Machine Learning for Finance

UCEMA 2020

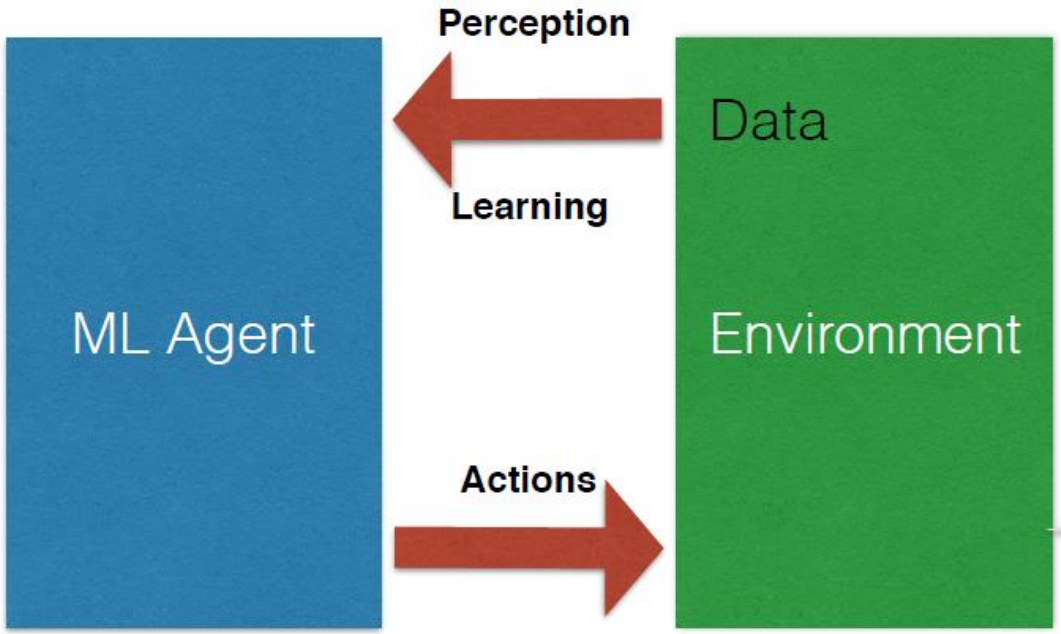
Lecture 2

- Bias-variance tradeoff
- Training and test sets
- Model capacity and overfitting
- Learning in linear regressions
- Regularizations
- Hyperparameters and cross validation
- Gradient descent-like algorithms
- Probabilistic models, MLE, MAP
- Probabilistic classifications
- Logistic regressions
- Scikit-Learn demo

Lectures Structure

- 1st block
  - ~1h: Presentation of subjects + code explanation.
  - ~15': Discussion + Q&A.
- 2nd block
  - ~1h: Presentation of subjects + code explanation.
  - ~15': Discussion + Q&A.

Generalization as a Goal of Learning



Observable/partially observable
Single agent/multi-agent
Stochastic or deterministic
Episodic or sequential
Discrete or continuous

replaced by static data in offline settings

Learning = ability to generalize

= perform well on unseen data

Generalization Error in Regression

Generalization Error in Regression

- Consider a **regression problem**

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

Generalization Error in Regression

- Consider a **regression problem**

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$
- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}]$

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}] =$

$$Var(y) + \mathbb{E}[y]^2 + Var(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2\mathbb{E}[y\hat{f}] =$$

$y = f(\boldsymbol{x}) + \varepsilon$

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over all possible data, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}] =$

$$Var(y) + \mathbb{E}[y]^2 + Var(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2\mathbb{E}[y\hat{f}] =$$

$$Var(y) + Var(\hat{f}) + \mathbb{E}[f - \hat{f}]^2$$

$y = f(\boldsymbol{x}) + \varepsilon$

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}] =$

$$Var(y) + \mathbb{E}[y]^2 + Var(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2\mathbb{E}[y\hat{f}] =$$

$$Var(y) + Var(\hat{f}) + \mathbb{E}[f - \hat{f}]^2$$

noise (σ^2)

$y = f(\boldsymbol{x}) + \varepsilon$

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}] =$

$$Var(y) + \mathbb{E}[y]^2 + Var(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2\mathbb{E}[y\hat{f}] =$$

$$Var(y) + Var(\hat{f}) + \mathbb{E}[f - \hat{f}]^2$$

$y = f(\boldsymbol{x}) + \varepsilon$

noise (σ^2) variance

Generalization Error in Regression

- Consider a regression problem

$$y = f(\boldsymbol{x}) + \varepsilon$$

- Here $\mathbb{E}[\varepsilon] = 0, \mathbb{E}[\varepsilon^2] = \sigma^2$

- This produces $\mathbb{E}[y] = f(x), Var(y) = \sigma^2$

- We look for approximation $f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x})$ such that the **generalization error** (expected loss) $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$ is minimized over <u>all possible data</u>, both seen and unseen!

- Evaluate: $\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}[y^2] + \mathbb{E}[\hat{f}^2] - 2\mathbb{E}[y\hat{f}] =$

$$Var(y) + \mathbb{E}[y]^2 + Var(\hat{f}) + \mathbb{E}[\hat{f}]^2 - 2\mathbb{E}[y\hat{f}] =$$

$$Var(y) + Var(\hat{f}) + \mathbb{E}[f - \hat{f}]^2$$

noise (σ^2) variance bias

$y = f(\boldsymbol{x}) + \varepsilon$

The Bias-Variance Decomposition

The Bias-Variance Decomposition

This is is bias-variance decomposition:

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

Here:

$$(bias)^2=\mathbb{E}\left[f-\hat{f}\right]^2$$

$$variance=Var(\hat{f})$$

$$\text{noise}=Var(y)=\sigma^2$$

The Bias-Variance Decomposition

This is is bias-variance decomposition:

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

Here:

$$(bias)^2=\mathbb{E}\left[f-\hat{f}\right]^2$$

$$variance=Var(\hat{f})$$

$$\text{noise}=Var(y)=\sigma^2$$

Bias: the (square of) expected difference of approximate predictor $\hat{f}(\boldsymbol{x})$ from the "true" predictor $f(\boldsymbol{x})$

The Bias-Variance Decomposition

This is is bias-variance decomposition:

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

Here:

$$(bias)^2=\mathbb{E}\left[f-\hat{f}\right]^2$$

$$variance=Var(\hat{f})$$

$$\text{noise}=Var(y)=\sigma^2$$

Bias: the (square of) expected difference of approximate predictor $\hat{f}(\boldsymbol{x})$ from the "true" predictor $f(\boldsymbol{x})$

Variance: sensitivity of $\hat{f}(\boldsymbol{x})$ to the choice of data set

The Bias-Variance Decomposition

This is is bias-variance decomposition:

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

Here:

$$(bias)^2=\mathbb{E}\left[f-\hat{f}\right]^2$$

$$variance=Var(\hat{f})$$

$$\text{noise}=Var(y)=\sigma^2$$

Bias: the (square of) expected difference of approximate predictor $\hat{f}(\boldsymbol{x})$ from the "true" predictor $f(\boldsymbol{x})$

Variance: sensitivity of $\hat{f}(\boldsymbol{x})$ to the choice of data set

Noise: a property of data; beyond our control

The Bias-Variance Tradeoff

The Bias-Variance Tradeoff

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

- This relation is mostly of a theoretical value: we do not know how to compute the bias and variance!
- But it show a general decomposition of expected loss of a generic regression model.

The Bias-Variance Tradeoff

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

- This relation is mostly of a theoretical value: we do not know how to compute the bias and variance!
- But it show a general decomposition of expected loss of a generic regression model.
- **Bias-variance tradeoff:**
  - Complex models (more features) tend to have low bias and high variance

The Bias-Variance Tradeoff

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

- This relation is mostly of a theoretical value: we do not know how to compute the bias and variance!
- But it show a general decomposition of expected loss of a generic regression model.
- **Bias-variance tradeoff:**
  - Complex models (more features) tend to have low bias and high variance
  - Simple models (less features) tend to have high bias and low variance

The Bias-Variance Tradeoff

$$\mathbb{E}\left[\left(y-\hat{f}(\boldsymbol{x})\right)^2\right]=(bias)^2+variance+noise$$

- This relation is mostly of a theoretical value: we do not know how to compute the bias and variance!
- But it show a general decomposition of expected loss of a generic regression model.
- **Bias-variance tradeoff:**
  - Complex models (more features) tend to have low bias and high variance
  - Simple models (less features) tend to have high bias and low variance
- Important to control **model complexity** to have an optimal tradeoff!

Data Representation

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Feature vector

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

Design Matrix

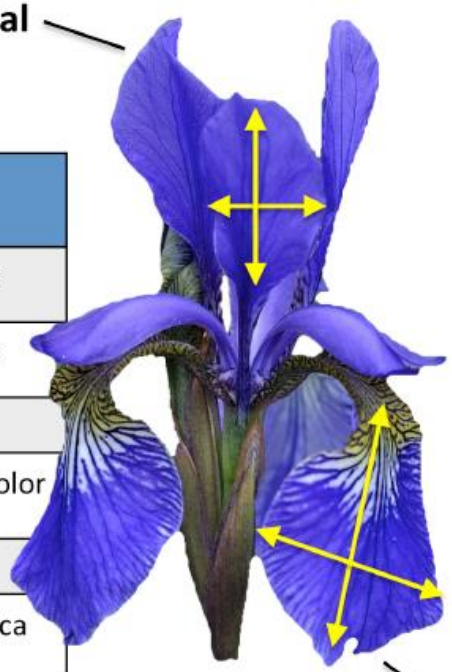
$$\mathbf{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \cdots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \cdots & x_m^{[2]} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \cdots & x_m^{[n]} \end{bmatrix}$$

Design Matrix

Structured Data

n instances

	Sepal length	Sepal width	Petal length	Petal width	
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica



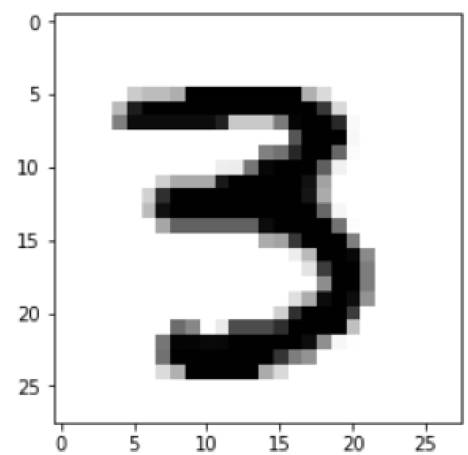
m features

n targets

Unstructured Data (Images)

"traditional methods"

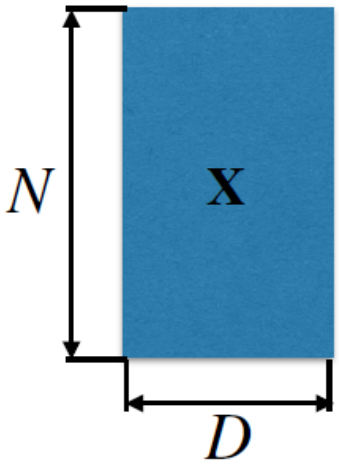

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.20392157, 0.2627451, 0.2627451, 0.31764707, 0.99607843, 0.99607843, 0.99607843
, 0.99607843, 1.0, 0.99607843, 0.99607843, 0.99607843, 0.30980393, 0.14509805, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.27450982, 0.8862745, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.9
9215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.7764706, 0.15294118, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5019608, 0.9529412, 0.9529412, 0.9529412, 0.9529412, 0.9529412,
0.9529412, 0.87058824, 0.21568628, 0.21568628, 0.21568628, 0.5176471, 0.98039216, 0.99215686, 0.99215686, 0.8392157
, 0.023529412, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.6627451, 0.99215686, 0.99215686, 0.99215686, 0.03137255, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.49803922, 0.5529412, 0.84705883, 0.99215686, 0.99
215686, 0.59607846, 0.015686275, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.06666667, 0.07450981, 0.5411765, 0.972549, 0.99215686, 0.99215686, 0.99215686, 0.627451, 0.05490196, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1882353, 0.33333334, 0.33333334, 0.33333334
, 0.8980392, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.92941177, 0.3647059, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.18039216, 0.8156863, 0.99215686, 0.99215686, 0.99215686, 0
.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.91764706, 0.32941177, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.26666668, 0.9411765, 0.99215686, 0.99215686, 0.99215686, 0
.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.59607846, 0.03529412, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.34901962, 0.61960787, 0.61960787, 0.61960787,
0.61960787, 0.61960787, 0.61960787, 0.9529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.54901963, 0.011764
706, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.325
4902, 0.3647059, 0.78431374, 0.99215686, 0.99215686, 0.99215686, 0.5019608, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07058824, 0.27058825, 0.99215686, 0.99
215686, 0.96862745, 0.45882353, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.10980392, 0.7647059, 0.99215686, 0.99215686, 0.50980395, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.44705883, 0.992156
86, 0.99215686, 0.50980395, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.1254902, 0.3254902, 0.9529412, 0.99215686, 0.9490196, 0.41960785, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.18431373, 0.8117647, 0.99215686,
0.99215686, 0.99215686, 0.76862746, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.57
64706, 0.47058824, 0.0, 0.08235294, 0.7019608, 0.7019608, 0.7019608, 0.81960785, 0.99215686, 0.99215686, 0.99215686
, 0.99215686, 0.24705882, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5372549, 0.984313
7, 0.98039216, 0.9607843, 0.9647059, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.9490196, 0.43921
57, 0.03137255, 0.003921569, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5529412, 0.992
15686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.7921569, 0.50980395, 0.41960785, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.14117648, 0.30980393, 0.99215
686, 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.32941177, 0.14901961, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Training and Test Sets

Training and Test Sets

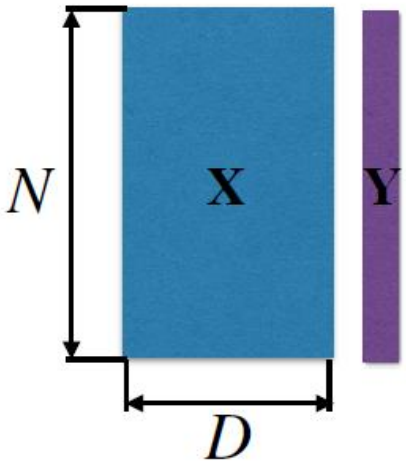
Design matrix (dimension $N \times D$):



I.i.d. samples from a data-generating distribution p_{data}

Training and Test Sets

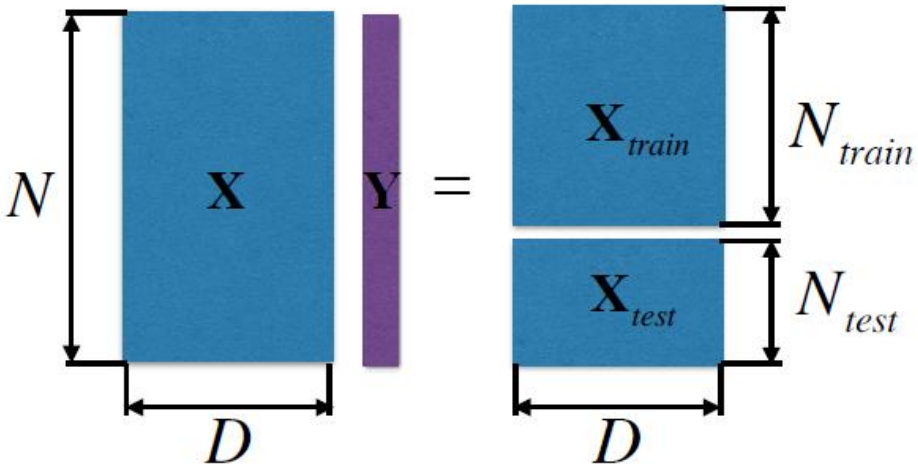
Design matrix (dimension $N \times D$):



I.i.d. samples from a data-generating distribution p_{data}

Training and Test Sets

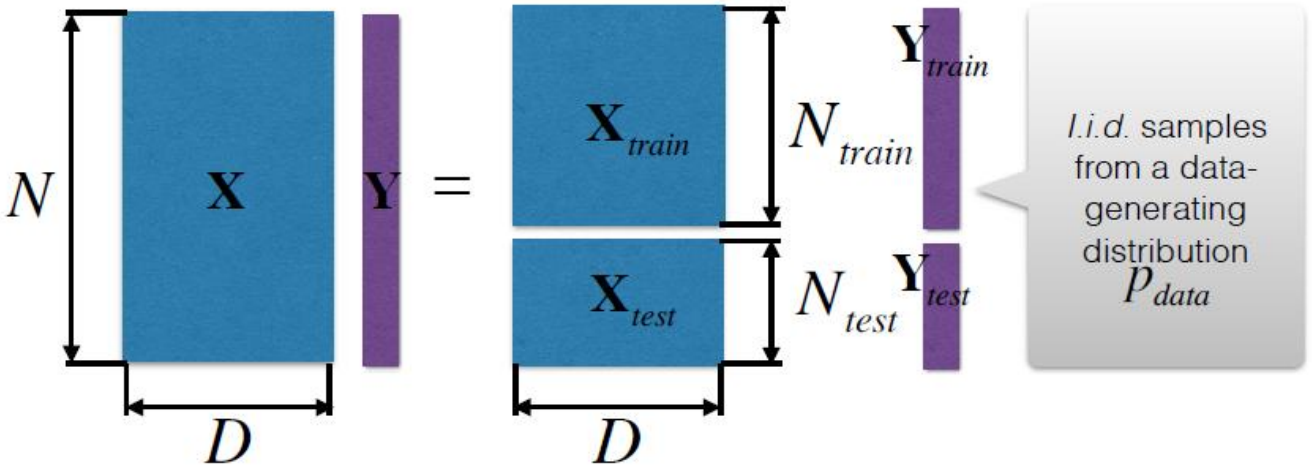
Design matrix (dimension $N \times D$):



I.i.d. samples from a data-generating distribution p_{data}

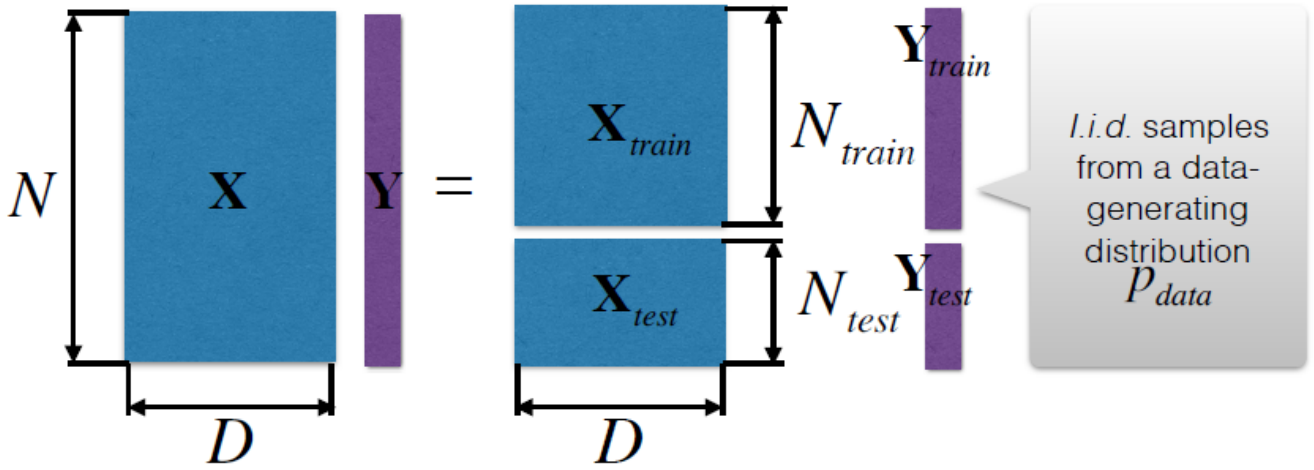
Training and Test Sets

Design matrix (dimension $N \times D$):



Training and Test Sets

Design matrix (dimension $N \times D$):

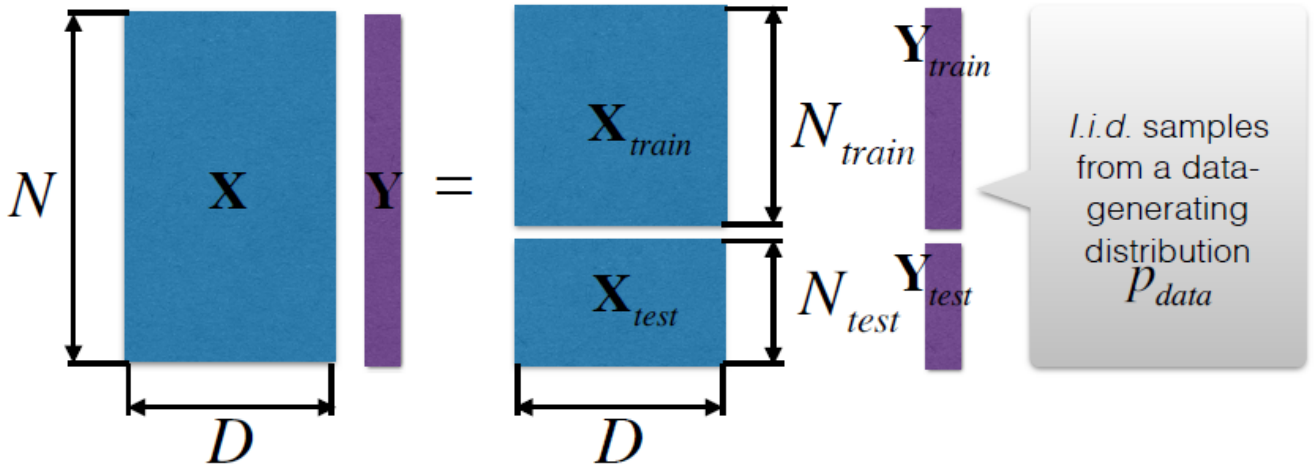


A model is trained using only a **training set** $(\mathbf{X}_{train}, y_{train}) \sim p_{data}$

A **test set** $(\mathbf{X}_{test}, y_{test}) \sim p_{data}$ is used to estimate algorithm's ability to **generalize**, i.e. perform well on unseen data.

Training and Test Sets

Design matrix (dimension $N \times D$):

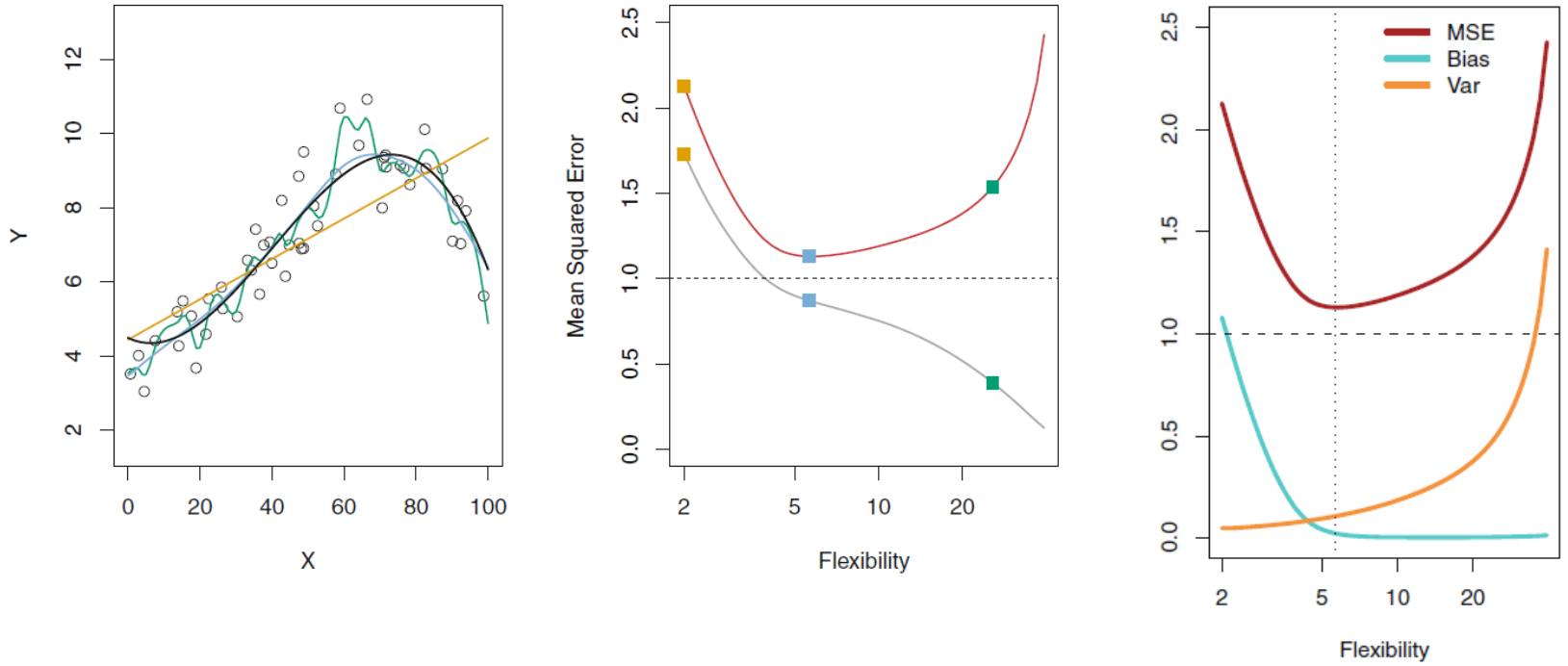


A model is trained using only a **training set** $(\mathbf{X}_{train}, y_{train}) \sim p_{data}$

A **test set** $(\mathbf{X}_{test}, y_{test}) \sim p_{data}$ is used to estimate algorithm's ability to **generalize**, i.e. perform well on unseen data.

More specifically, a test set is used to detect when a ML algorithm starts to **overfit** data, by estimating a generalization error by a test error.

The Bias-Variance Tradeoff

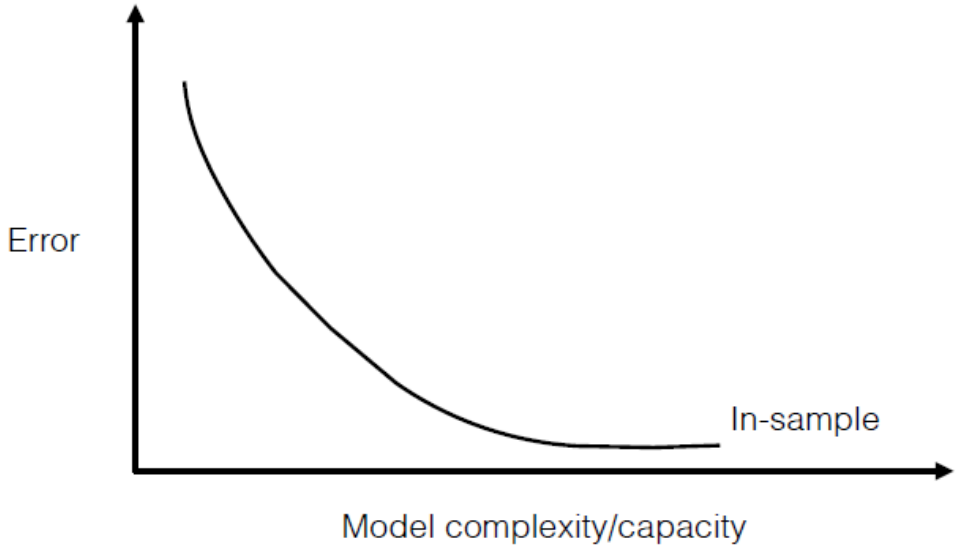


Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves). Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.*

Overfitting

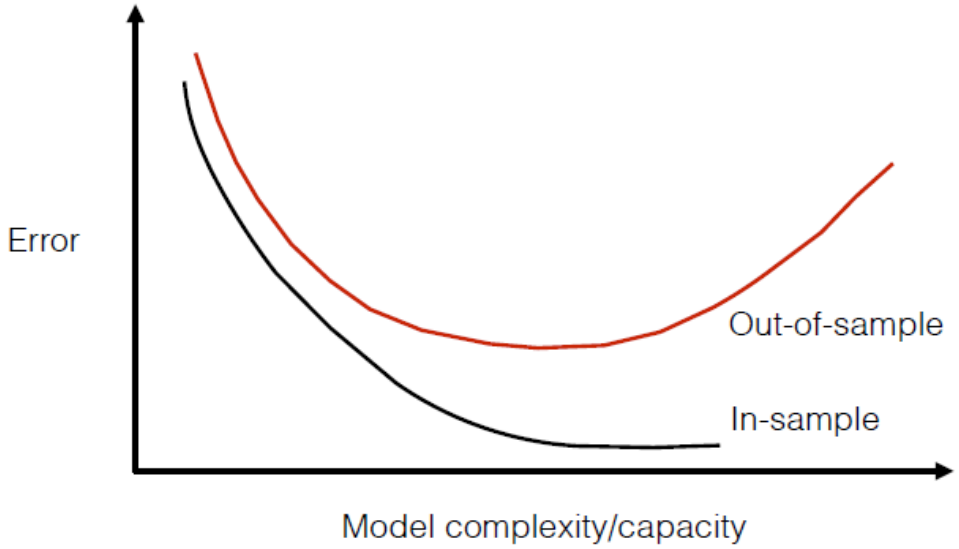
Overfitting

Trying to exactly match all available data is almost always a bad idea



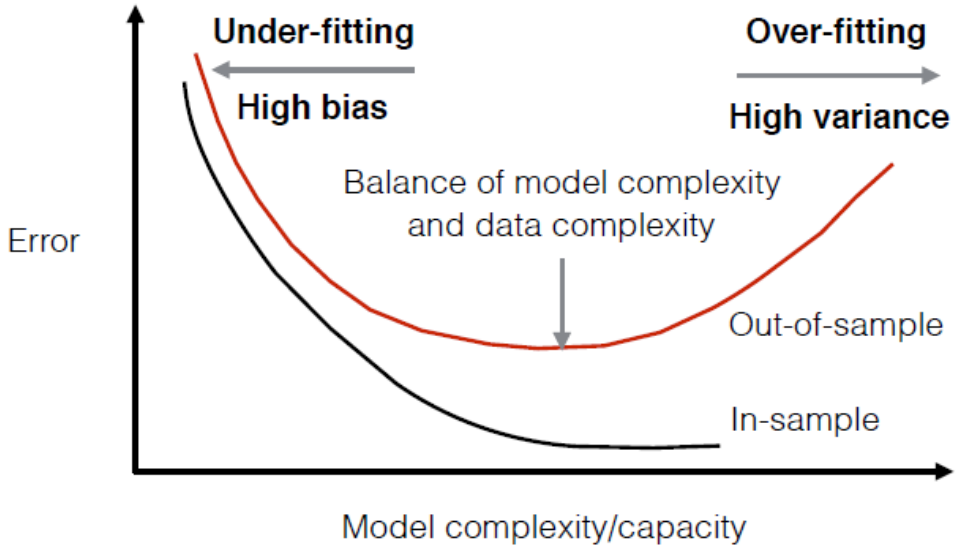
Overfitting

Trying to exactly match all available data is almost always a bad idea



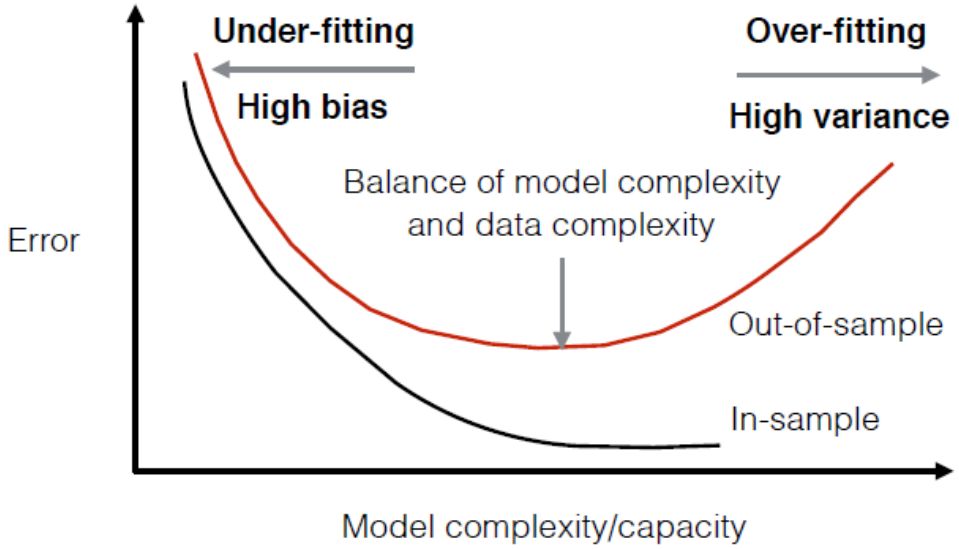
Overfitting

Trying to exactly match all available data is almost always a bad idea



Overfitting

Trying to exactly match all available data is almost always a bad idea



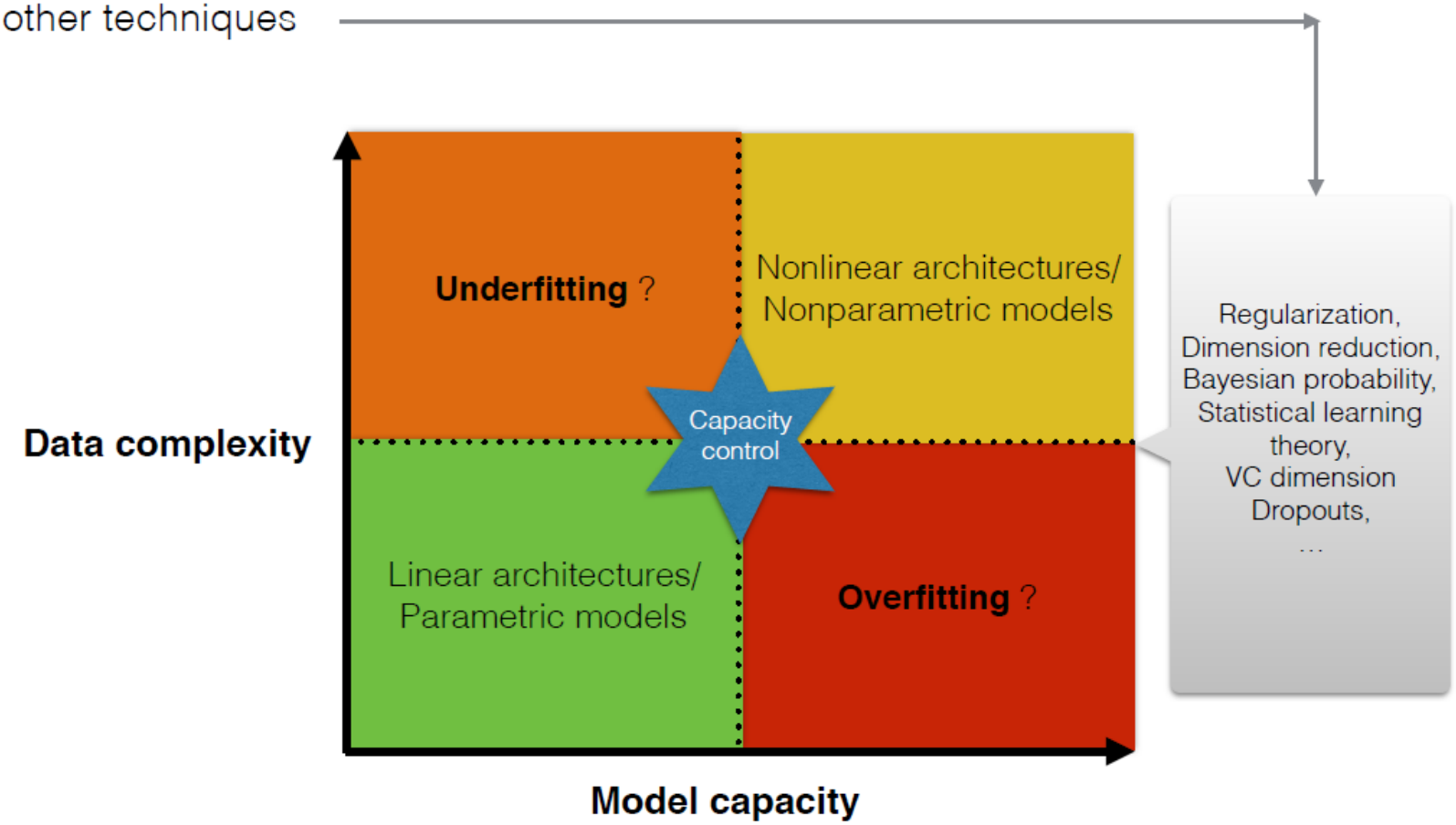
A good ML algorithm should achieve **two goals**:

1. Make the **training error** small (avoid under-fitting)
2. Make the gap between training and **test errors** small (avoid over-fitting)

Key ingredients: 1) data is i.i.d. $\sim p_{data}$, and 2) model **capacity control**

Model Capacity and Overfitting

- Model **capacity** controls model's ability to fit a wide variety of functions.
- Models with low capacity can **under-fit**, but models with high capacity can **over-fit**!
- Capacity is controlled by the choice of a **hypothesis space** (architecture), and other techniques



Linear Regression as a ML Task

Task: predict a scalar value $y \in \mathbb{R}$ from a vector of predictors ("**features**") $\mathbf{X} = (X_0, X_1, \ldots, X_{D-1}) \in \mathbb{R}^D$ ($D \geq 1$ is the dimension of the feature space, or the number of predictors).

Linear Regression as a ML Task

Task: predict a scalar value $y \in \mathbb{R}$ from a vector of predictors ("**features**") $\mathbf{X} = \left(X_0, X_1, \ldots, X_{D-1}\right) \in \mathbb{R}^D$ ($D \geq 1$ is the dimension of the feature space, or the number of predictors).

Given: a dataset $(\mathbf{X}, y)_{data} = \left[\left(\mathbf{X}_{train}, y_{train}\right), \left(\mathbf{X}_{test}, y_{test}\right)\right] \sim p_{data}$

Linear Regression as a ML Task

Task: predict a scalar value $y \in \mathbb{R}$ from a vector of predictors ("**features**") $\mathbf{X} = \left(X_0, X_1, \ldots, X_{D-1}\right) \in \mathbb{R}^D$ ($D \geq 1$ is the dimension of the feature space, or the number of predictors).

Given: a dataset $(\mathbf{X}, y)_{data} = \left[\left(\mathbf{X}_{train}, y_{train}\right), \left(\mathbf{X}_{test}, y_{test}\right)\right] \sim p_{data}$

Architecture: Linear

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{W}$$

N predicted values

$\mathbf{X}$ is a N x D design matrix

$\mathbf{W} \in \mathbb{R}^D$ is a vector of parameters

Linear Regression as a ML Task

Task: predict a scalar value $y \in \mathbb{R}$ from a vector of predictors ("**features**") $\mathbf{X} = \left(X_0, X_1, \ldots, X_{D-1} \right) \in \mathbb{R}^D$ ($D \geq 1$ is the dimension of the feature space, or the number of predictors).

Given: a dataset $(\mathbf{X}, y)_{data} = \left[(\mathbf{X}_{train}, y_{train}), (\mathbf{X}_{test}, y_{test}) \right] \sim p_{data}$

Architecture: Linear

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{W}$$

- N predicted values
- $\mathbf{X}$ is a N x D design matrix
- $\mathbf{W} \in \mathbb{R}^D$ is a vector of parameters

Example: $\mathbf{y}$ is a vector of N daily returns of AMZN, and $\mathbf{X}$ is a $N \times D$ design matrix made of $D-1$ daily market index returns (S&P 500, NASDAQ, VIX, etc.)

Learning in Linear Regression

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$?

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$?
Minimize MSE on the **training set**!
Why? Because both MSE_{train} and MSE_{test} estimate the same generalization (expected) error $\mathbb{E}\left[\left(\hat{\mathbf{y}} - \mathbf{y} \right)^2 \right]$ from the empirical distribution $\sim p_{data}$.

Note: In ML, we often minimize one function, while actually caring about minimization of another function. This makes ML different from optimization that typically focuses only on one function.

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test} \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train} \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

Simplify notation:
$\mathbf{X}^{train} \rightarrow \mathbf{X}, \mathbf{y}^{train} \rightarrow \mathbf{y}$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{W}^T \mathbf{X}^T \mathbf{X}\, \mathbf{W} - 2\mathbf{W}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \right)$$

Simplify notation:
$\mathbf{X}^{train} \rightarrow \mathbf{X}, \mathbf{y}^{train} \rightarrow \mathbf{y}$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train}\, \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

Simplify notation: $\mathbf{X}^{train} \rightarrow \mathbf{X}, \mathbf{y}^{train} \rightarrow \mathbf{y}$

$$\Rightarrow \nabla_W \left(\mathbf{W}^T \mathbf{X}^T \mathbf{X}\, \mathbf{W} - 2\mathbf{W}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \right) = 2\left(\mathbf{X}^T \mathbf{X} \right) \mathbf{W} - 2\left(\mathbf{X}^T \mathbf{y} \right) = 0$$

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test} \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train} \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

Simplify notation: $\mathbf{X}^{train} \rightarrow \mathbf{X}, \mathbf{y}^{train} \rightarrow \mathbf{y}$

$$\Rightarrow \nabla_W \left(\mathbf{W}^T \mathbf{X}^T \mathbf{X} \mathbf{W} - 2 \mathbf{W}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \right) = 2 \left(\mathbf{X}^T \mathbf{X} \right) \mathbf{W} - 2 \left(\mathbf{X}^T \mathbf{y} \right) = 0$$

$$\Rightarrow \boxed{\mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}}$$

Normal equation

Learning in Linear Regression

Performance measure P : mean square error (**MSE**) on the **test set**:

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test} \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

How to optimize parameters $\mathbf{W} \in \mathbb{R}^D$? Minimize MSE on the **training set**! Why? Because both MSE_{test} and MSE_{train} estimate the generalization (expected) error.

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train} \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \nabla_W \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right)^T \left(\mathbf{X}^{train} \mathbf{W} - \mathbf{y}^{train} \right) = 0$$

Simplify notation: $\mathbf{X}^{train} \rightarrow \mathbf{X}, \mathbf{y}^{train} \rightarrow \mathbf{y}$

$$\Rightarrow \nabla_W \left(\mathbf{W}^T \mathbf{X}^T \mathbf{X} \mathbf{W} - 2 \mathbf{W}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \right) = 2 \left(\mathbf{X}^T \mathbf{X} \right) \mathbf{W} - 2 \left(\mathbf{X}^T \mathbf{y} \right) = 0$$

Here $\mathbf{X} = \mathbf{X}^{train}, \mathbf{y} = \mathbf{y}^{train}$

$$\Rightarrow \quad \mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

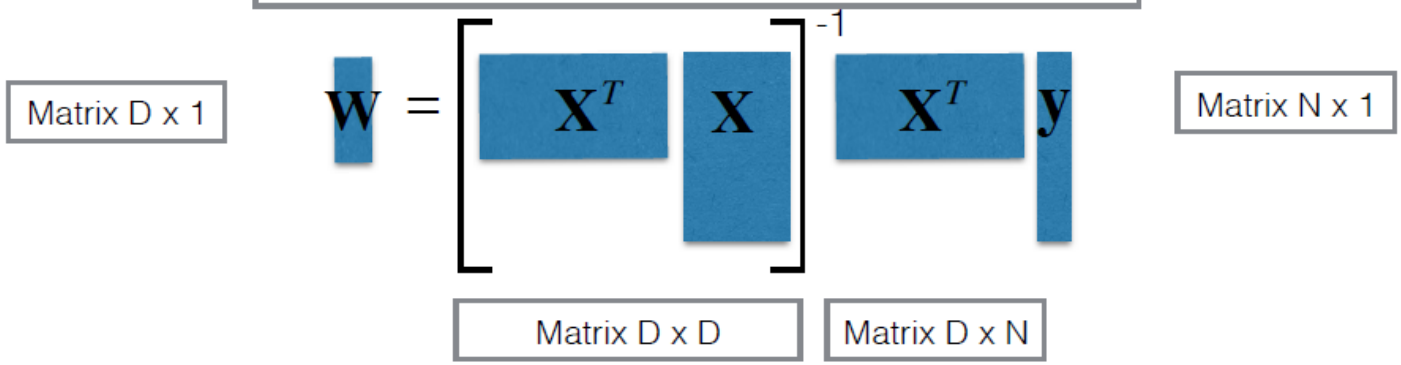
Normal equation

Learning in Linear Regression

Learning in Linear Regression

Visualize the result for optimal weights $\mathbf{W}$ in the vector form

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$



Prediction in-sample:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{W} = \underbrace{\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T}_{=\mathbf{H}}\mathbf{y} \equiv \mathbf{H}\mathbf{y}$$

The "hat" (projection) matrix

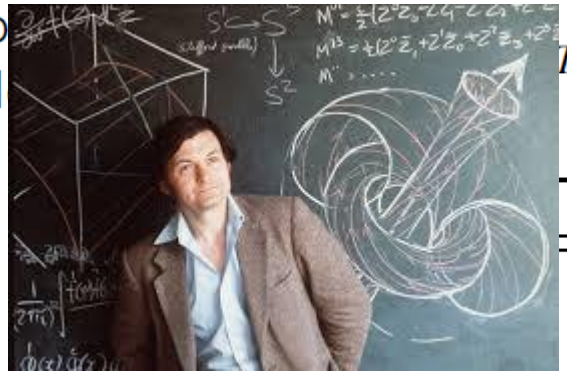
Learning in Linear Regression

Visualize the result for optimal weights $\mathbf{W}$ in the vector form

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Matrix D x 1 | Matrix D x D | Matrix D x N | Matrix N x 1

Sir R. Penrose
Nobel Prize in Physics 2020



Predictio
in-sampl

$$\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \equiv \mathbf{H}\mathbf{y}$$

$= \mathbf{H}$ The "hat" (projection) matrix

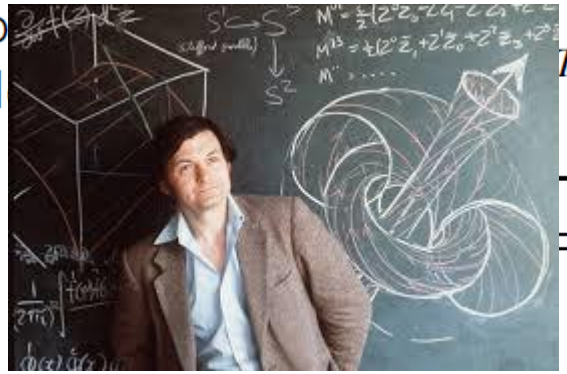
Learning in Linear Regression

Visualize the result for optimal weights $\mathbf{W}$ in the vector form

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Matrix D x 1 | Matrix D x D | Matrix D x N | Matrix N x 1

Sir R. Penrose
Nobel Prize in Physics 2020



Predictio
in-sampl

$$\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \equiv \mathbf{H}\mathbf{y}$$

Symmetric and idempotent: $\mathbf{H}^2 = \mathbf{H}$

$\mathbf{H}$

The "hat" (projection) matrix

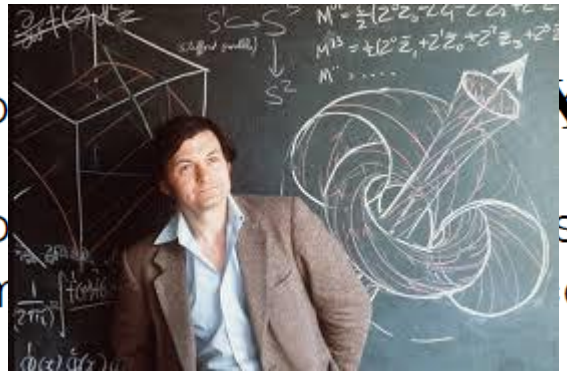
Learning in Linear Regression

Visualize the result for optimal weights $\mathbf{W}$ in the vector form

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Matrix D x 1 | Matrix D x D | Matrix D x N | Matrix N x 1

Sir R. Penrose
Nobel Prize in Physics 2020



Predictio $\mathbf{X}^{test}\mathbf{W}$

How to o ssion: add more features to the design m ontrol!

Regularization

Regularization

Recall that we minimize MSE_{train}, though want in fact to minimize MSE_{test}
The idea of regularization is to modify the objective function of minimization of MSE_{train} so that MSE_{test} has a smaller variance:

$$J(\mathbf{W}) = MSE_{train}(\mathbf{W}) + \lambda \Omega(\mathbf{W})$$

- Regularized loss function
- Train loss
- Regularization parameter
- Regularizer

Regularization

Recall that we minimize MSE_{train}, though want in fact to minimize MSE_{test}
The idea of regularization is to modify the objective function of minimization of MSE_{train} so that MSE_{test} has a smaller variance:

$$J(\mathbf{W}) = MSE_{train}(\mathbf{W}) + \lambda \Omega(\mathbf{W})$$

- Regularized loss function
- Train loss
- Regularization parameter
- Regularizer

Popular choices for the **regularizer**:

- L_2 regularization $\Omega(\mathbf{W}) = \mathbf{W}^T\mathbf{W} = ||\mathbf{W}||_2$ — Penalizes large weights
- L_1 regularization $\Omega(\mathbf{W}) = \sum_i |W_i| = ||\mathbf{W}||_1$ — Enforces a sparse solution
- Entropy regularization $\Omega(\mathbf{W}) = \sum_i W_i \log W_i \quad (W_i \geq 0, \sum_i W_i = 1)$ — Motivated by Bayesian statistics

Hyperparameters and Validation Set

Hyperparameters and Validation Set

- **Hyperparameters** are any quantitative features of ML models that are not directly optimized by minimizing in-sample loss such as MSE_{train}
- **Hyperparameters control model capacity**

Hyperparameters and Validation Set

- **Hyperparameters** are any quantitative features of ML models that are not directly optimized by minimizing in-sample loss such as MSE_{train}

- **Hyperparameters control model capacity**

- Examples of hyperparameters:
  - Degree of a polynomial regression (linear, quadratic, cubic, etc.)
  - Regularization parameter λ
  - Number of levels in a decision tree
  - Number of layers and nodes per layer in neural networks
  - Learning rate
  - …

Hyperparameters and Validation Set

- **Hyperparameters** are any quantitative features of ML models that are not directly optimized by minimizing in-sample loss such as MSE_{train}
- **Hyperparameters control model capacity**
- Examples of hyperparameters:
  - Degree of a polynomial regression (linear, quadratic, cubic, etc.)
  - Regularization parameter λ
  - Number of levels in a decision tree
  - Number of layers and nodes per layer in neural networks
  - Learning rate
  - …
- How to choose hyperparameters:
  - Split a training set into training and **validation** sets (e.g. as 80:20)

Hyperparameters and Validation Set

- **Hyperparameters** are any quantitative features of ML models that are not directly optimized by minimizing in-sample loss such as MSE_{train}

- **Hyperparameters control model capacity**

- Examples of hyperparameters:
  - Degree of a polynomial regression (linear, quadratic, cubic, etc.)
  - Regularization parameter λ
  - Number of levels in a decision tree
  - Number of layers and nodes per layer in neural networks
  - Learning rate
  - …

- How to choose hyperparameters:
  - Split a training set into training and **validation** sets (e.g. as 80:20)
  - Use **cross-validation**

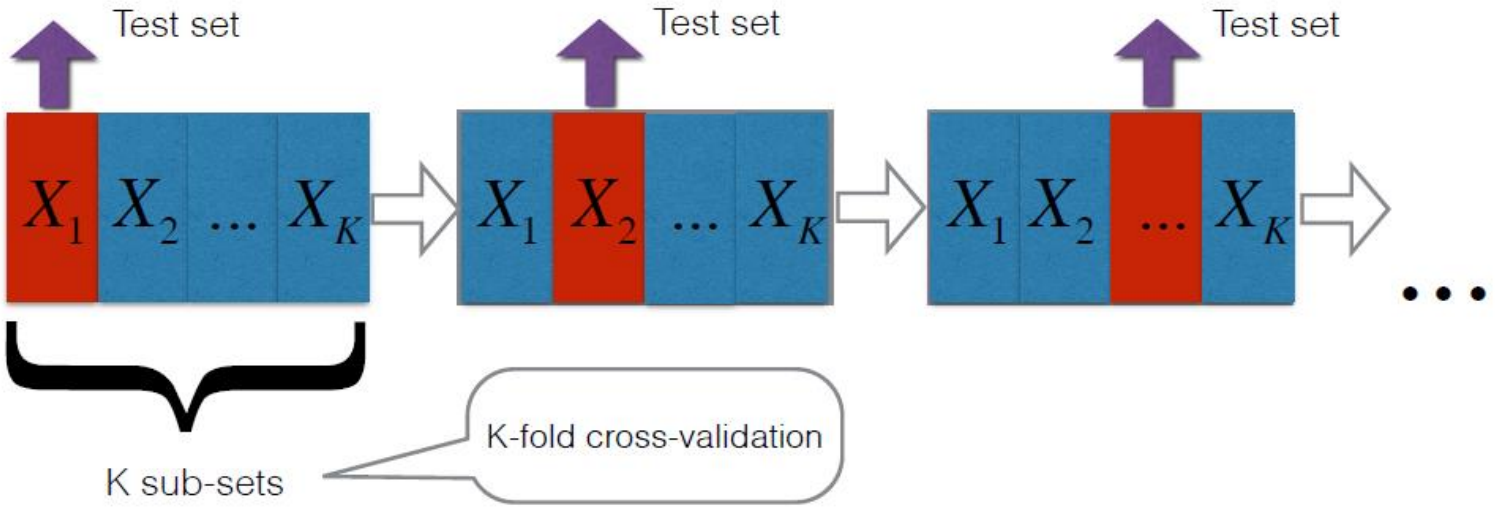
Cross-Validation

Cross-Validation

- Assume we are given N samples, but N is small, so setting aside a fixed test set is problematic.
- We want to use all samples for training!
- This is achieved using cross-validation:

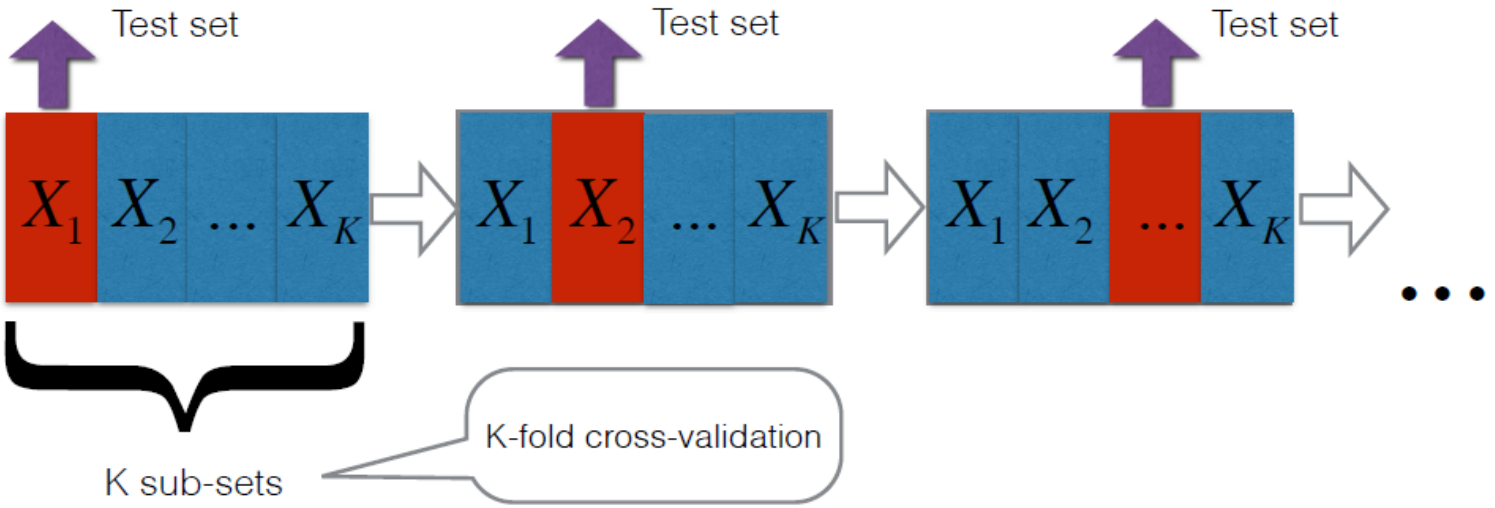
Cross-Validation

- Assume we are given N samples, but N is small, so setting aside a fixed test set is problematic.
- We want to use all samples for training!
- This is achieved using cross-validation:



Cross-Validation

- Assume we are given N samples, but N is small, so setting aside a fixed test set is problematic.
- We want to use all samples for training!
- This is achieved using cross-validation:

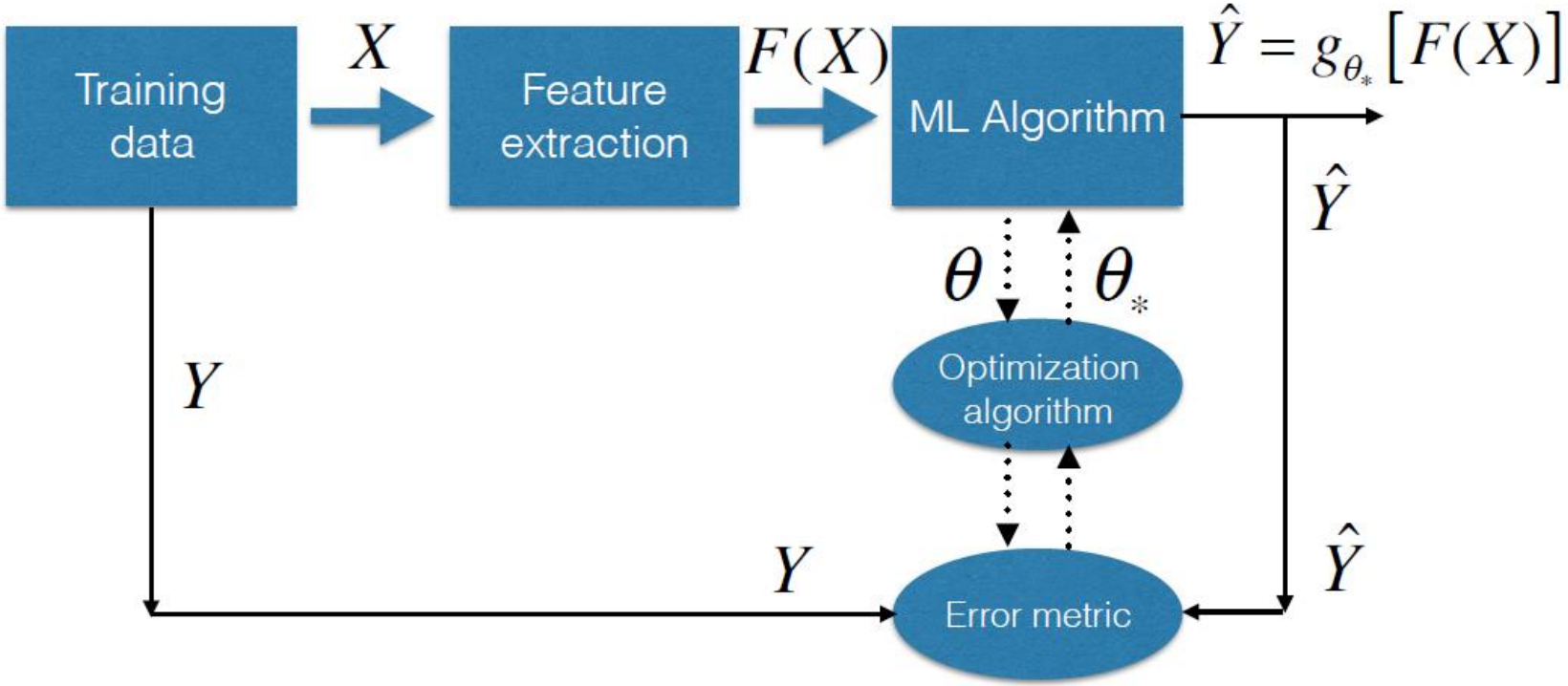


Special cases:

- K=1: no test sub-set!
- K= N: leave-on-out cross-validation

Supervised Learning Diagram

General diagram for training supervised learning algorithms



Analytical Learning in Linear Reg

Analytical Learning in Linear Reg

Performance measure P : mean square error (**MSE**) MSE_{test} :

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

Parameters $\mathbf{W} \in \mathbb{R}^D$ are found by minimizing MSE_{train}

$$\mathrm{MSE}_{train} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \left(\hat{y}_i^{train} - y_i^{train} \right)^2 = \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2$$

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \quad \boxed{\mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}}$$

Normal equation

Analytical Learning in Linear Reg

Performance measure P : mean square error (**MSE**) MSE_{test} :

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

Parameters $\mathbf{W} \in \mathbb{R}^D$ are found by minimizing MSE_{train}

$$\mathrm{MSE}_{train} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \left(\hat{y}_i^{train} - y_i^{train} \right)^2 = \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2$$

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \quad \mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

Normal equation

Only works analytically for linear regression

Analytical Learning in Linear Reg

Performance measure P : mean square error (**MSE**) MSE_{test} :

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test}\, \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

Parameters $\mathbf{W} \in \mathbb{R}^D$ are found by minimizing MSE_{train}

$$\mathrm{MSE}_{train} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \left(\hat{y}_i^{train} - y_i^{train} \right)^2 = \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2$$

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train}\, \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \quad \mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

Normal equation

Only works analytically for linear regression

Complexity $O\left(N^3\right)$ to $O\left(N^{2.4}\right)$

Analytical Learning in Linear Reg

Performance measure P : mean square error (**MSE**) MSE_{test} :

$$\mathrm{MSE}_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \left(\hat{y}_i^{test} - y_i^{test} \right)^2 = \frac{1}{N_{test}} \left\| \hat{\mathbf{Y}}^{test} - \mathbf{Y}^{test} \right\|_2^2 = \frac{1}{N_{test}} \left\| \mathbf{X}^{test} \mathbf{W} - \mathbf{Y}^{test} \right\|_2^2$$

Parameters $\mathbf{W} \in \mathbb{R}^D$ are found by minimizing MSE_{train}

$$\mathrm{MSE}_{train} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \left(\hat{y}_i^{train} - y_i^{train} \right)^2 = \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \left\| \mathbf{X}^{train} \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2$$

For optimal weights $\mathbf{W}$, the gradient of MSE_{train} should be 0:

$$\nabla_W \mathrm{MSE}_{train} = \nabla_W \frac{1}{N_{train}} \left\| \hat{\mathbf{Y}}^{train} - \mathbf{Y}^{train} \right\|_2^2 = \frac{1}{N_{train}} \nabla_W \left\| \mathbf{X}^{train} \mathbf{W} - \mathbf{Y}^{train} \right\|_2^2 = 0$$

$$\Rightarrow \quad \mathbf{W} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

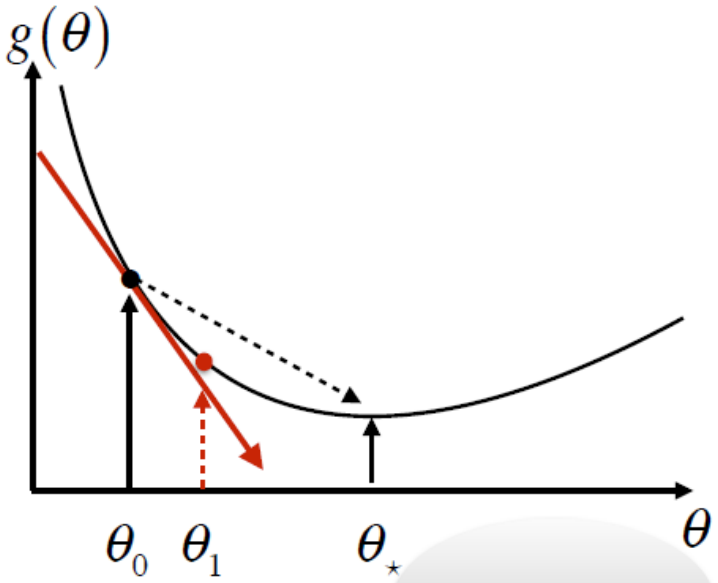
Only works analytically for linear regression

Complexity $O\left(N^3\right)$ to $O\left(N^{2.4}\right)$

Gradient Descent

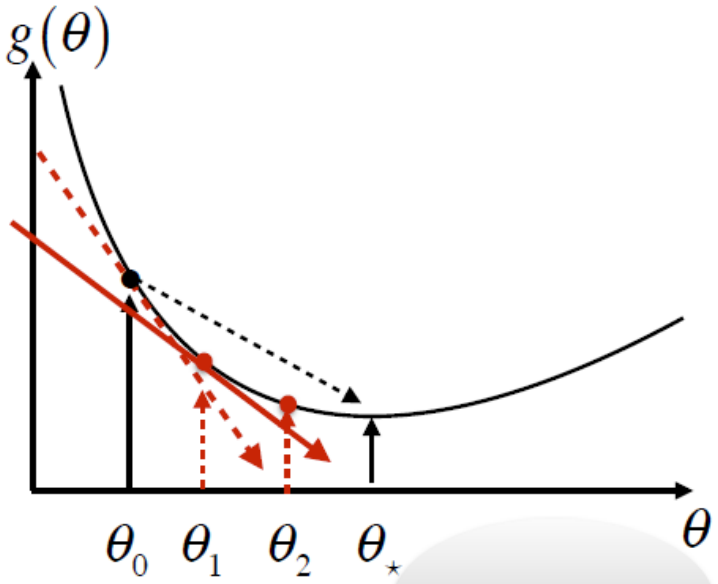
Gradient Descent

Optimize weights $\mathbf{W}$ **gradually** by moving against the gradients of MSE_{train}



Gradient Descent

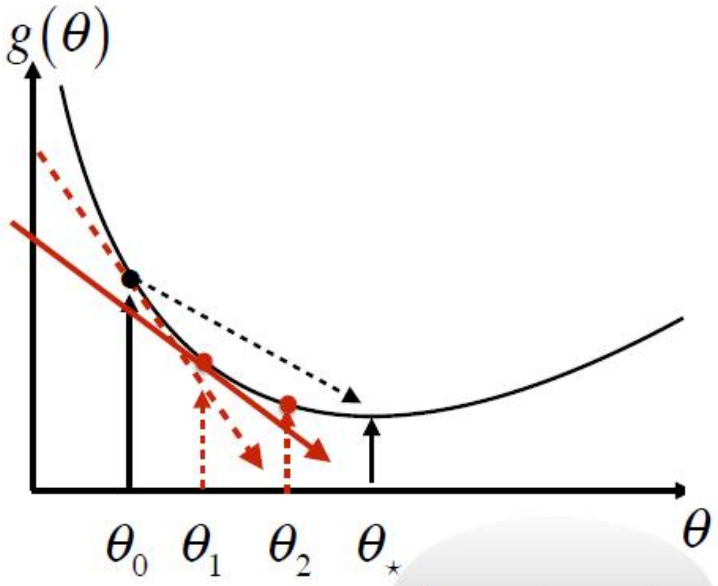
Optimize weights $\mathbf{W}$ **gradually** by moving against the gradients of MSE_{train}



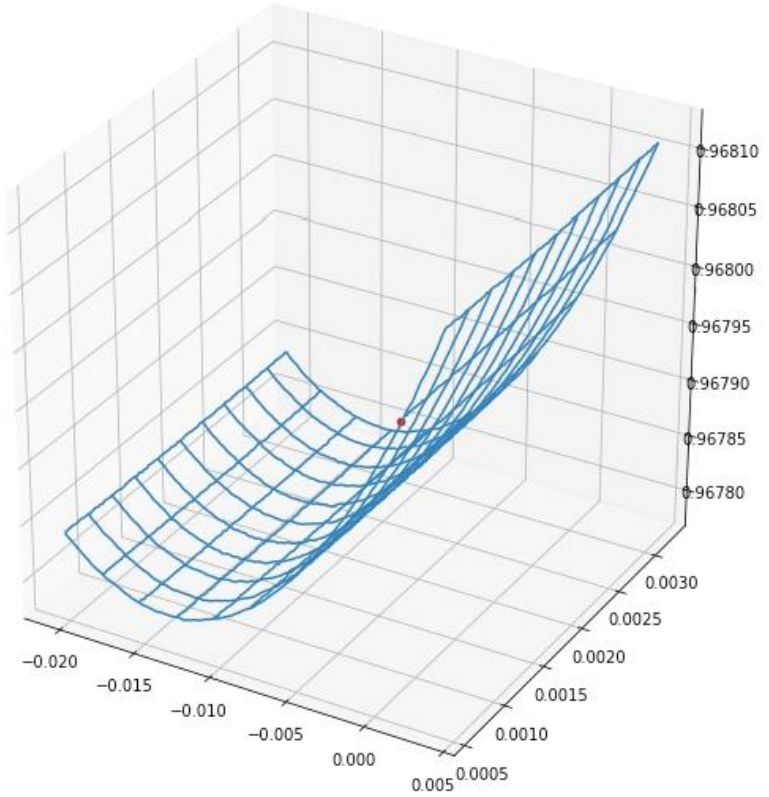
$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$

Gradient Descent

Optimize weights **W** **gradually** by moving against the gradients of MSE_{train}

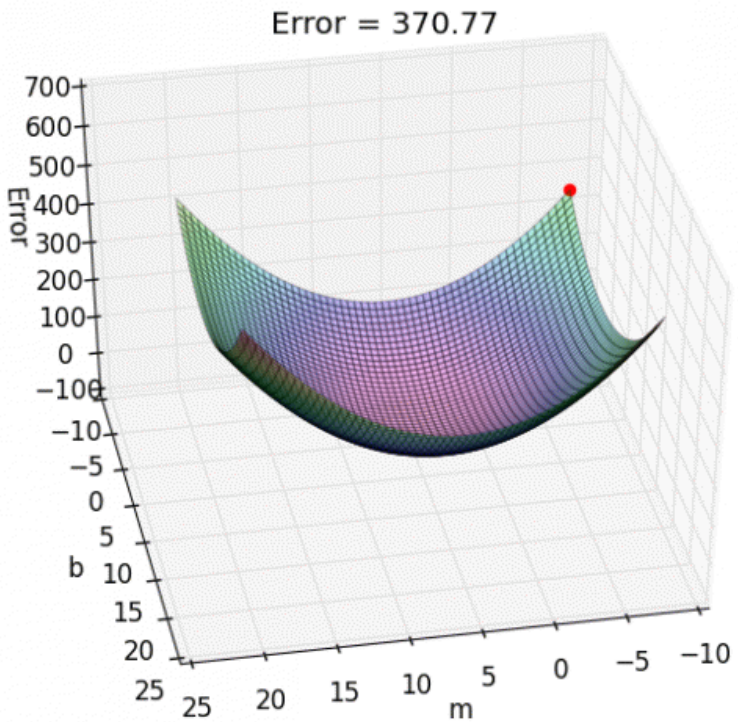


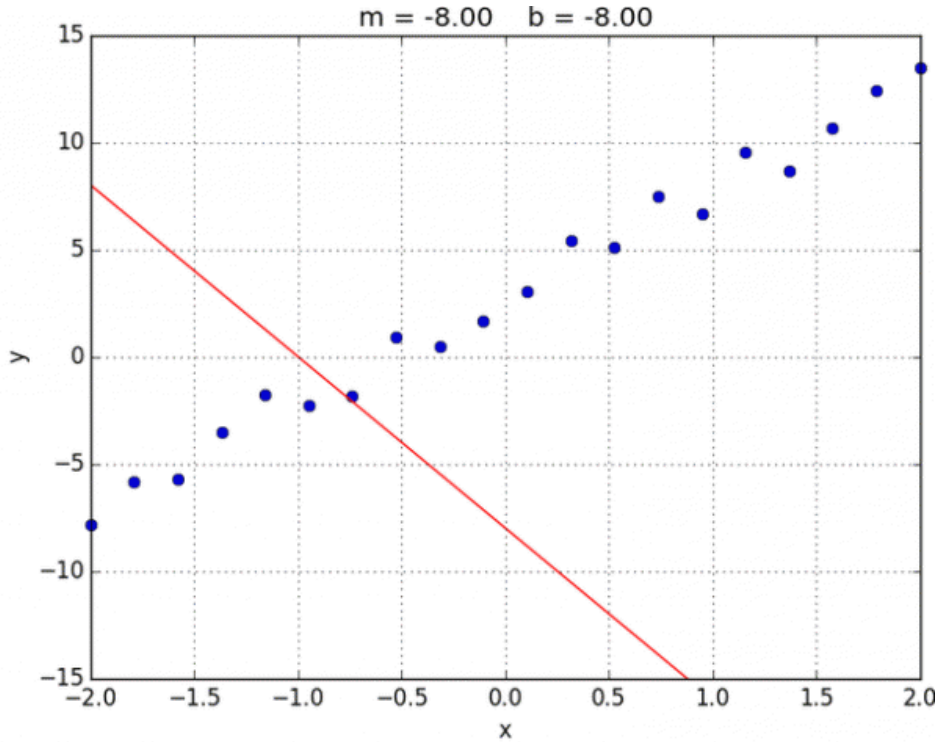
$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$



Gradient Descent

Gradient Descent

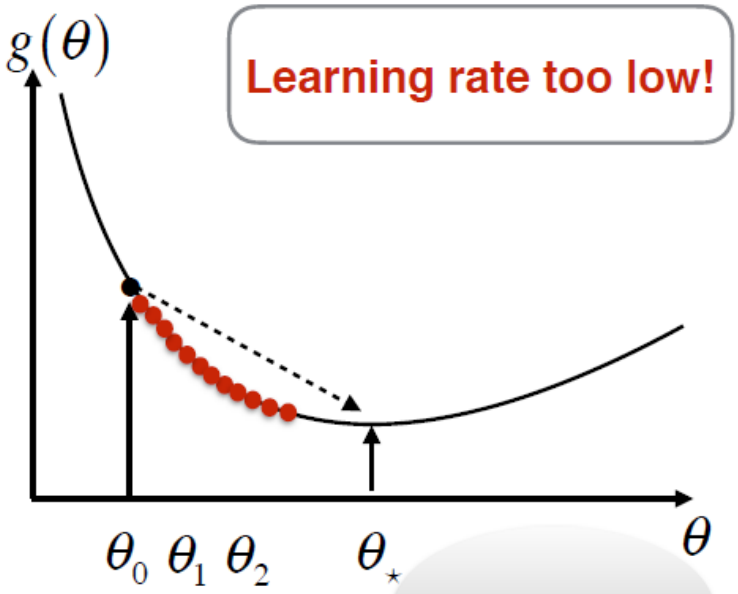




Choice of Learning Rate

Choice of Learning Rate

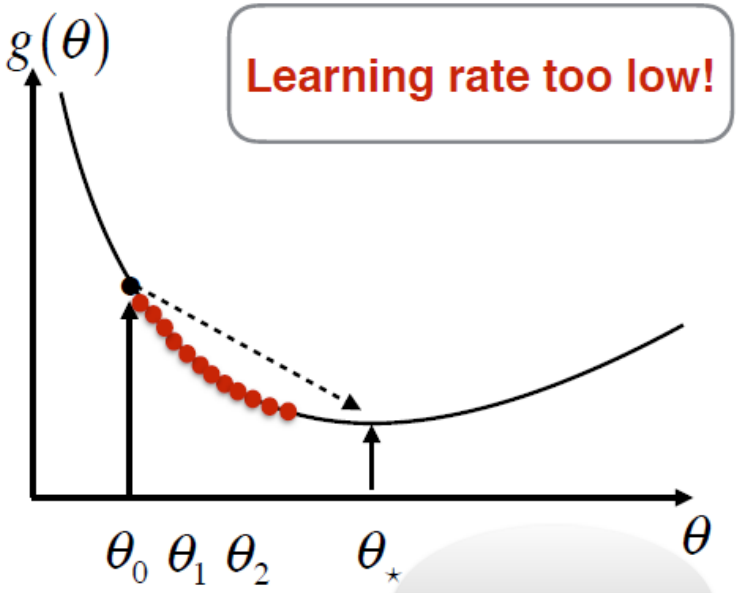
Optimize weights **W** **gradually** by moving against the gradients of MSE_{train}

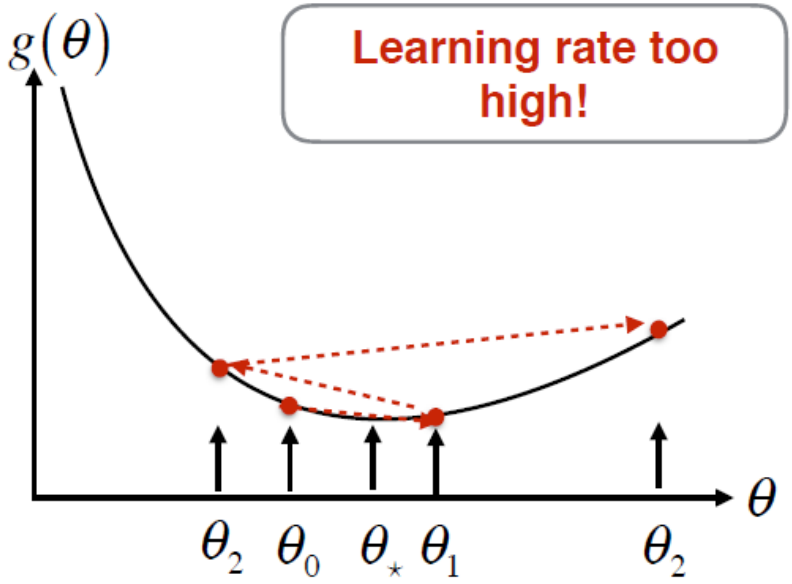


$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$

Choice of Learning Rate

Optimize weights **W** **gradually** by moving against the gradients of MSE_{train}

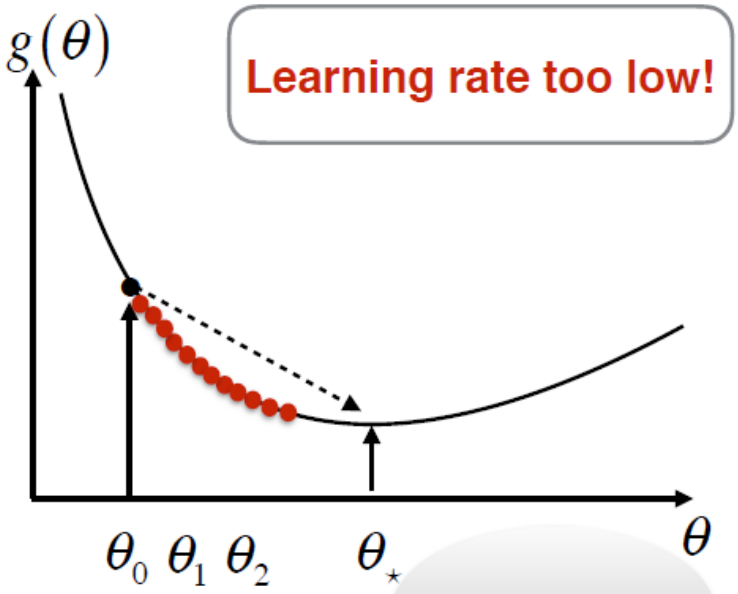




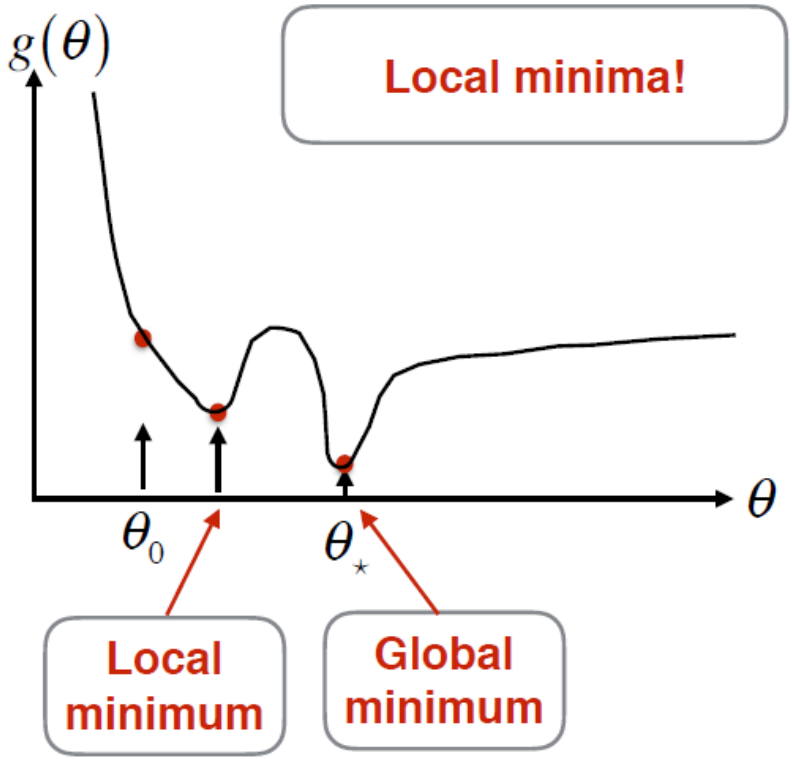
$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$

Choice of Learning Rate

Optimize weights **W** **gradually** by moving against the gradients of MSE_{train}

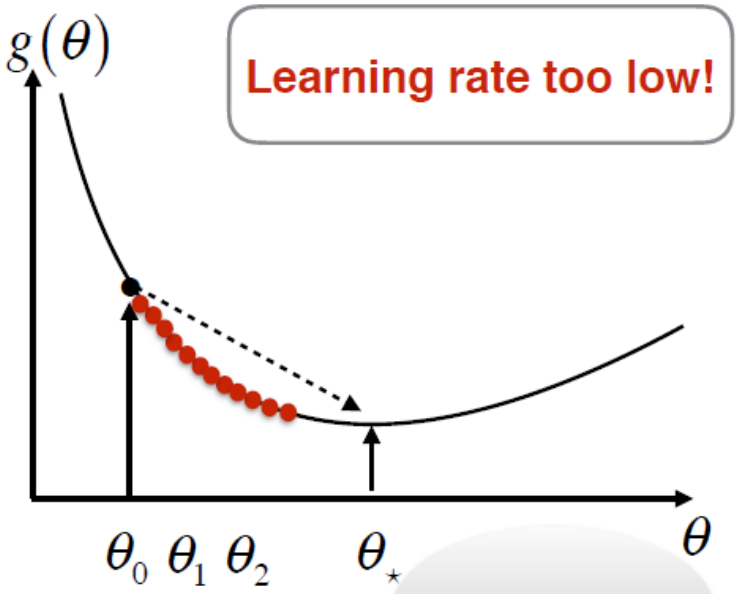


$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$

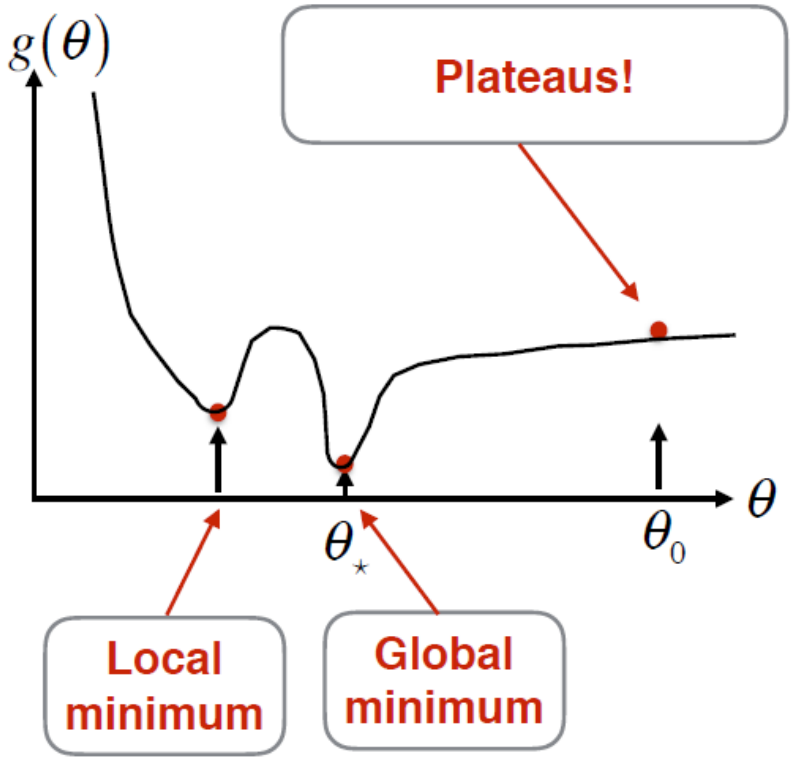


Choice of Learning Rate

Optimize weights **W** **gradually** by moving against the gradients of MSE_{train}



$$\theta_1 = \theta_0 - \eta g'_\theta(\theta_0)$$
$$\theta_2 = \theta_1 - \eta g'_\theta(\theta_1)$$
$$\vdots$$



Stochastic Gradient Descent

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

:

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

Gradient Descent:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

:

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

Gradient Descent:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

Stochastic Gradient Descent (SGD) with randomly selected **mini-batches** M_k of some fixed size $N_{MB} \ll N$:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i \in M_k} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

Gradient Descent:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

Stochastic Gradient Descent (SGD) with randomly selected mini-batches M_k of some fixed size $N_{MB} \ll N$:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i \in M_k} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

As mini-batches are randomly drawn from a data-generating distribution, **SGD minimizes the generalization error plus noise:**

$$w^{(k+1)} = w^{(k)} - \eta \nabla_w E[f] + \varepsilon^{(k)}$$

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

Gradient Descent:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

Stochastic Gradient Descent (SGD) with randomly selected mini-batches M_k of some fixed size $N_{MB} \ll N$:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i \in M_k} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

As mini-batches are randomly drawn from a data-generating distribution, SGD minimizes the generalization error plus noise:

$$w^{(k+1)} = w^{(k)} - \eta \nabla_w E[f] + \varepsilon^{(k)}$$

On-line SGD ($N_{MB} = 1$): convergence is slow, but guaranteed under certain conditions on η

Stochastic Gradient Descent

Gradient descent minimizes a train error - an approximation to the generalization error

$$E[f] = \int L(f(x), y)\, d\, P(x, y) \Rightarrow E_n[f] = \frac{1}{N} \sum_{i=1}^{N} L\left(f(x_i, w), y_i\right)$$

Gradient Descent:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

Stochastic Gradient Descent (SGD) with randomly selected mini-batches M_k of some fixed size $N_{MB} \ll N$:

$$w^{(k+1)} = w^{(k)} - \eta \frac{1}{N} \sum_{i \in M_k} \nabla_w L\left(f(x_i, w^{(k)}), y_i\right)$$

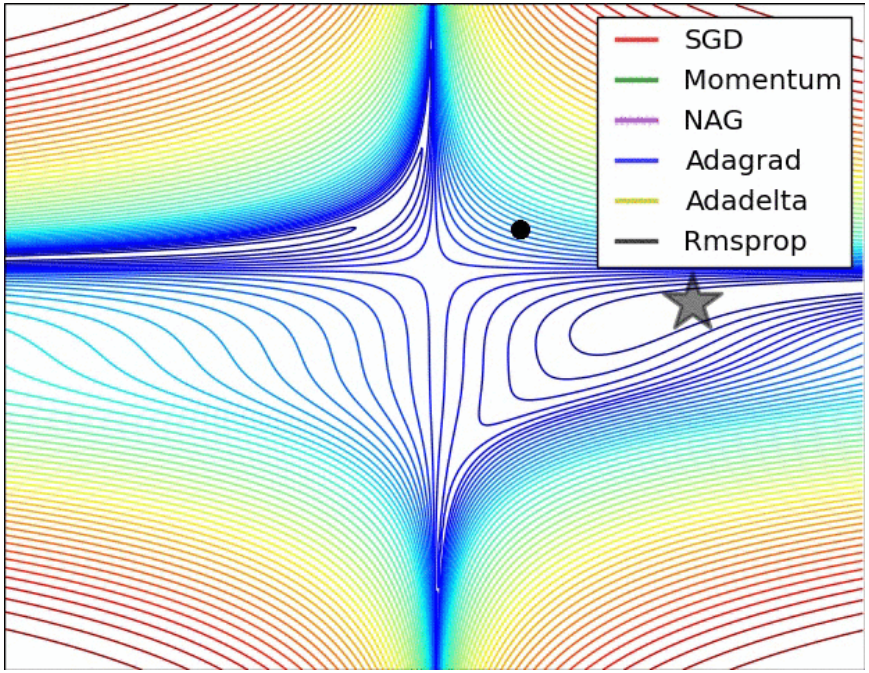
As mini-batches are randomly drawn from a data-generating distribution, SGD minimizes the generalization error plus noise:

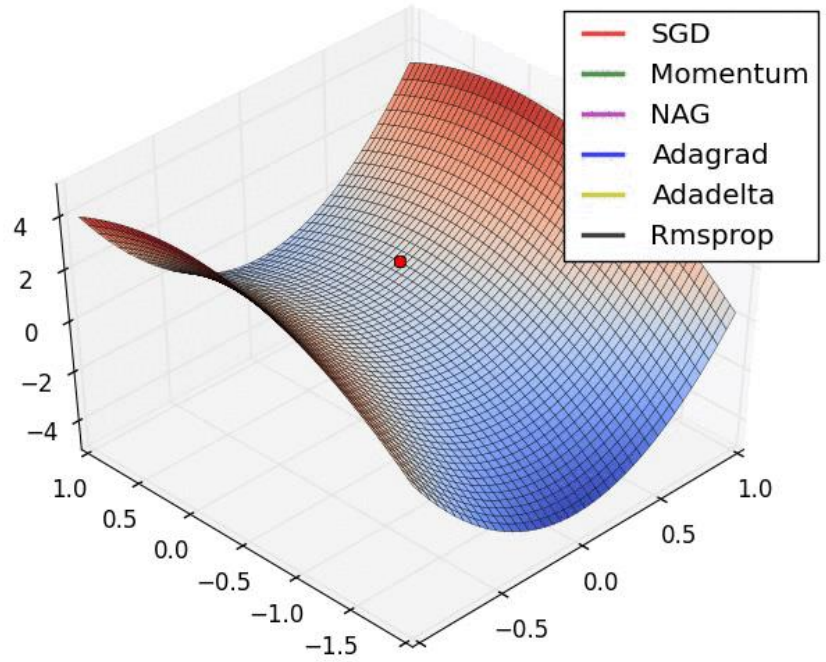
$$w^{(k+1)} = w^{(k)} - \eta \nabla_w E[f] + \varepsilon^{(k)}$$

On-line SGD ($N_{MB} = 1$): convergence is slow, but guaranteed under certain conditions on η

SGD is one of the most important algorithms for neural networks, and for ML in general!

Advanced Variants of SGD





The Bayesian Recipe

Sum

$$p(A+B) = p(A) + p(B) - p(AB)$$

Product

$$p(AB) = p(A|B)p(B) = p(B|A)p(A)$$

Bayes' theorem

$$p(A|B) = \frac{p(B|A)\,p(A)}{p(B)}$$

Law of Total Probability

$$p(A|I) = \sum_i p(A|B_i, I)p(B_i|I)$$

for $\{B_i\}$ exclusive and exhaustive

Normalisation

$$\sum_i p(B_i|I) = 1$$

for $\{B_i\}$ exclusive and exhaustive

ML Training for Probabilistic Models

ML Training for Probabilistic Models

ML essentially amounts to a **model estimation** problem: What are the most probable values of parameters Θ, given the model $\mathbf{M} = f(\mathbf{X}, \Theta)$ and data $\mathbf{D}$? That is, we need to find Θ that maximizes $p(\Theta \mid \mathbf{D}, \mathbf{M})$
Use **Bayes' rule**: (assuming our model is a **probabilistic model!**)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})}$$

ML Training for Probabilistic Models

ML essentially amounts to a **model estimation** problem: What are the most probable values of parameters Θ, given the model $\mathbf{M} = f(\mathbf{X}, \Theta)$ and data $\mathbf{D}$? That is, we need to find Θ that maximizes $p(\Theta \mid \mathbf{D}, \mathbf{M})$
Use **Bayes' rule**: (assuming our model is a **probabilistic model!**)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})}$$

$$= \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{\int p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})\, d\Theta}$$

ML Training for Probabilistic Models

ML essentially amounts to a **model estimation** problem: What are the most probable values of parameters Θ, given the model $\mathbf{M} = f(\mathbf{X}, \Theta)$ and data $\mathbf{D}$? That is, we need to find Θ that maximizes $p(\Theta \mid \mathbf{D}, \mathbf{M})$
Use **Bayes' rule**: (assuming our model is a **probabilistic model!**)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})}$$

$$= \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{\int p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})\, d\Theta}$$

$$= \max_{\Theta} \frac{Likelihood \times Prior}{Evidence}$$

ML Training for Probabilistic Models

ML essentially amounts to a **model estimation** problem: What are the most probable values of parameters Θ, given the model $\mathbf{M} = f(\mathbf{X}, \Theta)$ and data $\mathbf{D}$? That is, we need to find Θ that maximizes $p(\Theta \mid \mathbf{D}, \mathbf{M})$
Use **Bayes' rule**: (assuming our model is a **probabilistic model!**)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})}$$

$$= \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{\int p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})\, d\Theta}$$

$$= \max_{\Theta} \frac{Likelihood \times Prior}{Evidence}$$

Measures the match between the data and prediction of the model

Measures how well the model can describe the data

Prior beliefs about which values of the model are most reasonable

Maximum Likelihood Estimation MLE

Bayes' rule produced a general relation:

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})} = \max_{\Theta} \frac{Likelihood \times Prior}{Evidence}$$

Special cases:

- The **model is fixed**, can drop conditioning on $\mathbf{M}$: $p(\mathbf{D} \mid \mathbf{M}) \rightarrow p(\mathbf{D})$
- The model is fixed, and a flat prior $p(\Theta \mid \mathbf{M}) = const$ is used:
  Maximum Likelihood Estimation (MLE)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} p(\mathbf{D} \mid \mathbf{M}, \Theta)$$

Why MLE

Maximum Likelihood Estimation (MLE):

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} p(\mathbf{D} \mid \mathbf{M}, \Theta)$$

Properties:

- **Consistent** estimator (i.e. $\hat{\Theta}_{N \to \infty} \to \Theta$) **if**:
  - The true distribution $p_{\text{data}}(\mathbf{x})$ lies within the family $\mathcal{F}_{\Theta}$
  - There is a unique value of Θ that corresponds to $p_{\text{data}}(\mathbf{x})$
- Asymptotically (as $N \to \infty$), has the **lowest possible MSE** among all consistent estimators (Cramer-Rao lower bound, 1945-1946)
- For **finite** N, biased estimators (e.g. a regularized MLE) can be more efficient (i.e. reach the same level of generalization error with a smaller number of samples N)

Maximum A-Posteriori Estimation

Bayes' rule produced a general relation:

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \frac{p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})}{p(\mathbf{D} \mid \mathbf{M})} = \max_{\Theta} \frac{Likelihood \times Prior}{Evidence}$$

Special cases:

- The **model is fixed**, can drop conditioning on $\mathbf{M}$: $p(\mathbf{D} \mid \mathbf{M}) \rightarrow p(\mathbf{D})$
- The model is fixed, and a flat prior $p(\Theta \mid \mathbf{M}) = const$ is used: Maximum Likelihood Estimation (MLE)

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} p(\mathbf{D} \mid \mathbf{M}, \Theta)$$

- The model is fixed, and a non-flat prior $p(\Theta \mid \mathbf{M})$ is used: **Maximum A-Posteriori (MAP) estimation**

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})$$

MLE and Least Squares Loss

MLE and Least Squares Loss

Assume we want to fit *N* noisy measurements of quantity y_n as a function of variable $\mathbf{x_n}$: $y_n = f(\mathbf{x_n}, \Theta) + \varepsilon_n$

Assuming that errors ε_n are i.i.d. and have a Gaussian distribution with variance σ_n^2, the likelihood is

$$p(\mathbf{D} \mid \mathbf{M}, \Theta) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(y_n - f(\mathbf{x_n}, \Theta))^2}{2\sigma_n^2}}$$

MLE and Least Squares Loss

Assume we want to fit *N* noisy measurements of quantity y_n as a function of variable $\mathbf{x_n}$: $y_n = f(\mathbf{x_n}, \Theta) + \varepsilon_n$

Assuming that errors ε_n are i.i.d. and have a Gaussian distribution with variance σ_n^2, the likelihood is

$$p(\mathbf{D} \mid \mathbf{M}, \Theta) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{\left(y_n - f(\mathbf{x_n}, \Theta)\right)^2}{2\sigma_n^2}}$$

Maximization of $p(\mathbf{D} \mid \mathbf{M}, \Theta)$ is equivalent to minimization of **negative log-likelihood** (**NLL**):

$$\min_{\Theta} \left(-\log\, p(\mathbf{D} \mid \mathbf{M}, \Theta)\right) = \min_{\Theta} \sum_{n-1}^{N} \frac{\left(y_n - f(\mathbf{x_n}, \Theta)\right)^2}{2\sigma_n^2} + \frac{1}{2}\log(2\pi\sigma_n^2)$$

MLE and Least Squares Loss

Assume we want to fit N noisy measurements of quantity y_n as a function of variable $\mathbf{x_n}$: $y_n = f(\mathbf{x_n}, \Theta) + \varepsilon_n$

Assuming that errors ε_n are i.i.d. and have a Gaussian distribution with variance σ_n^2, the likelihood is

$$p(\mathbf{D} \mid \mathbf{M}, \Theta) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(y_n - f(\mathbf{x_n}, \Theta))^2}{2\sigma_n^2}}$$

Maximization of $p(\mathbf{D} \mid \mathbf{M}, \Theta)$ is equivalent to minimization of **negative log-likelihood** (**NLL**):

$$\min_{\Theta} \left(-\log p(\mathbf{D} \mid \mathbf{M}, \Theta)\right) = \min_{\Theta} \sum_{n-1}^{N} \frac{\left(y_n - f(\mathbf{x_n}, \Theta)\right)^2}{2\sigma_n^2} + \frac{1}{2}\log(2\pi\sigma_n^2)$$

When variances σ_n^2 are constant, this is equivalent to a **minimum least squares error (MSE)** function

$$\min_{\Theta} \sum_{n-1}^{N} \left(y_n - f(\mathbf{x_n}, \Theta)\right)^2$$

Becomes linear regression if

$$f(\mathbf{x_n}, \Theta) = \sum \Theta_k x_{nk} = \Theta^T \mathbf{x}$$

Kullback-Leible (KL) Divergence

Another interpretation of MLE : minimization of the KL divergence between the model distribution $p_{\text{model}}(\mathbf{x},\Theta)$ and the true distribution $p_{\text{data}}(\mathbf{x})$:

$$D_{KL}\left(\mathrm{p}_{\text{data}} \,\|\, p_{\text{model}}\right) = \mathbb{E}_{x\sim p_{data}}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x})}\right]$$

$$= \mathbb{E}_{x\sim p_{data}}\left[\log p_{\text{data}}(\mathbf{x}) - \log p_{\text{model}}(\mathbf{x})\right]$$

$$= \underbrace{\mathbb{E}_{x\sim p_{\text{data}}}\left[\log p_{\text{data}}(\mathbf{x})\right]}_{\textit{independent of } p_{\text{model}}} - \mathbb{E}_{x\sim p_{\text{data}}}\left[\log p_{\text{model}}(\mathbf{x})\right]$$

$$\simeq -\mathbb{E}_{x\sim \hat{p}_{\text{data}}}\left[\log p_{\text{model}}(\mathbf{x})\right] + \ldots = -\sum_{n} \log p_{\text{model}}(\mathbf{x}_n,\Theta)$$

KL divergence between two distributions $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ measures their dissimilarity:

$$D_{KL}\left(p_1 \,\|\, p_2\right) \geq 0$$

$$D_{KL}\left(p_1 \,\|\, p_2\right) = 0 \quad \textit{iff } p_1(\mathbf{x}) = p_2(\mathbf{x})$$

KL divergence is widely used in ML!

MAP and Regularization

MAP and Regularization

Assume we want to fit *N* noisy measurements of quantity y_n as a function of variable $\mathbf{X}_n$:

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} \; p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})$$

Negative log-likelihood for our problem $y_n = f(\mathbf{X}_n, \Theta) + \varepsilon_n$:

$$\min_{\Theta} \left(-\log p(\mathbf{D} \mid \mathbf{M}, \Theta) \right) = \min_{\Theta} \sum_{n-1}^{N} \frac{\left(y_n - f(\mathbf{x_n}, \Theta) \right)^2}{2\sigma_n^2} - \log p(\Theta \mid \mathbf{M}) + \ldots$$

Training error function

Regularizer

MAP and Regularization

Recall the definition of the Maximum A-Posteriori (MAP) Estimation of model parameters:

$$\max_{\Theta} p(\Theta \mid \mathbf{D}, \mathbf{M}) = \max_{\Theta} p(\mathbf{D} \mid \mathbf{M}, \Theta)\, p(\Theta \mid \mathbf{M})$$

Negative log-likelihood for our problem $y_n = f(\mathbf{x_n}, \Theta) + \varepsilon_n$:

$$\min_{\Theta} \left(-\log p(\mathbf{D} \mid \mathbf{M}, \Theta)\right) = \min_{\Theta} \sum_{n-1}^{N} \frac{\left(y_n - f(\mathbf{x_n}, \Theta)\right)^2}{2\sigma_n^2} - \log p(\Theta \mid \mathbf{M}) + \ldots$$

Training error function

Regularizer

Examples:

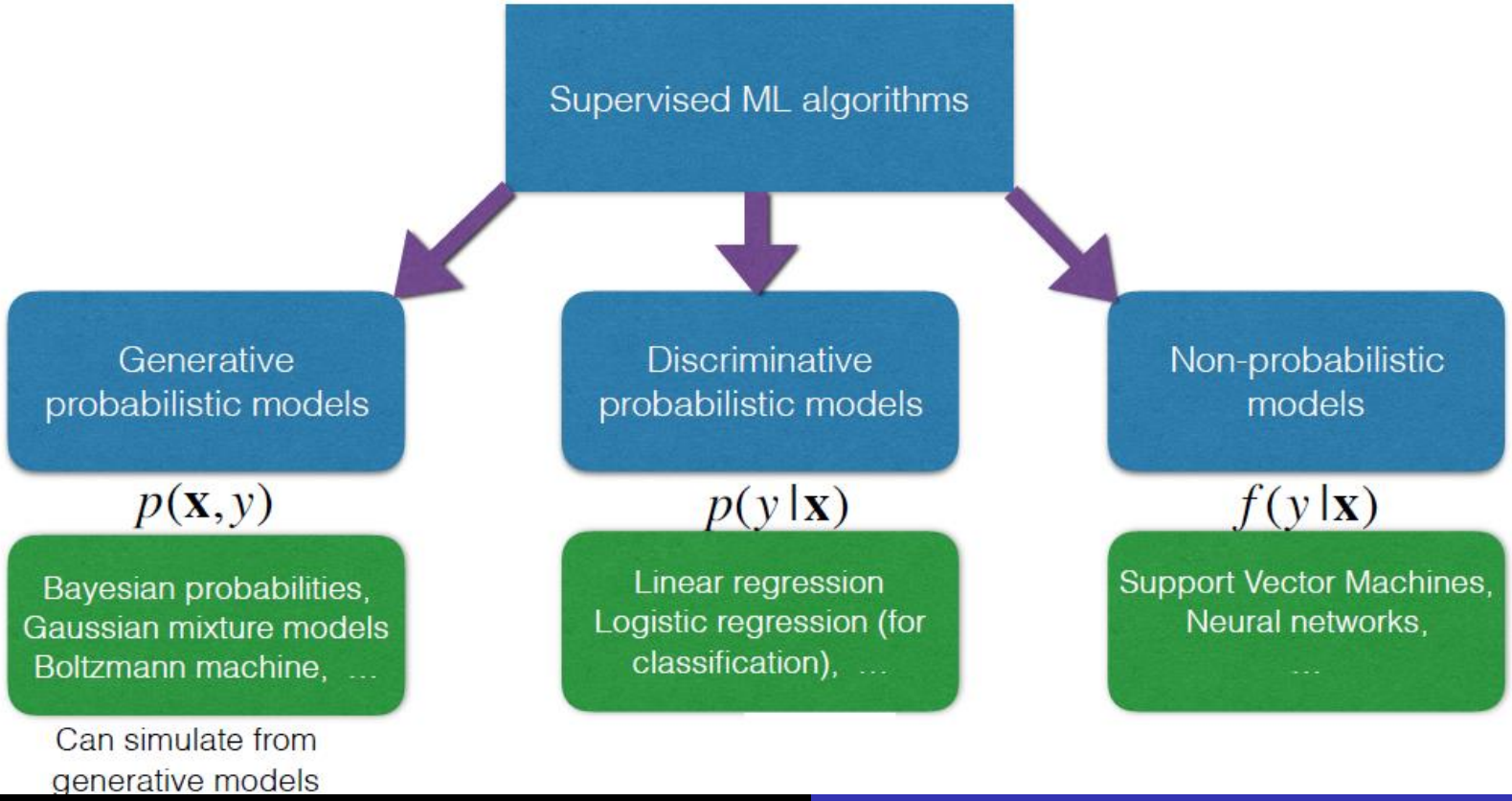
- Gaussian prior $\sim e^{-\lambda(\theta-\theta_0)^2} \Rightarrow L_2$ regularization $\lambda \left\| \Theta - \Theta_0 \right\|_2$
- Laplace prior $\sim e^{-\alpha|\theta-\theta_0|} \Rightarrow L_1$ regularization $\lambda \left\| \Theta - \Theta_0 \right\|_1$

Supervised Learning Algorithms

Most, but not all, supervised ML algorithms amount to **estimating a probability distribution** $p(y|\mathbf{x})$

Example: Linear Regression is equivalent to a **discriminative probabilistic model**

$$p(y|\mathbf{x}) = \mathcal{N}(y; \Theta^T \mathbf{x}; \sigma^2)$$



Probabilistic Classification Models

Probabilistic Classification Models

For probabilistic **regression** models, we estimate a probability distribution $p(y|\mathbf{x}), \quad y \in \mathbb{R}$

Probabilistic **classification** models:

$$p(y|\mathbf{x}), \quad y \in (C_1, \ldots, C_K)$$

Class labels

$\sum_k p(C_k|\mathbf{x}) = 1$

K classes

Probabilistic Classification Models

For probabilistic **regression** models, we estimate a probability distribution $p(y|\mathbf{x}), \quad y \in \mathbb{R}$

Probabilistic **classification** models:

$$p(y|\mathbf{x}), \quad y \in (C_1,...,C_K)$$

Class labels

$\sum_k p(C_k|\mathbf{x}) = 1$

K classes

Binary classification (K=2):
Here $y = \{0,1\}$, or $y = \{-1,1\}$, or $y = \{low\,risk, high\,risk\}$, e.t.c.
Only one unknown: the "positive class" probability

$$p(y = 1|\mathbf{x})$$

with $p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$

Probabilistic Classification Models

For probabilistic **regression** models, we estimate a probability distribution $p(y|\mathbf{x}), \quad y \in \mathbb{R}$

Probabilistic **classification** models:

$$p(y|\mathbf{x}), \quad y \in (C_1,...,C_K)$$

Class labels

$\sum_k p(C_k|\mathbf{x}) = 1$

K classes

Binary classification (K=2):
Here $y = \{0,1\}$, or $y = \{-1,1\}$, or $y = \{low\,risk, high\,risk\}$, e.t.c.
Only one unknown: the "positive class" probability

$$p(y = 1|\mathbf{x})$$

with $p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$

Examples of binary classification: bank failures, mortgage defaults, credit card fraud, anti-money laundering, price direction prediction

Logistic Regression

Logistic Regression

Recall that a Linear Regression model fits a linear function

$$y = \Theta^T \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^N, \boxed{y \in \mathbb{R}}$$

This is *not* on its own suitable to describe probabilities (as they should be numbers between 0 and 1).

Logistic Regression

Recall that a Linear Regression model fits a linear function

$$y = \Theta^T \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^N, \boxed{y \in \mathbb{R}}$$

This is *not* on its own suitable to describe probabilities (as they should be numbers between 0 and 1). **But!**

Let us apply a "squashing" function to y:

$$y \to h(y), \quad h: \mathbb{R} \to [0,1]$$

Nonlinear function

Logistic Regression

Recall that a Linear Regression model fits a linear function

$$y = \Theta^T \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^N, \boxed{y \in \mathbb{R}}$$

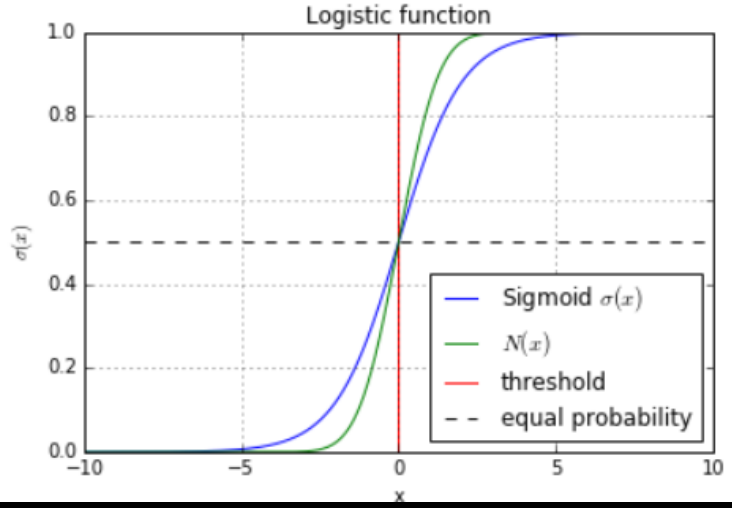
This is *not* on its own suitable to describe probabilities (as they should be numbers between 0 and 1). But!

Let us apply a "squashing" function to y:

$$y \to h(y), \quad h: \mathbb{R} \to [0,1]$$

Nonlinear function

Logistic Regression: take $h(y) = \sigma(y) = \dfrac{1}{1+\exp(-y)} \Rightarrow$



$$p(y=1 \mid \mathbf{x}) = \sigma(\theta^{\mathbf{T}}\mathbf{x}) = \frac{1}{1+\exp(-\theta^{\mathbf{T}}\mathbf{x})}$$

Can also be derived from other models

MLE for Logistic Regression

Logistic Regression:

$$p_n(\theta) = p(y_n = 1 \mid \mathbf{x}_n) = \sigma(\theta^{\mathrm{T}}\mathbf{x}_n) = \frac{1}{1+\exp(-\theta^{\mathrm{T}}\mathbf{x}_n)}$$

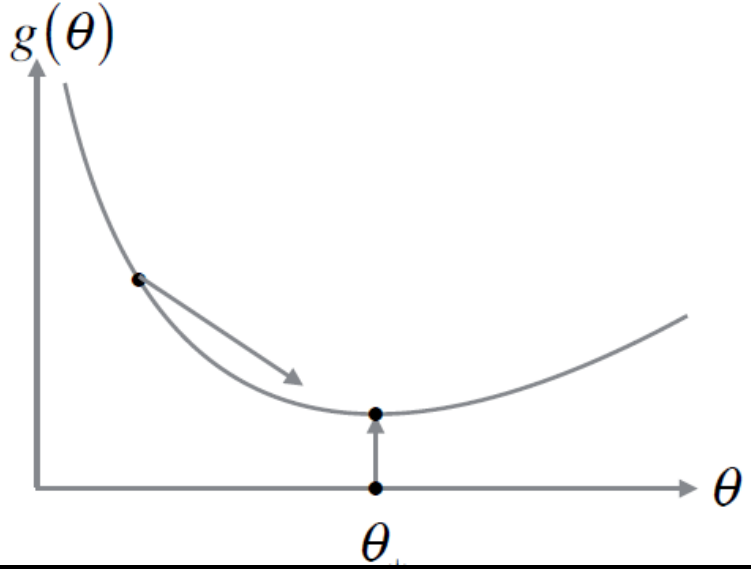
Likelihood:

$$p(\mathbf{D} \mid \mathbf{M}, \Theta) = \prod_{n=1}^{N} p_n(\theta)^{y_n} \left(1 - p_n(\theta)\right)^{1-y_n}$$

observed values {0,1}

Negative LL:

$$-\log p(\mathbf{D} \mid \mathbf{M}, \Theta) = \sum_{n-1}^{N} \left[y_n \log p_n(\theta) + (1-y_n) \log\left(1 - p_n(\theta)\right) \right]$$

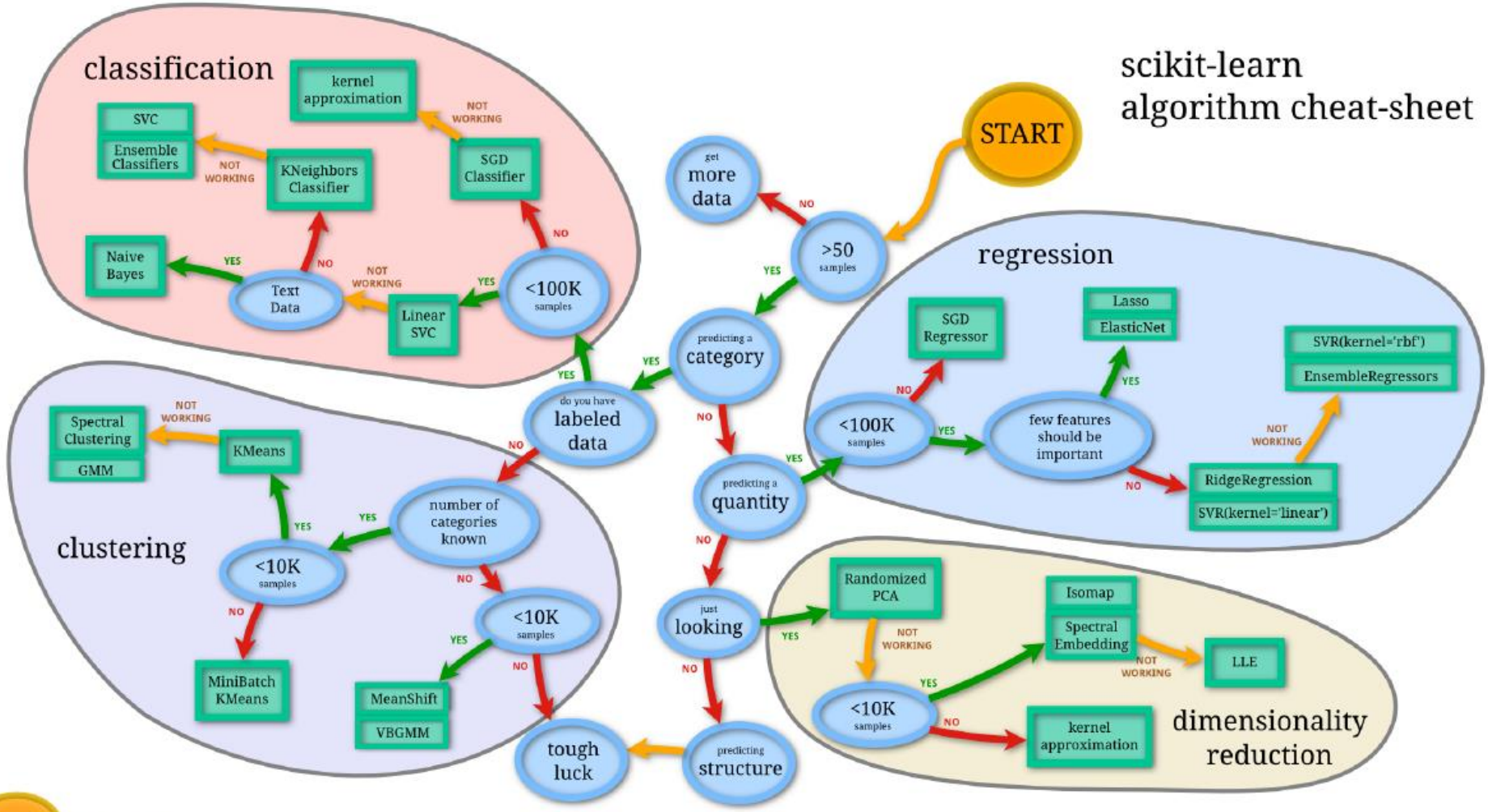


Can be minimized by gradient descent

Convex function of θ

Unique solution

Regression and Classification Demo



- We will use the scikit-learn package to illustrate all we have seen during this lecture