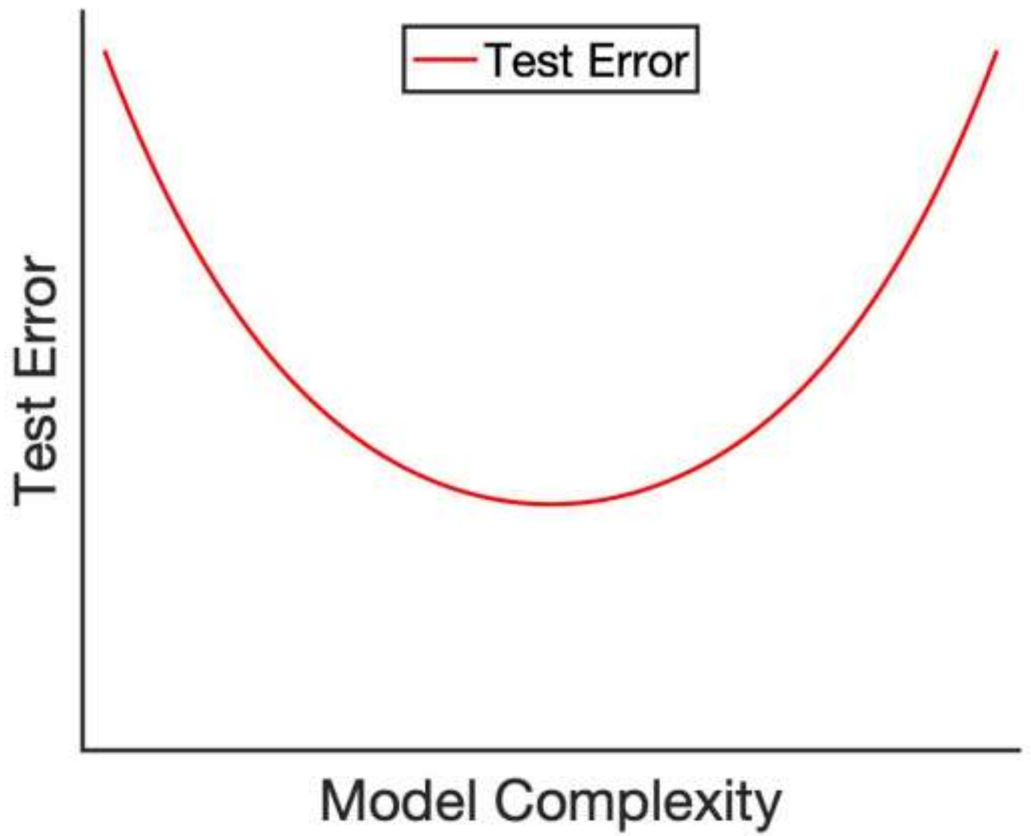
High Bias: blue predictions on average not close to red target
High Variance: high variability among predictions

Bias + Variance Trade off

- Test error = Bias Squared + Variance + Irreducible Noise

Bias + Variance Trade off

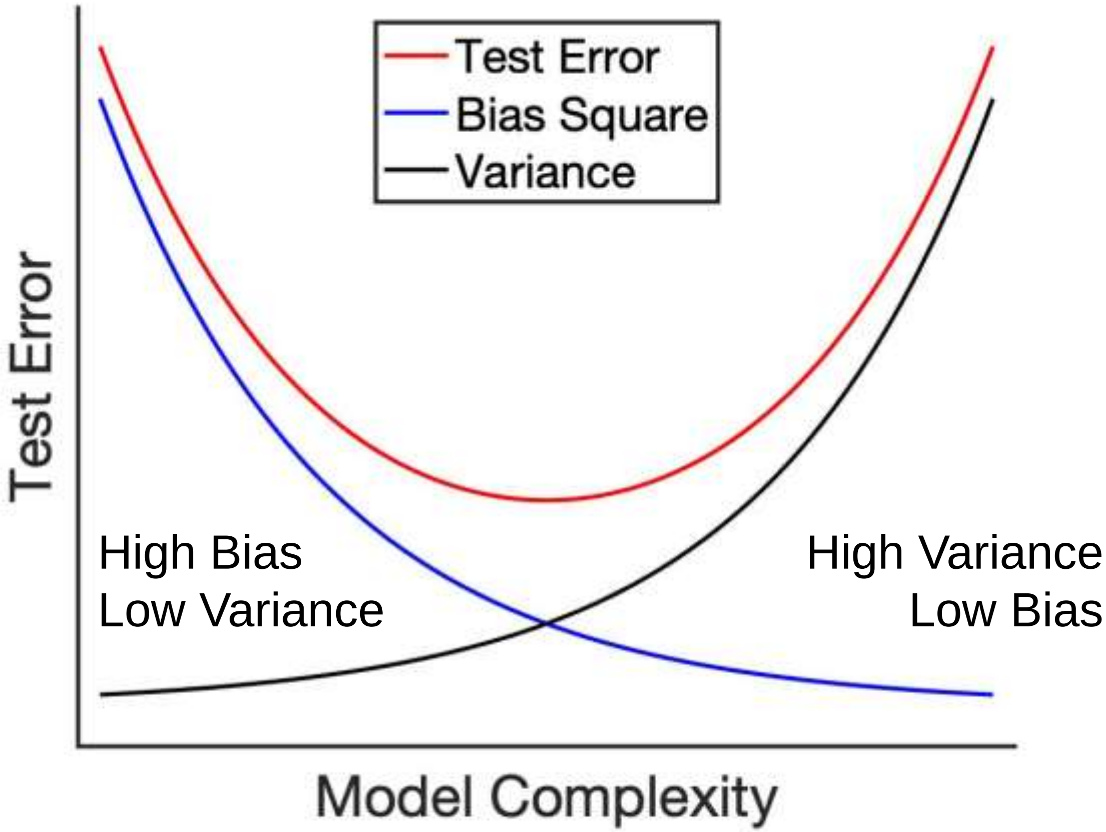
- Test error = Bias Squared + Variance + Irreducible Noise



or Number of Features

Bias + Variance Trade off

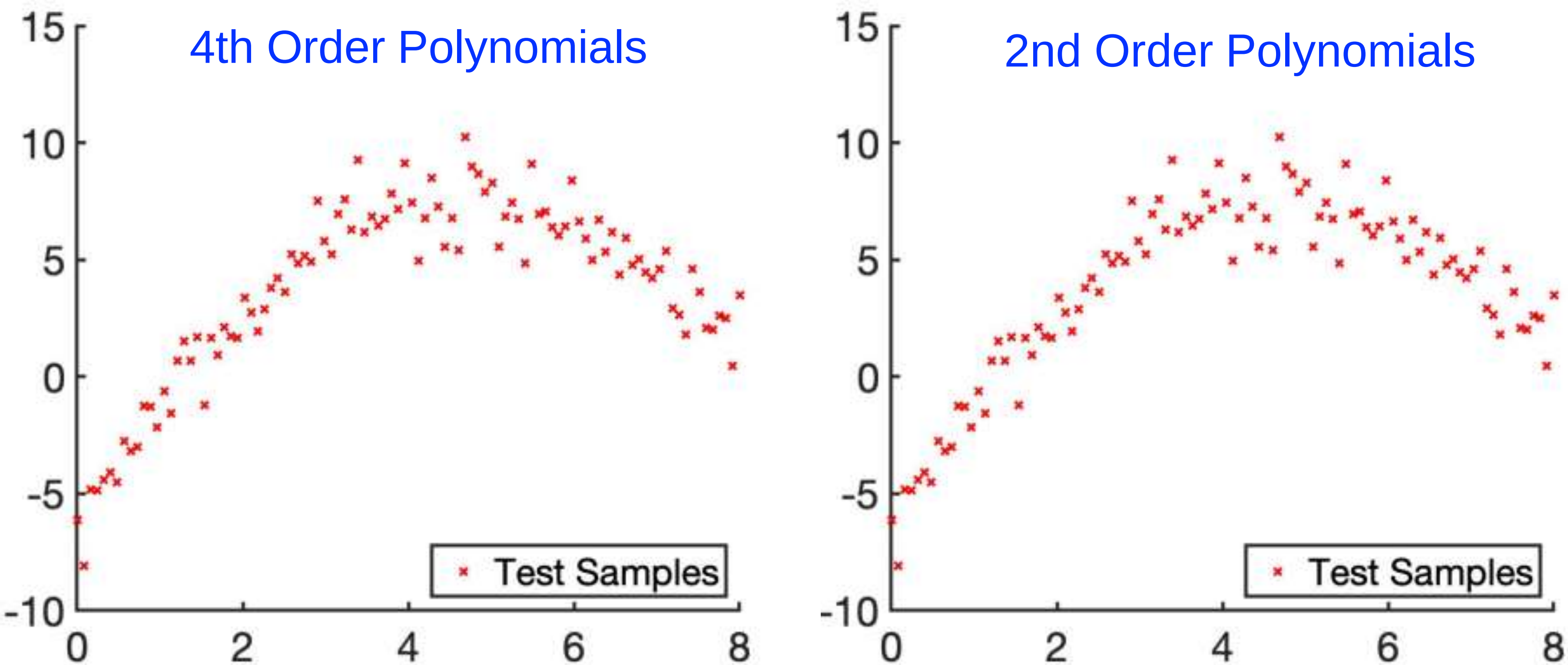
- Test error = Bias Squared + Variance + Irreducible Noise



or Number of Features

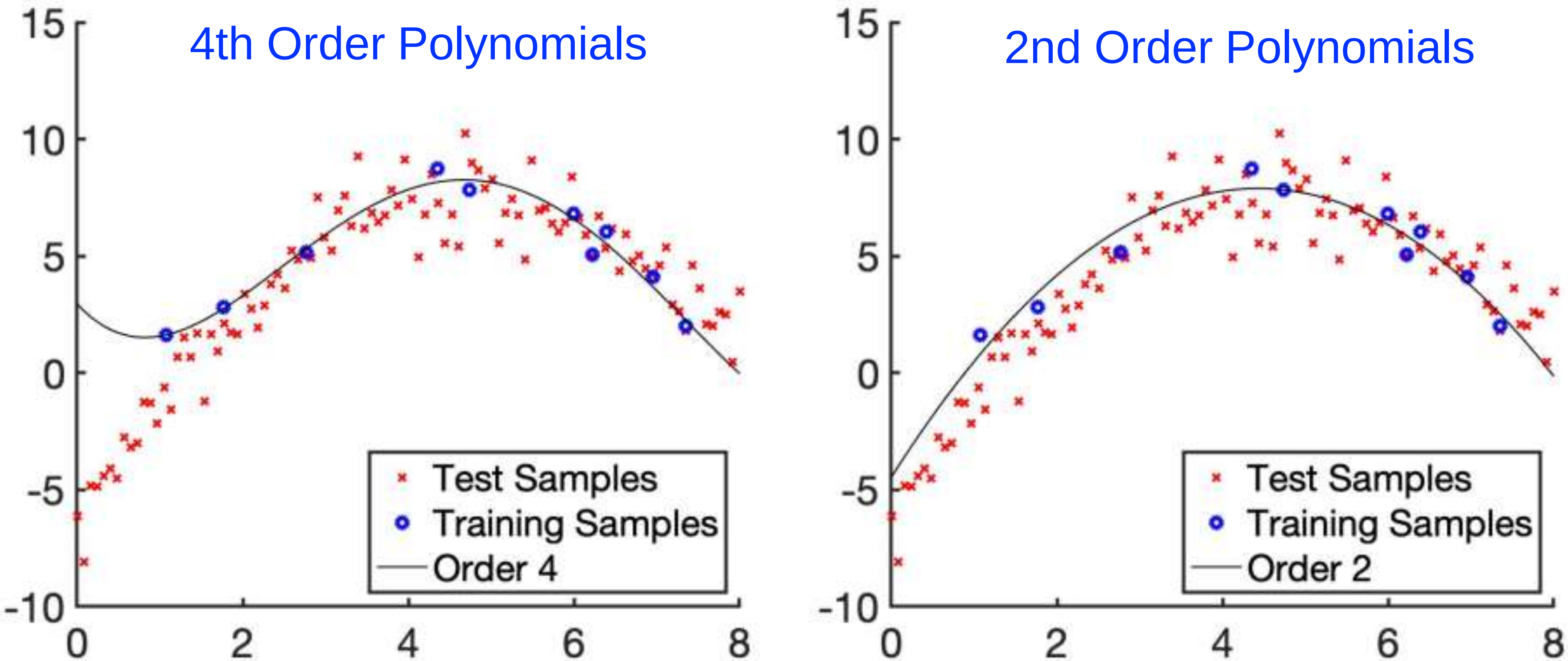
Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time



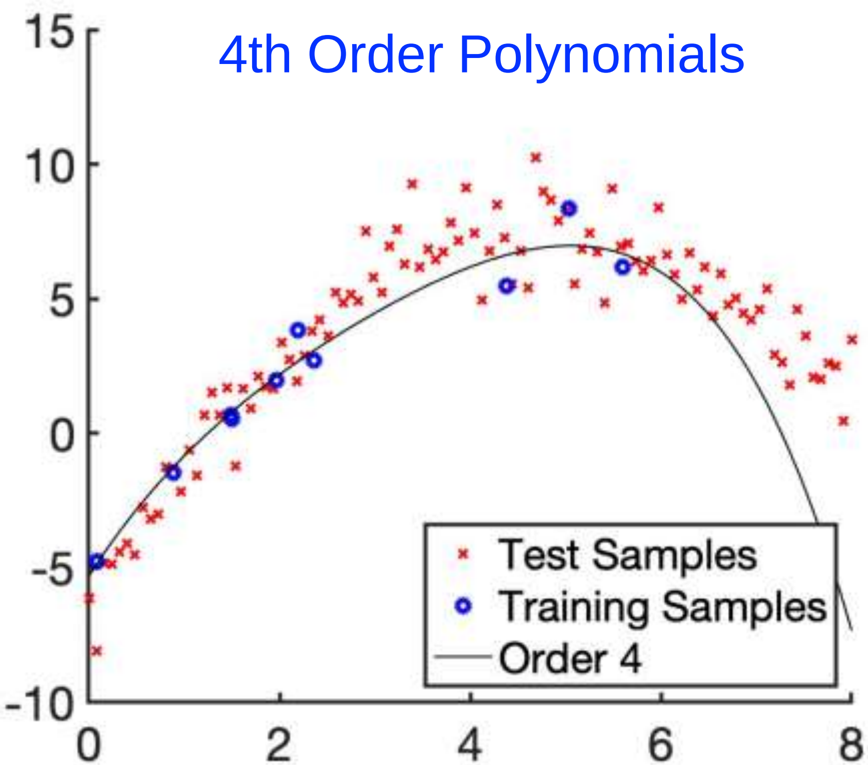
Bias + Variance Example

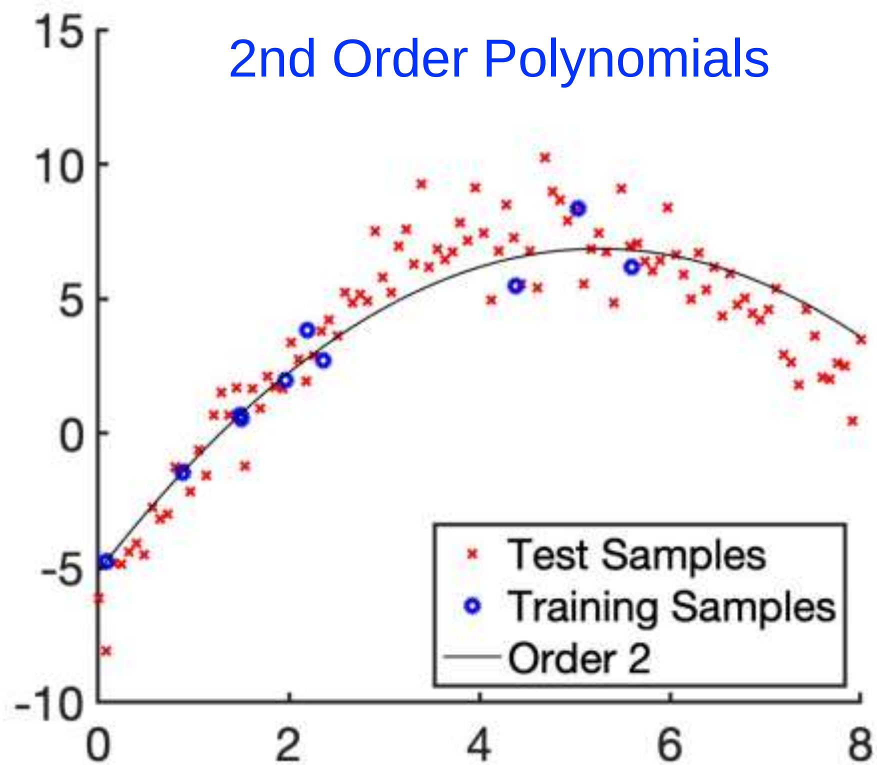
- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial



Bias + Variance Example

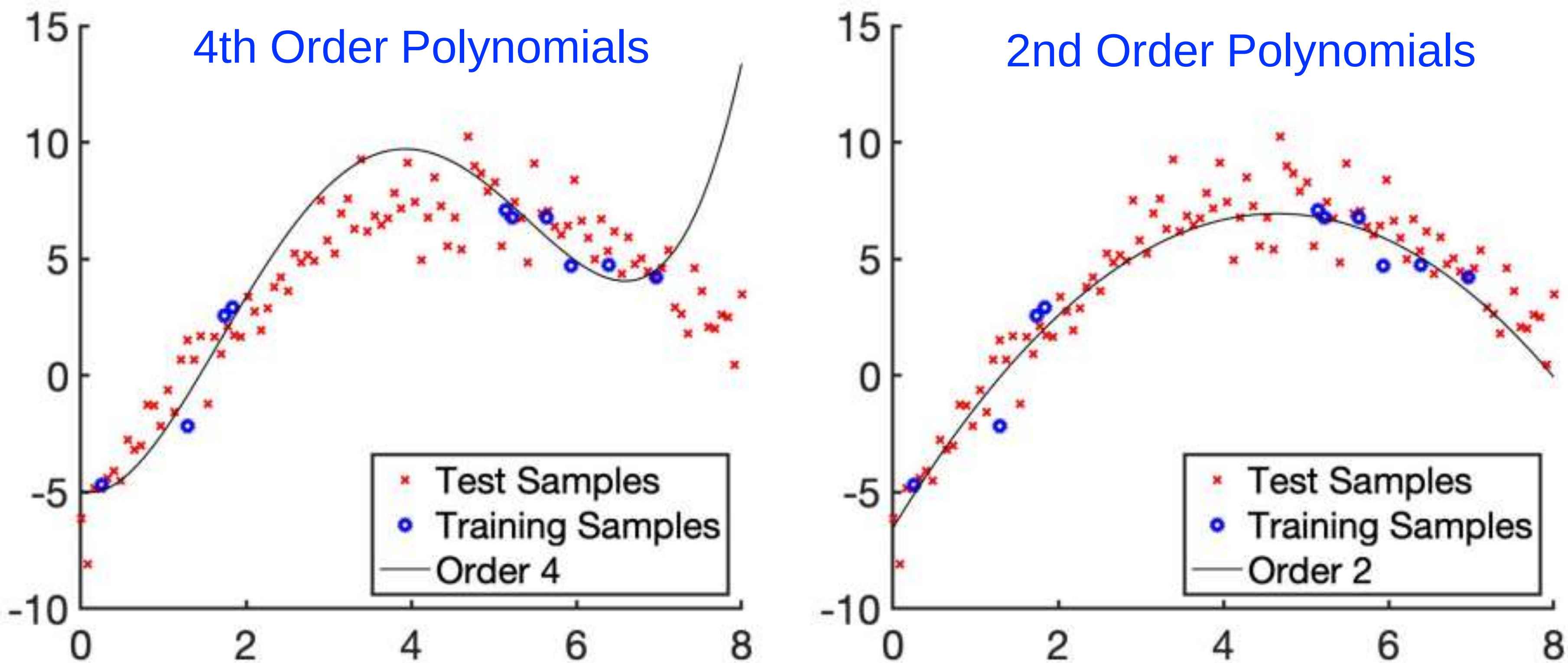
- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial





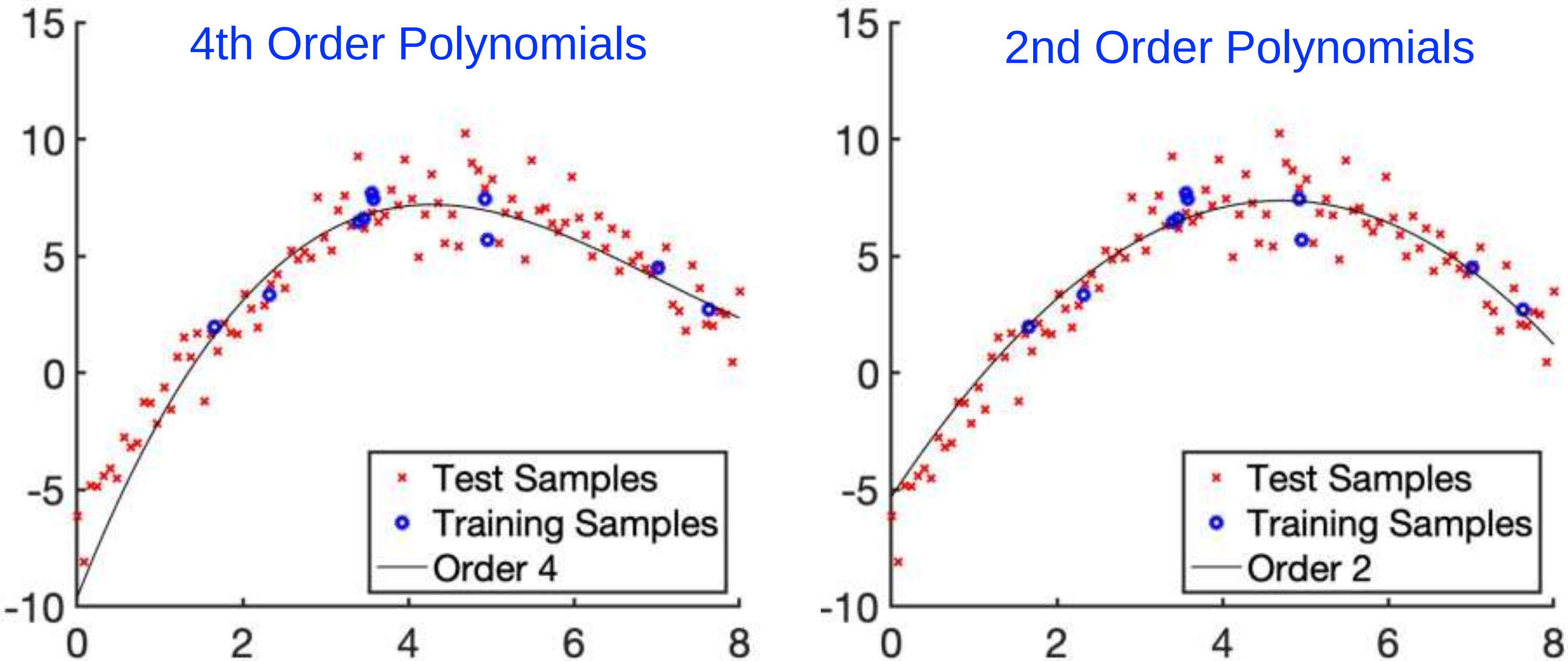
Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial



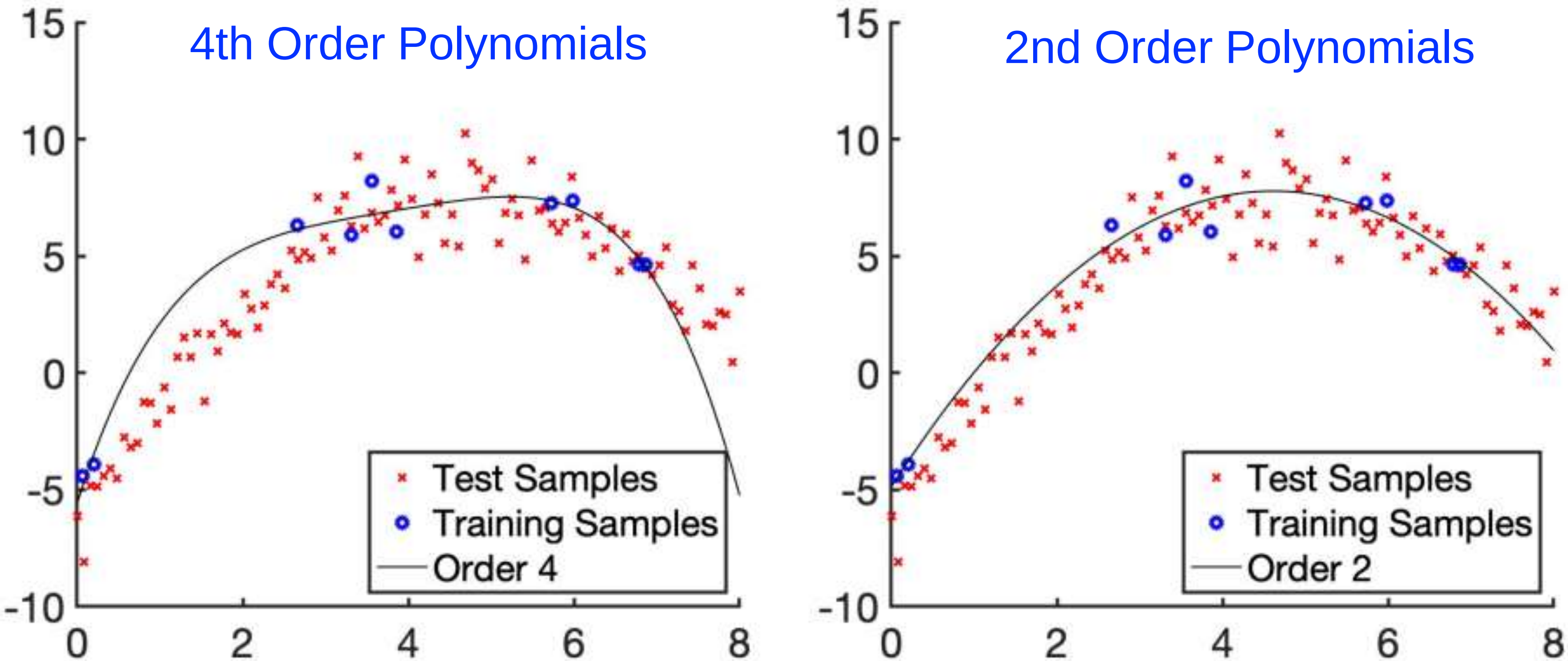
Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial



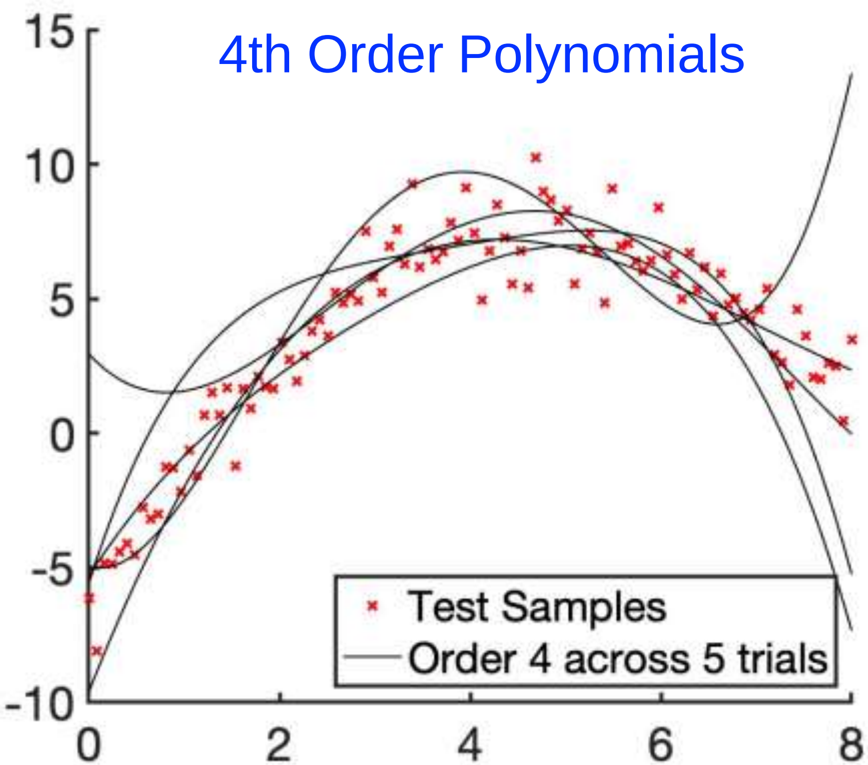
Bias + Variance Example

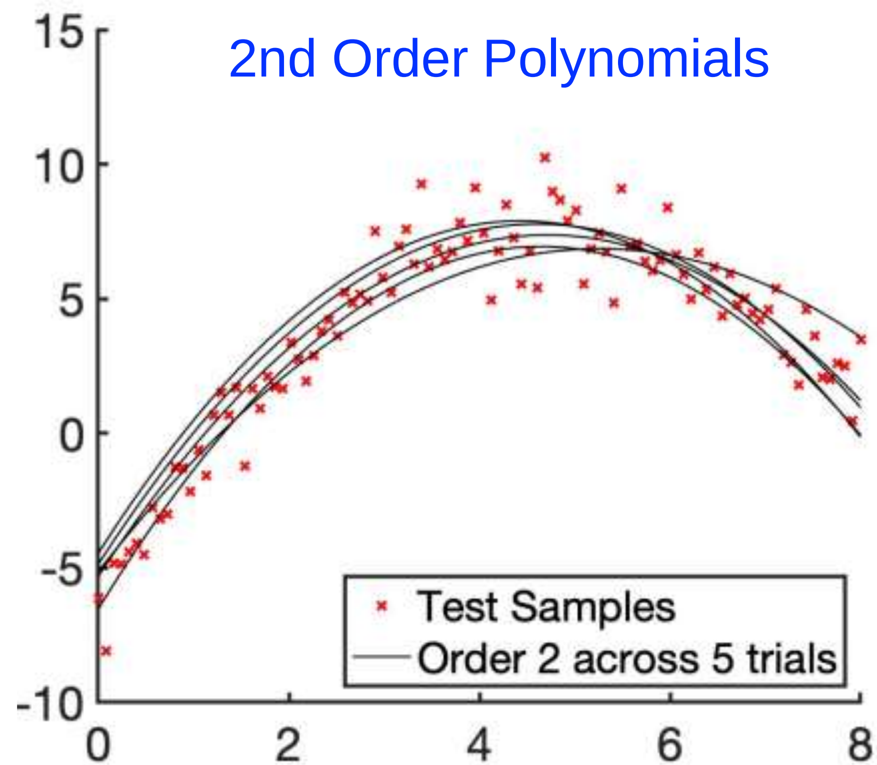
- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial



Bias + Variance Example

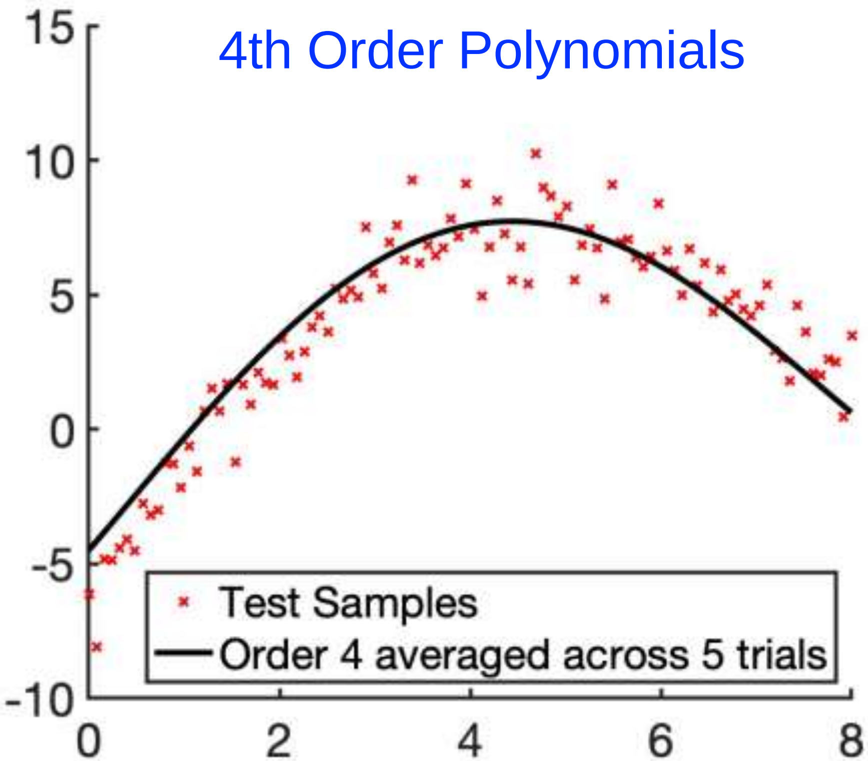
- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance
- Fit with order 4 polynomial: high variance

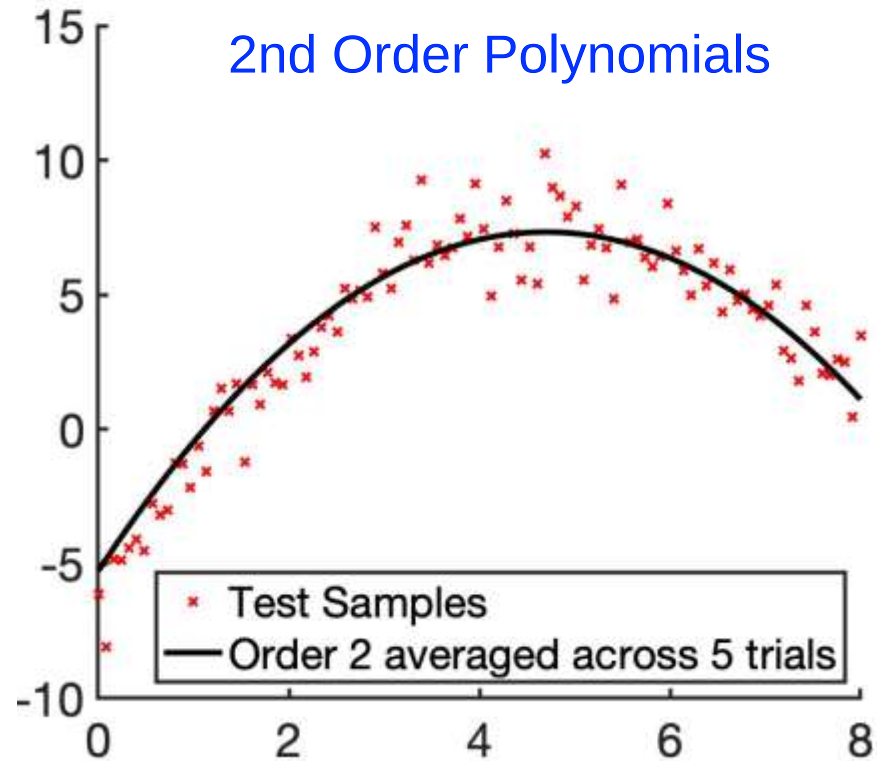




Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance, low bias
- Fit with order 4 polynomial: high variance, low bias

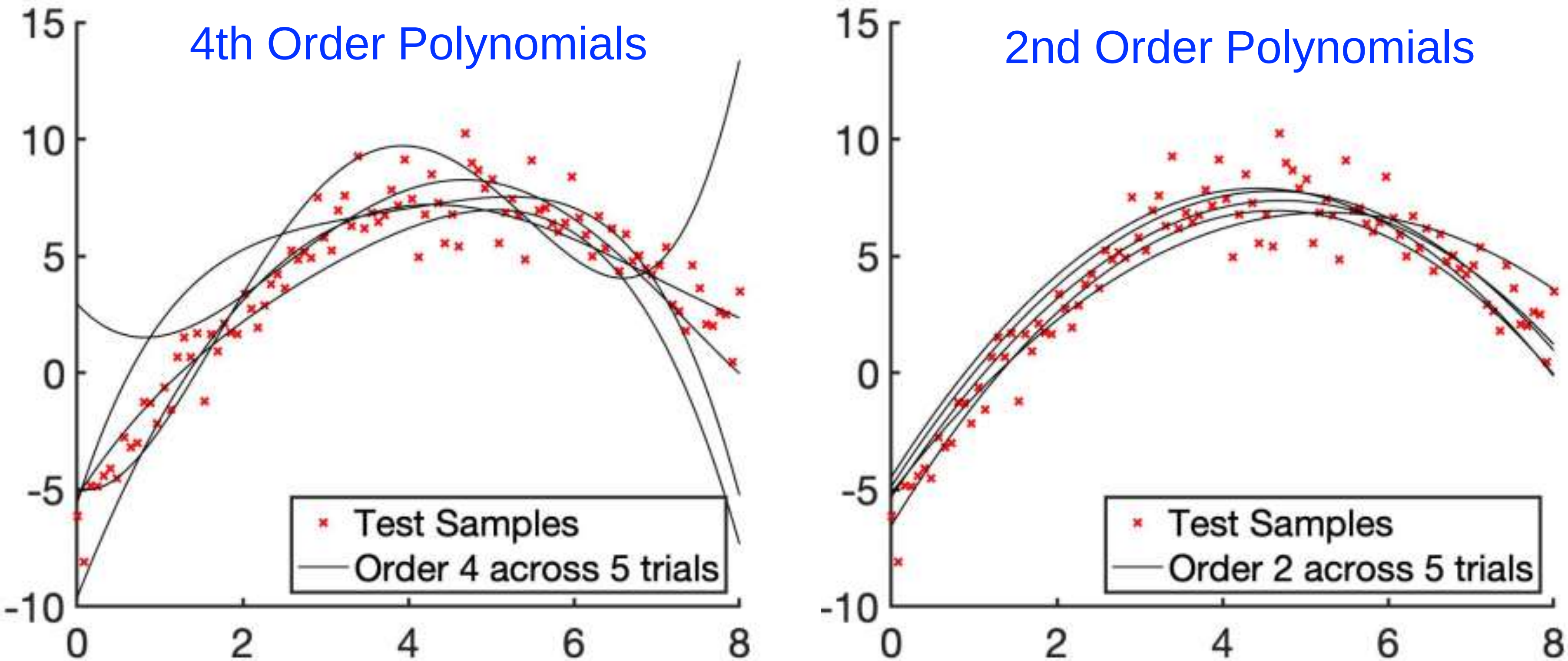




Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance, low bias
- Fit with order 4 polynomial: high variance, low bias

Order 2 Achieves Lower Test Error



Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise

Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise
- Suppose $y(x) = f(x) + \epsilon$, where f is deterministic & ϵ is random with mean 0 & variance σ^2.

Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise
- Suppose $y(x) = f(x) + \epsilon$, where f is deterministic & ϵ is random with mean 0 & variance σ^2.
- In previous example, we repeated our training 5 times. Each time, we randomly pick 10 different training samples. So training set D is random & there is a probability distribution function $p(D)$ associated with D.

Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise
- Suppose $y(x) = f(x) + \epsilon$, where f is deterministic & ϵ is random with mean 0 & variance σ^2.
- In previous example, we repeated our training 5 times. Each time, we randomly pick 10 different training samples. So training set D is random & there is a probability distribution function $p(D)$ associated with D.
- Using the training data D, we learn a function $\hat{f}_D(x)$. Note dependence of $\hat{f}_D$ on D because $\hat{f}_D$ will change when our training data changes

Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise
- Suppose $y(x) = f(x) + \epsilon$, where f is deterministic & ϵ is random with mean 0 & variance σ^2.
- In previous example, we repeated our training 5 times. Each time, we randomly pick 10 different training samples. So training set D is random & there is a probability distribution function $p(D)$ associated with D.
- Using the training data D, we learn a function $\hat{f}_D(x)$. Note dependence of $\hat{f}_D$ on D because $\hat{f}_D$ will change when our training data changes
- In previous example, we average the predictions across 5 trials. Here, let's perform ∞ trials and average predictions $\hat{f}_D(x)$ across ∞ trials (with different D) resulting in $\hat{f}_{avg}(x) = E_D(\hat{f}_D(x))$

Bias-Variance Decomposition Theorem

- Test error = Bias Squared + Variance + Irreducible Noise

- Suppose $y(x) = f(x) + \epsilon$, where f is deterministic & ϵ is random with mean 0 & variance σ^2.

- In previous example, we repeated our training 5 times. Each time, we randomly pick 10 different training samples. So training set D is random & there is a probability distribution function $p(D)$ associated with D.

- Using the training data D, we learn a function $\hat{f}_D(x)$. Note dependence of $\hat{f}_D$ on D because $\hat{f}_D$ will change when our training data changes

- In previous example, we average the predictions across 5 trials. Here, let's perform ∞ trials and average predictions $\hat{f}_D(x)$ across ∞ trials (with different D) resulting in $\hat{f}_{avg}(x) = E_D(\hat{f}_D(x))$

- E_D is expectation with respect to $p(D)$: think of this as averaging across ∞ trials

Bias-Variance Decomposition Theorem

- Think of E_D as averaging across ∞ trials

- See proof in extra uploaded material (won't be tested)

Bias-Variance Decomposition Theorem

- Think of E_D as averaging across ∞ trials
- According to the Bias-Variance Decomposition Theorem, the mean squared error for a new test sample x is given by

$$E_D\left[\left(y(x) - \underbrace{\hat{f}_D(x)}_{\hat{y}_{test}}\right)^2\right]$$

- See proof in extra uploaded material (won't be tested)

Bias-Variance Decomposition Theorem

- Think of E_D as averaging across ∞ trials
- According to the Bias-Variance Decomposition Theorem, the mean squared error for a new test sample x is given by

$$E_D\left[\left(y(x)-\hat{f}_D(x)\right)^2\right]=\text{Bias}(\hat{f})^2+\text{Var}(\hat{f})+\sigma^2,$$

- See proof in extra uploaded material (won't be tested)

Bias-Variance Decomposition Theorem

- Think of E_D as averaging across ∞ trials
- According to the Bias-Variance Decomposition Theorem, the mean squared error for a new test sample x is given by

$$E_D\left[\left(y(x) - \hat{f}_D(x)\right)^2\right] = \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \sigma^2,$$

where

$$\text{Bias}(\hat{f}) = \hat{f}_{avg}(x) - f(x),$$

- See proof in extra uploaded material (won't be tested)

Bias-Variance Decomposition Theorem

- Think of E_D as averaging across ∞ trials
- According to the Bias-Variance Decomposition Theorem, the mean squared error for a new test sample x is given by

$$E_D\left[\left(y(x) - \hat{f}_D(x)\right)^2\right] = \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \sigma^2,$$

where

$$\text{Bias}(\hat{f}) = \hat{f}_{avg}(x) - f(x),$$

and

$$\text{Var}(\hat{f}) = E_D\left[\left(\hat{f}_D(x) - \hat{f}_{avg}(x)\right)^2\right]$$

- See proof in extra uploaded material (won't be tested)

Numerical Example of Bias and Variance Tradeoff I

- We have randomly sampled a training set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ from a probability distribution $p(D)$.
- Using D, we training a regression model to predict y from $\mathbf{x}$.
- For simplicity, say that $\mathbf{x}$ and $\mathbf{x}_i$ are one-dimensional, so we write them as x and x_i for brevity.
- We repeat this process of sampling the dataset from $p(D)$ a total of 4 times, resulting in 4 trained models

$$\hat{f}_{D_1}, \hat{f}_{D_2}, \hat{f}_{D_3}, \hat{f}_{D_4}$$

  based on the datasets D_1, D_2, D_3, D_4.
- We have a new test sample $x_{\text{new}} = 3$ whose (unknown) target is $y_{\text{new}} = f(x_{\text{new}}) = 7.5$.
- The function $f(x)$ represents the real/true relation between y and x.

Numerical Example of Bias and Variance Tradeoff II

- Suppose now the predictions of x_{new} from $\hat{f}_{D_1}, \hat{f}_{D_2}, \hat{f}_{D_3}, \hat{f}_{D_4}$ are $\hat{y}_{\text{new},1} = 6$, $\hat{y}_{\text{new},2} = 7$, $\hat{y}_{\text{new},3} = 7$, $\hat{y}_{\text{new},4} = 8$ respectively. That is

$$
\begin{aligned}
\hat{y}_{\text{new},1} &= \hat{f}_{D_1}(x_{\text{new}}) = \hat{f}_{D_1}(3) = 6 \\
\hat{y}_{\text{new},2} &= \hat{f}_{D_2}(x_{\text{new}}) = \hat{f}_{D_2}(3) = 7 \\
\hat{y}_{\text{new},3} &= \hat{f}_{D_3}(x_{\text{new}}) = \hat{f}_{D_3}(3) = 7 \\
\hat{y}_{\text{new},4} &= \hat{f}_{D_4}(x_{\text{new}}) = \hat{f}_{D_4}(3) = 8
\end{aligned}
$$

- What is the bias and variance based on these models?

Numerical Example of Bias and Variance Tradeoff III

- Average prediction

$$\hat{f}_{\text{avg}}(x_{\text{new}}) = \hat{f}_{\text{avg}}(3) = \frac{1}{4}(6+7+7+8) = 7.$$

- Bias

$$\text{Bias} = \hat{f}_{\text{avg}}(x_{\text{new}}) - f(x_{\text{new}}) = \hat{f}_{\text{avg}}(x_{\text{new}}) - y_{\text{new}} = 7 - 7.5 = -\frac{1}{2}.$$

- Variance

$$\begin{aligned} \text{Variance} &= E_D\left[\left(\hat{f}_{\text{avg}}(x_{\text{new}}) - \hat{f}_D(x_{\text{new}})\right)^2\right] \\ &= \frac{1}{4}\left((6-7)^2 + (7-7)^2 + (7-7)^2 + (8-7)^2\right)^2 = \frac{1}{2} \end{aligned}$$

Numerical Example of Bias and Variance Tradeoff IV

- If there is no irreducible noise, using the bias-variance formula,

$$\mathrm{MSE} = \mathrm{Bias}^2 + \mathrm{Variance} = \left(-\frac{1}{2}\right)^2 + \frac{1}{2} = \frac{3}{4}.$$

- Now, using actual definition of MSE

$$\begin{aligned}
\mathrm{MSE} &= E_D\left[\left(\hat{f}_D(x_{\mathrm{new}}) - f(x_{\mathrm{new}})\right)^2\right] \\
&= \frac{1}{4}\left((6-7.5)^2 + (7-7.5)^2 + (7-7.5)^2 + (8-7.5)^2\right)^{\cancel{2}} \\
&= \frac{3}{4}
\end{aligned}$$

- Voila!

Bias-Variance Python Demo

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn import datasets
```

```
# preparing the dataset into inputs (feature matrix) and outputs (target vector)
data = datasets.load_boston()  # fetch the data
X = data.data  # feature matrix
y = data.target  # target vector

# split the data into training and test samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

- Here, we are loading the Boston Housing dataset
- We split the raw dataset into 70% training and 30% testing in the last line
- Use .shape to check the dimensions of the various matrices!
- X_train.shape should give (354,13)

Bias-Variance Python Demo

```
def draw_bootstrap_sample(rng, X, y):
    sample_indices = np.arange(X.shape[0])
    bootstrap_indices = rng.choice(
        sample_indices, size=sample_indices.shape[0], replace=True
    )
    return X[bootstrap_indices], y[bootstrap_indices]
```

- Here, we are drawing various datasets from the given training set
- We are using a technique known as bootstrapping
- <u>**You don’t need to know this!**</u>
- This simulates the effect of …

$$\text{Think of } E_D \text{ as averaging across } \infty \text{ trials}$$

Bias-Variance Python Demo

```
def bias_variance_decomp(estimator, X_train, y_train, X_test, y_test, num_rounds=200, random_seed=20):
    rng = np.random.RandomState(random_seed)

    all_pred = []
    for i in range(num_rounds):
        # do bootstrap sampling, i.e., sampling with replacement
        X_boot, y_boot = draw_bootstrap_sample(rng, X_train, y_train)

        # fit a model on bootstrap samples and make prediction on test samples
        pred = estimator.fit(X_boot, y_boot).predict(X_test)
        all_pred.append(pred)

    all_pred = np.array(all_pred)

    # calculate MSE
    avg_mse = ((all_pred - y_test[None,:])**2).mean() # y_test[None,:] will reshape y_test from (N,) to (1,N)

    # average prediction of all bootstrap models on test set
    avg_predictions = np.mean(all_pred, axis=0)

    # calculate bias squared
    avg_bias = np.sum((avg_predictions - y_test) ** 2) / y_test.size

    # calculate variance
    avg_var = np.sum((avg_predictions - all_pred) ** 2) / all_pred.size

    return avg_mse, avg_bias, avg_var
```

$$E_D\left[\left(y(x)-\hat{f}_D(x)\right)^2\right]$$

$$\mathrm{Bias}(\hat{f})=\hat{f}_{avg}(x)-f(x),$$

$$\mathrm{Var}(\hat{f})=E_D\left[\left(\hat{f}_D(x)-\hat{f}_{avg}(x)\right)^2\right]$$

- The different samples of the training dataset D are collected in all_pred
- The mean-squared error is calculated in avg_mse
- The bias squared is calculated in avg_bias
- The variance is calculated in avg_var

Bias-Variance Python Demo

```
# define the model
model = LinearRegression()

# estimating the bias and variance
avg_mse, avg_bias, avg_var = bias_variance_decomp(model, X_train,
                                                  y_train, X_test,
                                                  y_test,
                                                  num_rounds=500,
                                                  random_seed=0)

# summary of the results
print('Average mean-squared error: %.3f' % avg_mse)
print('Average bias: %.3f' % avg_bias)
print('Average variance: %.3f' % avg_var)
```

```
Average mean-squared error: 27.360
Average bias: 26.064
Average variance: 1.296
```

- Here, we train a linear regression model.
- The MSE is indeed the squared bias plus the variance. Proof by python!
- There is no residual noise in this model.

Bias-Variance Python Demo

```
# define the model
model = Ridge(alpha=1)

# estimating the bias and variance
avg_mse, avg_bias, avg_var = bias_variance_decomp(model, X_train,
                                                  y_train, X_test,
                                                  y_test,
                                                  num_rounds=500,
                                                  random_seed=0)

# summary of the results
print('Average mean-squared error: %.3f' % avg_mse)
print('Average bias: %.3f' % avg_bias)
print('Average variance: %.3f' % avg_var)
model.coef_
```

```
Average mean-squared error: 27.375
Average bias: 26.133
Average variance: 1.242
```

- Here, we train a ridge regression model with lambda = 1.0 (it is alpha in python)
- Bias goes up, variance goes down as expected.
- MSE may go up or down.
- Try experimenting with different alpha's (or lambda's)!