Machine Learning I

Mohamed Hussien

Polynomial Regression

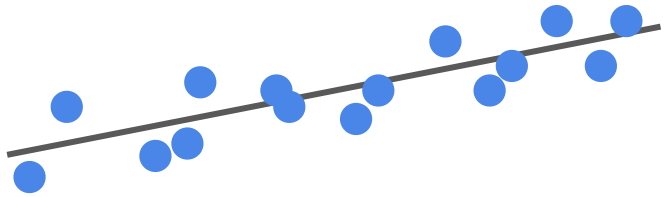
Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

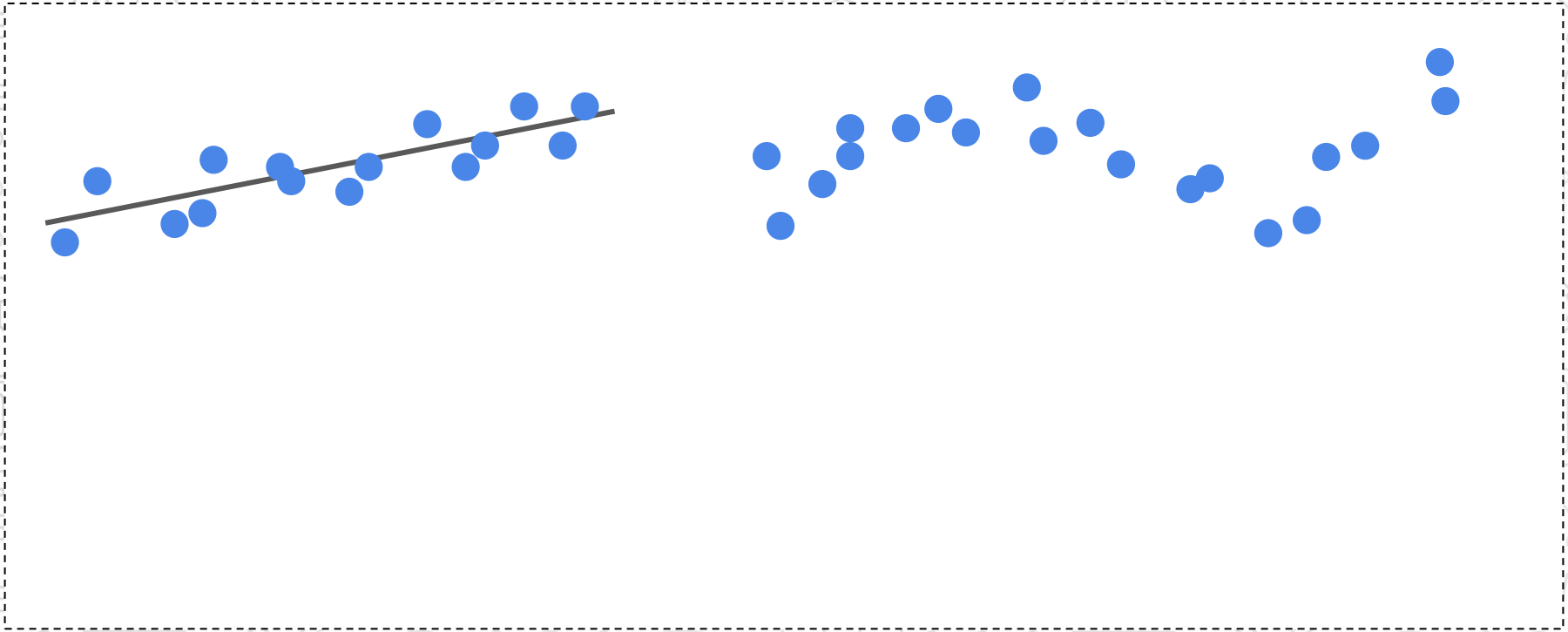
Polynomial Regression

Polynomial Regression

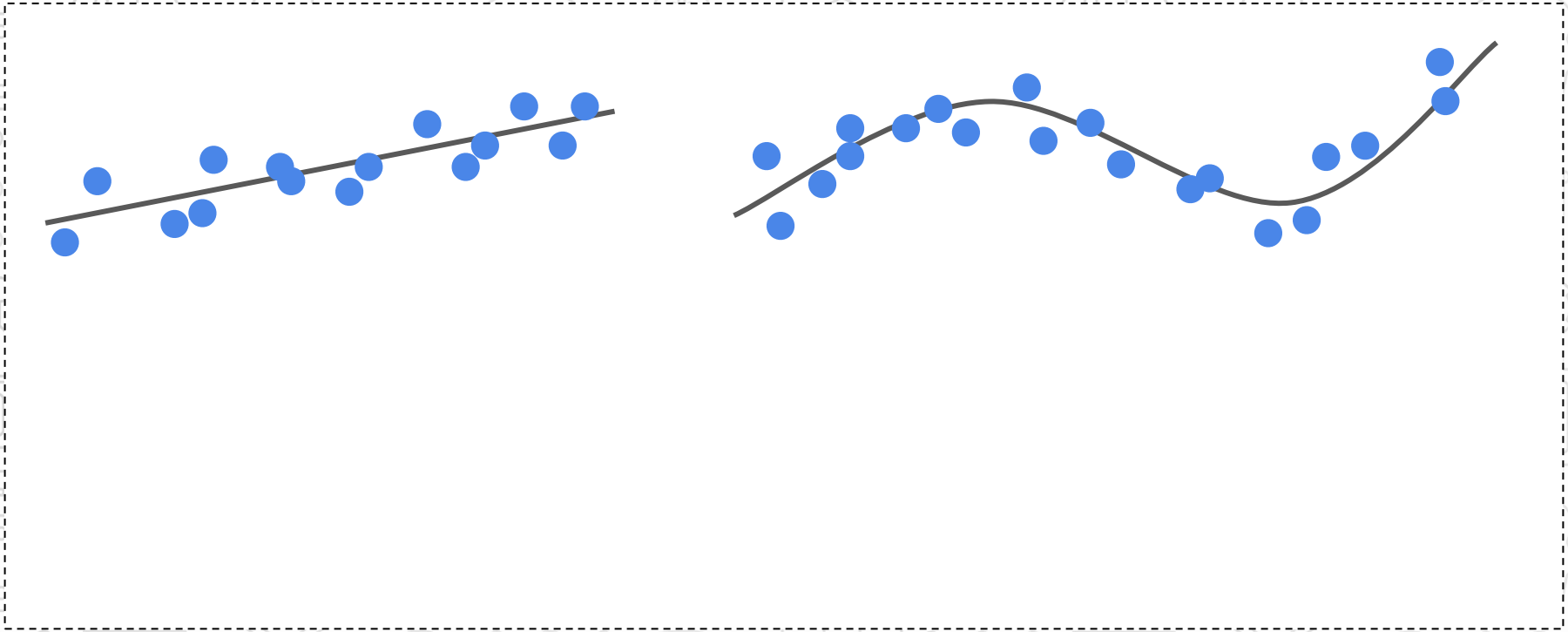
Polynomial Regression



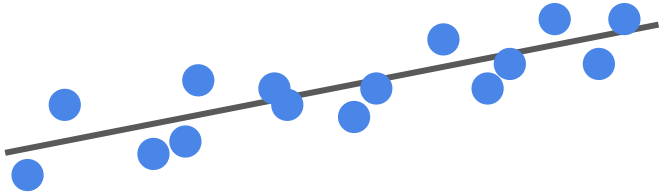
Polynomial Regression

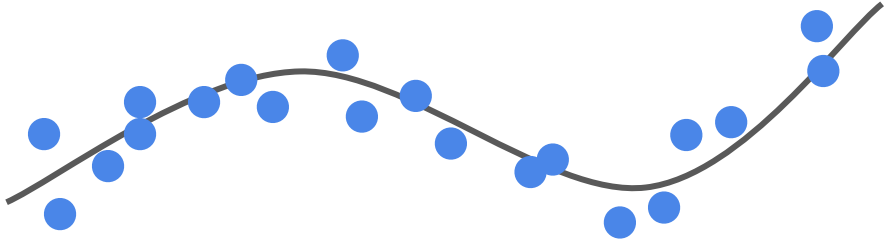


Polynomial Regression



Polynomial Regression

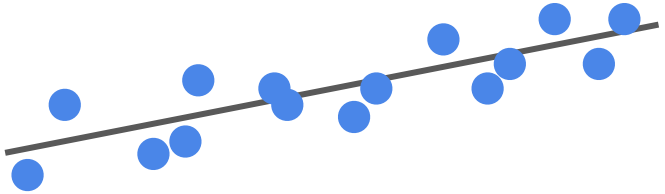


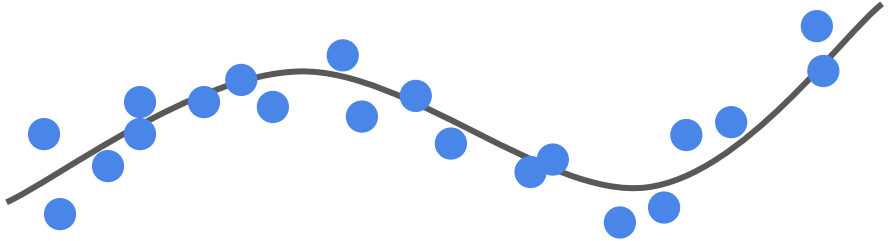


Examples of polynomials:

- $y = 4$
- $y = 3x + 2$
- $y = x^2 - 2x + 5$
- $y = 2x^3 + 8x^2 - 40$

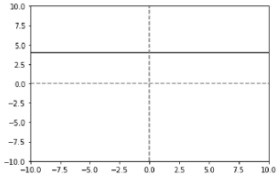
Polynomial Regression



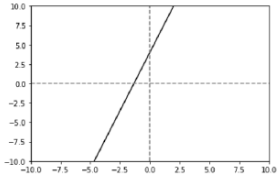


Examples of polynomials:

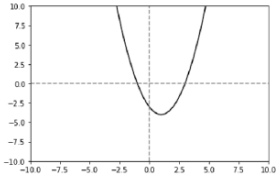
- $y = 4$
- $y = 3x + 2$
- $y = x^2 - 2x + 5$
- $y = 2x^3 + 8x^2 - 40$



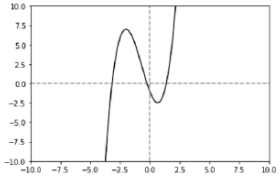
degree 0
$y = 4$



degree 1
$y = 3x + 4$

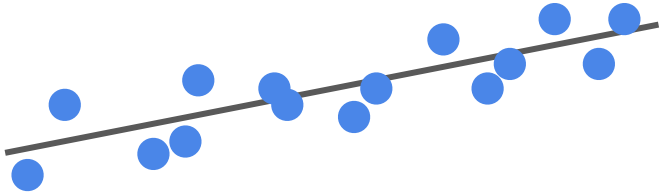


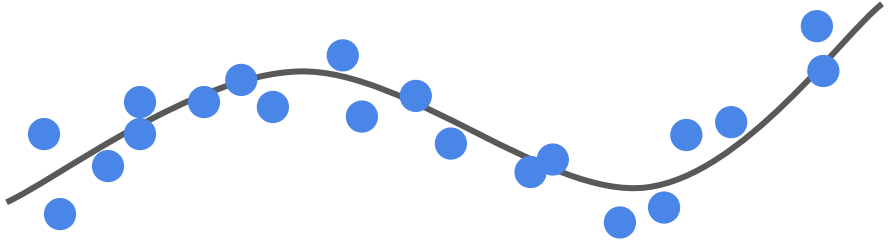
degree 2
$y = x^2 - 2x - 3$



degree 3
$y = x^3 + 2x^2 - 4x - 1$

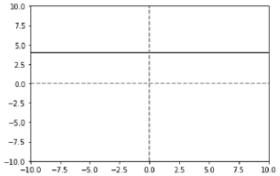
Polynomial Regression



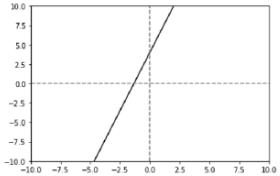


Examples of polynomials:

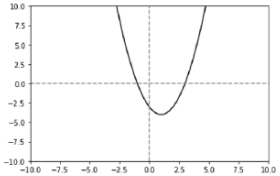
- $y = 4$
- $y = 3x + 2$
- $y = x^2 - 2x + 5$
- $y = 2x^3 + 8x^2 - 40$



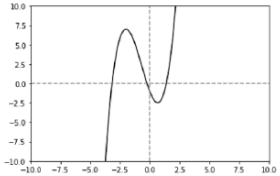
degree 0
$y = 4$



degree 1
$y = 3x + 4$



degree 2
$y = x^2 - 2x - 3$

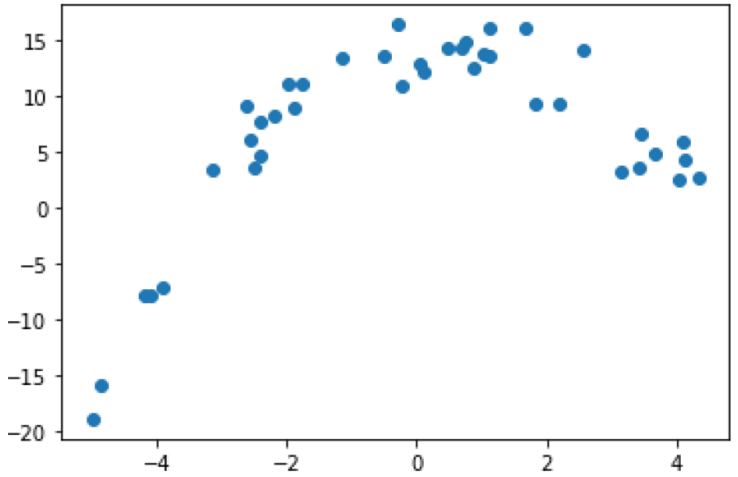


degree 3
$y = x^3 + 2x^2 - 4x - 1$

What degree of polynomial needed?

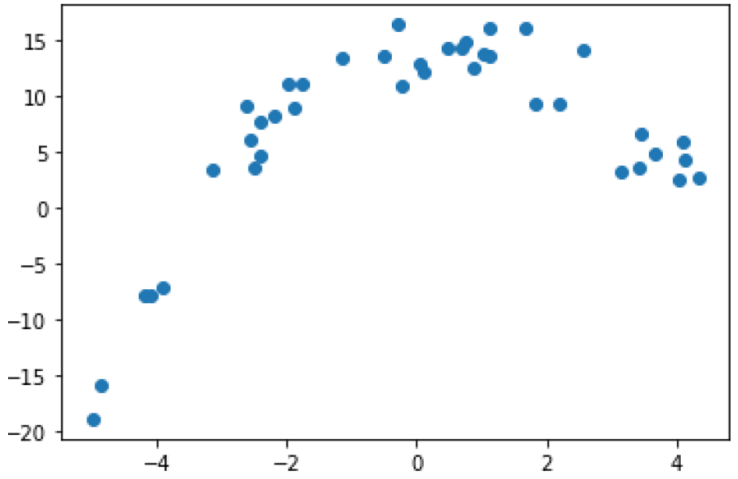
Transform Linear Regression to polynomial Regression

Transform Linear Regression to polynomial Regression



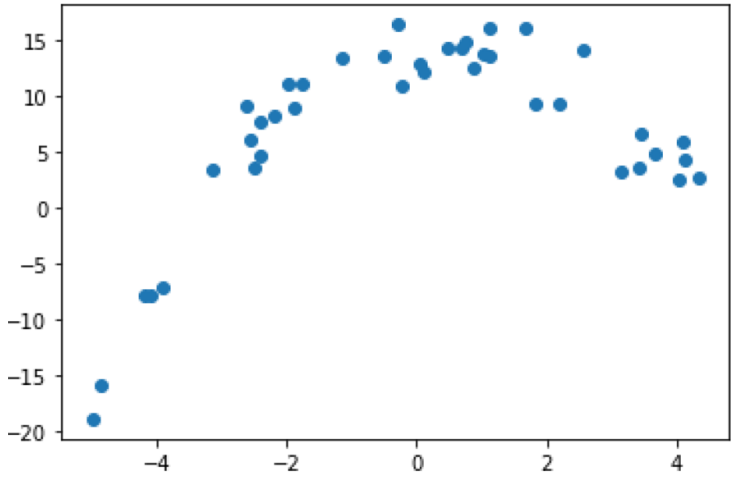
Transform Linear Regression to polynomial Regression

x	y
3.444218515250481 6	6.685961311021467
-2.410832497070366 3	4.690236225597948
0.112747213686085 42	12.205789026637378
-1.966872739210725 5	11.133217991032268



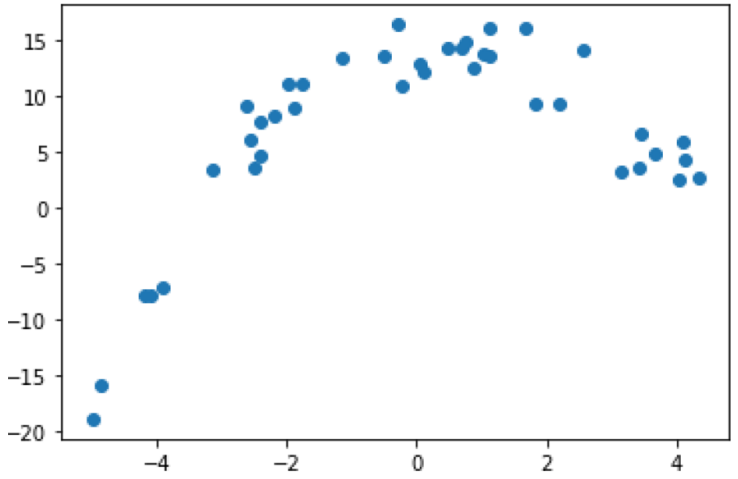
Transform Linear Regression to polynomial Regression

x	y	x^2	x^3	x^4
3.4442185152504816	6.685961311021467	11.862641180794233	40.85752839466433	140.72225578427518
-2.4108324970703663	4.690236225597948	5.812113328930538	-14.012031690041567	33.78066134833202
0.11274721368608542	12.205789026637378	0.012711934193975809	0.0014332351609316464	0.00016159327095197139
-1.9668727392107255	11.133217991032268	3.8685883722503025	-7.609021008606714	14.965975993910245



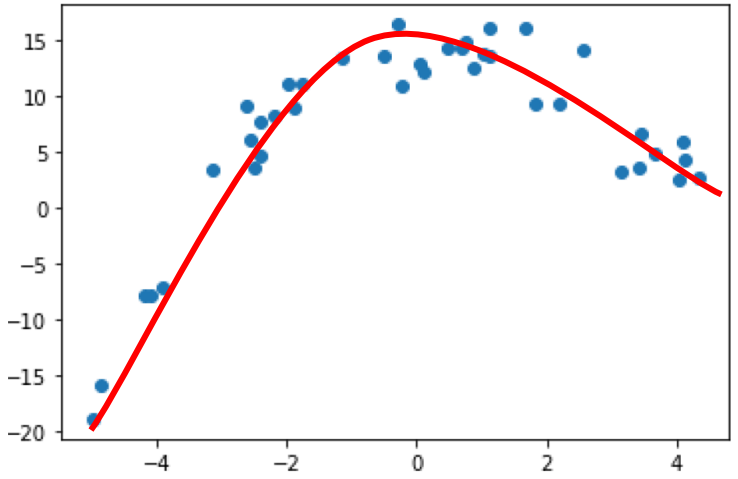
Transform Linear Regression to polynomial Regression

x	y	x^2	x^3	x^4
3.4442185152504816	6.685961311021467	11.862641180794233	40.85752839466433	140.72225578427518
-2.4108324970703663	4.690236225597948	5.812113328930538	-14.012031690041567	33.78066134833202
0.11274721368608542	12.205789026637378	0.012711934193975809	0.0014332351609316464	0.00016159327095197139
-1.9668727392107255	11.133217991032268	3.8685883722503025	-7.609021008606714	14.965975993910245



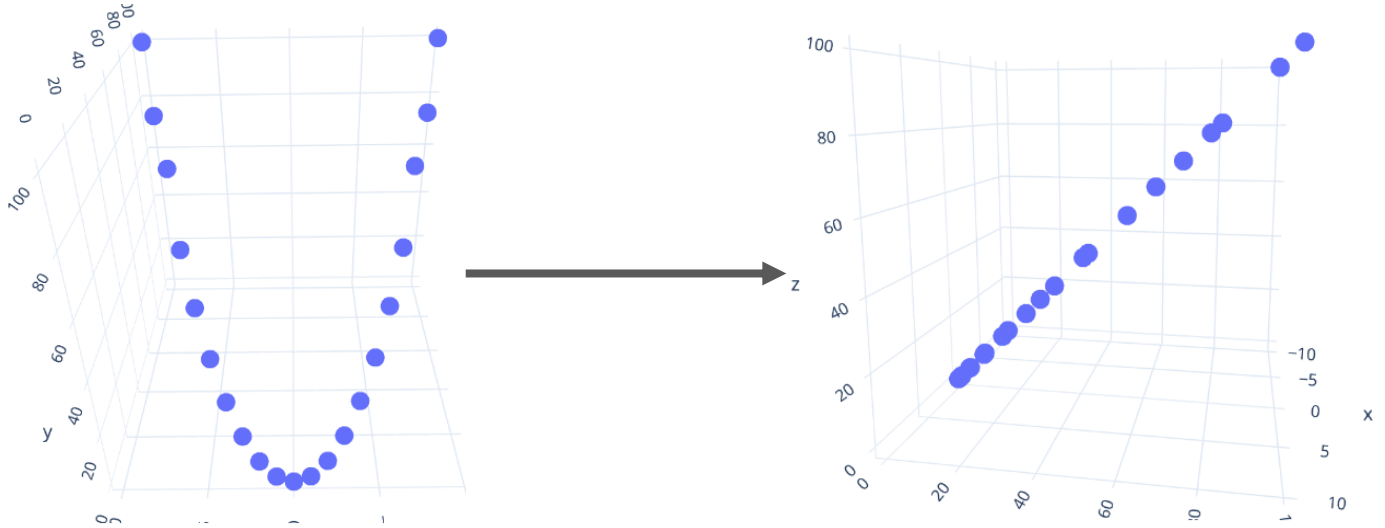
Transform Linear Regression to polynomial Regression

x	y	x^2	x^3	x^4
3.4442185152504816	6.685961311021467	11.862641180794233	40.85752839466433	140.72225578427518
-2.4108324970703663	4.690236225597948	5.812113328930538	-14.012031690041567	33.78066134833202
0.11274721368608542	12.205789026637378	0.012711934193975809	0.0014332351609316464	0.00016159327095197139
-1.9668727392107255	11.133217991032268	3.8685883722503025	-7.609021008606714	14.965975993910245



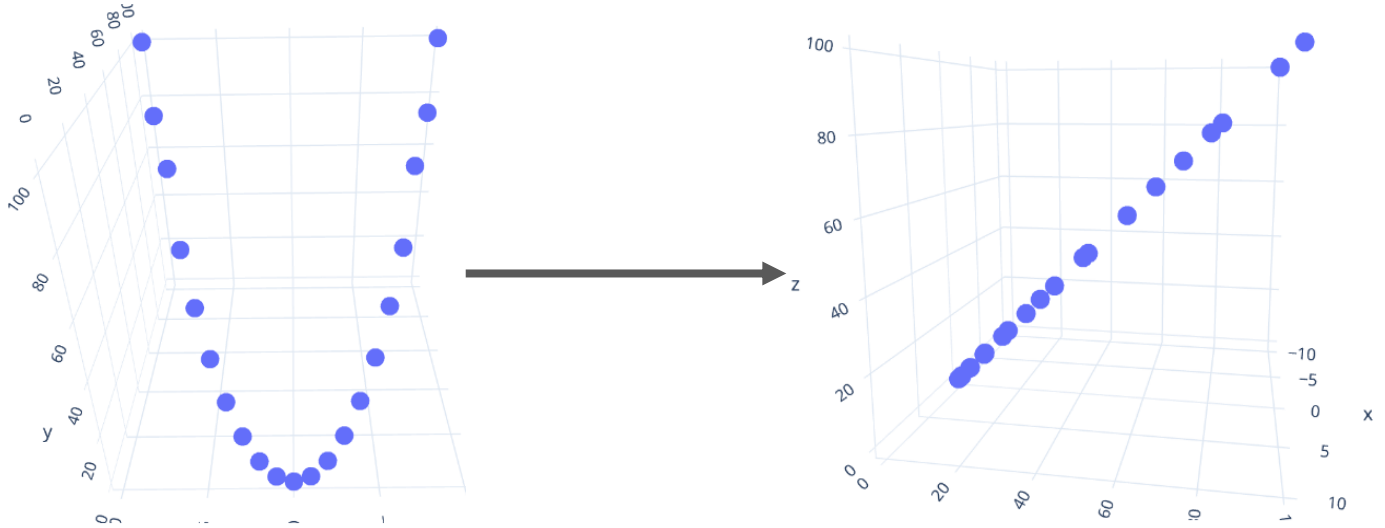
Polynomial Regression Intuition

https://plotly.com/~S7s91/1/



Polynomial Regression Intuition

https://plotly.com/~S7s91/1/



Polynomial of multivariate features?!

Lecture Overview

- **Polynomial Regression**
- **Underfitting and Overfitting**
- **Simple (Holdout) Cross Validation**
- **Model Complexity**
- **Early Stopping**
- **Regularization**
- **SKlearn [Polynomial Features]**
- **SKlearn [without regularization]**
- **SKlearn [Lasso regularization]**
- **SKlearn [Ridge regularization]**
- **K-fold Cross Validation**
- **Variance-Bias Tradeoff**

Underfitting and Overfitting problems

Underfitting and Overfitting problems

Underfitting Nabil

Overfitting Nagiba

Underfitting and Overfitting problems

Underfitting Nabil

Overfitting Nagiba

How does that happen in machine learning?

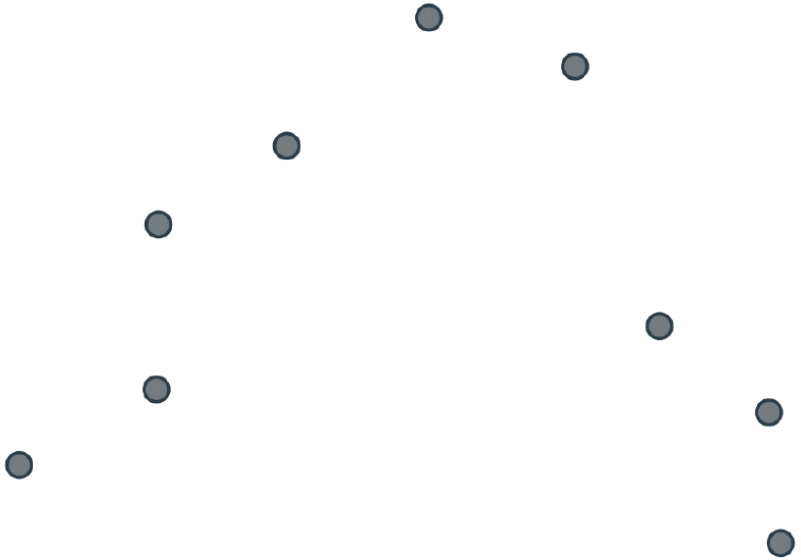
Underfitting and Overfitting problems

Underfitting and Overfitting problems

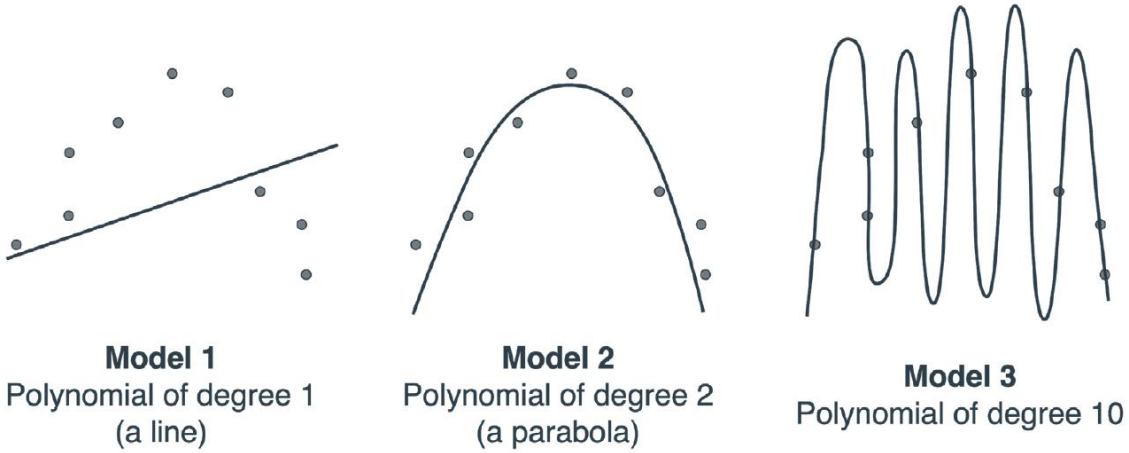
Which polynomial degree is suitable for this data?

It is **two**

How Machines know that?!!
Let’s try degree 1, 2 and 10



Underfitting and Overfitting problems



Model 1
Polynomial of degree 1
(a line)

Model 2
Polynomial of degree 2
(a parabola)

Model 3
Polynomial of degree 10

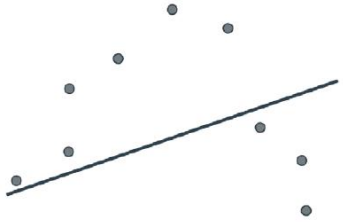
Underfitting and Overfitting problems

Very simple models tend to underfit. Very complex models tend to overfit.
The goal is to find a model that is neither too simple nor too complex.

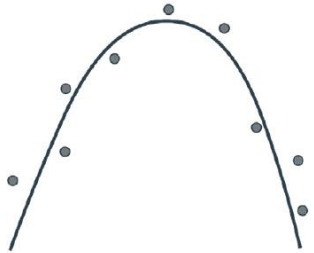
What is the model with the least error for each model?

How model 2 has error larger than what model 3 has, and model 2 is better for our data?

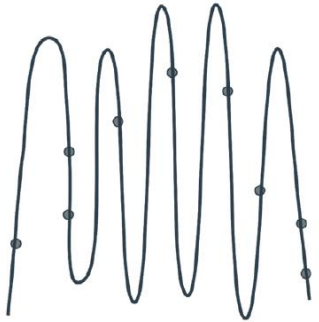
How do we know if our model overfit or underfit the data?



Model 1
Polynomial of degree 1
(a line)



Model 2
Polynomial of degree 2
(a parabola)



Model 3
Polynomial of degree 10

Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Underfitting and Overfitting problems [The **Testing set** solution]

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

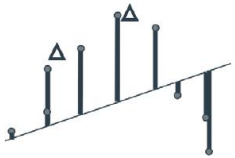
	Model 1 (degree 1)	**Model 2** (degree 2)	**Model 3** (degree 10)
Training error			
Testing error			

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

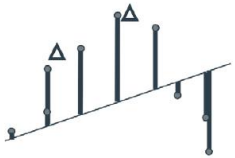
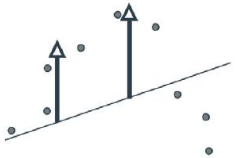
	Model 1 (degree 1)	**Model 2** (degree 2)	**Model 3** (degree 10)
Training error	 High		
Testing error			

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

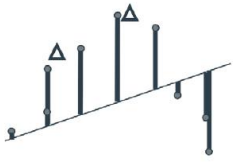
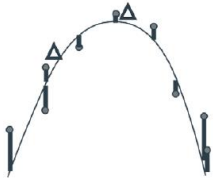
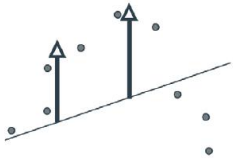
	Model 1 (degree 1)	**Model 2** (degree 2)	**Model 3** (degree 10)
Training error	 High		
Testing error	 High		

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

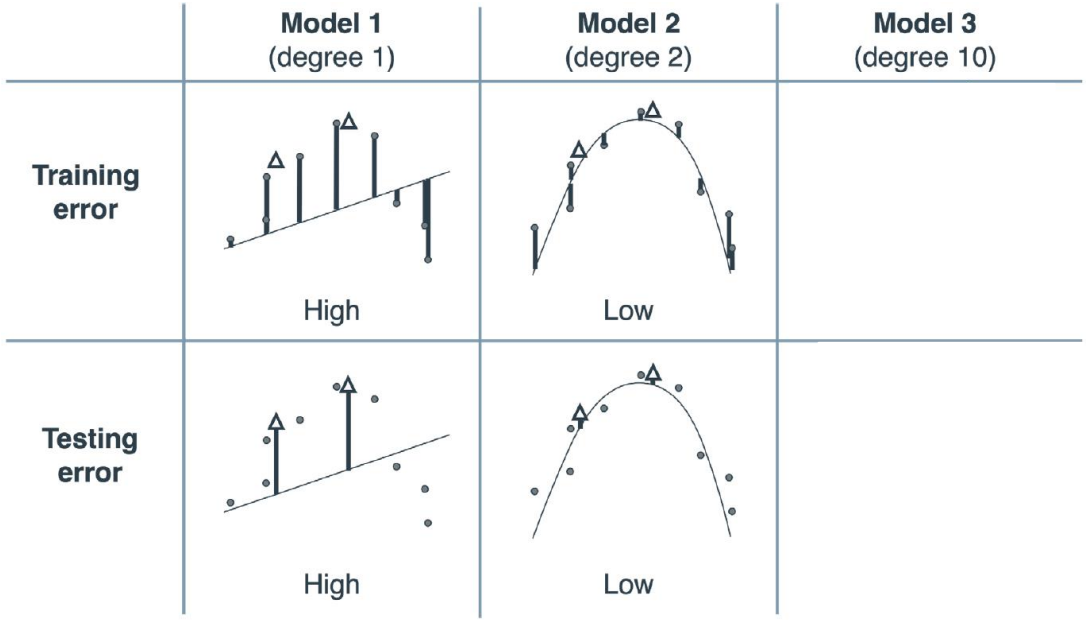
	Model 1 (degree 1)	**Model 2** (degree 2)	**Model 3** (degree 10)
Training error	 High	 Low	
Testing error	 High		

Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

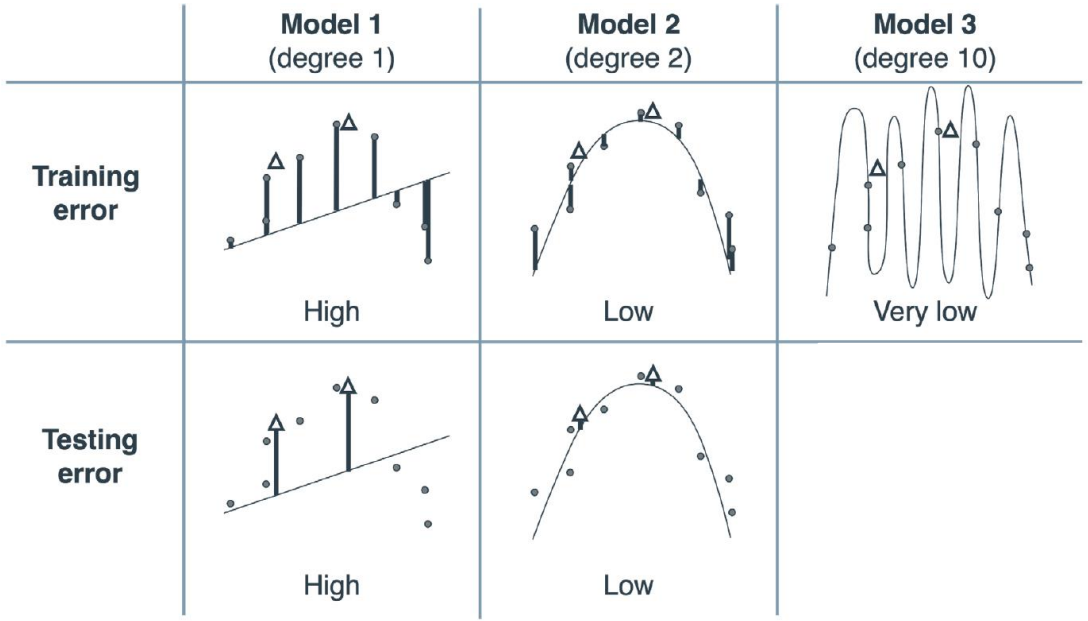


Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

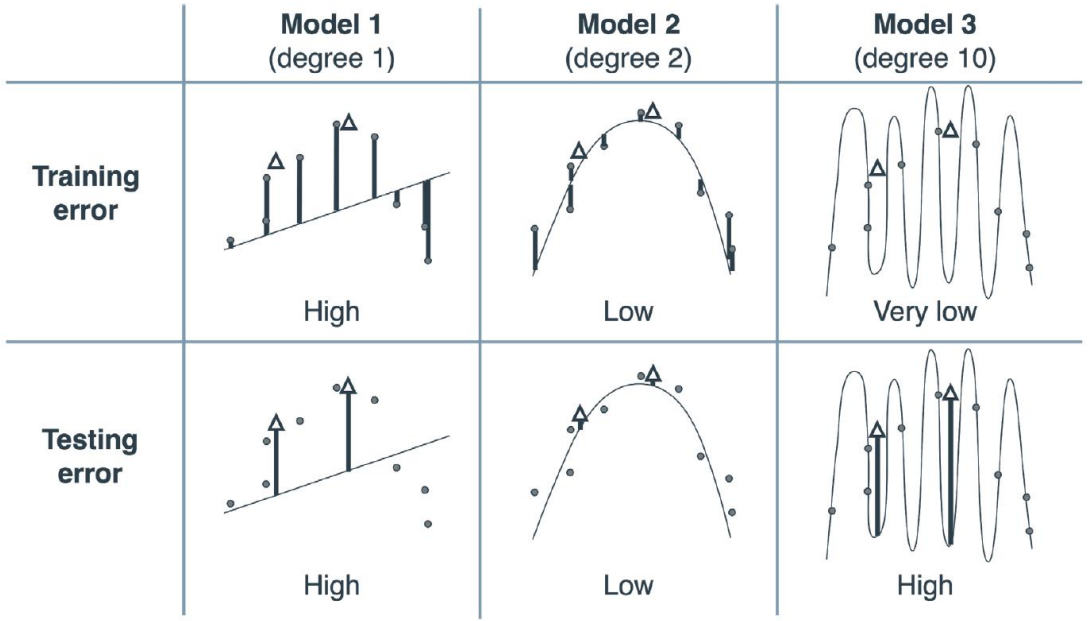


Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.



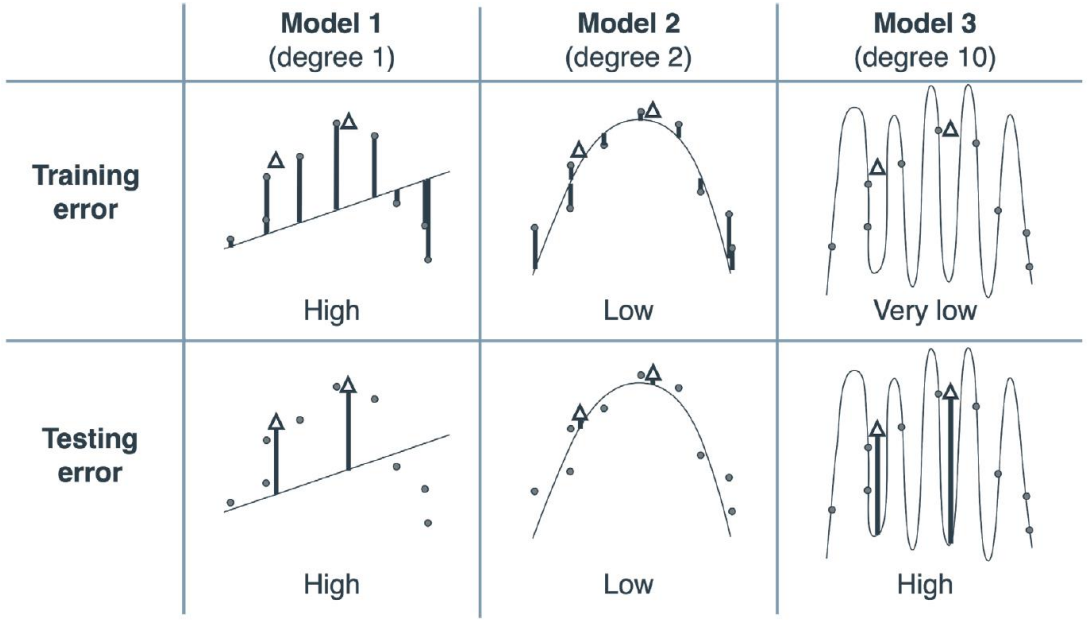
Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

[Golden Rule]
Never use testing data for training or to take a decision for the hyper parameters.



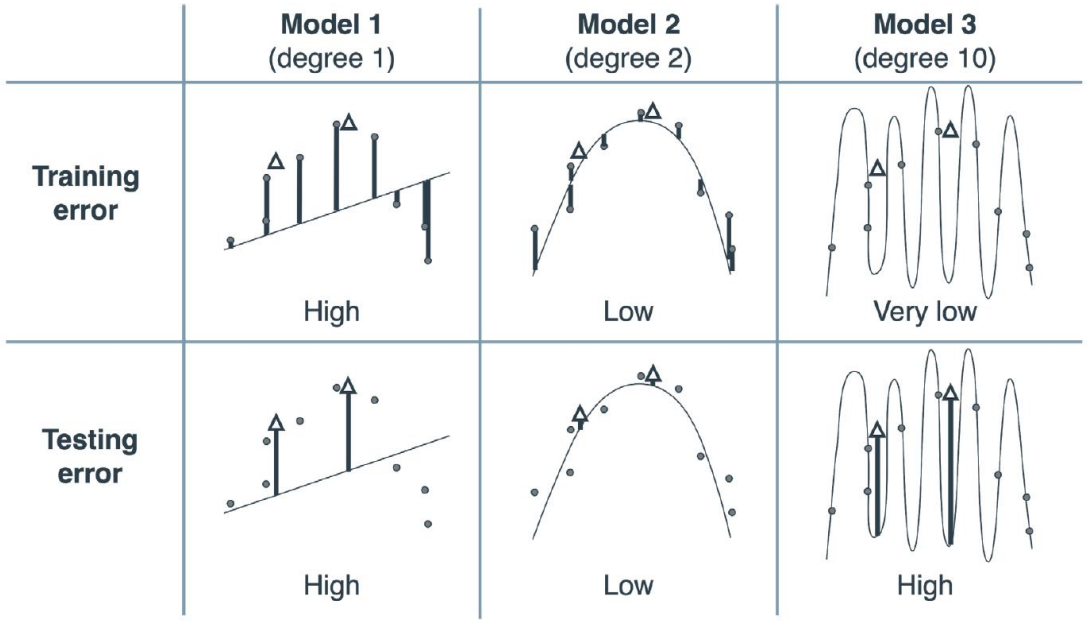
Underfitting and Overfitting problems [The **Testing set** solution]

We divide the data into 2 sets

- Training set
- Testing set

We can know if there is underfit or overfit problem from train and test errors and tune our hyper-params based on that.

[Golden Rule]
Never use testing data for training or to take a decision for the hyper parameters.



We already broke this rule!!!

Underfitting and Overfitting problems [The **Validation set** solution]

Underfitting and Overfitting problems [The **Validation set** solution]

We break our dataset into the following three sets:

- **Training set:** Used for training all our models.
- **Validation set (Dev):** Used for making decisions on which model to use.
- **Testing set:** Used to check how well our model did.

Underfitting and Overfitting problems [The **Validation set** solution]

We break our dataset into the following three sets:

- **Training set:** Used for training all our models.
- **Validation set (Dev):** Used for making decisions on which model to use.
- **Testing set:** Used to check how well our model did.

Underfitting and Overfitting problems [The **Validation set** solution]

We break our dataset into the following three sets:

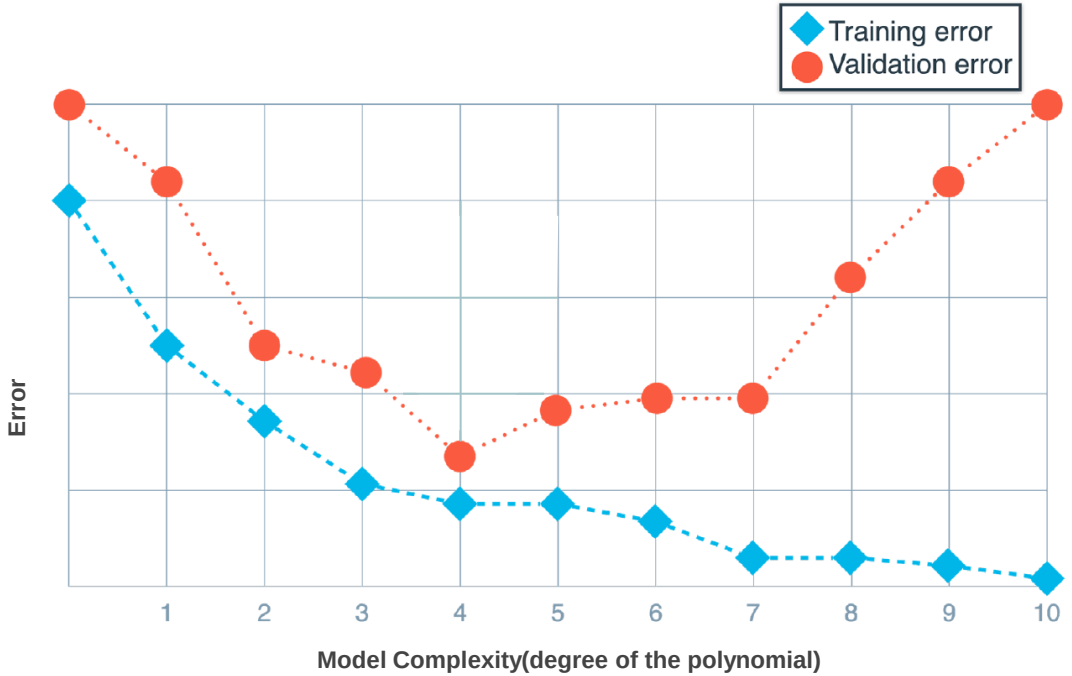
- **Training set:** Used for training all our models.
- **Validation set (Dev):** Used for making decisions on which model to use.
- **Testing set:** Used to check how well our model did.

This is called **Simple (Holdout) Cross Validation**

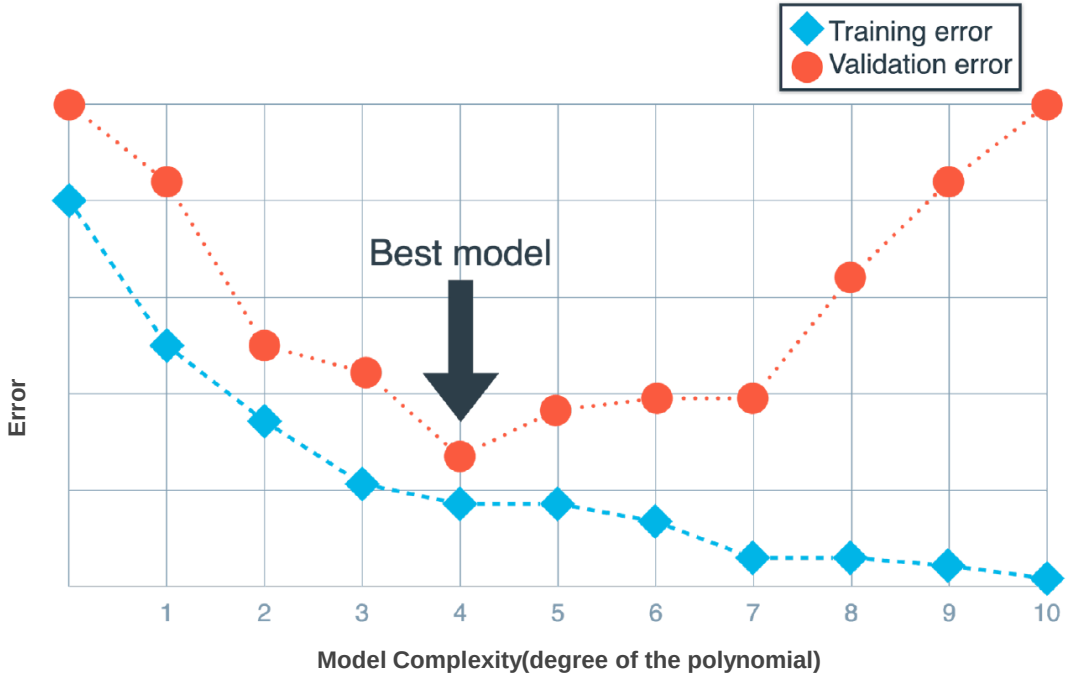
Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Solving overfitting/underfitting problem [**Model Complexity**]



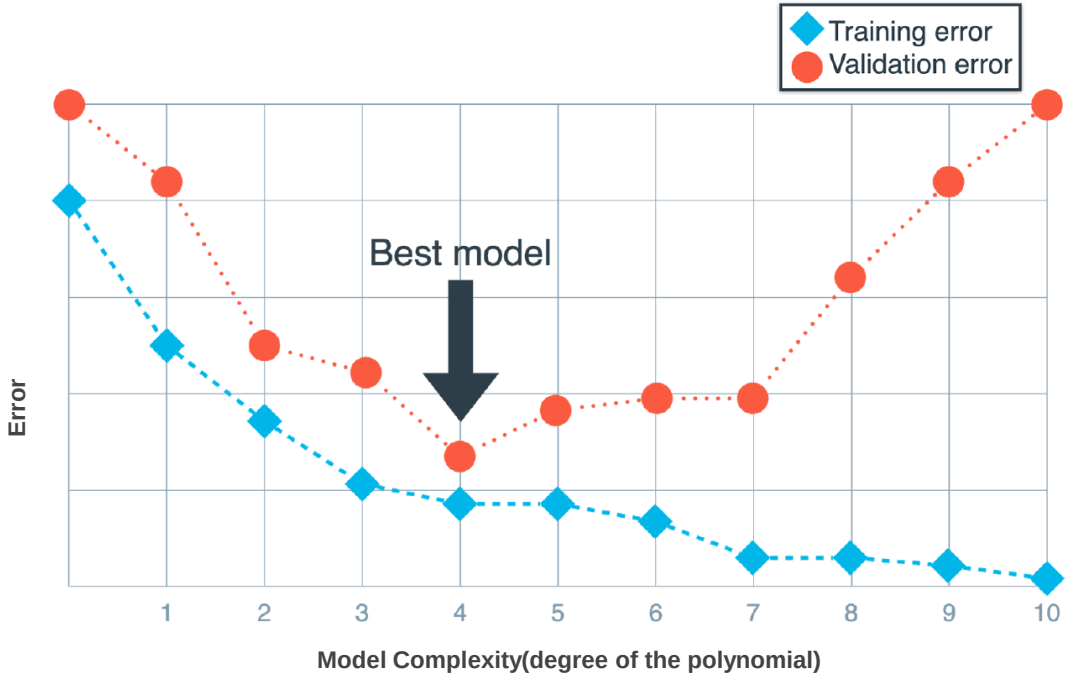
Solving overfitting/underfitting problem [**Model Complexity**]



Solving overfitting/underfitting problem [**Model Complexity**]

Model evaluation Graph

- Why validation error is always larger than the training error?
- Should we always pick the model with the least validation error?

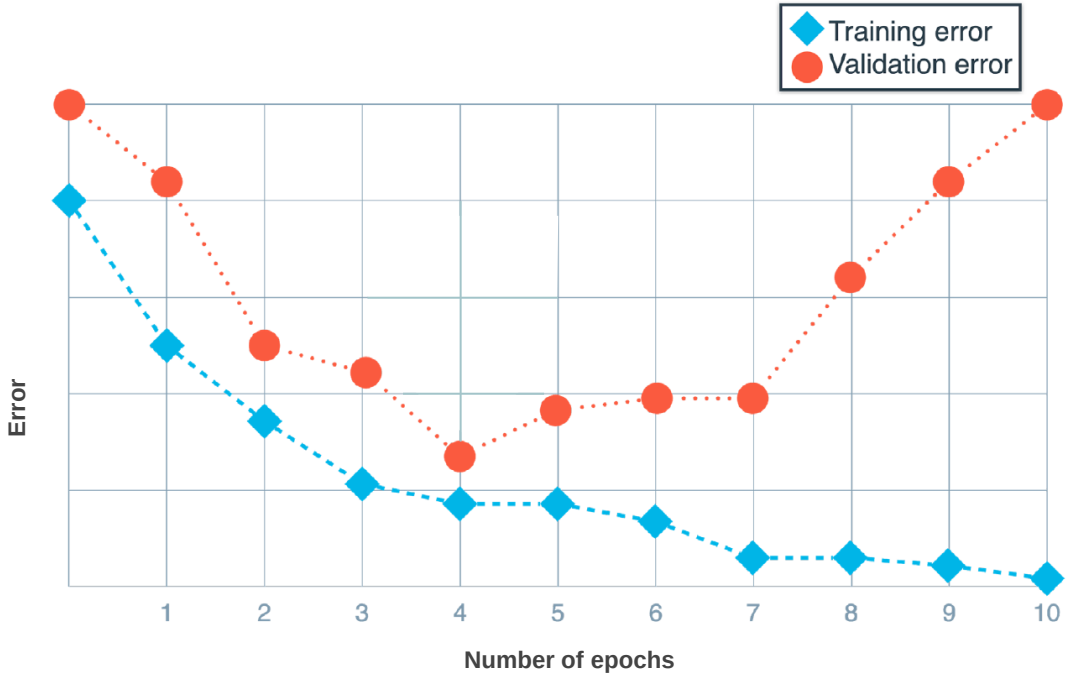


Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

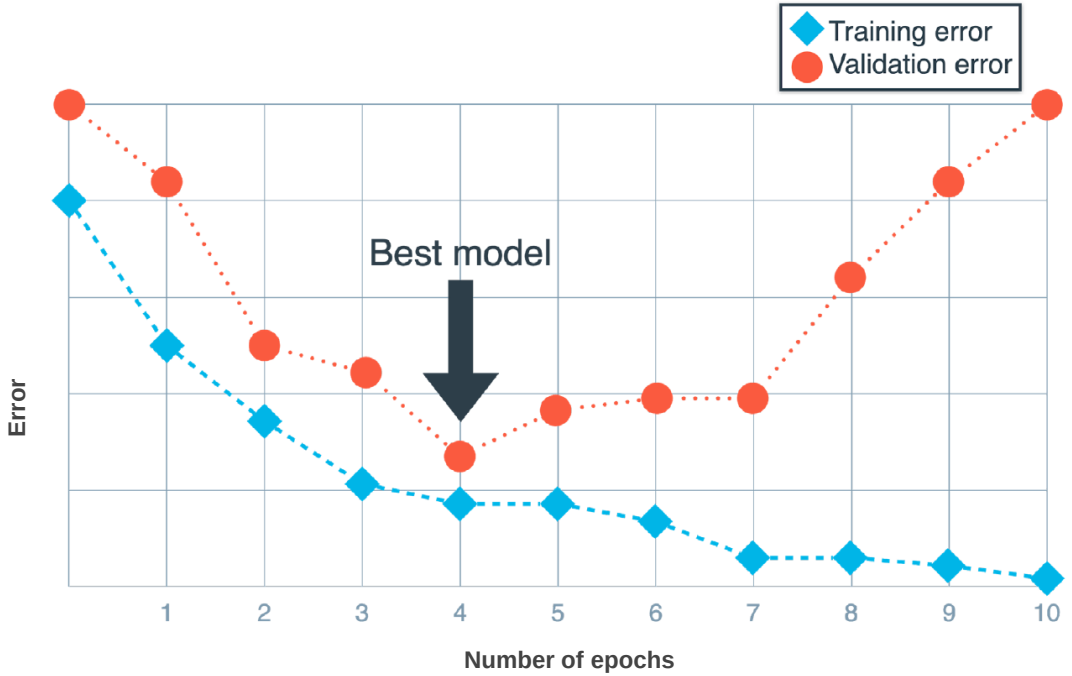
Solving overfitting/underfitting problem [**Early Stopping**]

Which checkpoint should we pick for the same model?



Solving overfitting/underfitting problem [**Early Stopping**]

Which checkpoint should we pick for the same model?



Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Solving overfitting/underfitting problem [**Regularization**]

Problem: Broken roof

Roofer 1
Solution: Bandage
(Underfitting)

Roofer 2
Solution: Shingles
(Correct)

Roofer 3
Solution: Titanium
(Overfitting)

Solving overfitting/underfitting problem [**Regularization**]

Performance cost (in ml of water leaked):

- Roofer 1: 1000 ml
- Roofer 2: 1 ml
- Roofer 3: 0 ml

Complexity cost (in price):

- Roofer 1: 1$
- Roofer 2: 100$
- Roofer 3: 100,000$

Performance cost + complexity cost:

- Roofer 1: 1001
- Roofer 2: 101
- Roofer 3: 100,000

Problem: Broken roof

Roofer 1
Solution: Bandage
(Underfitting)

Roofer 2
Solution: Shingles
(Correct)

Roofer 3
Solution: Titanium
(Overfitting)

In machine learning, what does represent the performance cost and the complexity cost?

Solving overfitting/underfitting problem [**Regularization**]

Solving overfitting/underfitting problem [**Regularization**]

Higher coefficients values (weights) => Higher complexity

Multiple regression example:

Solving overfitting/underfitting problem [**Regularization**]

Higher coefficients values (weights) => Higher complexity

Multiple regression example:

Model 1: $\hat{y} = 2x_3 + 1.4x_7 - 0.5x_9 + 4$

Model 2:

$$\hat{y} = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8$$

Solving overfitting/underfitting problem [**Regularization**]

Higher coefficients values (weights) => Higher complexity

Multiple regression example:

Model 1: $\hat{y} = 2x_3 + 1.4x_7 - 0.5x_9 + 4$

Model 2:

$$\hat{y} = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8$$

Polynomial regression example:

Solving overfitting/underfitting problem [**Regularization**]

Higher coefficients values (weights) => Higher complexity

Multiple regression example:

Model 1: $\hat{y} = 2x_3 + 1.4x_7 - 0.5x_9 + 4$

Model 2:

$$\hat{y} = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8$$

Polynomial regression example:

Model 1: $\hat{y} = 2x + 3.$

Model 2: $\hat{y} = -x^2 + 6x - 2.$

Model 3: $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10.$

Solving overfitting/underfitting problem [**Regularization**]

Model 1: $\hat{y} = 2x + 3.$

Model 2: $\hat{y} = -x^2 + 6x - 2.$

Model 3: $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10.$

Solving overfitting/underfitting problem [**Regularization**]

Measuring how complex a model is:

L1 Norm (lasso regularization): The sum of the absolute values of the coefficients (weights).

L2 Norm (ridge regularization): The sum of the squares of the coefficients (weights).

Model 1: $\hat{y} = 2x + 3.$

Model 2: $\hat{y} = -x^2 + 6x - 2.$

Model 3: $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10.$

L1 Norm:

Model 1: |2|=2
Model 2: |-1| + |6| =7
Model 3: |1| + |4| + |-9| + |3| + |-14| + |-2| + |-9| + |1| + |6| = 49

L2 Norm:

Model 1: $2^2=4$
Model 2: $(-1)^2 + 6^2 =37$
Model 3: $1^2 + 4^2 + (-9)^2 + 3^2 + (-14)^2 + (-2)^2 + (-9)^2 + 1^2 + 6^2 = 425$

Solving overfitting/underfitting problem [**Regularization**]

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2$$

Loss function

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

Lambda symbol(**λ**):
represents the regularization hyperparameter.
λ =0 means no regularization.

L1 Norm update: **m'=m - η * 1/n * (P'-P)r**

L2 Norm update: **m'=m - η * 1/n * (P'-P)r**

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

Lambda symbol(**λ**):
represents the regularization hyperparameter.
λ =0 means no regularization.

L1 Norm update: **m'=m - η * 1/n * (P'-P)r ∓ λ**

L2 Norm update: **m'=m - η * 1/n * (P'-P)r**

Solving overfitting/underfitting problem [**Regularization**]

How machines learn to reduce the complexity cost?

- Like performance cost, machine learning adds the regularization term(L1/L2 Norm) to the error
- And using gradient descent (derivative), it reduces the new cost function(including L1/L2 Norm term)

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

Lambda symbol(**λ**):
represents the regularization hyperparameter.
λ =0 means no regularization.

L1 Norm update: **m'=m - η * 1/n * (P'-P)r ∓ λ**

L2 Norm update: **m'=m - η * 1/n * (P'-P)r - 2λm**

Solving overfitting/underfitting problem [**Regularization**]

Solving overfitting/underfitting problem [**Regularization**]

Difference between L1 and L2 regularization:

L1 Norm update: **$m' = m - \eta * 1/n * (P'-P)r \mp \lambda$**

L2 Norm update: **$m' = m - \eta * 1/n * (P'-P)r - 2\lambda m$**

Solving overfitting/underfitting problem [**Regularization**]

Difference between L1 and L2 regularization:

L1 Norm update: **$m' = m - \eta * 1/n * (P'-P)r \mp \lambda$**

L2 Norm update: **$m' = m - \eta * 1/n * (P'-P)r - 2\lambda m$**

L1 Norm try to decrease the weight with constant value λ, so it can decrease multiple weights to zero

L2 Norm try to decrease the weight with a percentage of the weight itself, so it decreases the weights but can't decrease them to zero

So L1 Norm is used if a lot of features not related to the problem but L2 Norm used if features related

Solving overfitting/underfitting problem [**Regularization**]

Difference between L1 and L2 regularization:

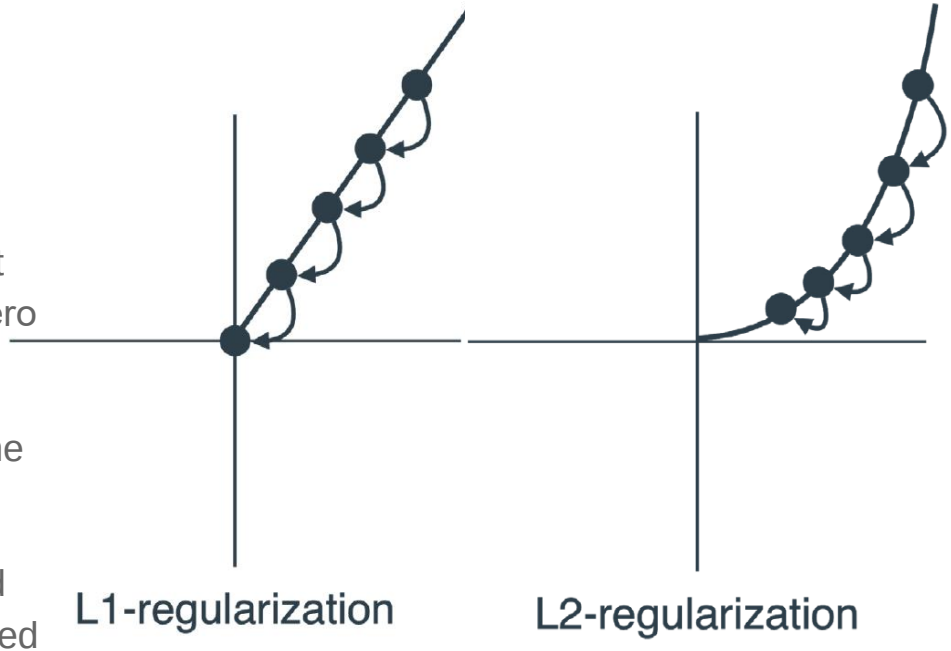
L1 Norm update: **m'=m - η * 1/n * (P'-P)r ∓ λ**

L2 Norm update: **m'=m - η * 1/n * (P'-P)r - 2λm**

L1 Norm try to decrease the weight with constant value λ, so it can decrease multiple weights to zero

L2 Norm try to decrease the weight with a percentage of the weight itself, so it decreases the weights but can't decrease them to zero

So L1 Norm is used if a lot of features not related to the problem but L2 Norm used if features related



Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Regularization Example

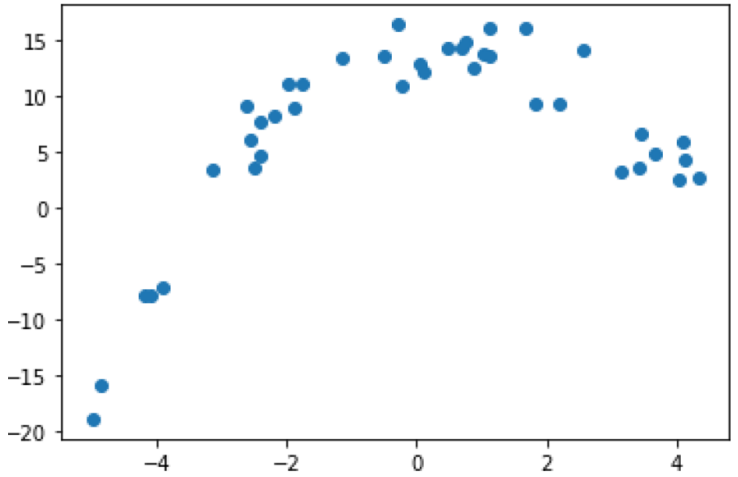
From a noisy parabola we draw our data

Then we converted our data to a 200-column data

Regularization Example

From a noisy parabola we draw our data

Then we converted our data to a 200-column data

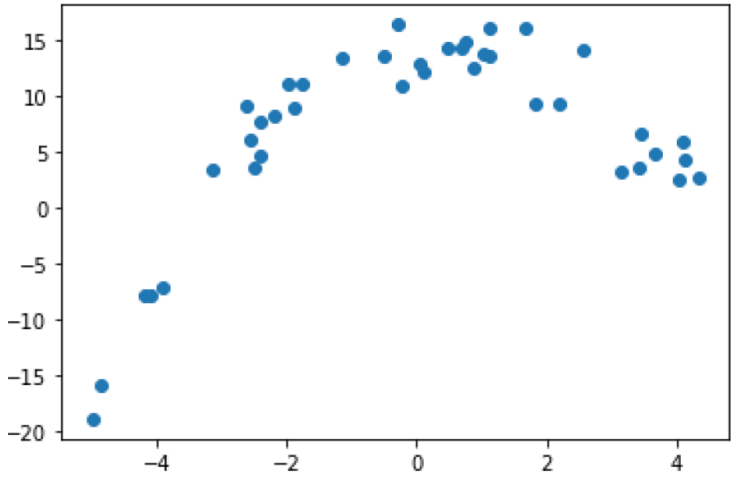


Regularization Example

From a noisy parabola we draw our data

Then we converted our data to a 200-column data

x	y
3.444218515250481 6	6.685961311021467
-2.410832497070366 3	4.690236225597948
0.112747213686085 42	12.205789026637378
-1.966872739210725 5	11.133217991032268

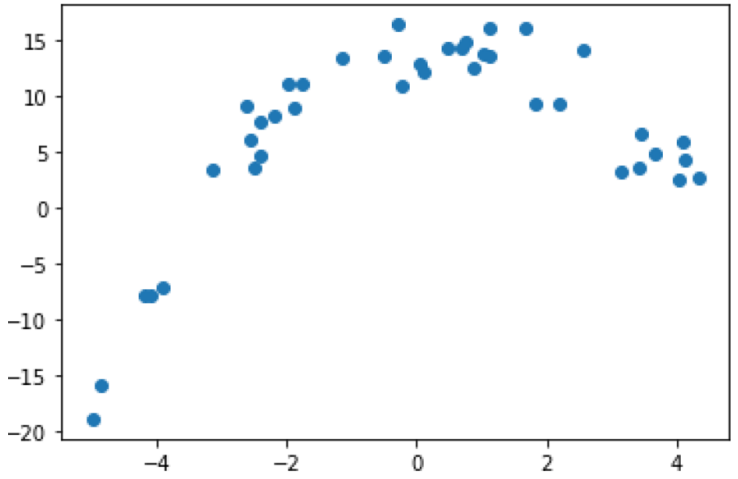


Regularization Example

From a noisy parabola we draw our data

Then we converted our data to a 200-column data

x	y	x^2	x^3	x^4
3.4442185152504816	6.685961311021467	11.862641180794233	40.85752839466433	140.72225578427518
-2.4108324970703663	4.690236225597948	5.812113328930538	-14.012031690041567	33.78066134833202
0.11274721368608542	12.205789026637378	0.012711934193975809	0.0014332351609316464	0.00016159327095197139
-1.9668727392107255	11.133217991032268	3.8685883722503025	-7.609021008606714	14.965975993910245



Regularization Example[Polynomial Features - Data Split]

Use colab to open this github notebook:

“s7s/machine_learning_1/polynomial_regression/Polynomial_regression_regularization.ipynb”

SKLearn [Polynomial Features]

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
>>> poly = PolynomialFeatures(interaction_only=True)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.],
       [ 1.,  2.,  3.,  6.],
       [ 1.,  4.,  5., 20.]])
```

SKLearn [Data Split]

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Regularization Example[Without Regularization]

Use colab to open this github notebook:

“s7s/machine_learning_1/polynomial_regression/Polynomial_regression_regularization.ipynb”

SKLearn [Without Regularization]

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Regularization Example[Lasso Regularization]

Use colab to open this github notebook:

“s7s/machine_learning_1/polynomial_regression/Polynomial_regression_regularization.ipynb”

SKLearn [Lasso Regularization]

```
>>> from sklearn import linear_model
>>> clf = linear_model.Lasso(alpha=0.1)
>>> clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lasso(alpha=0.1)
>>> print(clf.coef_)
[0.85 0.  ]
>>> print(clf.intercept_)
0.15...
```

Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Regularization Example[Ridge Regularization]

Use colab to open this github notebook:

“s7s/machine_learning_1/polynomial_regression/Polynomial_regression_regularization.ipynb”

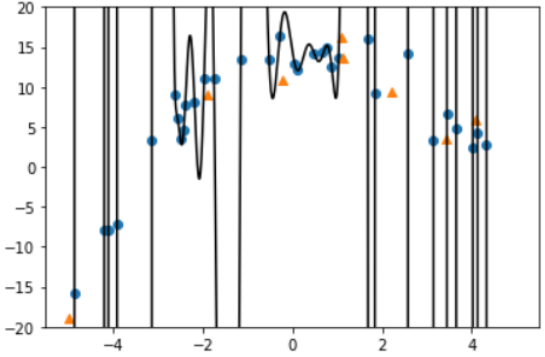
SKLearn [Ridge Regularization]

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge()
```

Regularization Example

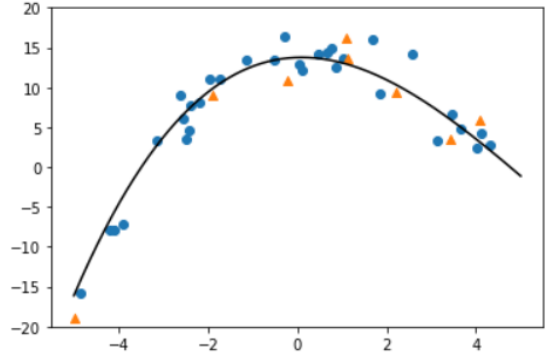
No regularization

Training error (rmse): 0.2044564193855381
Testing error (rmse): 2292653405740632.5



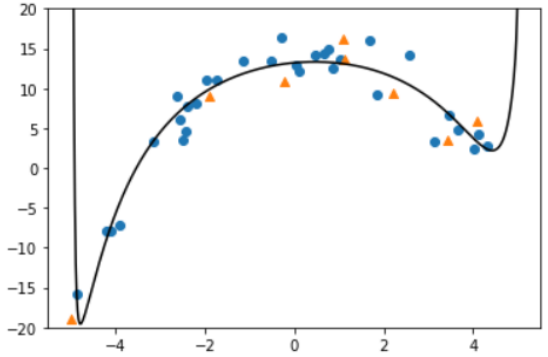
L1 regularization

Training error (rmse): 2.044243867351526
Testing error (rmse): 2.3139777217431914



L2 regularization

Training error (rmse): 2.0076495283381877
Testing error (rmse): 96.69221694896068



What do you notice?

Regularization Example

Coefficient	model_no_reg	model_L1_reg	model_L2_reg
$x^0 = 1$	8.41	0.57	13.24
x^1	15.87	0.07	0.87
x^2	108.87	-0.004	-0.52
x^3	-212.89	0.0002	0.006
x^4	-97.13	-0.0002	-0.02

Regularization Example

Model with no regularization:
All coefficients have values

Model with L1 regularization:
Most of coefficients ≃ zero

Model with L2 regularization:
Coefficients equal small values

Coefficient	model_no_reg	model_L1_reg	model_L2_reg
$x^0 = 1$	8.41	0.57	13.24
x^1	15.87	0.07	0.87
x^2	108.87	-0.004	-0.52
x^3	-212.89	0.0002	0.006
x^4	-97.13	-0.0002	-0.02

Lecture Overview

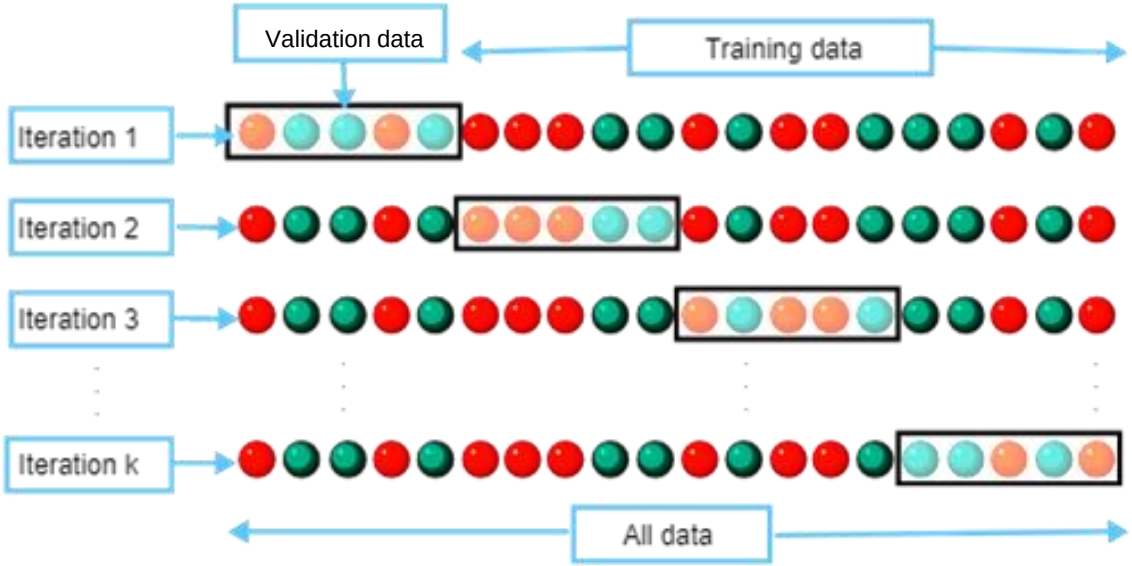
- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

K-fold Cross Validation

How we divide our data if we have small dataset?

K-fold Cross Validation

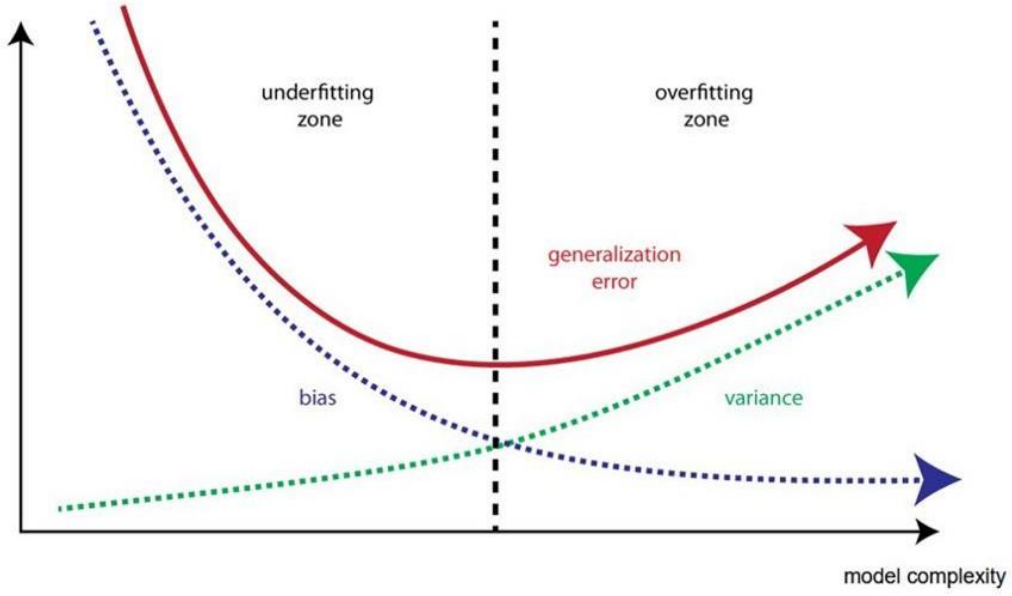
How we divide our data if we have small dataset?



Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

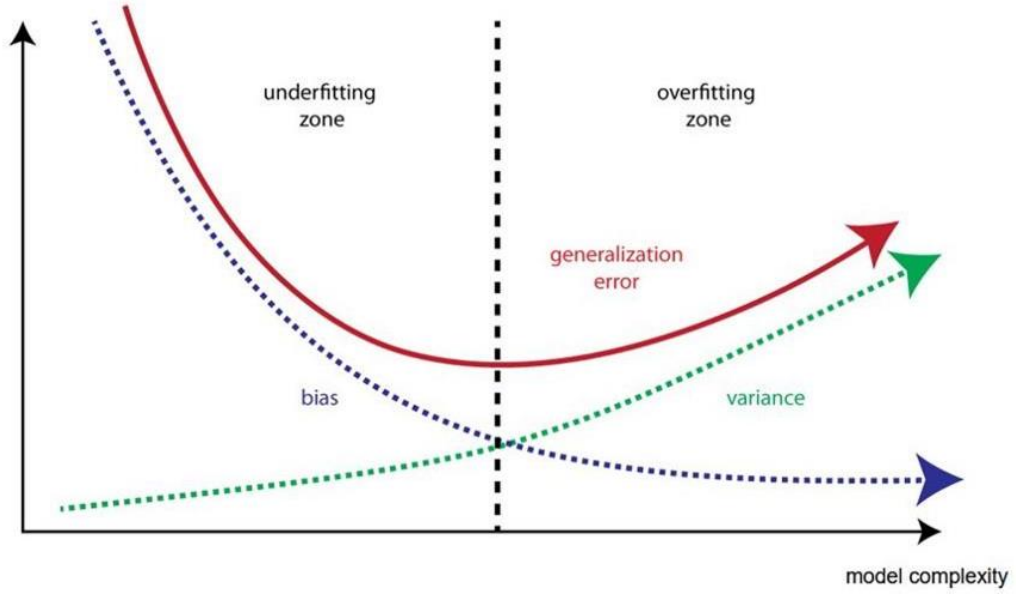
Variance-Bias Tradeoff



Variance-Bias Tradeoff

Overfit model: High Variance model

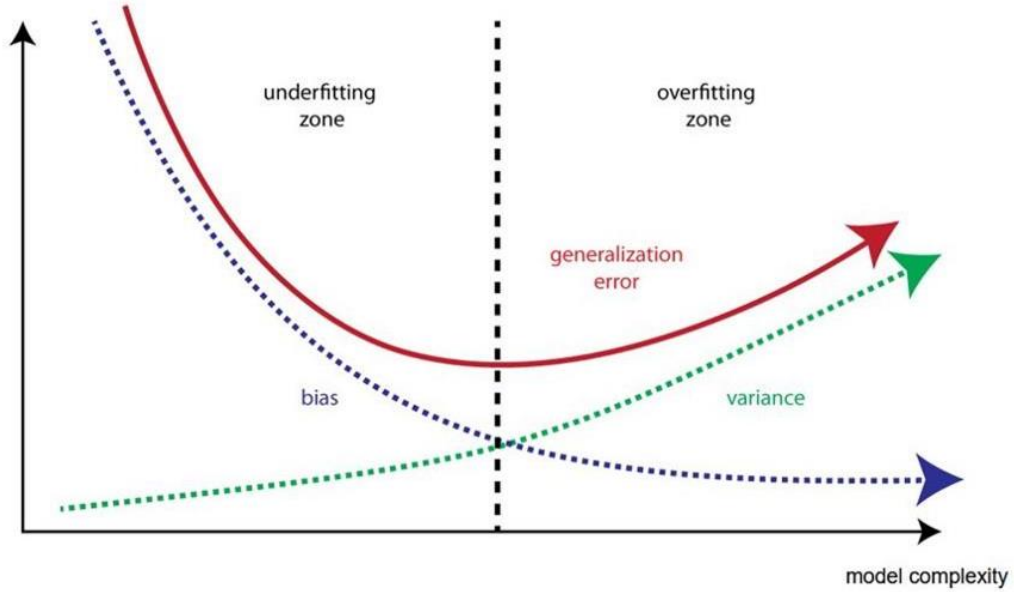
Underfit model: High Bias model



Variance-Bias Tradeoff

Overfit model: High Variance model

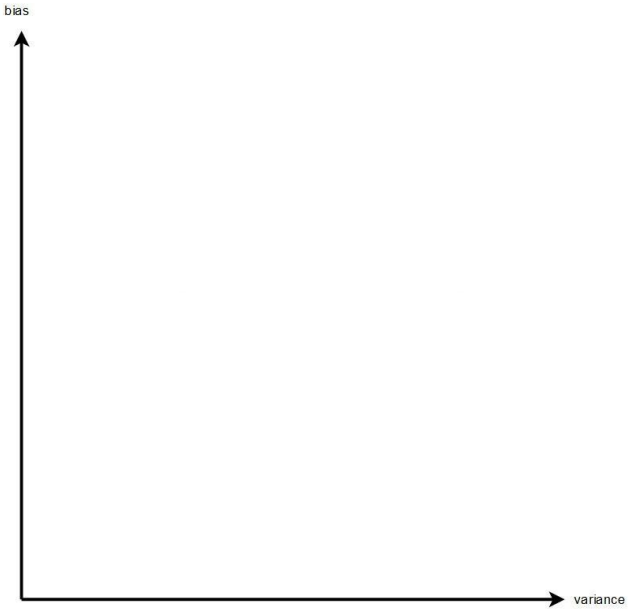
Underfit model: High Bias model



Is complexity the only factor of high bias or high variance models?

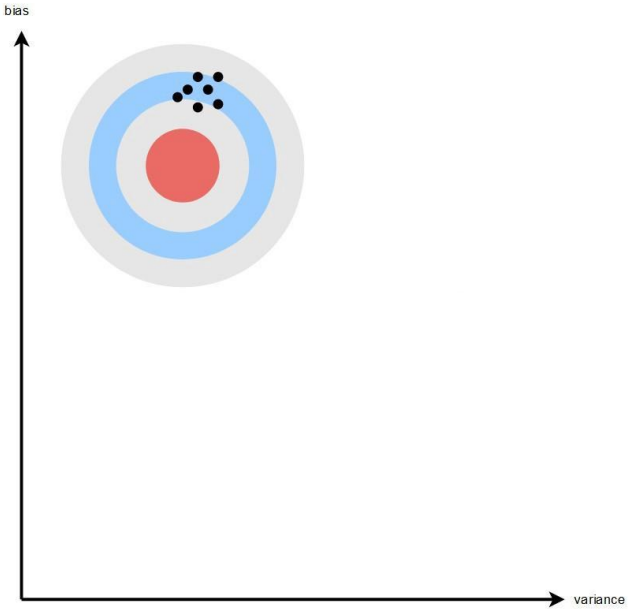
Variance-Bias Tradeoff

Train a new models with different data



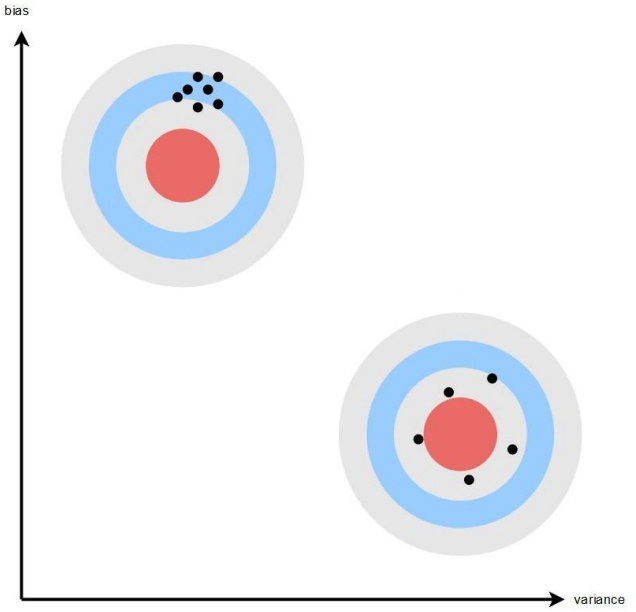
Variance-Bias Tradeoff

Train a new models with different data



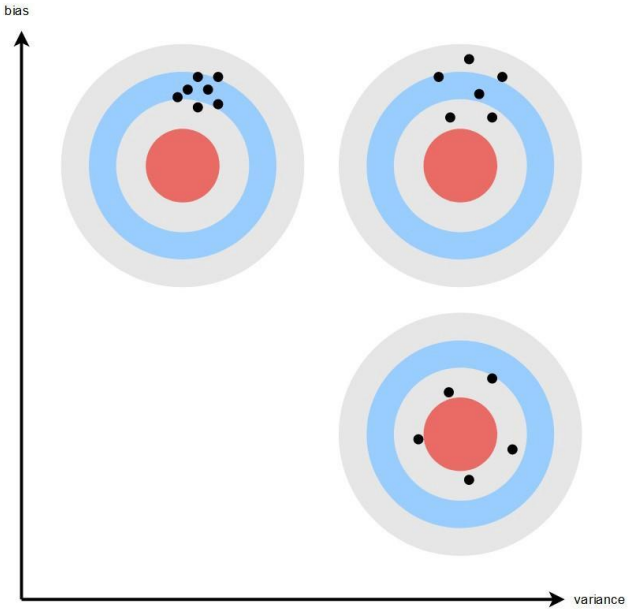
Variance-Bias Tradeoff

Train a new models with different data



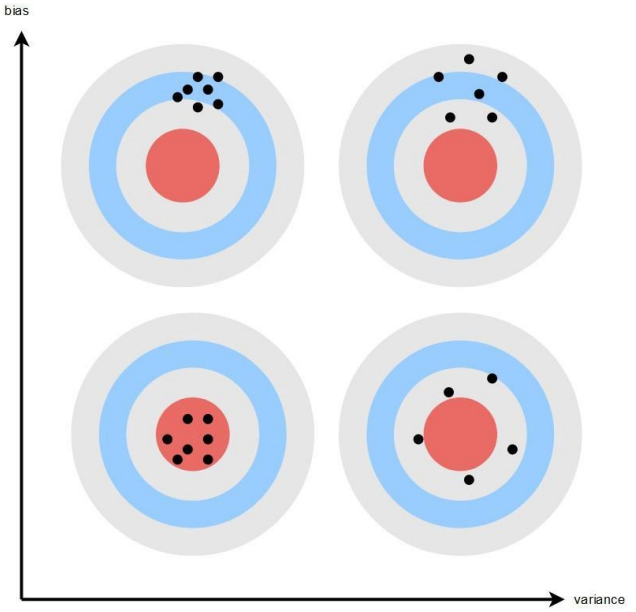
Variance-Bias Tradeoff

Train a new models with different data



Variance-Bias Tradeoff

Train a new models with different data

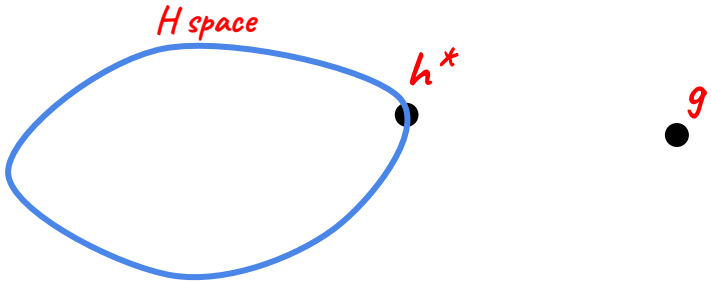


Variance-Bias Tradeoff

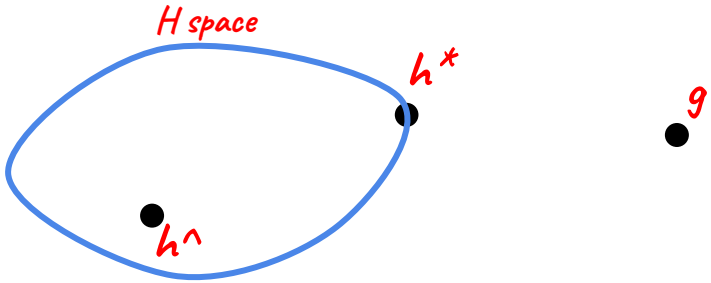
Variance-Bias Tradeoff

Variance-Bias Tradeoff

Variance-Bias Tradeoff



Variance-Bias Tradeoff



Variance-Bias Tradeoff

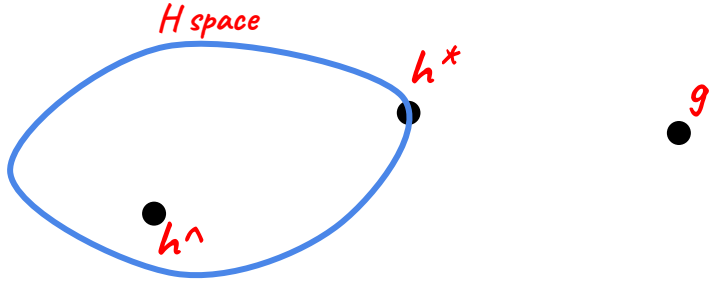
g: Best possible model

h*: Best model in H space
(infinite data)

h^: model learnt from finite data

ξ(h): Generalization Error/Risk
Error for all unseen data
(couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data



Variance-Bias Tradeoff

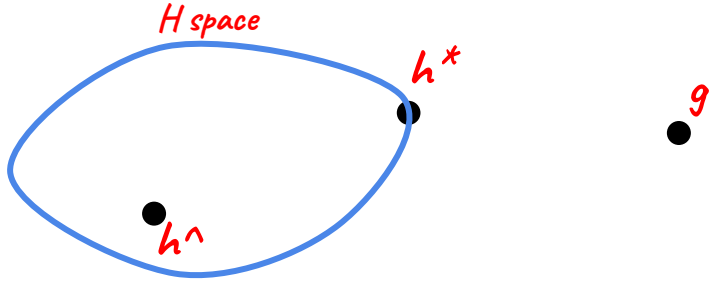
g: Best possible model

h*: Best model in H space
(infinite data)

h^: model learnt from finite data

ξ(h): Generalization Error/Risk
Error for all unseen data
(couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data



Variance-Bias Tradeoff

g: Best possible model

h*: Best model in H space (infinite data)

h^: model learnt from finite data

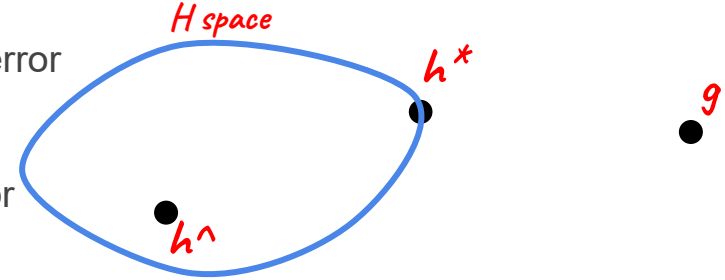
ξ(h): Generalization Error/Risk
Error for all unseen data (couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data

ξ(g) = Irreducible error

ξ(h*)-ξ(g) = approximation error
Complexity error

ξ(h^)-ξ(h*) = estimation error
Finite data error



ξ(h^) = estimation error + approximation error + Irreducible error

Variance-Bias Tradeoff

g: Best possible model

h*: Best model in H space (infinite data)

h^: model learnt from finite data

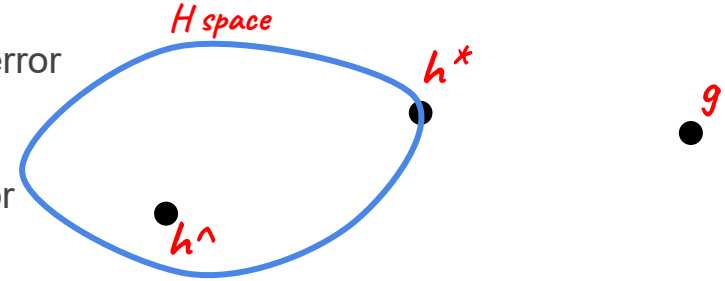
ξ(h): Generalization Error/Risk
Error for all unseen data (couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data

ξ(g) = Irreducible error

ξ(h*)-ξ(g) = approximation error
Complexity error

ξ(h^)-ξ(h*) = estimation error
Finite data error



ξ(h^) = estimation error + approximation error + Irreducible error
(finite data error) (complexity error)

Variance-Bias Tradeoff

g: Best possible model

h*: Best model in H space (infinite data)

h^: model learnt from finite data

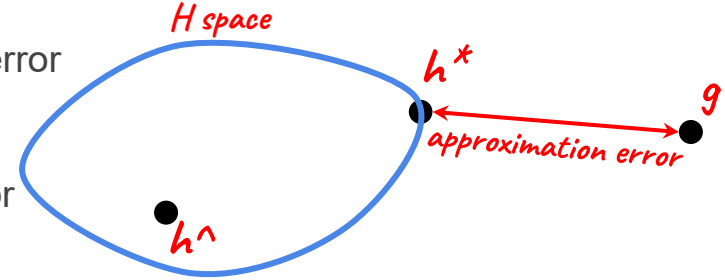
ξ(h): Generalization Error/Risk
Error for all unseen data (couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data

ξ(g) = Irreducible error

ξ(h*)-ξ(g) = approximation error
Complexity error

ξ(h^)-ξ(h*) = estimation error
Finite data error



ξ(h^) = estimation error + approximation error + Irreducible error
(finite data error) (complexity error)

Variance-Bias Tradeoff

g: Best possible model

h*: Best model in H space (infinite data)

h^: model learnt from finite data

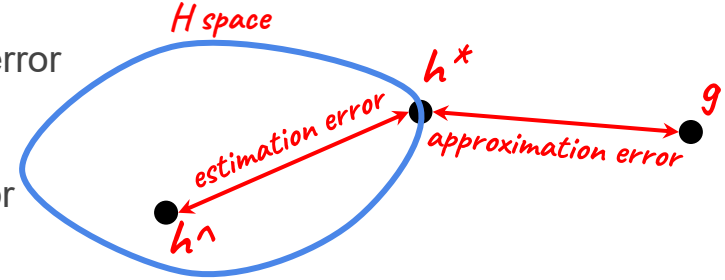
ξ(h): Generalization Error/Risk
Error for all unseen data (couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data

ξ(g) = Irreducible error

ξ(h*)-ξ(g) = approximation error
Complexity error

ξ(h^)-ξ(h*) = estimation error
Finite data error



ξ(h^) = estimation error + approximation error + Irreducible error
(finite data error) (complexity error)

Variance-Bias Tradeoff

g: Best possible model

h*: Best model in H space (infinite data)

h^: model learnt from finite data

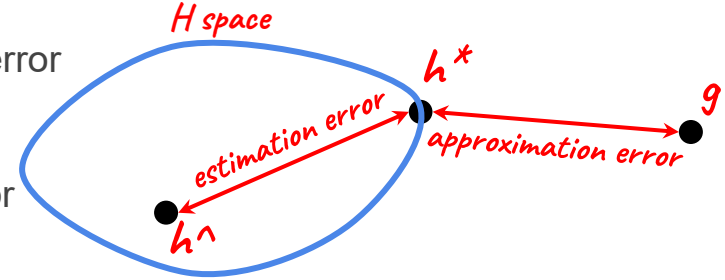
ξ(h): Generalization Error/Risk
Error for all unseen data (couldn't be calculated)

ξ^(h): Empirical Error/Risk
Error of finite data

ξ(g) = Irreducible error

ξ(h*)-ξ(g) = approximation error
Complexity error

ξ(h^)-ξ(h*) = estimation error
Finite data error



ξ(h^) = estimation error + approximation error + Irreducible error
(finite data error) (complexity error)

ξ(h^) = Variance + Bias + Irreducible error

Lecture Overview

- Polynomial Regression
- Underfitting and Overfitting
- Simple (Holdout) Cross Validation
- Model Complexity
- Early Stopping
- Regularization
- SKlearn [Polynomial Features]
- SKlearn [without regularization]
- SKlearn [Lasso regularization]
- SKlearn [Ridge regularization]
- K-fold Cross Validation
- Variance-Bias Tradeoff

Exercise

We have trained four models in the same dataset with different hyperparameters. In the following table we have recorded the training and testing errors for each of the models.

Model	Training error	Testing error
1	0.1	1.8
2	0.4	1.2
3	0.6	0.8
4	1.9	2.3

a) Which model would you select for this dataset?
b) Which model looks like it's underfitting the data?