

L9. Machine Learning (Classification)

- **Non-Bayesian approaches**
 - Discriminative model
 - ✓ Logistic regression
 - ✓ Neural Network
 - Generative model
 - ✓ Gaussian Discriminative Analysis
 - ✓ Naïve Bayes classification
- **Full Bayesian approach for classification**
 - Bayesian Logistic regression
 - Bayesian Neural Network

Non-Bayesian vs Bayesian

- **Non-Bayesian approaches**

- ✓ *discriminative* probabilistic classification

$$p(y|x) = f(w^T x)$$

Directly model posterior $p(y|x)$ using parameteric form

- ✓ *Generative* probabilistic classification

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in Y} P(x|y)P(y)}$$

Model $P(x|y)$ and $P(y)$ and combined them in Bayes' rule

$$\hat{y} = \operatorname{argmax}_{y \in Y} p(y|x)$$

- **Full Bayesian approach for classification**

1. Construct prior $p(w)$
2. Construct likelihood $p(D|w)$, where $D = \{(x_i, y_i)\}_{i=1}^m$
3. Construct posterior $p(w|D) = \frac{p(D|w)p(w)}{p(D)}$
4. Posterior predictive distribution $p(y_*|x_*, D) = \int_w p(y_*|x_*, w)p(w|D)dw$

University admission committee

High school grades

실업·가점			자유	비고			
기술년	기술	선택	선택	총점	평균	학급	학년
가점(년)	가점	(9/10)	(·)			석차	석차
수				55	5.00	1	1
				55	5.00	54	428
수				60	5.00	1	1
				60	5.00	54	432
		수		60	5.00	1	1
				60	5.00	51	417
가점	3	학습 의욕이 강하며 착한 학생					

National Exam score

〈2016학년도 대학수학능력시험 성적표(예시)〉						
수험번호	성명	생년월일	성별	출신교고 (반 또는 졸업년도)		
12345678	홍길동	97.09.05.	남	한국고등학교 (9)		
구분	국어 영역	수학 영역	영어 영역	사회탐구 영역		제2외국어/한문 영역
	B형	A형		생활과 윤리	사회·문화	일본어 I
표준점수	131	137	141	53	64	69
백분위	93	95	97	75	93	95
등급	2	2	1	4	2	2

2015. 12. 2.
한국교육과정평가원장

Rejected

Student 1

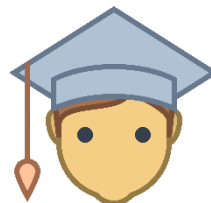
- Exam: 3/10
- Grades: 4/10



?

Student 2

- Exam: 7/10
- Grades: 6/10



Accepted

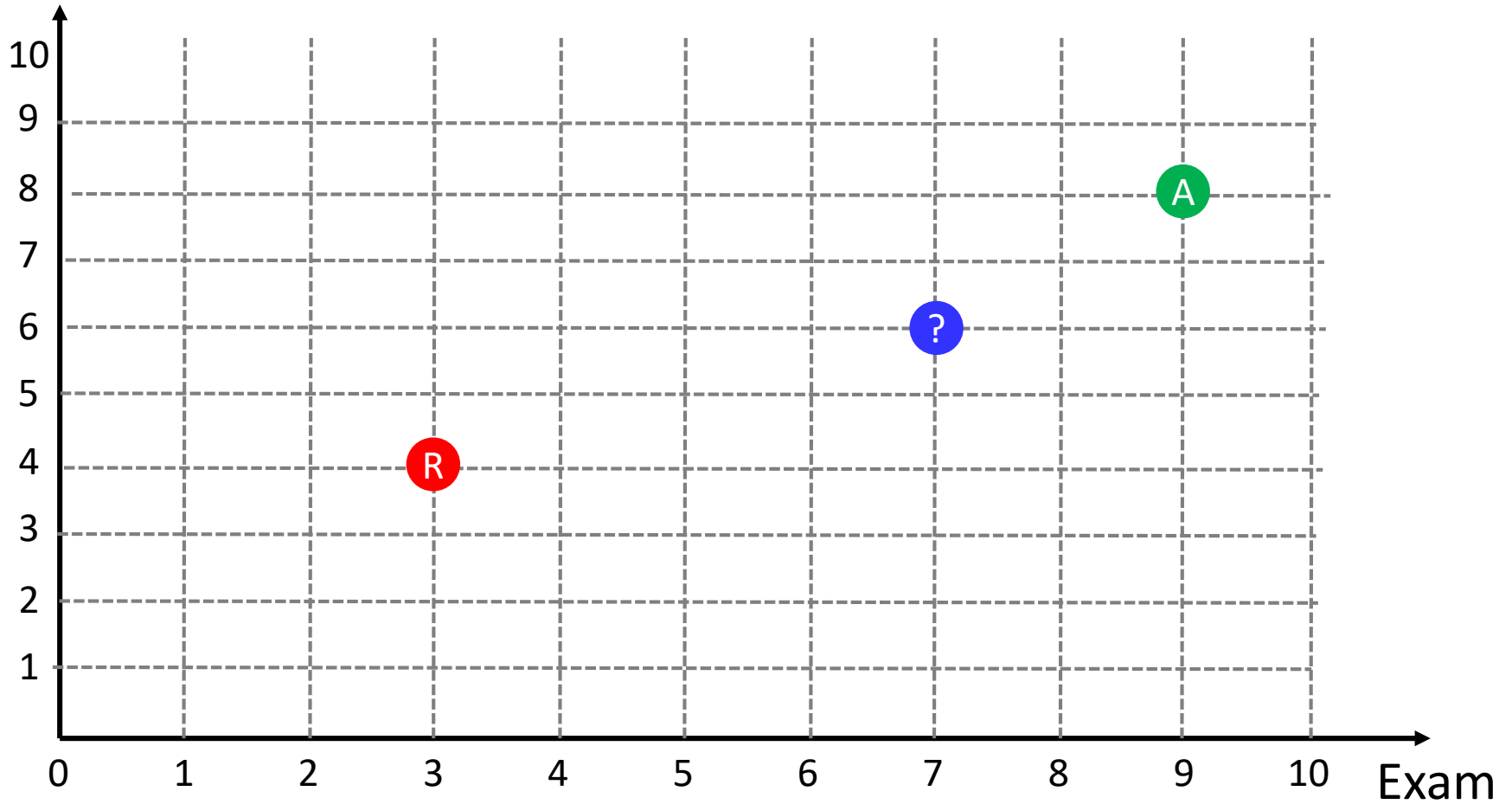
Student 3

- Exam: 9/10
- Grades: 8/10



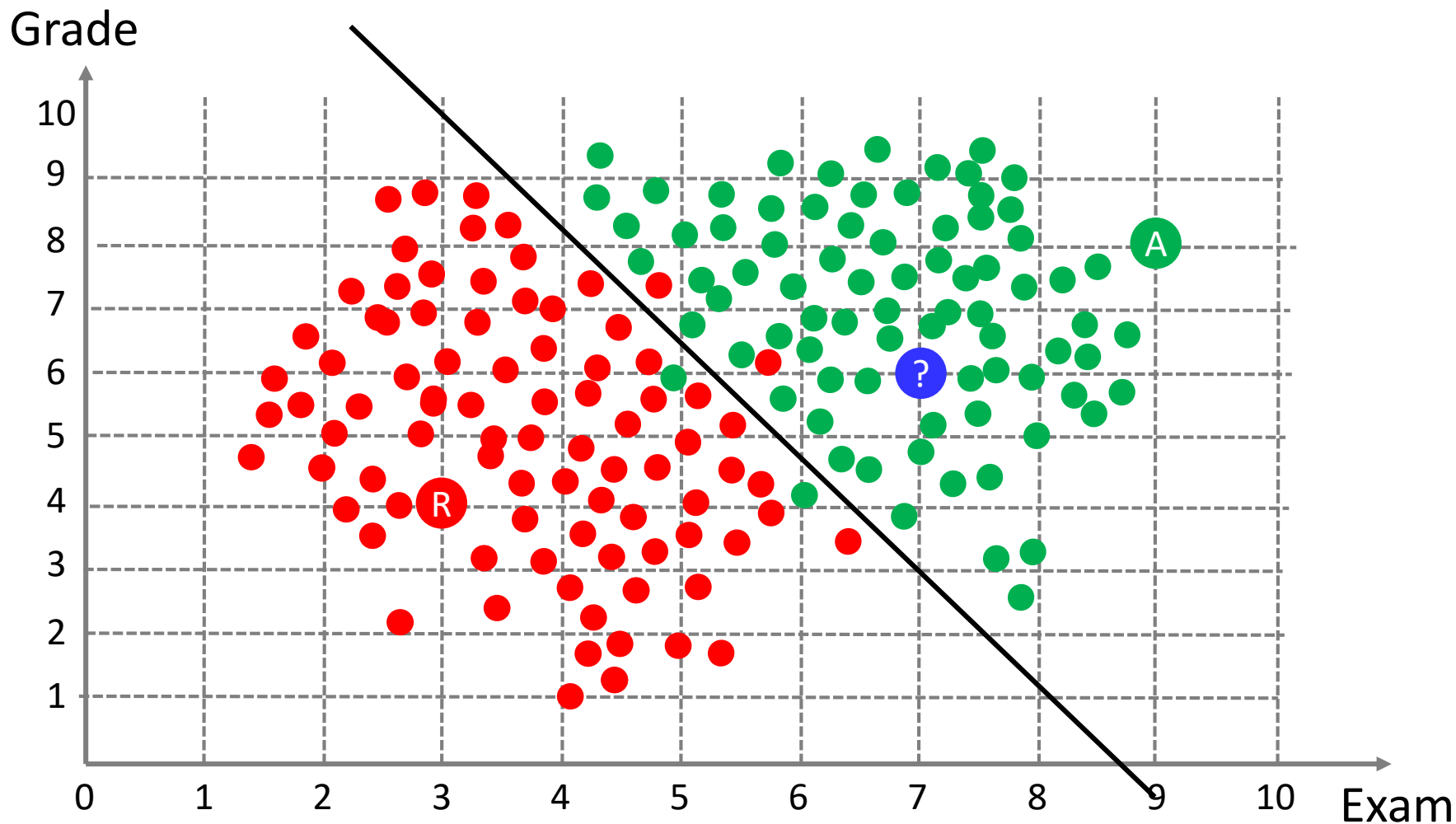
University admission committee

Grade



University admission committee

Look at the **historical data** on the admission results



Logistic regression

- Logistic regression is *discriminative* probabilistic linear classification : $p(y|x) = g(w^T x)$

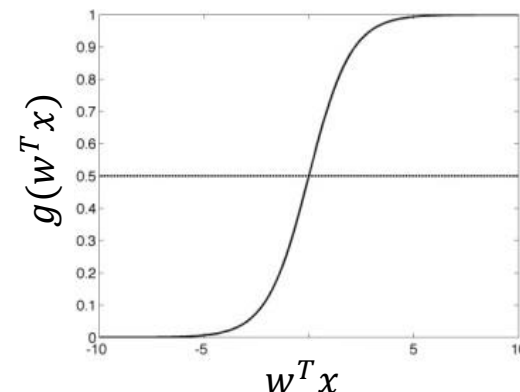
Let's denote p a probability of having $y = 1$

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = w^T x$$

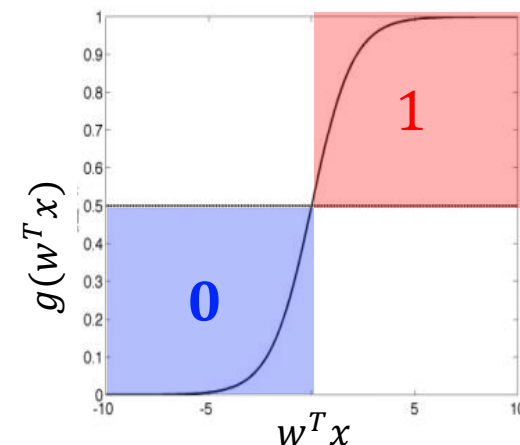
$$\frac{p}{1-p} = \exp(w^T x)$$

$$p = \frac{\exp(w^T x)}{1 + \exp(w^T x)} = \frac{1}{1 + \exp(-w^T x)} = g(w^T x)$$

- Larger $w^T x \rightarrow$ larger $\rightarrow g(w^T x) \rightarrow$ higher p for $y = 1$
- Smaller $w^T x \rightarrow$ smaller $\rightarrow g(w^T x) \rightarrow$ lower p for $y = 1$



$$g(z) = \frac{1}{(1 + \exp(-w^T x))}$$

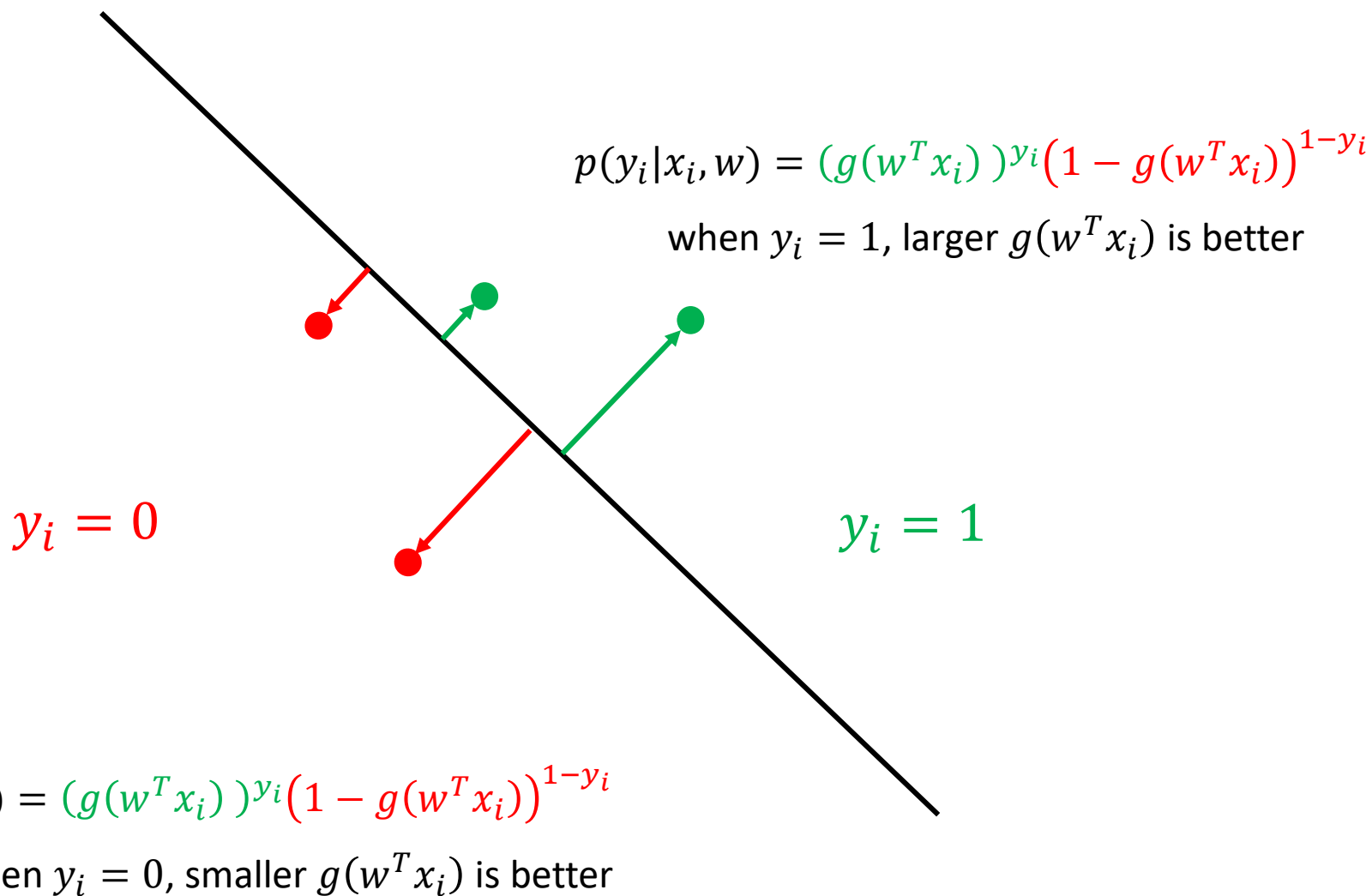


- Classification rule:

$$y = \begin{cases} 0, & \text{if } p(Y = 1|x) = g(w^T x) < 0.5 \Leftrightarrow w^T x < 0 \\ 1, & \text{if } p(Y = 1|x) = g(w^T x) \geq 0.5 \Leftrightarrow w^T x \geq 0 \end{cases}$$

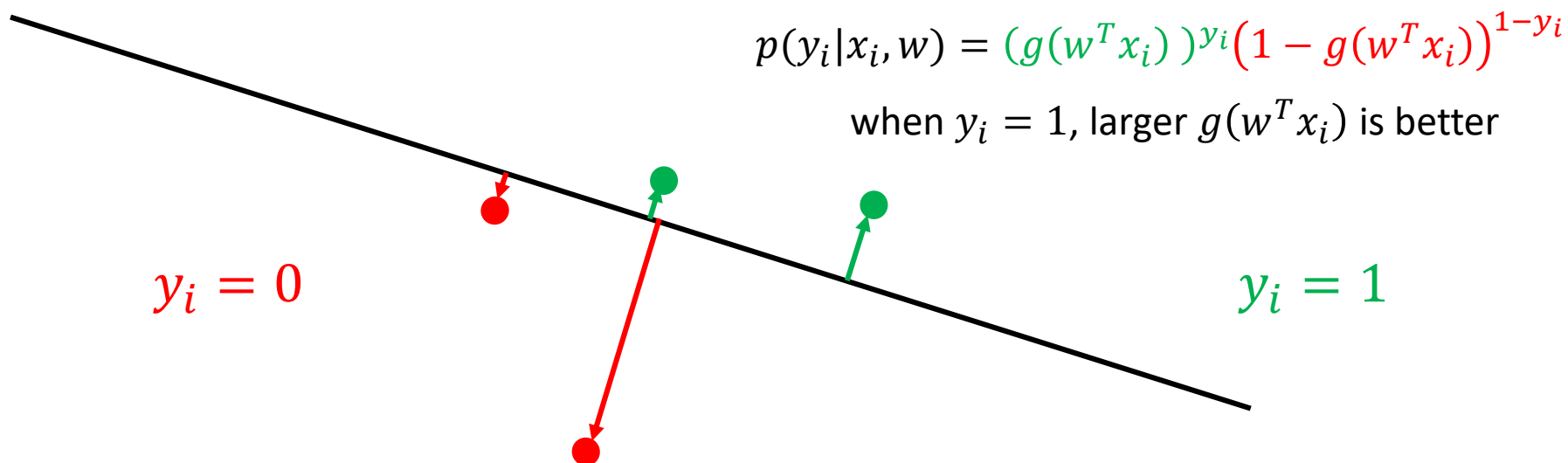
University admission committee

How to draw **a separating line** ?



University admission committee

How to draw **a separating line** ?



$$p(y_i|x_i, w) = (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

When $y_i = 0$, smaller $g(w^T x_i)$ is better

Logistic regression – objective function

- Likelihood for **a single point** (x_i, y_i) can be specified as

$$p(y_i|x_i, w) = (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

- Likelihood for **whole training data** (X, y) can be specified as

$$p(y|X, w) = \prod_i^m p(y_i|x_i, w) = \prod_{i=1}^m (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

Note that this is similar to the likelihood of Binomial dist.

- **Log**-likelihood

$$L(w) = \log \prod_i^m p(y_i|x_i, w) = \sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i))$$

Logistic regression – learning (optimization)

- **Log**-likelihood

$$L(w) = \log \prod_i^m p(y_i | x_i, w) = \sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i))$$

- We can find the parameters that maximizes the log-likelihood function

$$w^* = \operatorname{argmax}_w L(w)$$

- **Gradient ascent** algorithm

Repeat until convergence{

$$w_j := w_j + \alpha \frac{\partial}{\partial w_j} L(w) \text{ (for every } j)$$

α : learning rate

}

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^m (y_i - g(w^T x_i)) x_{ij}$$

Logistic regression – learning (optimization)

- **Log**-likelihood

$$L(w) = \log \prod_i^m p(y_i | x_i, w) = \sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i))$$

- We can find the parameters that maximizes the log-likelihood function

$$w^* = \operatorname{argmax}_w L(w)$$

- **Stochastic gradient ascent** algorithm

Repeat until convergence{

for $i = 1, \dots, m$ {

$w_j := w_j + \alpha (y_i - g(w^T x_i)) x_{ij}$ (for every j)

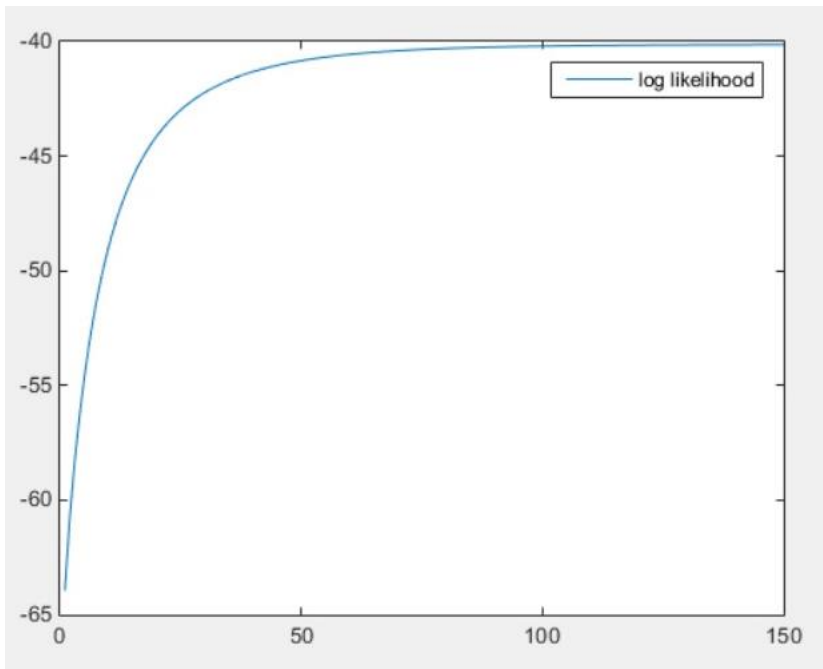
}

α : learning rate

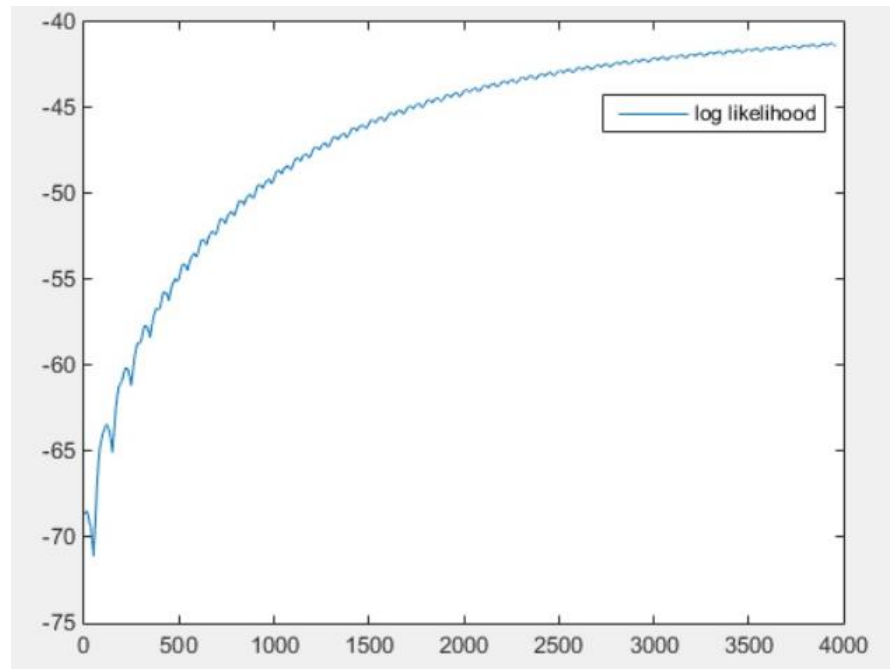
}

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^m (y_i - g(w^T x_i)) x_{ij} \sim (y_i - g(w^T x_i)) x_{ij}$$

Logistic regression – learning (optimization)

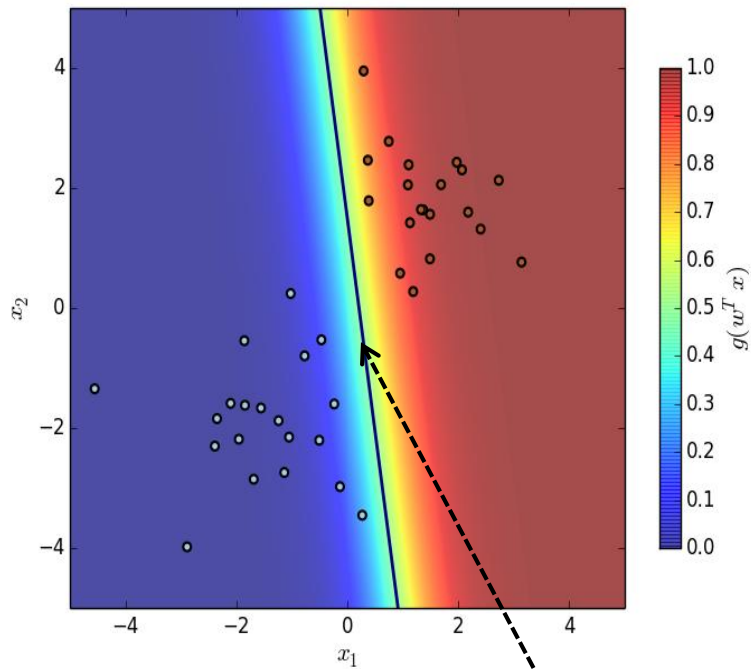


Gradient ascent의 log-likelihood 수렴

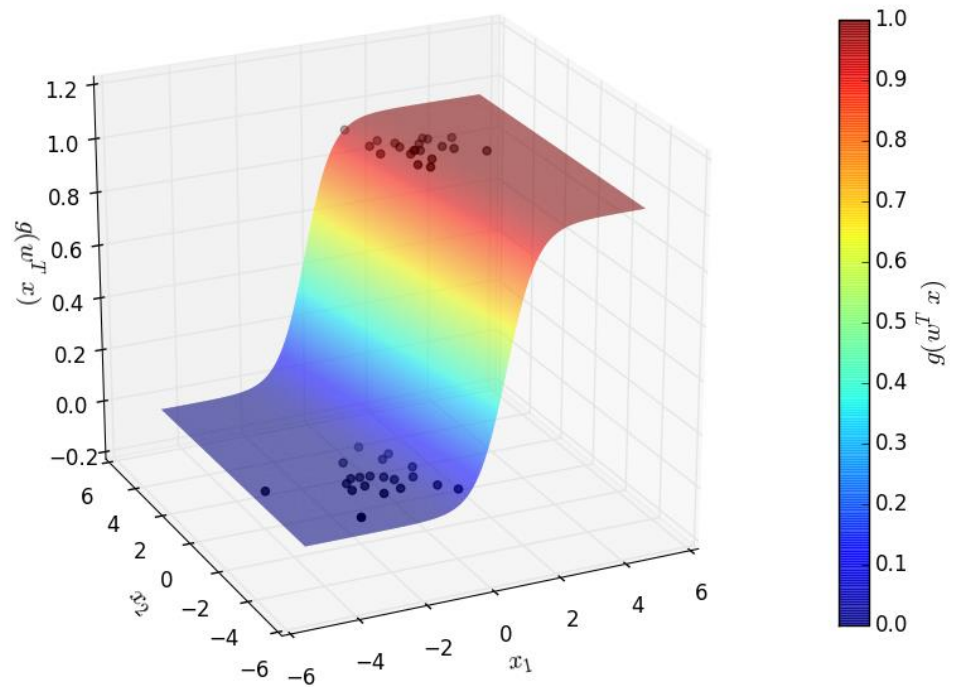


Stochastic gradient ascent의 log-likelihood 수렴

Logistic regression



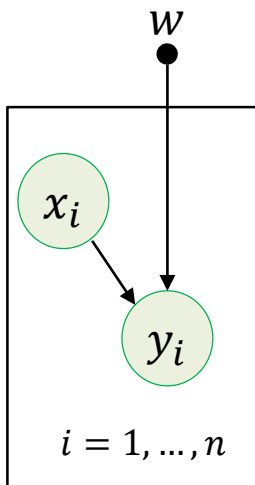
Classification line $w^T x = 0$



Jupyter Demo Simulation

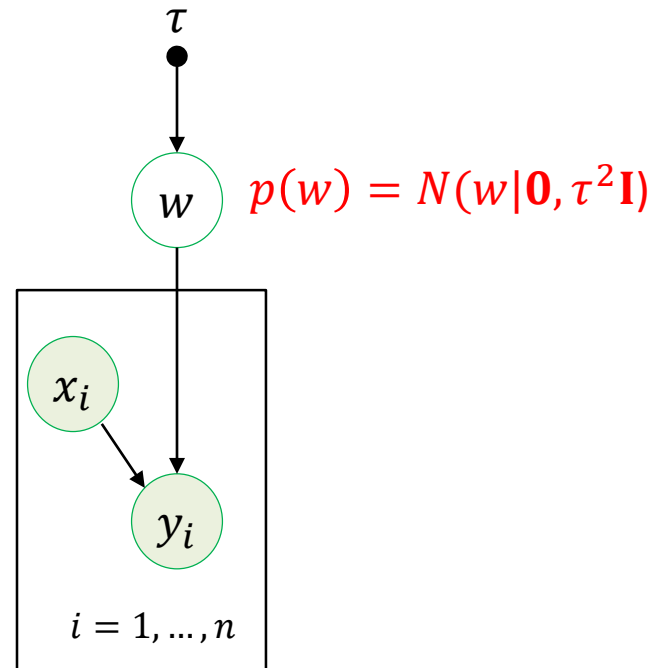
Logistic Regression

Fixed parameter
(to be determined)



Bayesian Logistic Regression

Fixed hyper-parameter



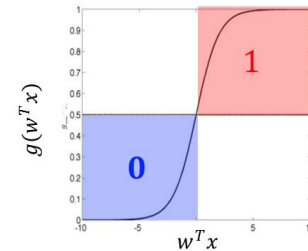
$$y_i = \begin{cases} 0, & \text{if } g(w^T x_i) < 0.5 \Leftrightarrow w^T x_i < 0 \\ 1 & \text{if } g(w^T x_i) \geq 0.5 \Leftrightarrow w^T x_i \geq 0 \end{cases}$$

Bayesian Logistic Regression with Gaussian Prior (Ridge Logistic Regression)

- We have a logistic regression model :

$$p(Y = 1|x) = g(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

$$p(Y = 0|x) = 1 - g(w^T x)$$



- Likelihood** can be specified as

$$p(y_i|x_i, w) = (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

for $y = (y_1, \dots, y_m)$

$$p(y|X, w) = \prod_{i=1}^m p(y_i|x_i, w) = \prod_{i=1}^m (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

- Prior** on parameter w can be specified as

$$p(w_j) = N(w_j|0, \tau_j^2) = \frac{1}{\sqrt{2\pi\tau_j^2}} \exp\left(-\frac{w_j^2}{2\tau_j^2}\right)$$

for $w = (w_1, \dots, w_n)$

$$p(w) = \prod_{j=1}^n N(w_j|0, \tau_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi\tau_j^2}} \exp\left(-\frac{w_j^2}{2\tau_j^2}\right)$$

- ✓ τ_j^2 quantifies our belief that w_j is close to 0.
- ✓ For simple case, $\tau_j^2 = \tau^2$ for $j = 1, \dots, n$

Bayesian Logistic Regression with Gaussian Prior (Ridge Logistic Regression)

- We need to compute **the posterior**: (For simple case, $\tau_j^2 = \tau^2$ for $j = 1, \dots, n$)

$$\begin{aligned} p(w|X, y) &= p(y|X, w)p(w) \\ &= \prod_{i=1}^m (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i} \prod_{j=1}^n \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{w_j^2}{2\tau^2}\right) \end{aligned}$$

$$\log p(w|X, y) = \sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i)) + n \log\left(\frac{1}{\sqrt{2\pi\tau^2}}\right) - \sum_{j=1}^n \frac{w_j^2}{2\tau^2}$$

- The **MAP** estimate of w is then simply

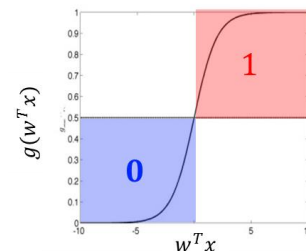
$$\begin{aligned} \hat{w} &= \operatorname{argmax}_w p(w|X, y) \\ &= \operatorname{argmax}_w \log p(w|X, y) \\ &= \operatorname{argmax}_w \underbrace{\sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i))}_{\text{Data fitness}} - \underbrace{\lambda \|w\|_2^2}_{\text{complexity}} \end{aligned}$$

Bayesian Logistic Regression with Laplace Prior (Lasso Logistic Regression)

- We have a logistic regression model :

$$p(Y = 1|x) = g(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

$$p(Y = 0|x) = 1 - g(w^T x)$$



- Likelihood** can be specified as

$$p(y_i|x_i, w) = (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

for $y = (y_1, \dots, y_m)$

$$p(y|X, w) = \prod_i^m p(y_i|x_i, w) = \prod_{i=1}^m (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i}$$

- Prior** on parameter w can be specified using Laplacian as

$$p(w_j) = \frac{\lambda_j}{2} \exp(-\lambda_j |w_j|)$$

for $w = (w_1, \dots, w_n)$

$$p(w) = \prod_{j=1}^n \frac{\lambda_j}{2} \exp(-\lambda_j |w_j|)$$

- ✓ τ_j^2 quantifies our belief that w_j is close to 0.
- ✓ For simple case, $\tau_j^2 = \tau^2$ for $j = 1, \dots, n$

Bayesian Logistic Regression with Laplace Prior (Lasso Logistic Regression)

- We need to compute **the posterior**: (For simple case, $\tau_j^2 = \tau^2$ for $j = 1, \dots, n$)

$$\begin{aligned} p(w|X, y) &= p(y|X, w)p(w) \\ &= \prod_{i=1}^m (g(w^T x_i))^{y_i} (1 - g(w^T x_i))^{1-y_i} \prod_{j=1}^n \frac{\lambda}{2} \exp(-\lambda |w_j|) \end{aligned}$$

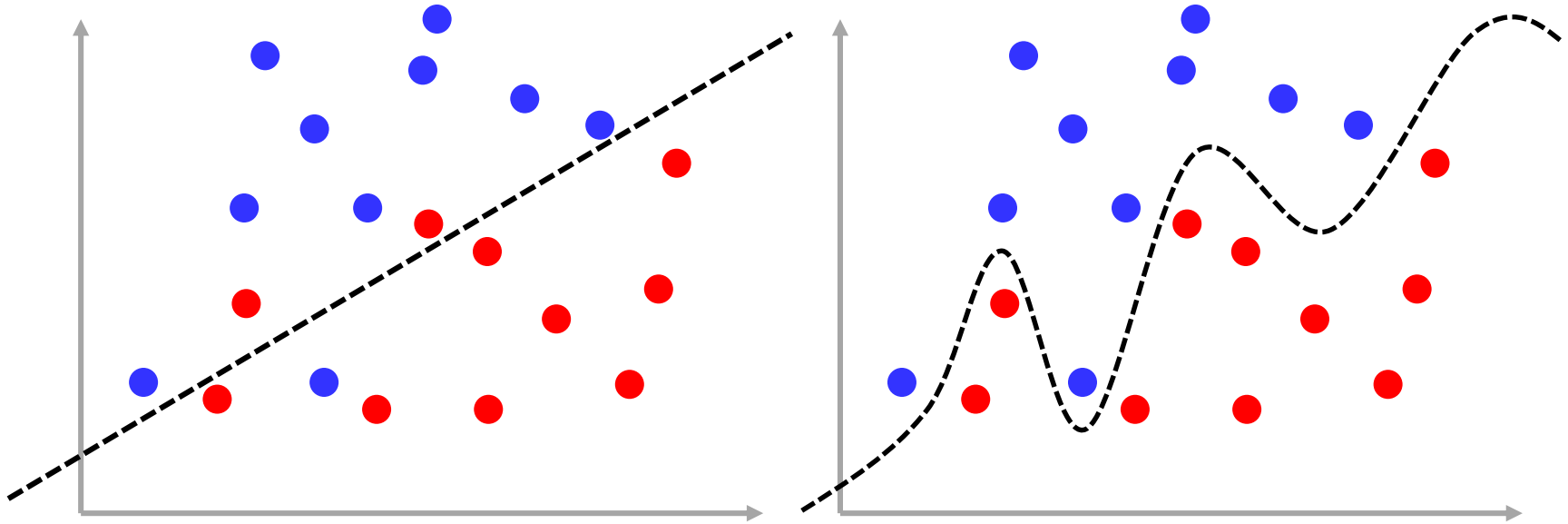
$$\log p(w|X, y) = \sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i)) + n \log\left(\frac{\lambda}{2}\right) - \lambda \sum_{j=1}^n |w_j|$$

- The **MAP** estimate of w is then simply

$$\begin{aligned} \hat{w} &= \operatorname{argmax}_w p(w|X, y) \\ &= \operatorname{argmax}_w \log p(w|X, y) \\ &= \operatorname{argmax}_w \underbrace{\sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i))}_{\text{Data fitness}} - \underbrace{\lambda \sum_{j=1}^n |w_j|}_{\text{Complexity (sparsity)}} \end{aligned}$$

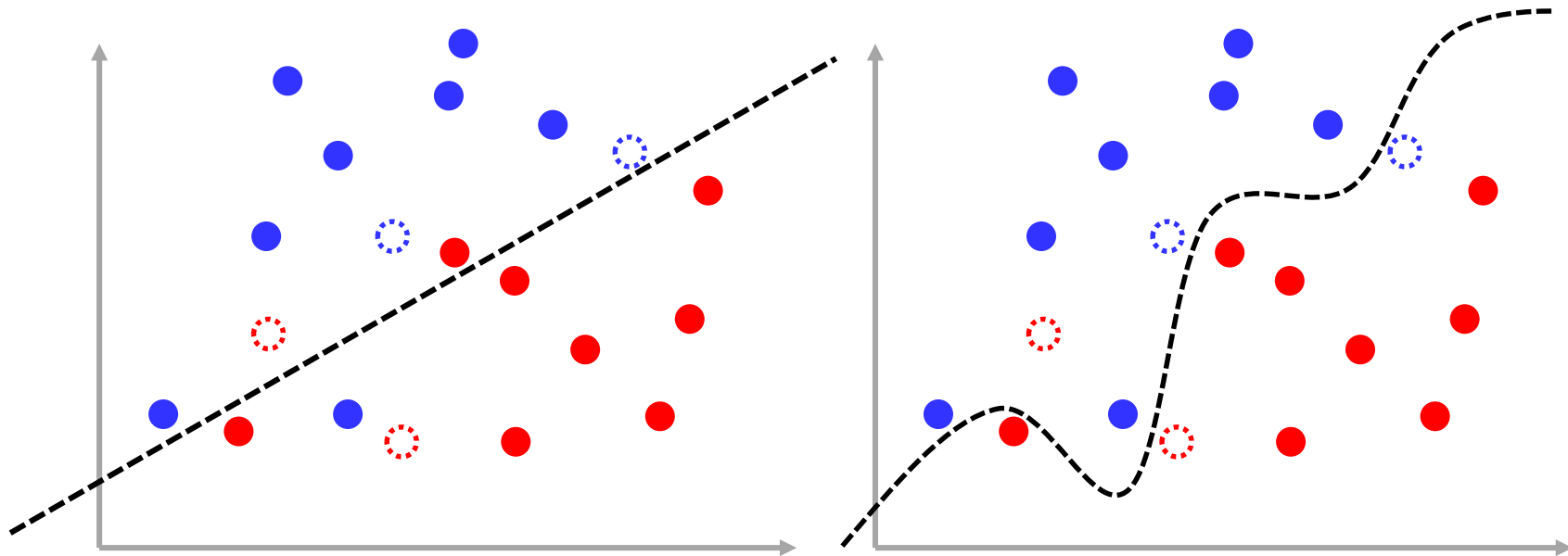
Model Selection and Evaluation

Which model is better



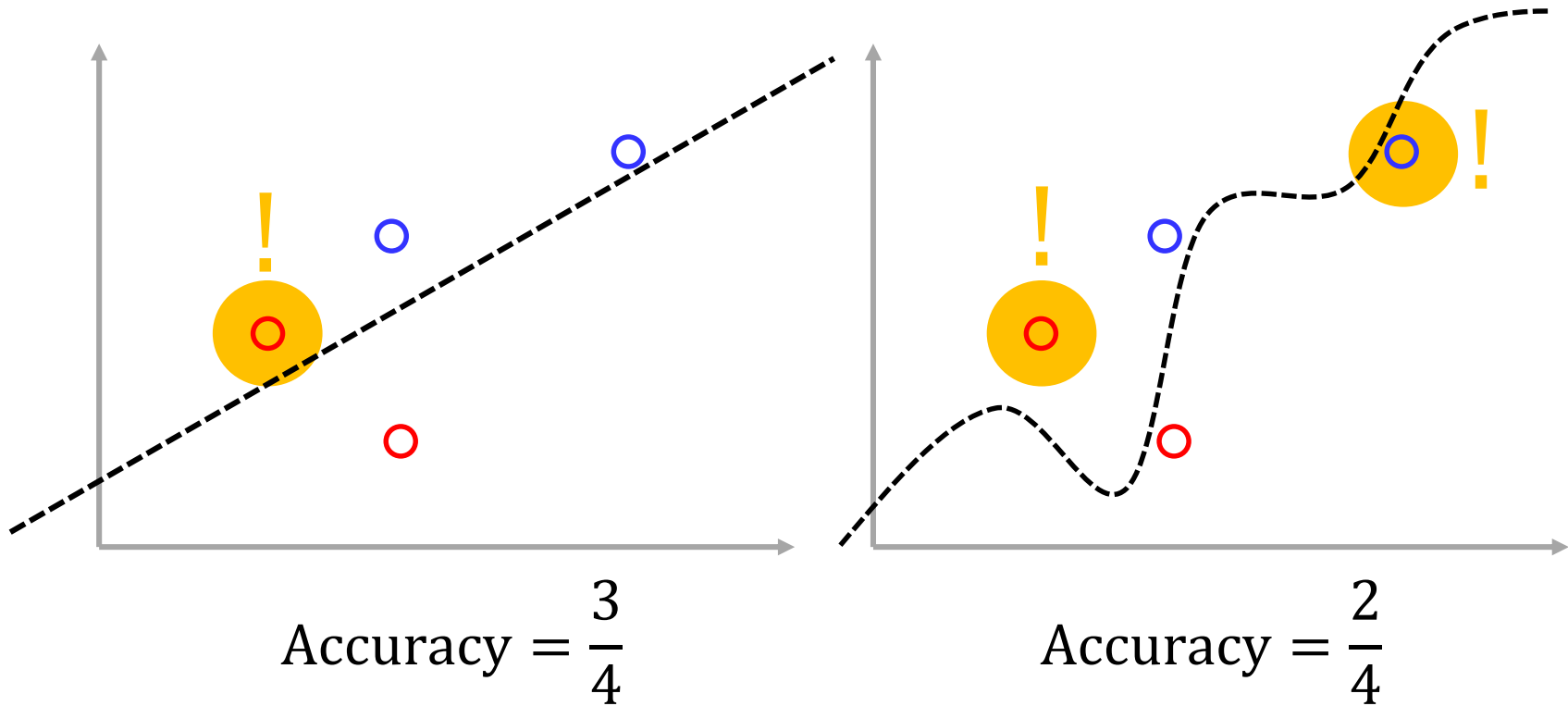
Which model is better

● ● Train data
○ ○ Test data



Which model is better

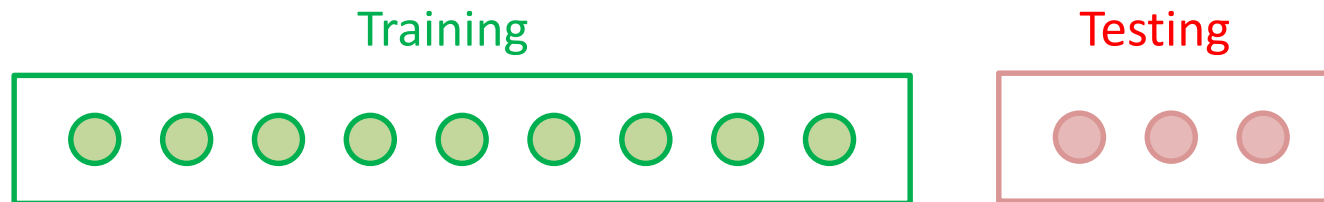
● ● Train data
○ ○ Test data



Golden rule for machine learning:

Never use test data to train your model!

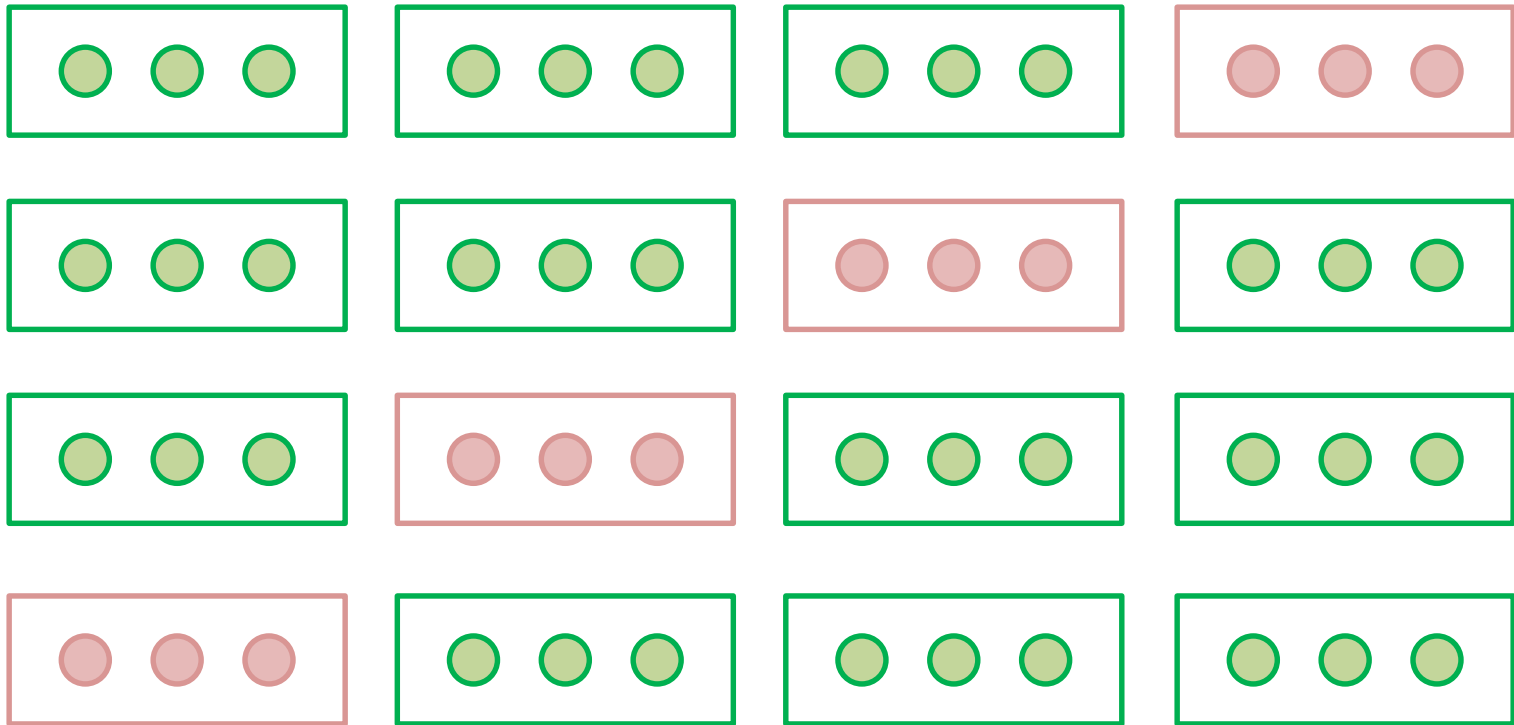
How do we not 'lose' the training data?



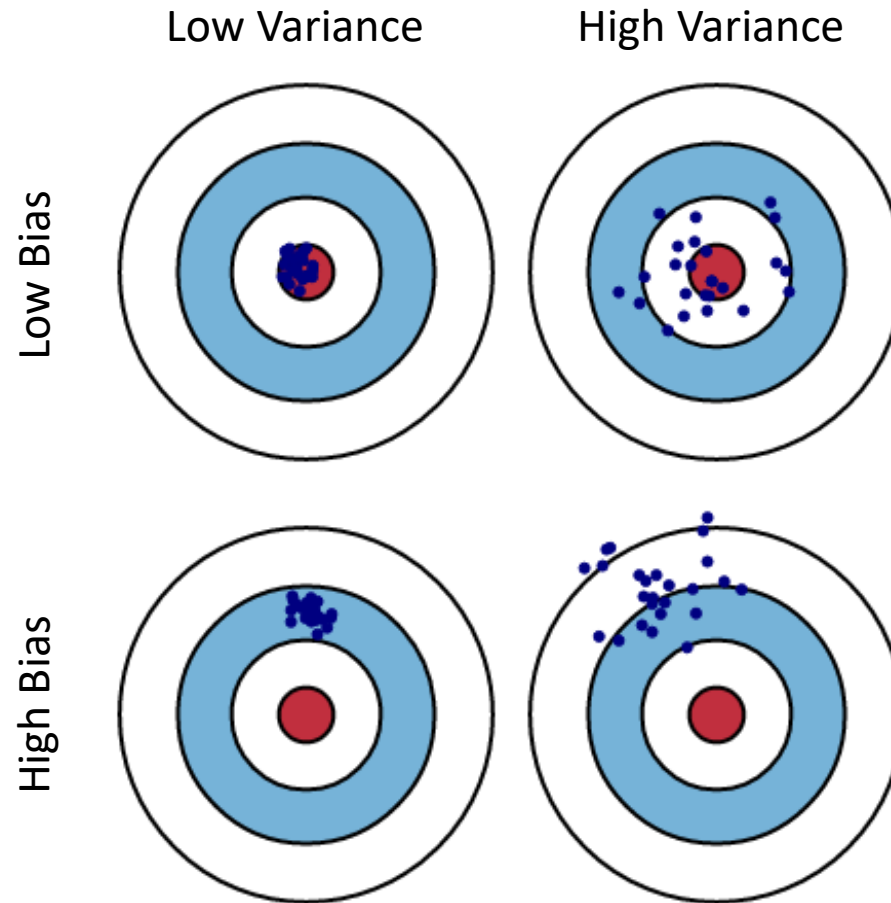
K-Fold Cross Validation

Training

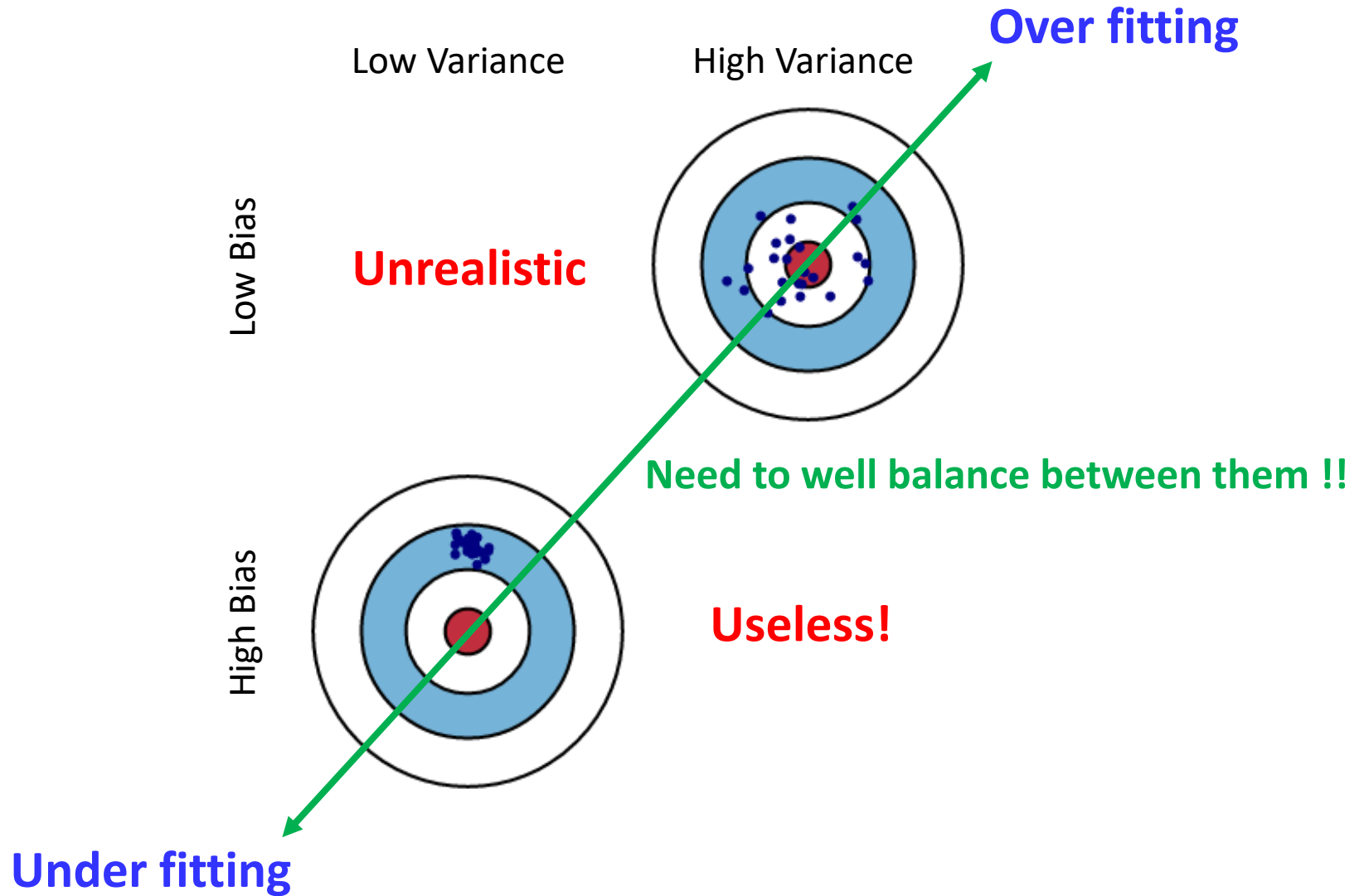
Testing



Under fitting and Over fitting : The Bias and Variance Trade off



Under fitting and Over fitting : The Bias and Variance Trade off



Model selection and training

Training set



Training the model

- Fit the model parameters

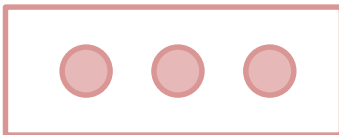
Validation set



Make decision about the model

- Select hyper parameters
 - Degree
 - Features,
 - Structures...

Test set

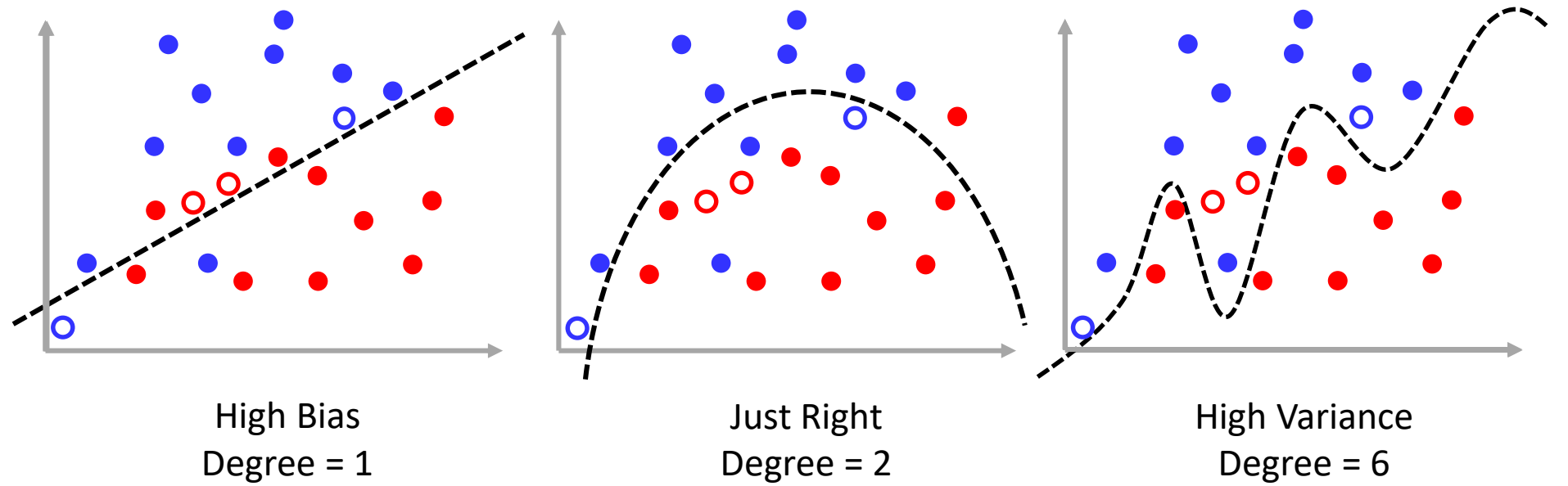


Final testing

- **Never make decision based on test set**
- its just for evaluation!

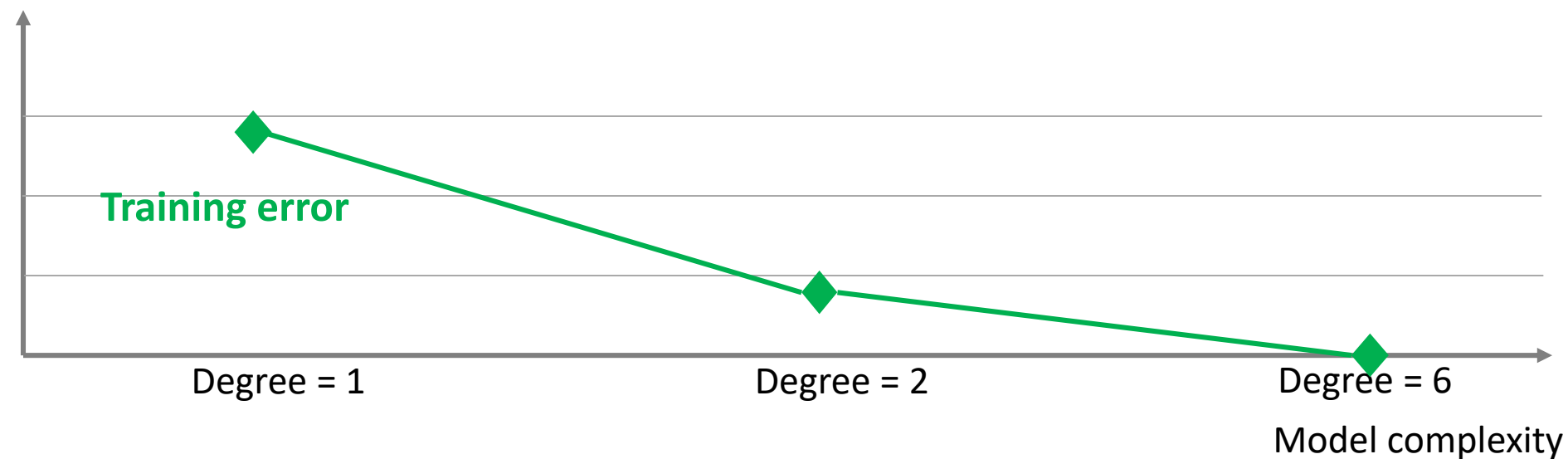
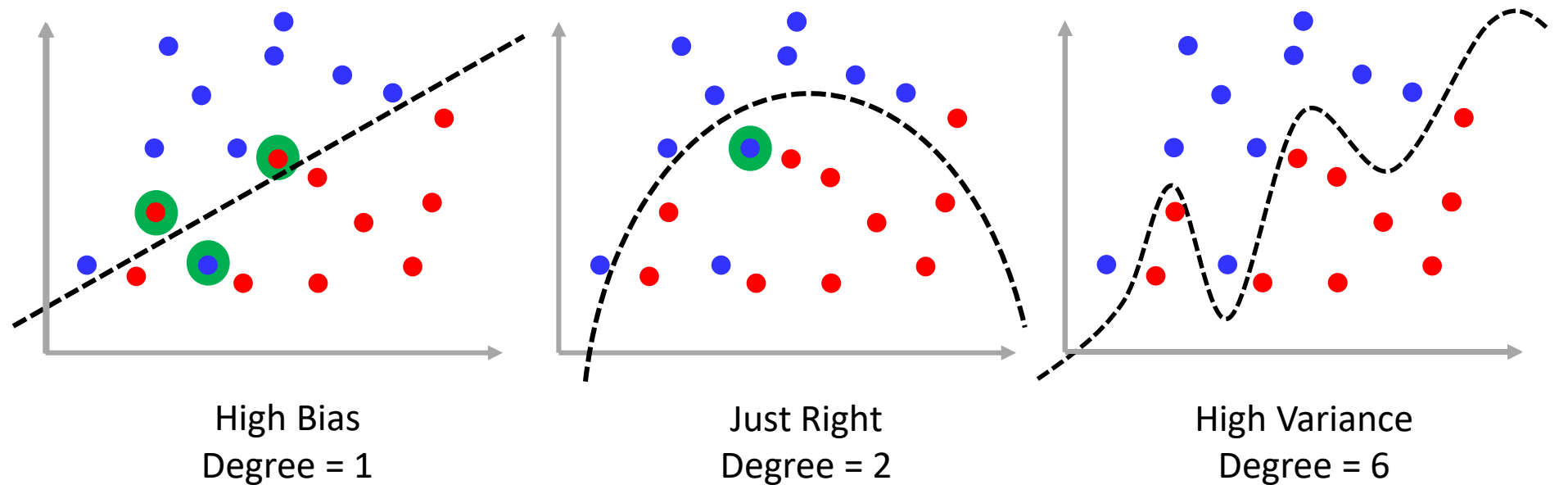
Model Complexity Graph

● ● Train data ○ ○ Validation data



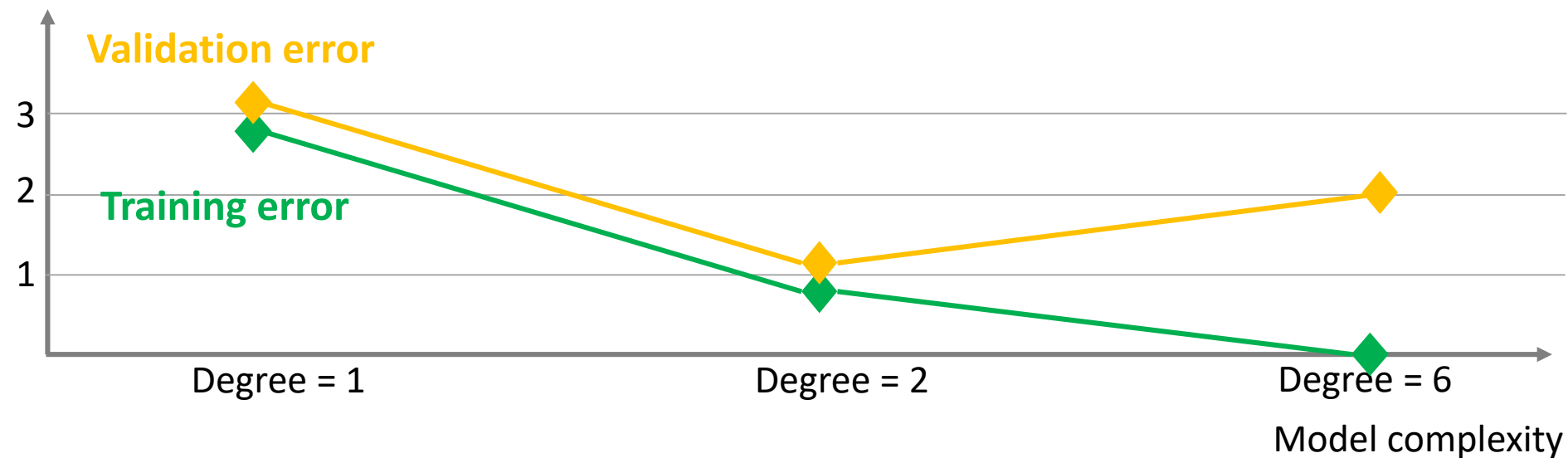
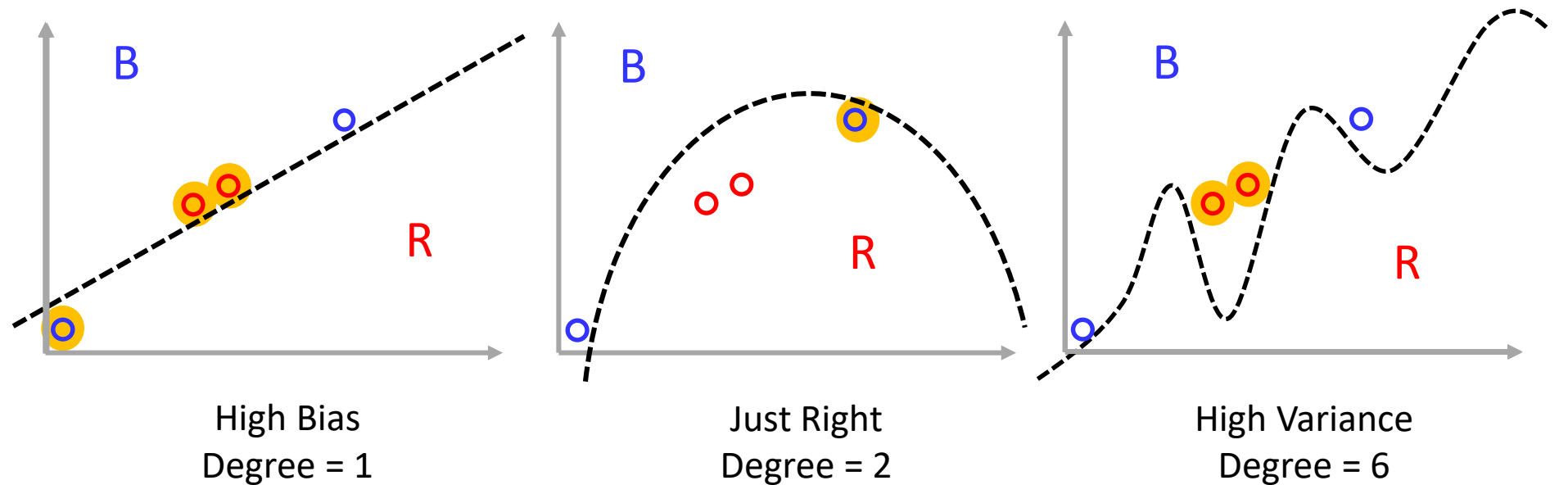
Model Complexity Graph

● ● Train data ○ ○ Validation data



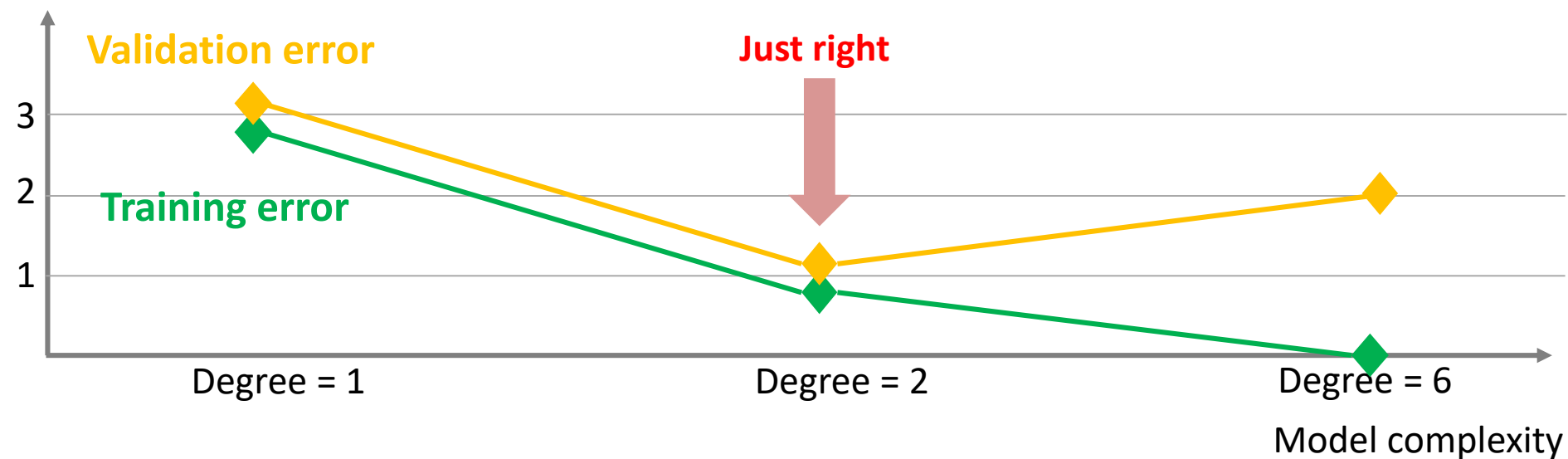
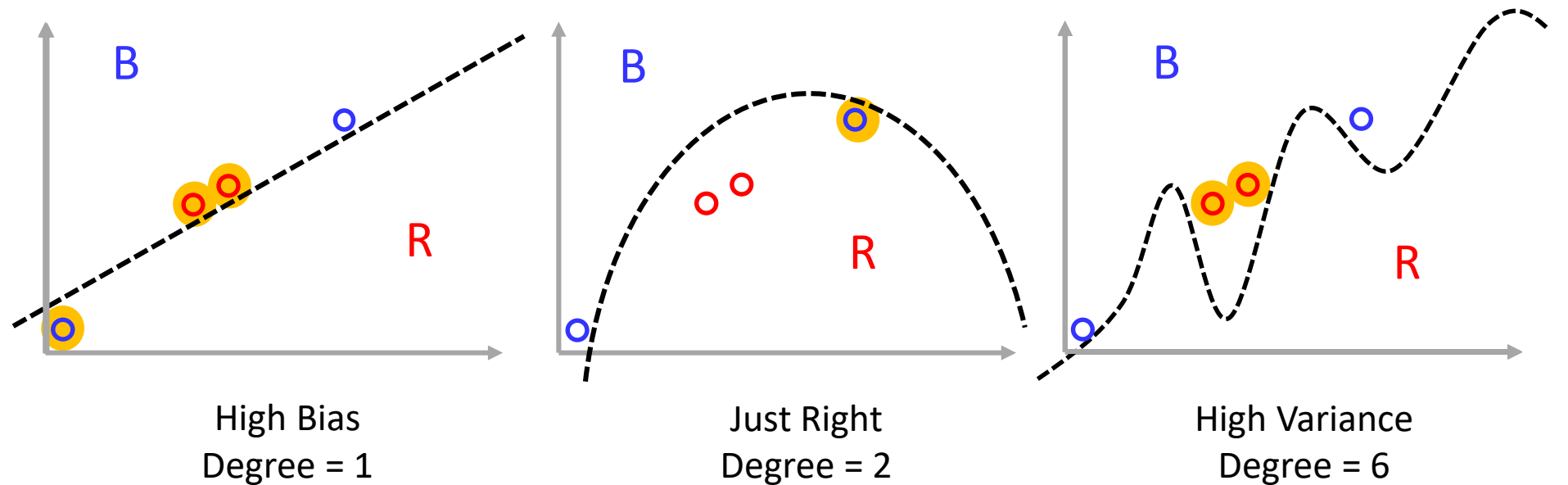
Model Complexity Graph

● ● Train data ○ ○ Validation data

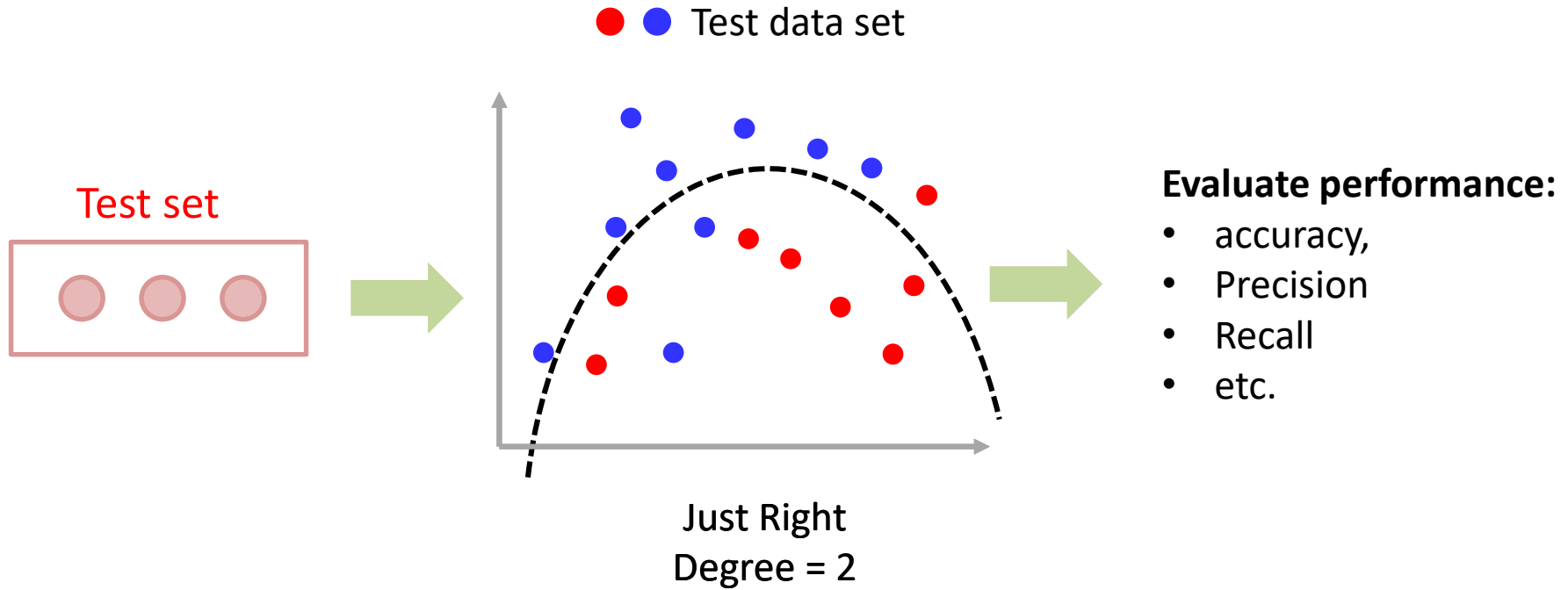


Model Complexity Graph

● ● Train data ○ ○ Test data



Model Complexity Graph



Bayesian Logistic Regression with Gaussian Prior (Ridge Logistic Regression)

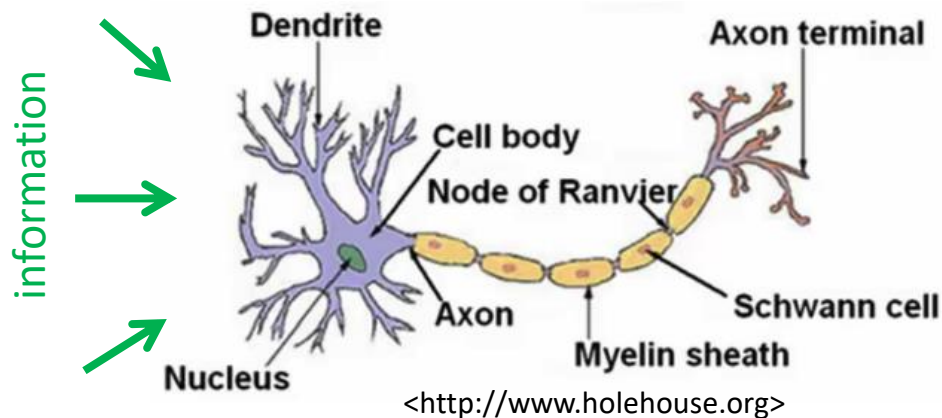
Jupyter Demo Simulation

Jupyter Demo Simulation (working on)

Neural Network

Concept

Neuron

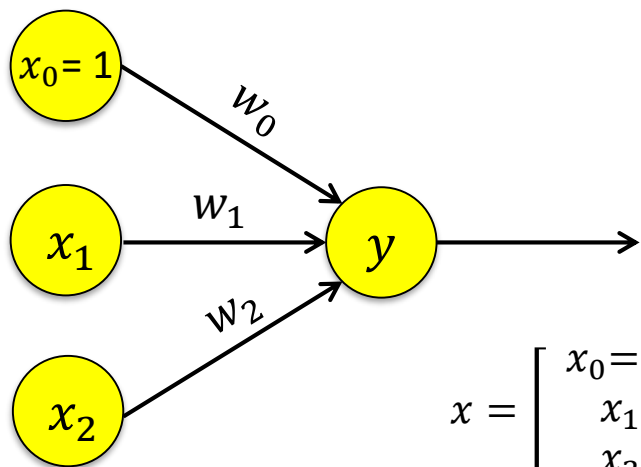


Dendrite: receive signal from multiple neurons

Cell body: Process signal

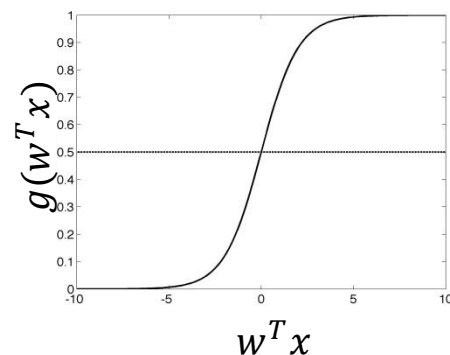
Axon: Send signal to other neuron

Logistic regression mimics the functionality of a single neuron



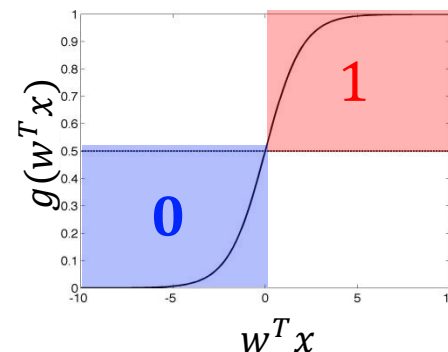
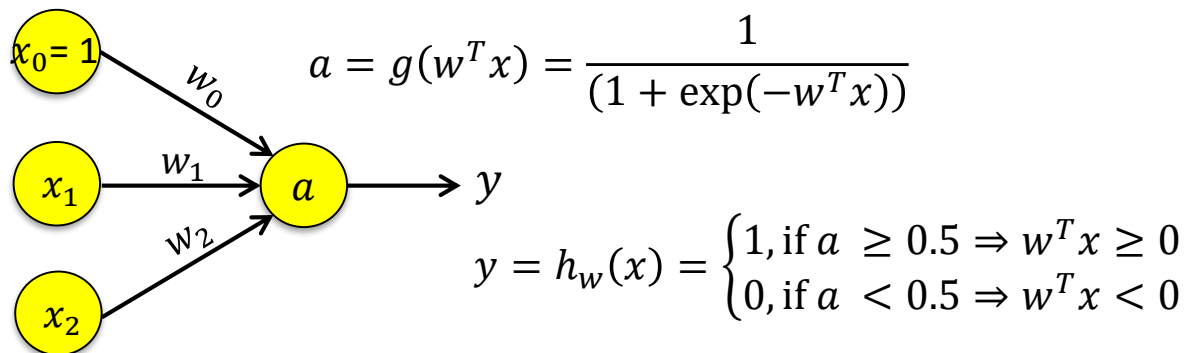
$$y = g(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

$$x = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$



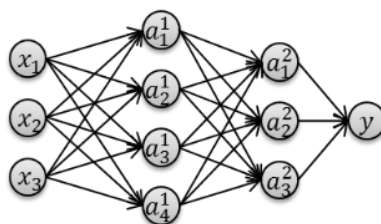
Concept

Classification with logistic regression



How to obtain non-linear decision boundaries?

Use multilayer neural networks



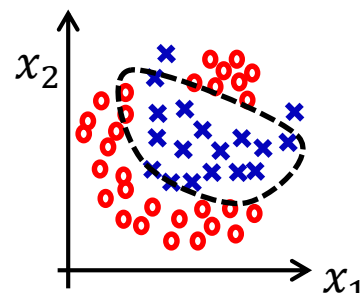
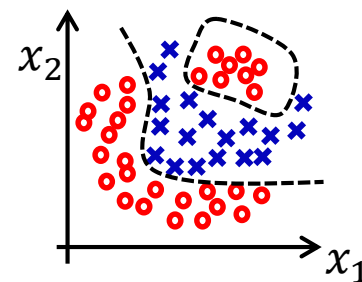
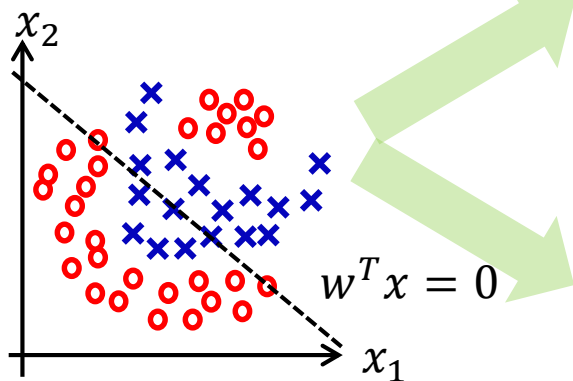
Any continuous function can be approximated well with a growing number of hidden units.

Use non-linear feature mapping

$$\phi: \chi \mapsto \mathcal{H}$$

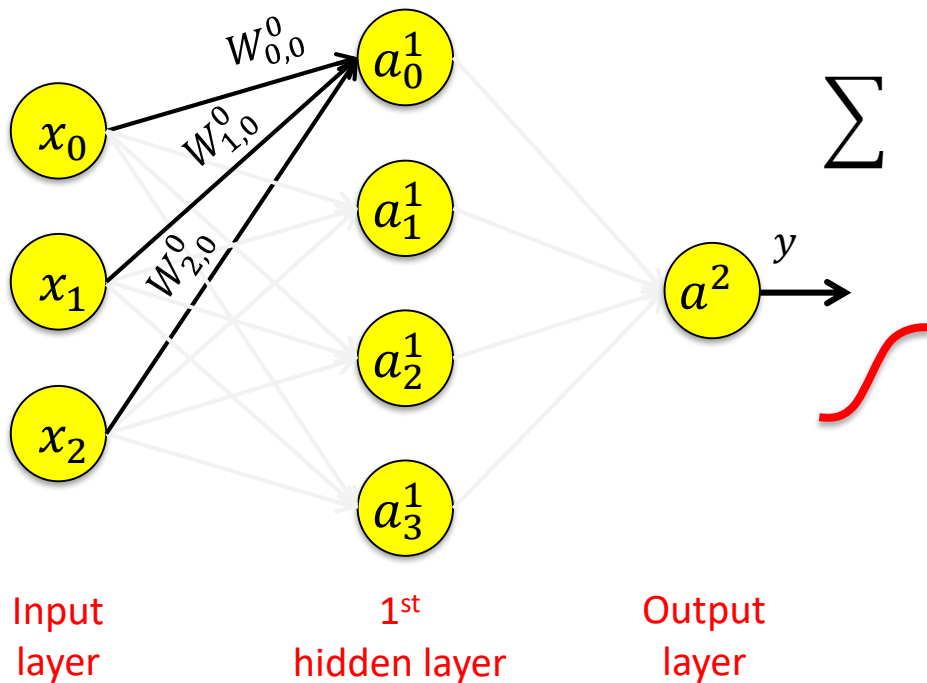
e.g., $\phi(x) = (x_1, x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$

Use kernel method (simplify the computation)



Linear decision boundary by single logistic regression

Concept



Σ

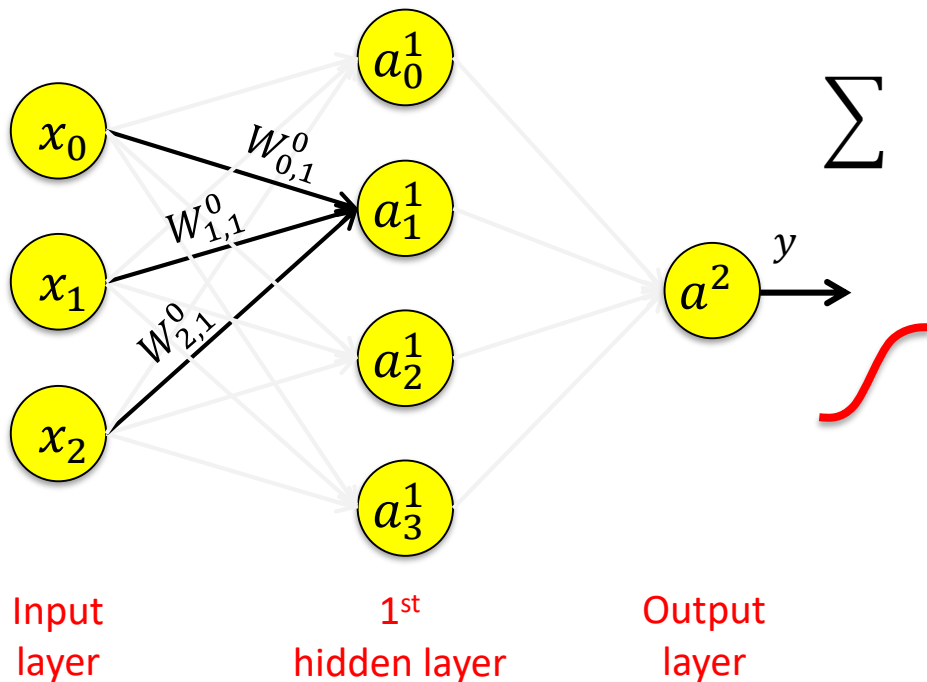
Weighted sum of the previous node values:

$$z_0^1 = W_{0,0}^0 x_0 + W_{1,0}^0 x_1 + W_{2,0}^0 x_2$$

Logistic transform:

$$a_0^1 = g(z_0^1) = g(W_{0,0}^0 x_0 + W_{1,0}^0 x_1 + W_{2,0}^0 x_2)$$

Concept

 Σ

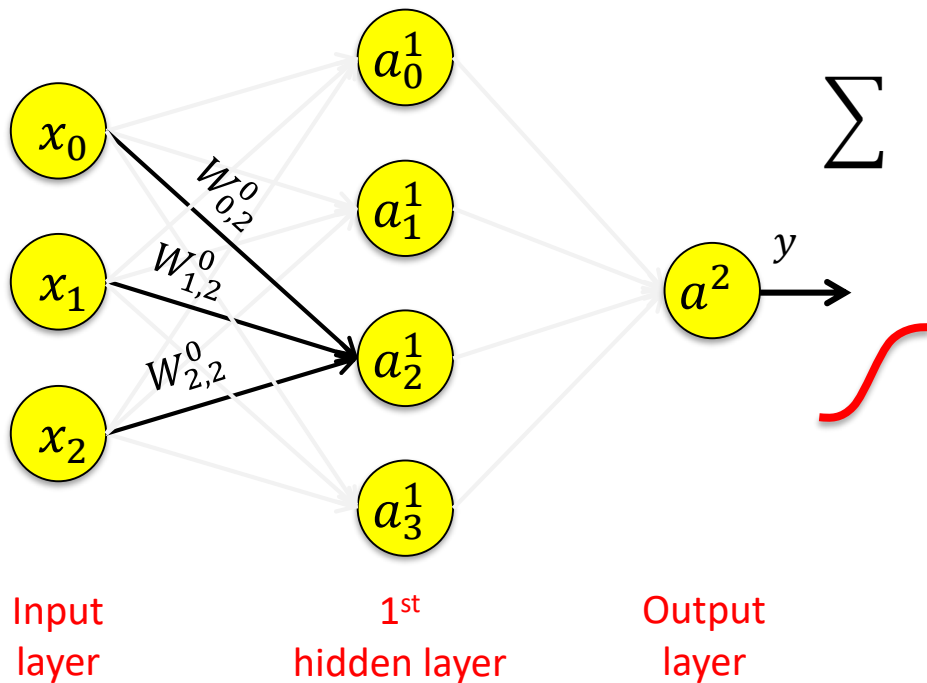
Weighted sum of the previous node values:

$$z_1^1 = W_{0,1}^0 x_0 + W_{1,1}^0 x_1 + W_{2,1}^0 x_2$$

Logistic transform:

$$a_1^1 = g(z_1^1) = g(W_{0,1}^0 x_0 + W_{1,1}^0 x_1 + W_{2,1}^0 x_2)$$

Concept

 Σ

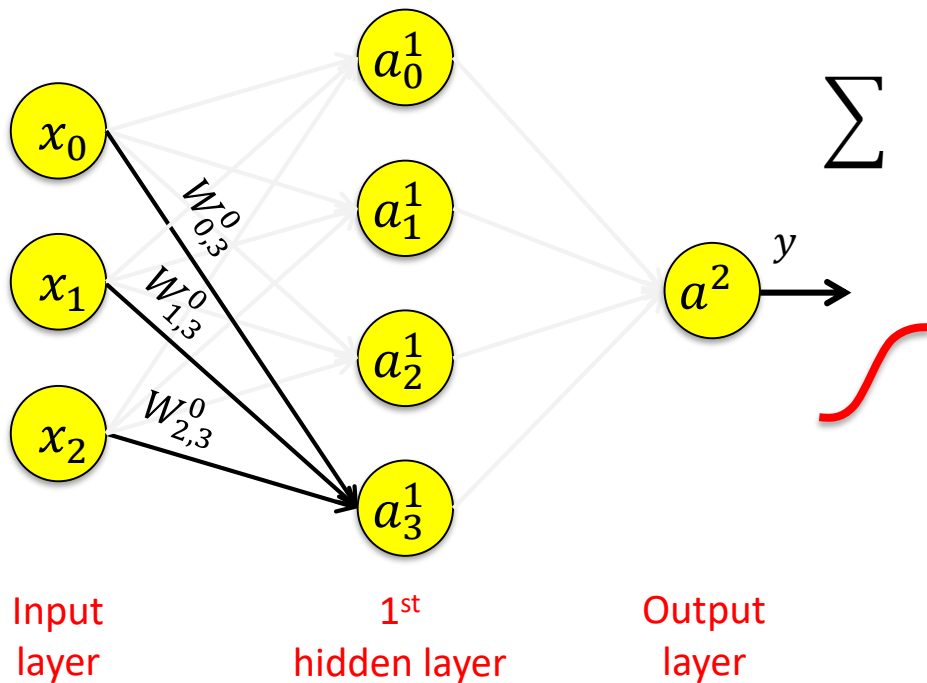
Weighted sum of the previous node values:

$$z_2^1 = W_{0,2}^0 x_0 + W_{1,2}^0 x_1 + W_{2,2}^0 x_2$$

Logistic transform:

$$a_2^1 = g(z_2^1) = g(W_{0,2}^0 x_0 + W_{1,2}^0 x_1 + W_{2,2}^0 x_2)$$

Concept



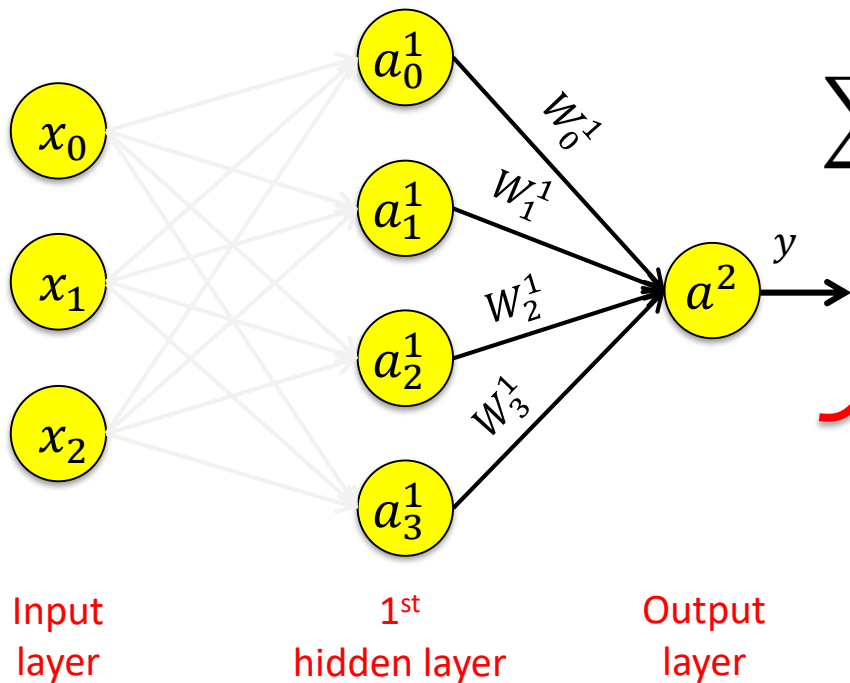
Σ Weighted sum of the previous node values:

$$z_3^1 = W_{0,3}^0 x_0 + W_{1,3}^0 x_1 + W_{2,3}^0 x_2$$

Logistic transform:

$$a_3^1 = g(z_3^1) = g(W_{0,3}^0 x_0 + W_{1,3}^0 x_1 + W_{2,3}^0 x_2)$$

Concept



Σ Weighted sum of the previous node values:
$$z^2 = W_0^1 a_0^1 + W_1^1 a_1^1 + W_2^1 a_2^1 + W_3^1 a_3^1$$

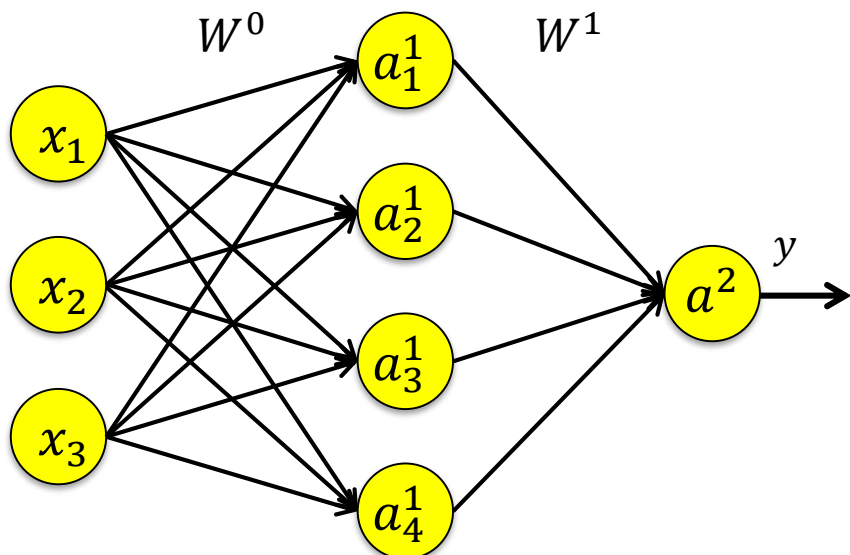
Logistic transform:

$$a^2 = g(z^2)$$
$$= g(W_0^1 a_0^1 + W_1^1 a_1^1 + W_2^1 a_2^1 + W_3^1 a_3^1)$$

Output node:

$$y = \begin{cases} 1, & \text{if } a^2 \geq 0.5 \\ 0, & \text{if } a^2 < 0.5 \end{cases}$$

Concept



Input
layer

1st
hidden layer

Output
layer

Σ



In every layer, two computations, weighted sum and the evaluations of sigmoid functions, are conducted to find the values at the next layer

Input layer \rightarrow 1st hidden layer

Linear combination

$$\begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \\ z_3^1 \end{bmatrix} = \begin{bmatrix} W_{0,0}^0 & W_{1,0}^0 & W_{2,0}^0 \\ W_{0,1}^0 & W_{1,1}^0 & W_{2,1}^0 \\ W_{0,2}^0 & W_{1,2}^0 & W_{2,2}^0 \\ W_{0,3}^0 & W_{1,3}^0 & W_{2,3}^0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$z^1 = W^1 x$$

Sigmoid

$$\begin{bmatrix} a_0^1 \\ a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix} = g \left(\begin{bmatrix} z_0^1 \\ z_1^1 \\ z_2^1 \\ z_3^1 \end{bmatrix} \right)$$

$$a^1 = g(z^1)$$

1st hidden layer \rightarrow output layer

Linear combination

$$[z^2] = [W_0^1 W_1^1 W_2^1 W_3^1] \begin{bmatrix} a_0^1 \\ a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix}$$

$$z^2 = W^2 a^1$$

Sigmoid

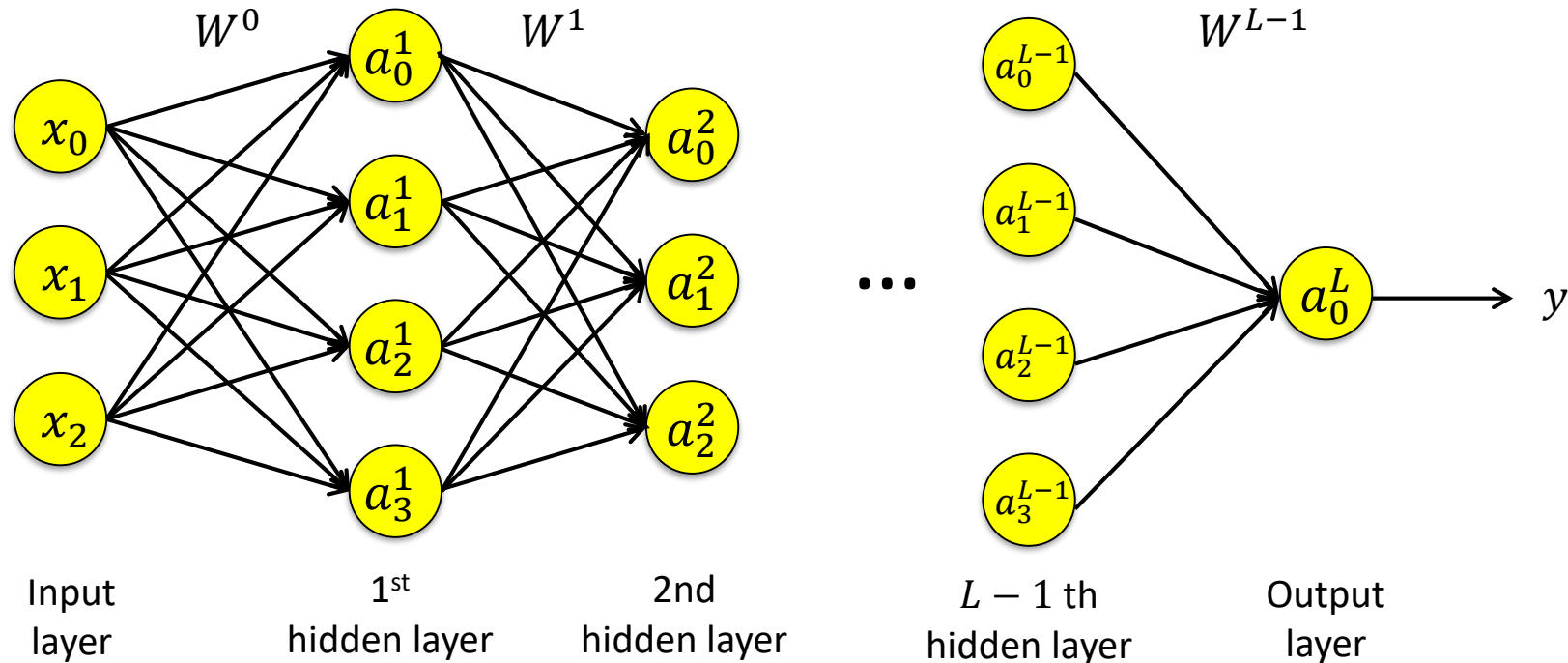
$$[a^2] = g([z^2])$$

$$a^2 = g(z^2)$$

Output layer

$$y = h_W(x) = \begin{cases} 1 & \text{if } a^2 \geq 0.5 \\ 0 & \text{if } a^2 < 0.5 \end{cases}$$

Prediction



Prediction (Forward propagation)

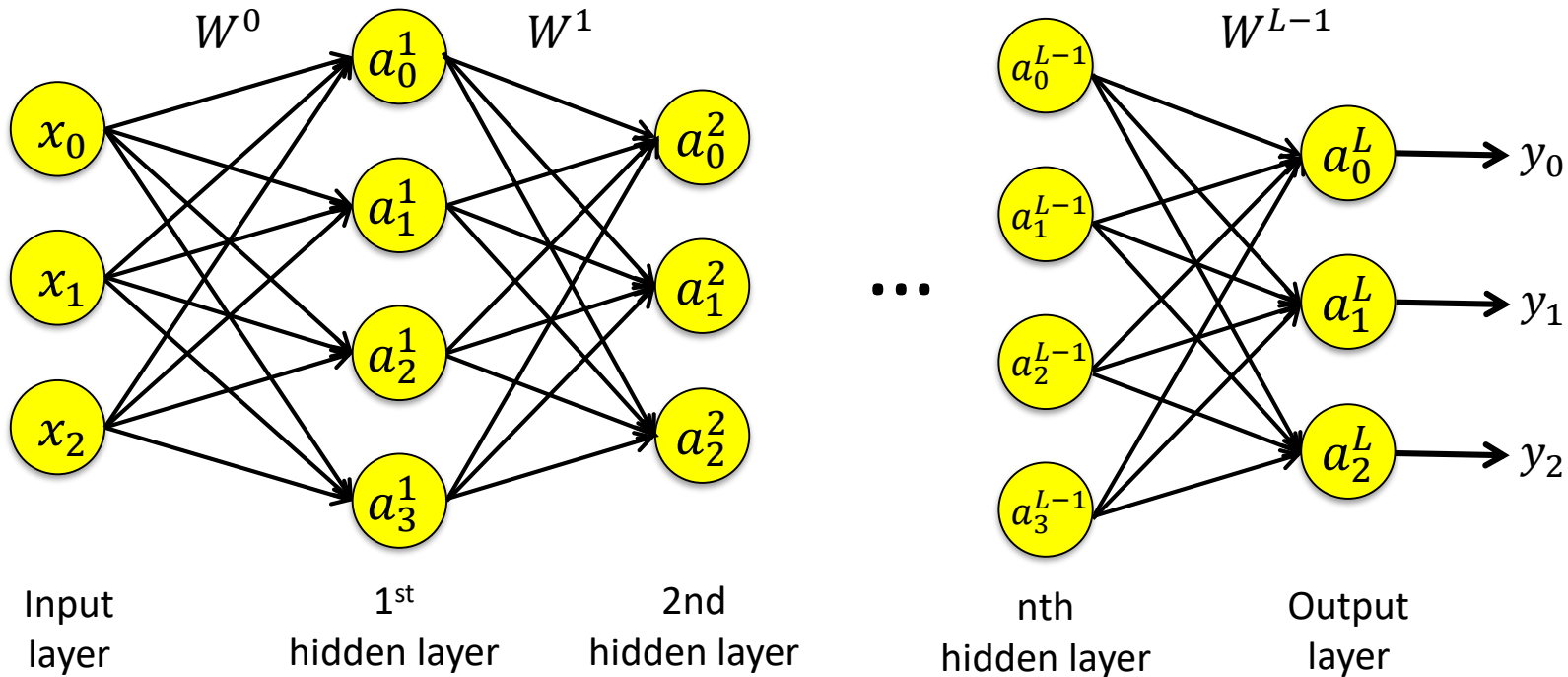
Input layer	$z^1 = W^1 x, a^1 = g(z^1)$
-------------	-----------------------------

Hidden layer	$\begin{cases} \text{for } l = 1, \dots, L \\ z^l = W^{l-1} a^{l-1}, a^l = g(z^l) \end{cases}$
--------------	------------------------------------------------------------------------------------------------

Output layer	$y = h_W(x) = \begin{cases} 1 & \text{if } a^L \geq 0.5 \\ 0 & \text{if } a^L < 0.5 \end{cases}$
--------------	--------------------------------------------------------------------------------------------------

- By adding more hidden layers, more complex features can be constructed.
- Prediction is conducted by sequence of matrix multiplication and the evaluations of logistic function

Multi-labels Classification



$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = h_W(x) = \text{one of } \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- Multiple binary classifications are executed for a certain class and the rest class.
(one vs. all method)

이 문장 의미 추가 부탁드립니다.

Cost Functions For Neural Network

- Log-likelihood of a logistic regression

$$\sum_{i=1}^m y_i \log g(w^T x_i) + (1 - y_i) \log(1 - g(w^T x_i)) - \lambda \sum_{i=1}^n w_i^2$$

Penalizing parameters

$$g(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

- Log-likelihood of a Neural Network

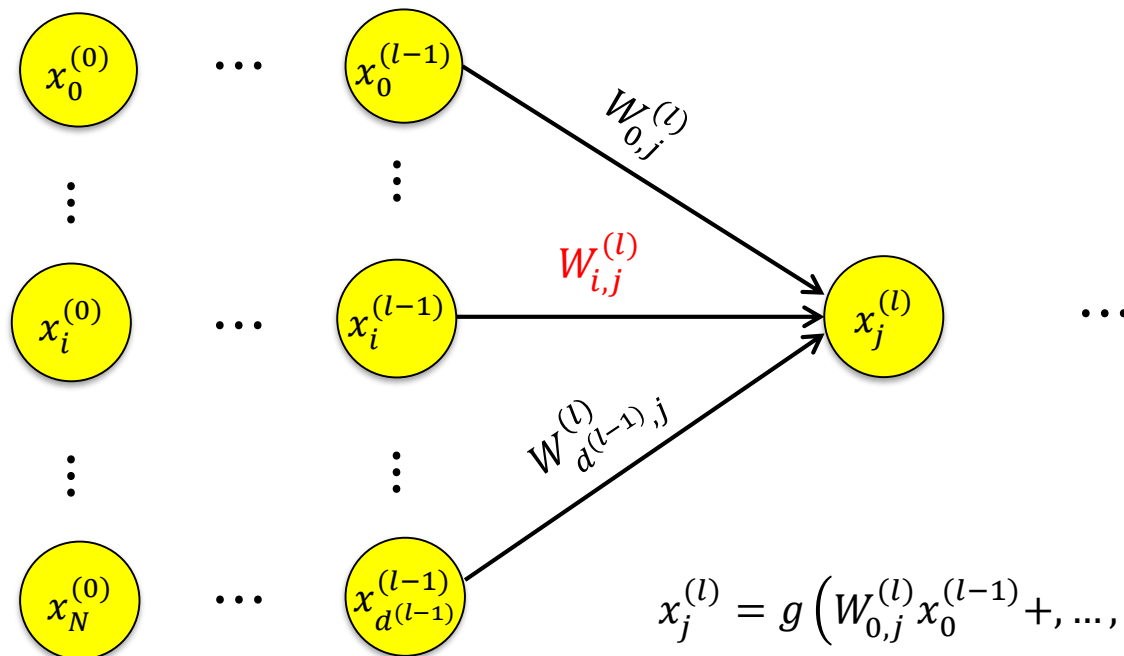
$$\sum_{i=1}^m y_i \log G_W(x_i) + (1 - y_i)(1 - \log(G_W(x_i))) - \lambda \sum_l \sum_i \sum_j (W_{i,j}^l)^2$$

Penalizing parameters

$G_W(x) = g(W_3^T g(W_2^T g(W_1^T x)))$ is a nested function of sigmoid functions

→ Training is difficult!!

Forward Propagation



$$x_j^{(l)} = g \left(W_{0,j}^{(l)} x_0^{(l-1)} +, \dots, + W_{i,j}^{(l)} x_i^{(l-1)} +, \dots +, W_{d^{(l-1)},j}^{(l)} x_{d^{(l-1)}}^{(l-1)} \right)$$

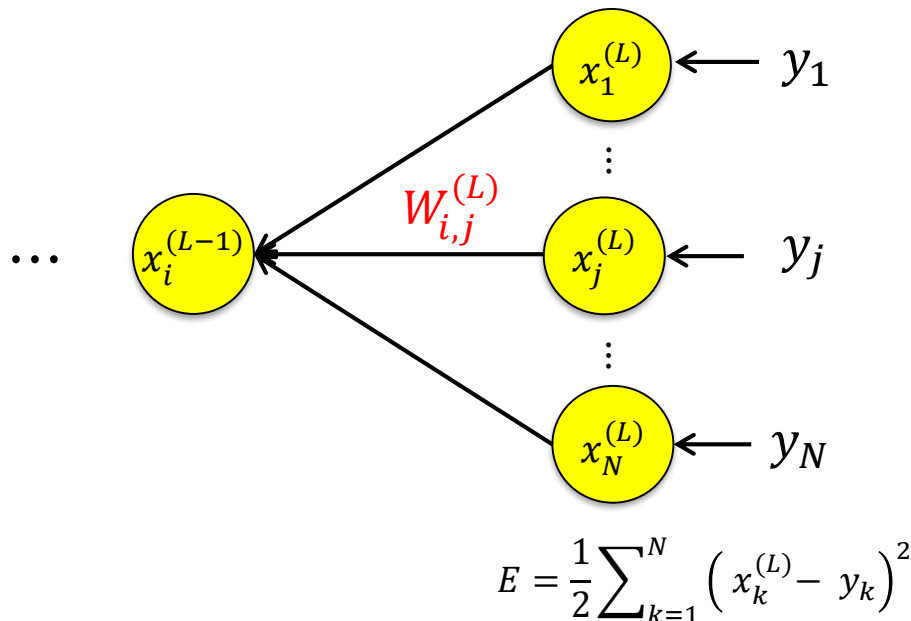
$$= g \left(\sum_{k=0}^{d^{(l-1)}} W_{k,j}^{(l)} x_k^{(l-1)} \right)$$

$$= g \left(\left(W_{\cdot,j}^{(l)} \right)^T \mathbf{x}^{(l-1)} \right)$$

$$= h_{W_{\cdot,j}^{(l)}} \left(\mathbf{x}^{(l-1)} \right)$$

The value for each node is determined by logistic regression for a single layer

Backward Propagation



At output node

$$E = \frac{1}{2} \sum_{k=1}^N (x_k^{(L)} - y_k)^2$$

$$\frac{\partial E}{\partial W_{i,j}^{(L)}} = (x_j^{(L)} - y_j) \frac{\partial}{\partial W_{i,j}^{(L)}} x_j^{(L)}$$

$$= e_j^{(L)} \frac{\partial}{\partial W_{i,j}^{(L)}} g(z_j^{(L)})$$

$$= e_j^{(L)} g'(z_j^{(L)}) \frac{\partial}{\partial W_{i,j}^{(L)}} z_j^{(L)}$$

$$= e_j^{(L)} g'(z_j^{(L)}) x_i^{(L-1)}$$

$$= x_i^{(L-1)} \delta_j^{(L)}$$

where $\delta_j^{(L)} = e_j^{(L)} g'(z_j^{(L)})$

Define the summed variable for simplicity

$$z_j^{(L)} = \sum_{k=0}^{d^{(L-1)}} W_{k,j}^{(L)} x_k^{(L-1)}, \text{ then } x_j^{(L)} = g(z_j^{(L)})$$

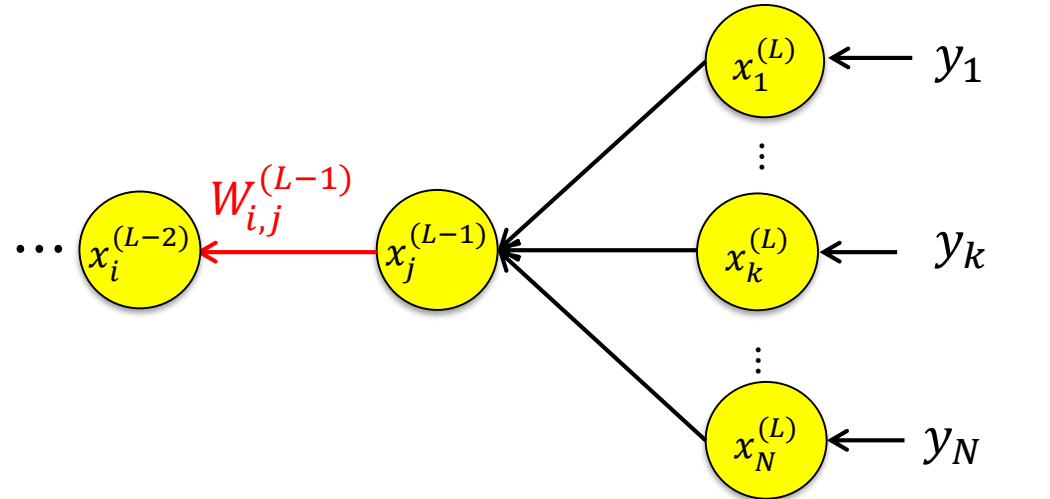
Backward Propagation

At hidden layer

$$E = \frac{1}{2} \sum_{k=1}^N \left(x_k^{(L)} - y_k \right)^2$$

$$\begin{aligned} \frac{\partial E}{\partial W_{i,j}^{(L-1)}} &= \sum_{k=1}^N \left(x_k^{(L)} - y_k \right) \frac{\partial}{\partial W_{i,j}^{(L-1)}} x_k^{(L)} \\ &= \sum_{k=1}^N e_k^{(L)} \frac{\partial}{\partial W_{i,j}^{(L-1)}} g \left(z_k^{(L)} \right) \\ &= \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) \frac{\partial}{\partial W_{i,j}^{(L-1)}} z_k^{(L)} \\ &= \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) \frac{\partial}{\partial W_{i,j}^{(L-1)}} W_{j,k}^{(L)} x_j^{(L-1)} \\ &= \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) \frac{\partial W_{j,k}^{(L)} x_j^{(L-1)}}{\partial x_j^{(L-1)}} \frac{\partial x_j^{(L-1)}}{\partial W_{i,j}^{(L-1)}} \\ &= \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) W_{j,k}^{(L)} g' \left(z_j^{(L-1)} \right) x_i^{(L-2)} \\ &= x_i^{(L-2)} g' \left(z_j^{(L-1)} \right) \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) W_{j,k}^{(L)} \\ &= x_i^{(L-2)} \delta_j^{(L-1)} \end{aligned}$$

where $\delta_j^{(L-1)} = g' \left(z_j^{(L-1)} \right) \sum_{k=1}^N e_k^{(L)} g' \left(z_k^{(L)} \right) W_{j,k}^{(L)} = g' \left(z_j^{(L-1)} \right) \sum_{k=1}^N \delta_j^{(L)} W_{j,k}^{(L)}$



$$e_k^{(L)} = x_k^{(L)} - y_k$$

$$z_k^{(L)} = \sum_{r=0}^{d^{(L-1)}} W_{r,k}^{(L)} x_r^{(L-1)}$$

$$E = \frac{1}{2} \sum_{k=1}^N \left(x_k^{(L)} - y_k \right)^2$$

$$\frac{\partial x_j^{(L-1)}}{\partial W_{i,j}^{(L-1)}} = g' \left(z_j^{(L-1)} \right) x_i^{(L-2)} \text{ from previous slide}$$

Forward Backward Propagation algorithm

Forward propagation

$$x_j^{(l)} = g \left(\sum_{k=0}^{a^{(l-1)}} W_{k,j}^{(l)} x_k^{(l-1)} \right)$$

$$g(z) = \frac{1}{(1 + \exp(-z))}$$

Backward propagation

$$\delta_j^{(l)} = \begin{cases} e_j^{(l)} g' \left(z_j^{(l)} \right) & \text{If } l = L \text{ (output layer)} \\ g' \left(z_j^{(l)} \right) \sum_{k=1}^N \delta_j^{(l+1)} W_{j,k}^{(l+1)} & \text{If } l < L \text{ (hidden layer)} \end{cases}$$

$$g' \left(z_j^{(l)} \right) = x_j^{(l)} \left(1 - x_j^{(l)} \right)$$

Forward Backward Propagation algorithm

- 1 Initialize all weights $W_{i,j}^{(l)}$ at random
- 2 For $t = 0, 1, 2, \dots$ do
- 3 Pick a single data point in $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, \dots, m\}$
- 4 **Forward propagation** : compute all $x_j^{(l)}$
- 5 **Backward propagation** : compute all $\delta_j^{(l)}$
- 6 Update the weights $W_{i,j}^{(l)} \leftarrow W_{i,j}^{(l)} - \alpha x_i^{(l-1)} \delta_j^{(l)}$
- 7 Iterate until $W_{i,j}^{(l)}$ converges
- 8 Return the final weights $W_{i,j}^{(l)}$

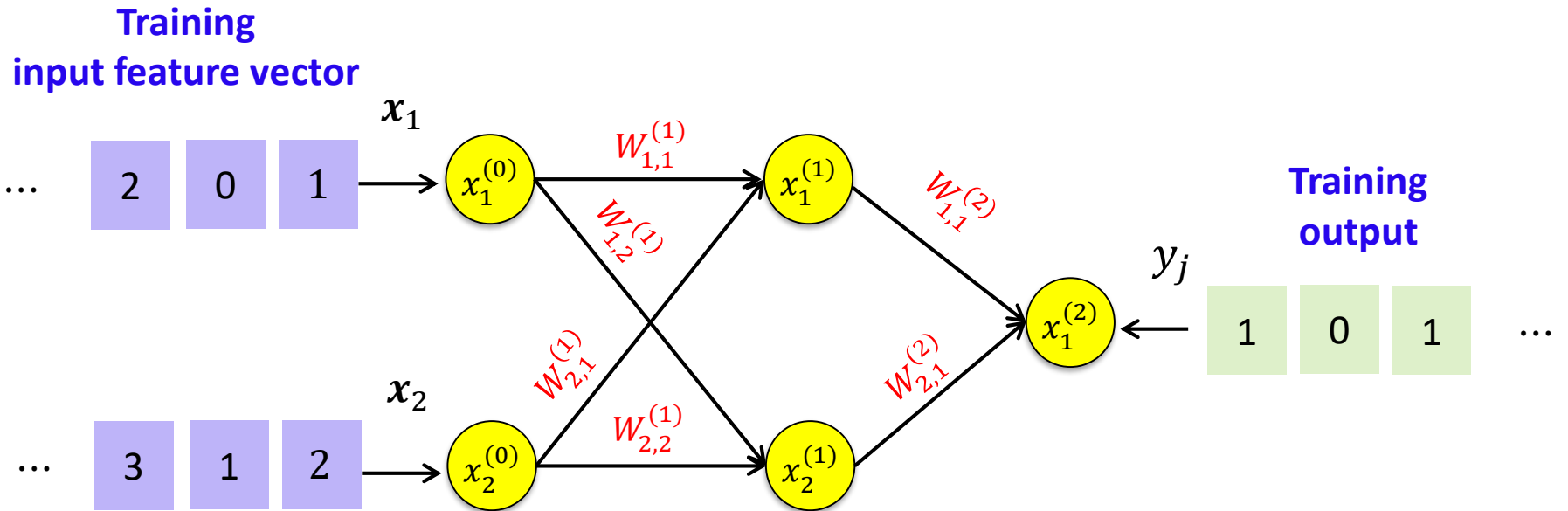
Applying gradient decent

$$W_{i,j}^{(l)} = W_{i,j}^{(l)} - \alpha \frac{\partial E}{\partial W_{i,j}^{(l)}}$$

$$\frac{\partial E}{\partial W_{i,j}^{(l)}} = x_i^{(l-1)} \delta_j^{(l)}$$

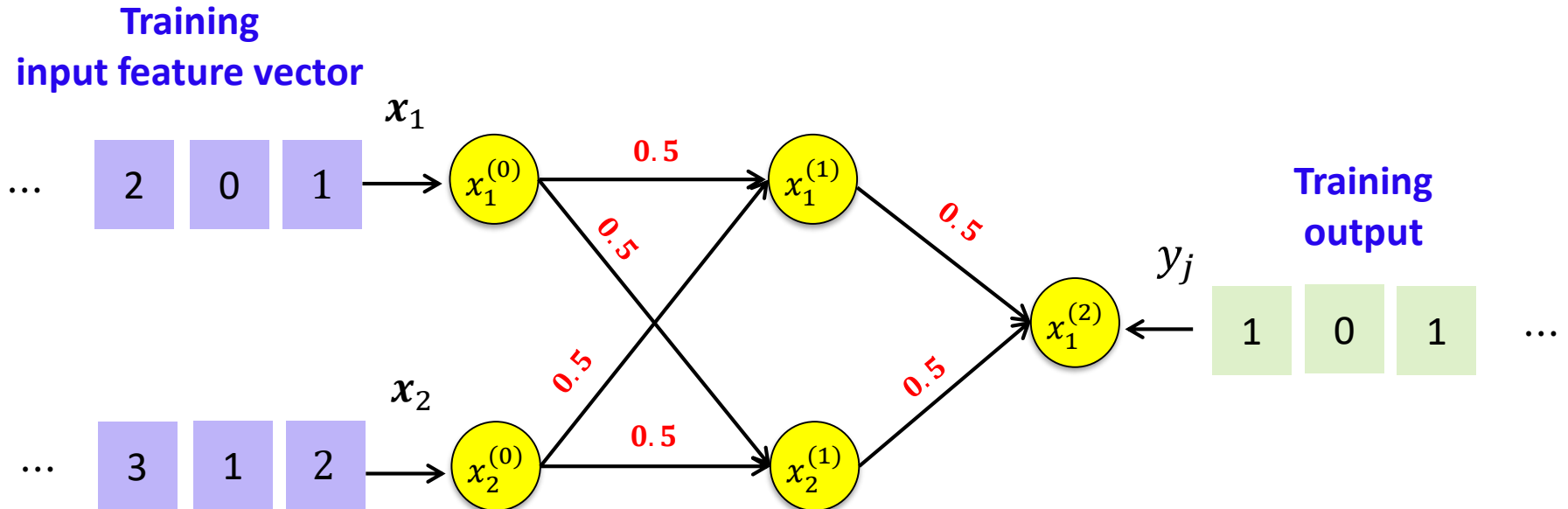
α is learning rate

Illustrations



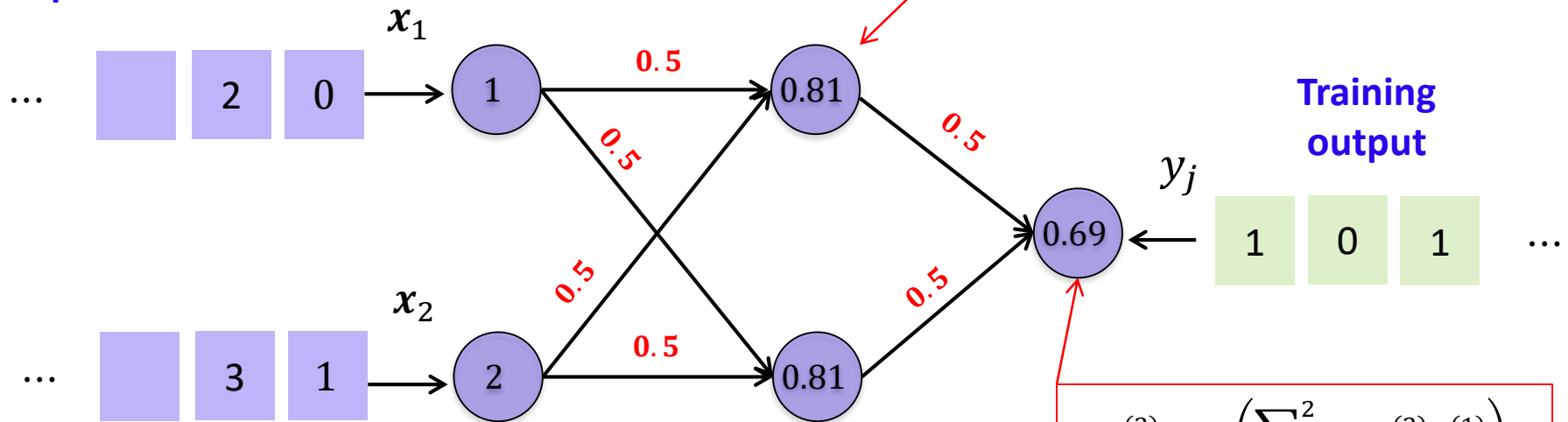
Illustrations

Initialize weight parameters $w_{i,j}^{(L-1)}$ (iteration = 0)



Forward Propagation (iteration = 1)

Training
input feature vector



$$x_1^{(1)} = g \left(\sum_{k=1}^2 W_{k,1}^{(1)} x_k^{(0)} \right) = \frac{1}{1 + \exp \left(-[0.5, 0.5]^T \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)}$$

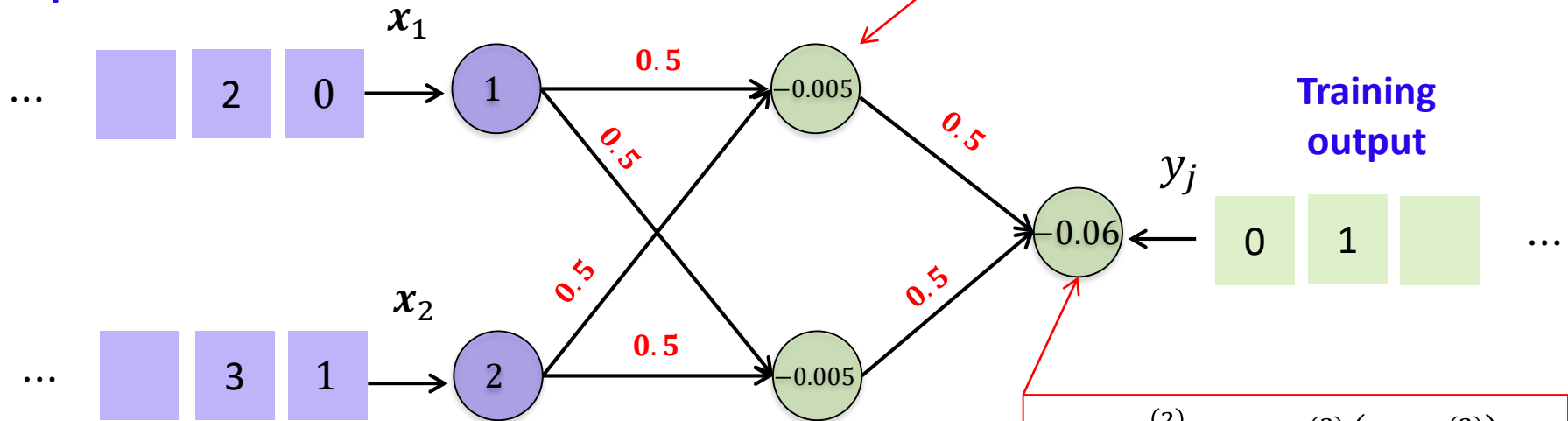
Forward propagation

$$x_j^{(l)} = g \left(\sum_{k=0}^{d^{(l-1)}} W_{k,j}^{(l)} x_k^{(l-1)} \right)$$

$$x_1^{(2)} = g \left(\sum_{k=1}^2 W_{k,1}^{(2)} x_k^{(1)} \right) = \frac{1}{1 + \exp \left(-[0.5, 0.5]^T \begin{bmatrix} 0.81 \\ 0.81 \end{bmatrix} \right)}$$

Backward Propagation (iteration = 1)

Training
input feature vector



$$x_1^{(1)}(1 - x_1^{(1)}) \sum_{k=1}^1 \delta_1^{(2)} W_{1,1}^{(2)} = 0.81(1 - 0.81)(-0.06)0.5$$

Backward propagation

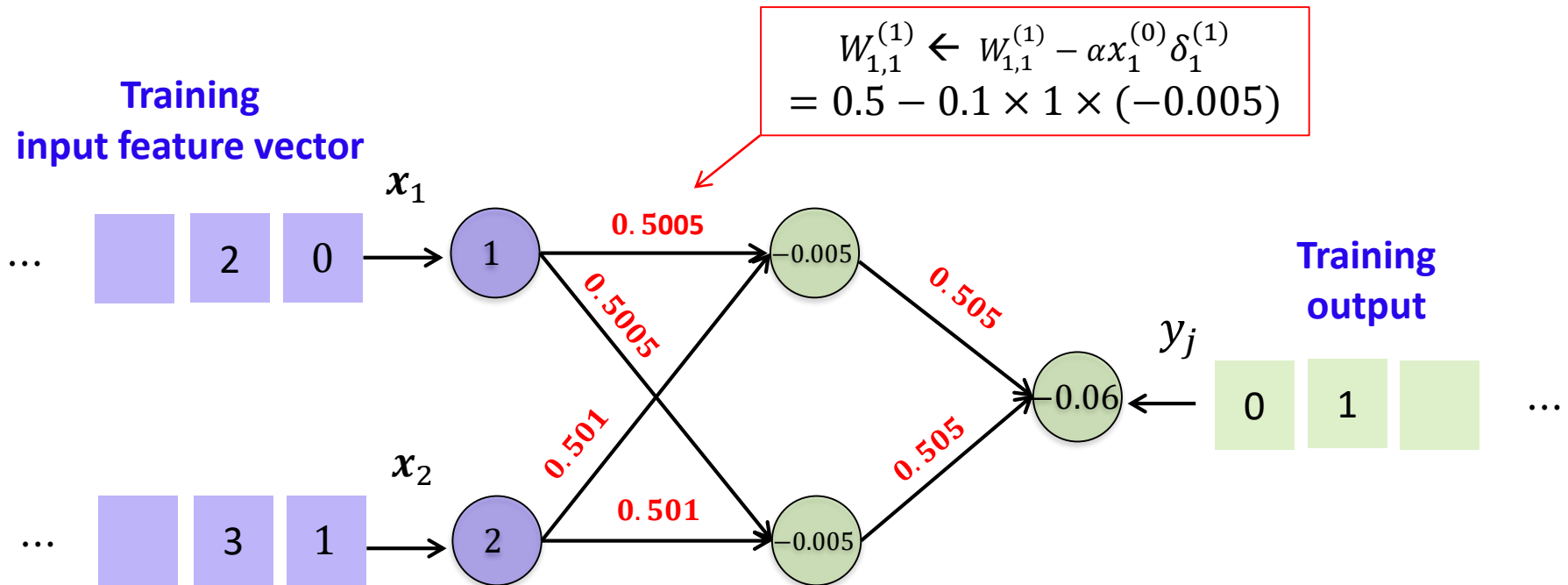
$$(x_1^{(2)} - y_1) x_1^{(2)}(1 - x_1^{(2)}) = (0.69 - 1)0.69(1 - 0.69)$$

$$\delta_j^{(l)} = \begin{cases} e_j^{(l)} g'(z_j^{(l)}) \\ g'(z_j^{(l)}) \sum_{k=1}^N \delta_j^{(l+1)} W_{j,k}^{(l+1)} \\ g'(z_j^{(l)}) = x_j^{(l)}(1 - x_j^{(l)}) \end{cases}$$

Illustrations

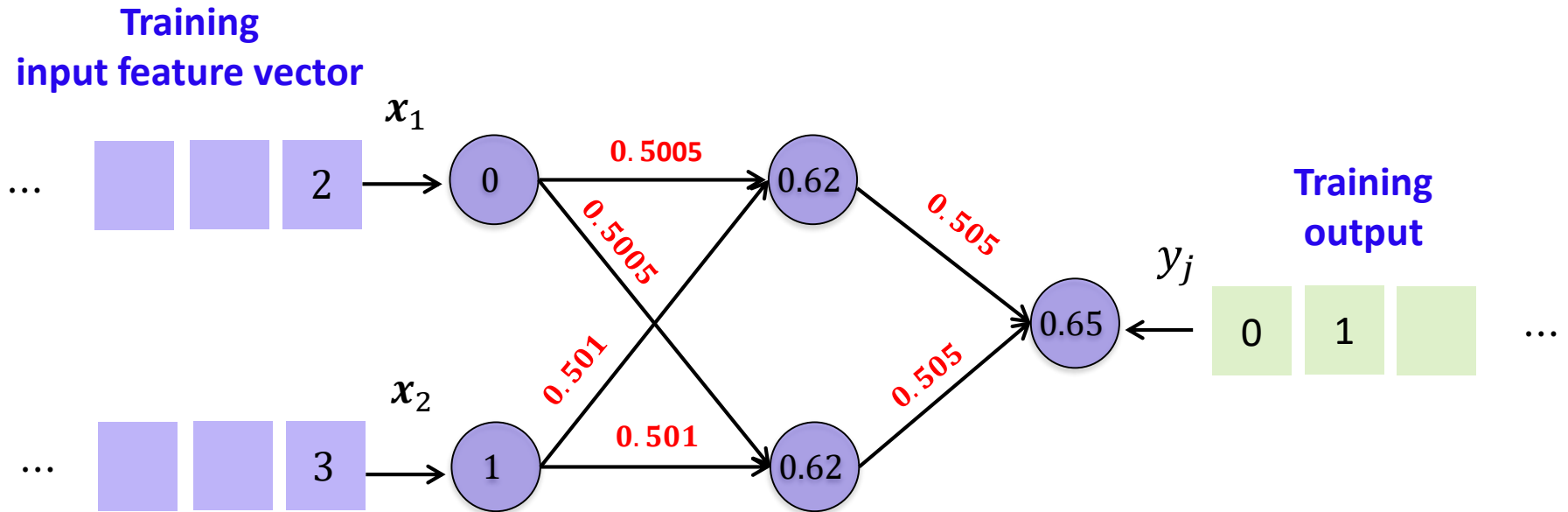
Parameters updating (iteration = 1)

$\alpha = 0.1$



Update the weights $W_{i,j}^{(l)} \leftarrow W_{i,j}^{(l)} - \alpha x_i^{(l-1)} \delta_j^{(l)}$

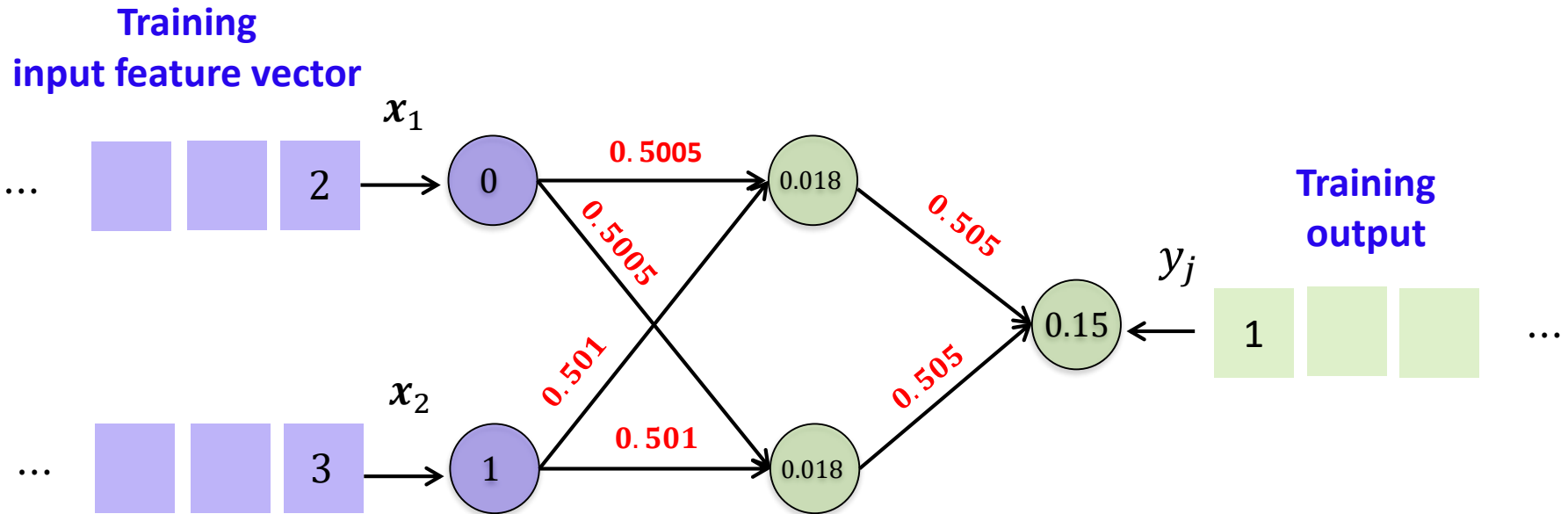
Forward Propagation (iteration = 2)



Forward propagation

$$x_j^{(l)} = g \left(\sum_{k=0}^{d^{(l-1)}} W_{k,j}^{(l)} x_k^{(l-1)} \right)$$

Backward Propagation (iteration = 2)



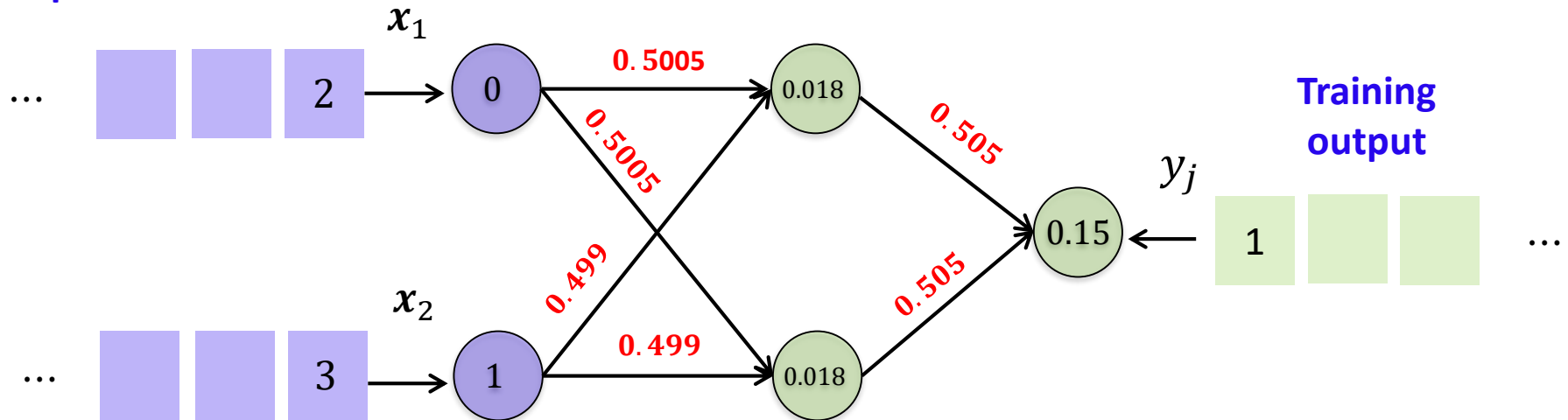
Backward propagation

$$\delta_j^{(l)} = \begin{cases} e_j^{(l)} g' \left(z_j^{(l)} \right) \\ g' \left(z_j^{(l)} \right) \sum_{k=1}^N \delta_k^{(l+1)} W_{j,k}^{(l+1)} \end{cases}$$

$$g' \left(z_j^{(l)} \right) = x_j^{(l)} \left(1 - x_j^{(l)} \right)$$

Backward Propagation (iteration = 2)

Training
input feature vector



Update the weights $W_{i,j}^{(l)} \leftarrow W_{i,j}^{(l)} - \alpha x_i^{(l-1)} \delta_j^{(l)}$

Generative model

Generative model

1. Define Class prior $p(y)$ and likelihood $P(x|y)$
2. Learn the parameters of the models, $P(y)$ and $P(x|y)$
3. Express posterior distribution on class y given the input vector x

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in Y} P(x|y)P(y)}$$

4. Prediction step: any new input feature vector x_{new} can be classified according to the maximum a posteriori detection principle (MAP)

$$\begin{aligned}\hat{y} &= \operatorname{argmax}_y P(y|x_{new}) = \operatorname{argmax}_y \frac{P(x_{new}|y)P(y)}{\sum_y P(x_{new}|y)P(y)} \\ &= \operatorname{argmax}_y P(x_{new}|y)P(y)\end{aligned}$$

Generative model

1. Define prior and likelihood

$$p(x|y = \text{dog}), p(y = \text{dog})$$
$$p(x|y = \text{cat}), p(y = \text{cat})$$

2. Learn the parameters for the models

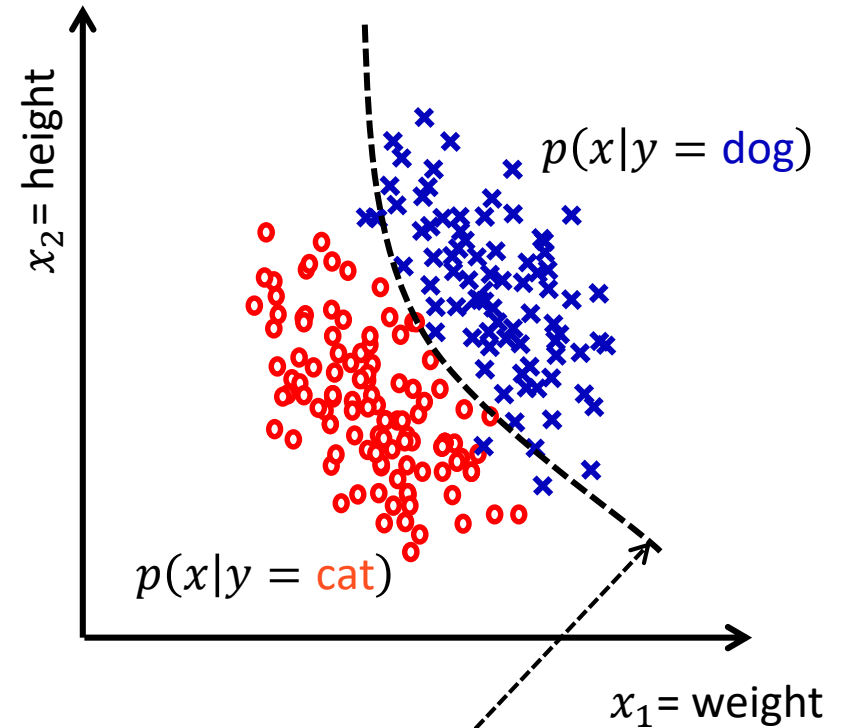
3. Construct the posterior distribution on class

$$P(y = \text{dog}|x) = \frac{p(x|y = \text{dog})p(y = \text{dog})}{\sum_{y \in \{\text{dog}, \text{cat}\}} p(x|y)p(y)}$$

$$P(y = \text{cat}|x) = \frac{p(x|y = \text{cat})p(y = \text{cat})}{\sum_{y \in \{\text{dog}, \text{cat}\}} p(x|y)p(y)}$$

4. Classify animal based on MAP estimation:

$$\hat{y} = \operatorname{argmax}_{y \in Y} P(y|x^{\text{new}})$$



Decision boundary

$$p(y = \text{dog}|x) = p(y = \text{cat}|x)$$

The shape of a decision boundary changes depending on the assumptions on the model (ex., linear, quadratic, ...)

Multivariate Gaussian Distribution

Univariate Gaussian

$$N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Mean $\mu = E[X]$

variance $\sigma^2 = \text{var}(X) = E[(X - E[X])^2]$

Multivariate Gaussian

$$N(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

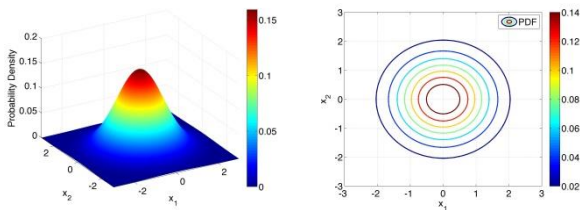
Mean vector

Covariance matrix

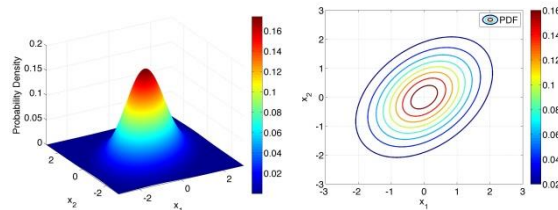
$$\boldsymbol{\mu} = \begin{bmatrix} E[X_1] \\ \vdots \\ E[X_n] \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \text{Cov}[X_1, X_1] & \cdots & \text{Cov}[X_1, X_n] \\ \vdots & \ddots & \vdots \\ \text{Cov}[X_n, X_1] & \cdots & \text{Cov}[X_n, X_n] \end{bmatrix}$$

$$\text{Cov}[X, Z] = E[(X - E[X])(Z - E[Z])]$$

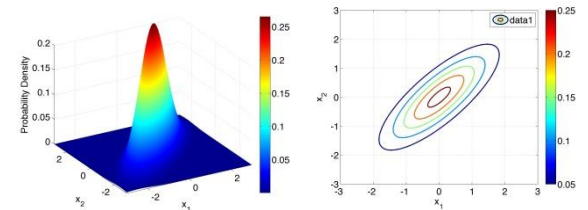
$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

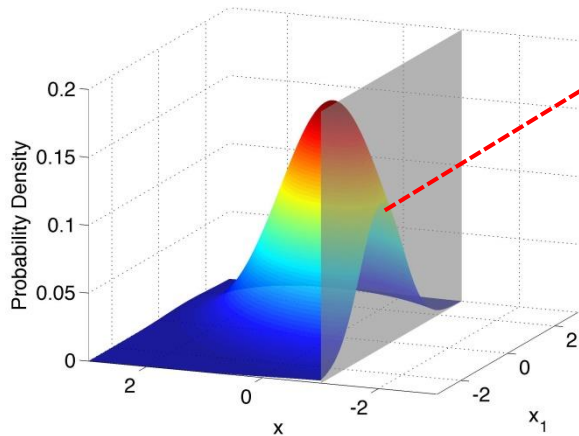


$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$



Multivariate Gaussian Distribution

Conditionalization → Gaussian

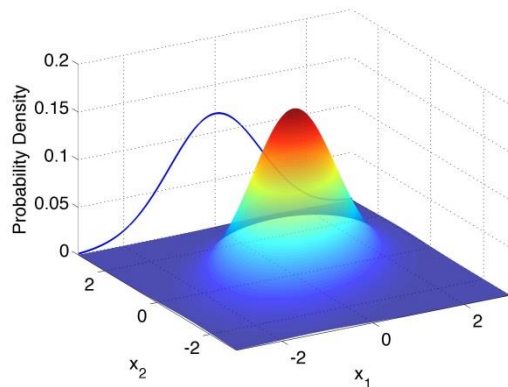


$$P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} \quad (\text{Graph does not show normalization})$$

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim N \left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

$$X_1 | \{X_2 = x_2\} \sim N(\Sigma_{21}\Sigma_{11}^{-1}(x_2 - \mu_1) + \mu_1, \Sigma_{11} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$$

Marginalization → Gaussian



$$P(X_1) = \int_{x_2=-\infty}^{x_2=\infty} P(X_1, X_2 = x_2) dx_2$$

(Graph does not show normalization)

Properties of the Covariance Matrix

The covariance matrix of a random vector $\mathbf{X} \in \mathbf{R}^n$ with mean vector \mathbf{m}_x is defined via:

$$\mathbf{C}_x = E[(\mathbf{X} - \mathbf{m})(\mathbf{X} - \mathbf{m})^T].$$

The $(i, j)^{\text{th}}$ element of this covariance matrix \mathbf{C}_x is given by

$$C_{ij} = E[(X_i - m_i)(X_j - m_j)] = \sigma_{ij}.$$

The diagonal entries of this covariance matrix \mathbf{C}_x are the variances of the components of the random vector \mathbf{X} , i.e.,

$$C_{ii} = E[(X_i - m_i)^2] = \sigma_i^2.$$

Since the diagonal entries are all positive the trace of this covariance matrix is positive, i.e.,

$$\text{Trace}(\mathbf{C}_x) = \sum_{i=1}^n C_{ii} > 0.$$

This covariance matrix \mathbf{C}_x is symmetric, i.e., $\mathbf{C}_x = \mathbf{C}_x^T$ because :

$$C_{ij} = \sigma_{ij} = \sigma_{ji} = C_{ji}.$$

The covariance matrix \mathbf{C}_x is positive semidefinite, i.e., for $\mathbf{a} \in \mathbf{R}^n$:

$$\begin{aligned} E\{[(\mathbf{X} - \mathbf{m})^T \mathbf{a}]^2\} &= E\{[(\mathbf{X} - \mathbf{m})^T \mathbf{a}]^T [(\mathbf{X} - \mathbf{m})^T \mathbf{a}]\} \geq 0 \\ E[\mathbf{a}^T (\mathbf{X} - \mathbf{m})(\mathbf{X} - \mathbf{m})^T \mathbf{a}] &\geq 0, \quad \mathbf{a} \in \mathbf{R}^n \\ \mathbf{a}^T \mathbf{C}_x \mathbf{a} &\geq 0, \quad \mathbf{a} \in \mathbf{R}^n. \end{aligned}$$

Concept

1. The class **prior** is represented as **multinomial distribution**

$$p(y = j) = \phi_j, \quad (\sum_{j=1}^N \phi_j = 1)$$

2. The distribution of input feature \mathbf{x} conditional on the output class y is modeled as **multivariate Gaussian distribution**

$$p(\mathbf{x}|y = j) = N(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_j|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right)$$

$\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$: the mean vector and covariance matrix for the j th class

Gaussian Discriminant Analysis

Parameter learning for GDA

- Using the training data $\mathbf{D} = \{(x_i, y_i); i = 1, \dots, m\}$, the parameter sets for GDA are:

$\boldsymbol{\phi} = \{\phi_1, \dots, \phi_N\}$: set of priors

$\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_N\}$: set of mean vectors

$\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_N\}$: set of covariance matrices

- The parameters are found as ones maximizing the log-likelihood of data

$$\begin{aligned}\log p(\mathbf{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\phi}) &= \log \prod_{i=1}^m p(x_i|y_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(y_i|\boldsymbol{\phi}_i) \\ &= \sum_{i=1}^m \log p(x_i|y_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(y_i|\boldsymbol{\phi}_i)\end{aligned}$$

The log-likelihood function is **concave function** in terms of the parameters

→ the optimum parameters are analytically derived as

$$\phi_j = \frac{1}{m} \sum_{i=1}^m 1\{y_i = j\}$$

$$\boldsymbol{\mu}_j = \frac{\sum_{i=1}^m 1\{y_i = j\} x_i}{\sum_{i=1}^m 1\{y_i = j\}}$$

$$\boldsymbol{\Sigma}_j = \frac{1}{\sum_{i=1}^m 1\{y_i = j\}} \sum_{i=1}^m 1\{y_i = j\} (x_i - \boldsymbol{\mu}_j)(x_i - \boldsymbol{\mu}_j)^T$$

Indication function

$$1\{y_i = j\} = \begin{cases} 1, & \text{if } y_i = j \\ 0, & \text{otherwise} \end{cases}$$

Class Prediction

- The probability of class $y = j$ given the new input x^{new} can be computed

$$\begin{aligned} P(y = j | x^{new}) &\sim P(x^{new} | y = j) P(y = j) & p(y | x^{new}) &= \frac{P(x^{new} | y) P(y)}{P(x^{new})} \\ &= \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} \exp\left(-\frac{1}{2} (x^{new} - \mu_j)^T \Sigma_j^{-1} (x^{new} - \mu_j)\right) \phi_j \end{aligned}$$

- The class can be selected using MAP estimation:

$$\hat{y} = \operatorname{argmax}_{y \in Y} p(y | x^{new})$$

- The boundary surface between two neighboring classes i and j (i. e., $P(y = i | x) = P(y = j | x)$)

$$\frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} \exp\left(-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right) \phi_i = \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} \exp\left(-\frac{1}{2} (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j)\right) \phi_j$$

$$\rightarrow x^T (\Sigma_i^{-1} - \Sigma_j^{-1}) x - 2(\mu_i^T \Sigma_i^{-1} - \mu_j^T \Sigma_j^{-1}) x + \mu_i^T \Sigma_i^{-1} \mu_i - \mu_j^T \Sigma_j^{-1} \mu_j + \log \frac{\phi_j |\Sigma_i|}{\phi_i |\Sigma_j|} = 0$$

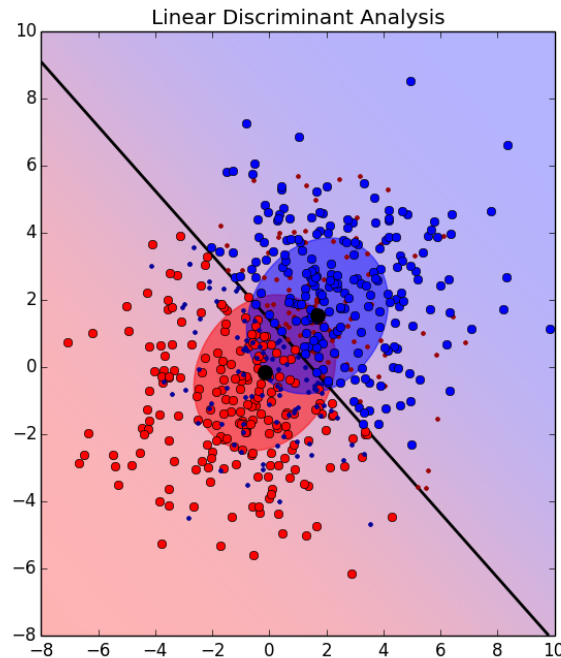
Gaussian Discriminant Analysis

Example (binary-classes)

The boundary surface between two neighboring classes i and j (i. e., $P(y = i|\mathbf{x}) = P(y = j|\mathbf{x})$)

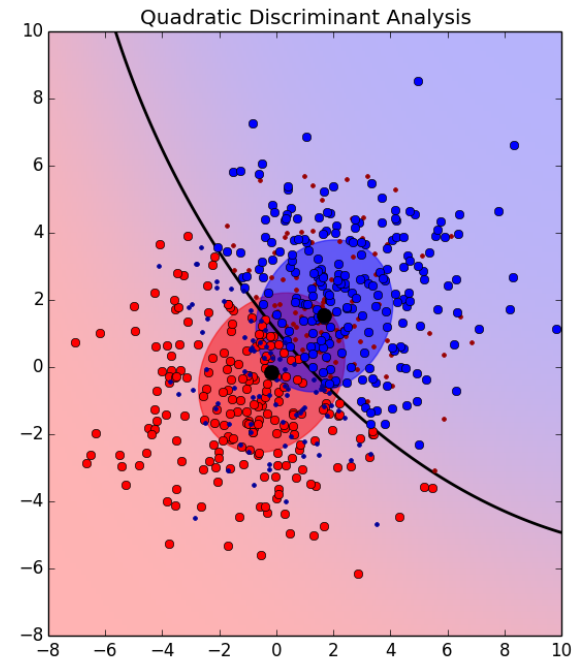
$$\mathbf{x}^T(\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\Sigma}_j^{-1})\mathbf{x} - 2(\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\mu}_j^T\boldsymbol{\Sigma}_j^{-1})\mathbf{x} + \boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \boldsymbol{\mu}_j^T\boldsymbol{\Sigma}_j^{-1}\boldsymbol{\mu}_j + \log\frac{\phi_j|\boldsymbol{\Sigma}_i|}{\phi_i|\boldsymbol{\Sigma}_j|} = 0$$

LDA vs QDA



Linear discriminant analysis

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} \text{ for all } i$$



Quadratic discriminant analysis

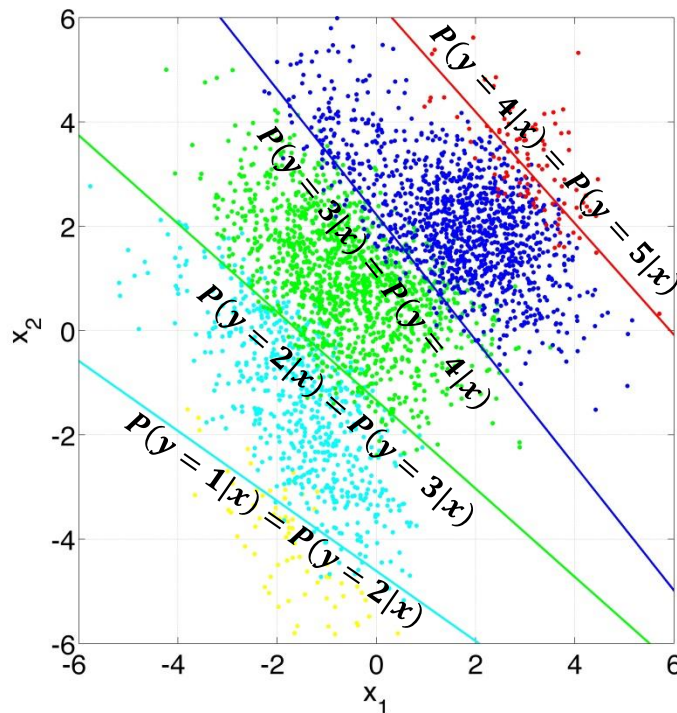
$$\boldsymbol{\Sigma}_i \text{ for each } i$$

Gaussian Discriminant Analysis

Example (multi-classes)

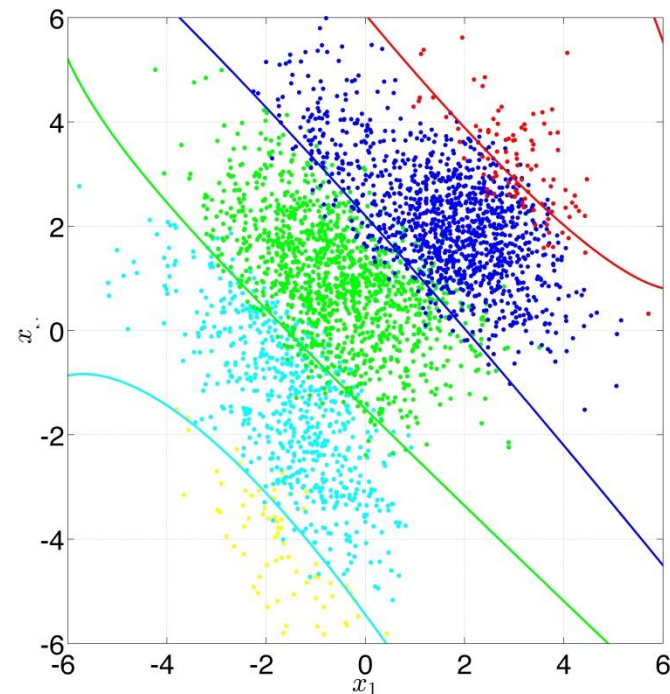
The boundary between the two neighboring classes i and j by setting $P(y = i|\mathbf{x}) = P(y = j|\mathbf{x})$, which yields

$$\mathbf{x}^T (\boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\Sigma}_j^{-1}) \mathbf{x} - 2(\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} - \boldsymbol{\mu}_j^T \boldsymbol{\Sigma}_j^{-1}) \mathbf{x} + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i - \boldsymbol{\mu}_j^T \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\mu}_j + \log \frac{|\boldsymbol{\Sigma}_i|}{|\boldsymbol{\Sigma}_j|} = 0$$



Linear discriminant analysis

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma} \text{ for all } i$$



Quadratic discriminant analysis

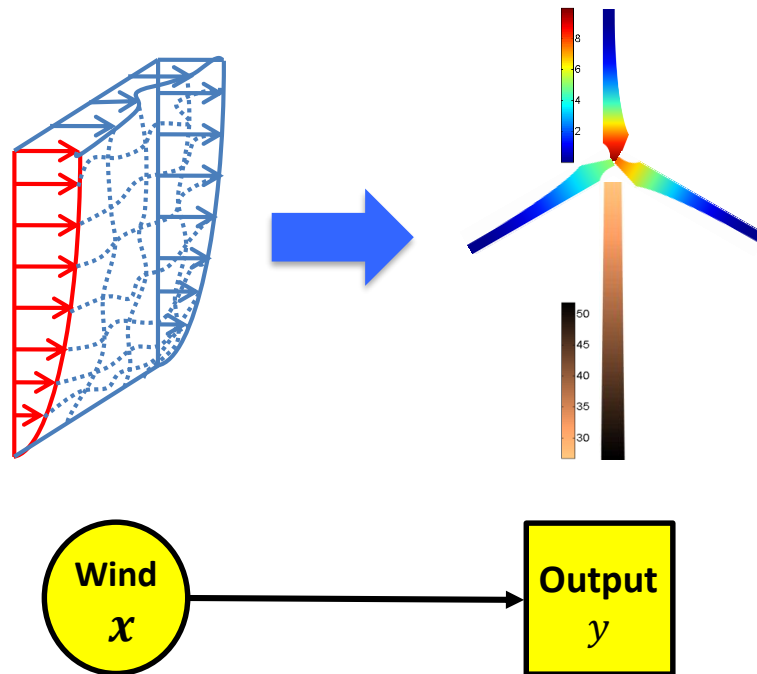
$$\boldsymbol{\Sigma}_i \text{ for each } i$$

Application to wind turbine monitoring data

Study how wind field characteristics affect wind turbine response class.

Wind field characteristics

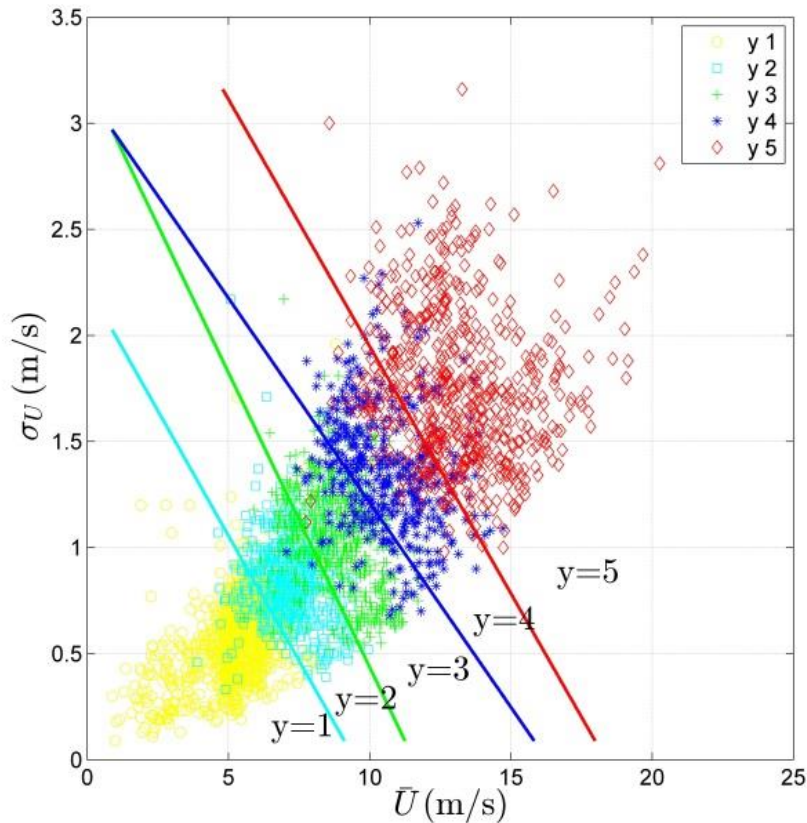
Wind turbine load



→ Construct the posterior probability mass function for the response class $p(y|x)$

Gaussian Discriminant Analysis

Application to wind turbine monitoring data



Classification boundaries between the i th and the j th class is determined as:

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

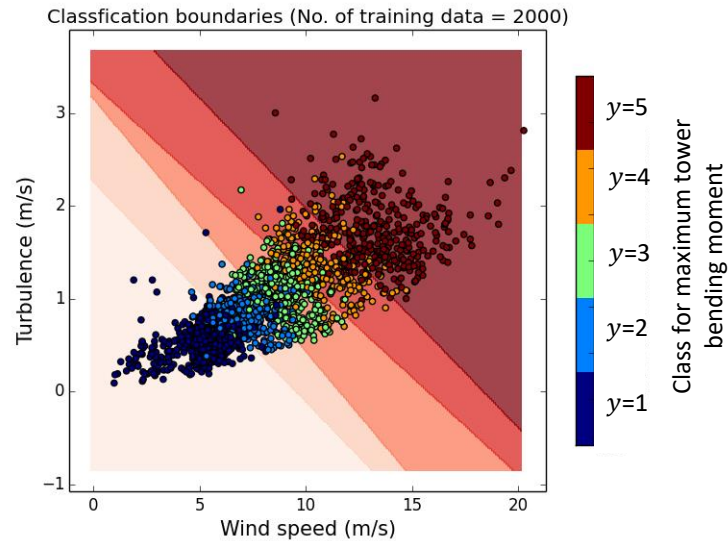
- Higher wind speed and higher turbulence tend to cause higher blade bending moment.
- Accuracy for the classification is approximately 80 %.
- Including more input features can increase the accuracy ratio.

Gaussian Discriminant Analysis

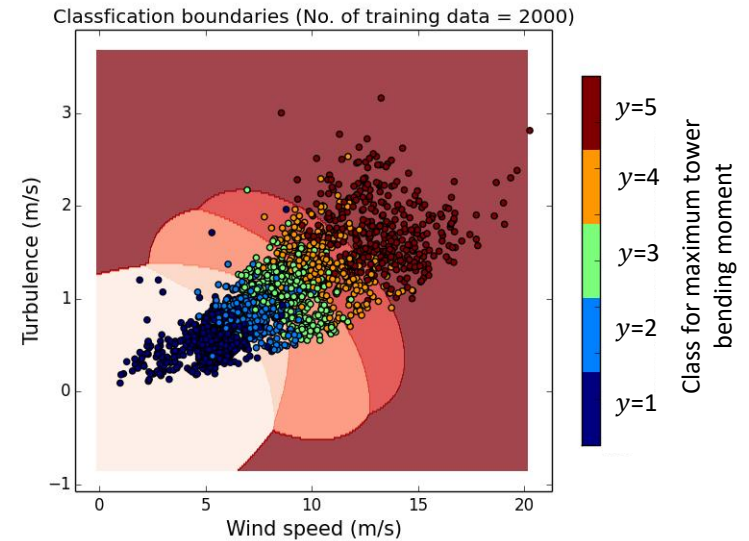
Application to wind turbine monitoring data

Linear discriminant analysis

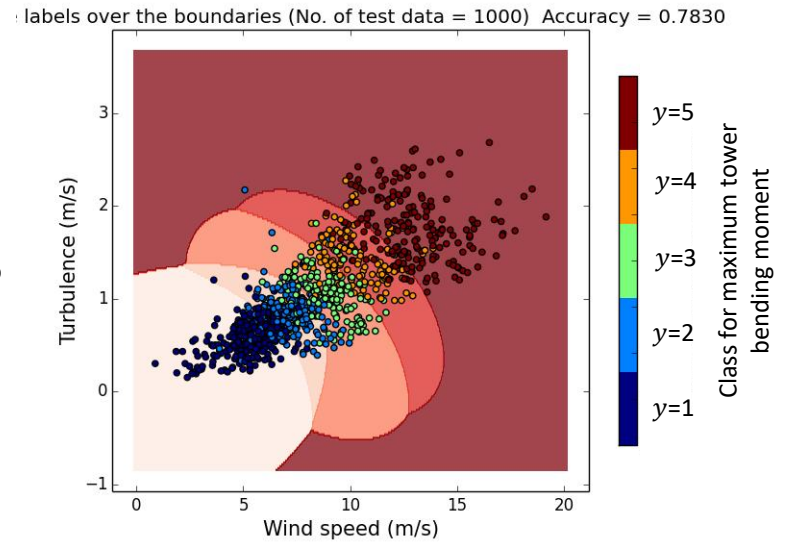
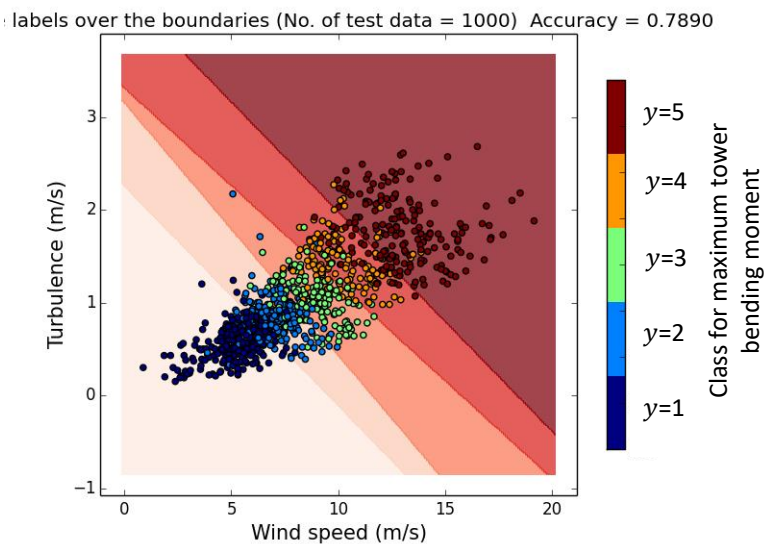
Training



Quadratic discriminant analysis



Testing



Detecting Spam e-mails

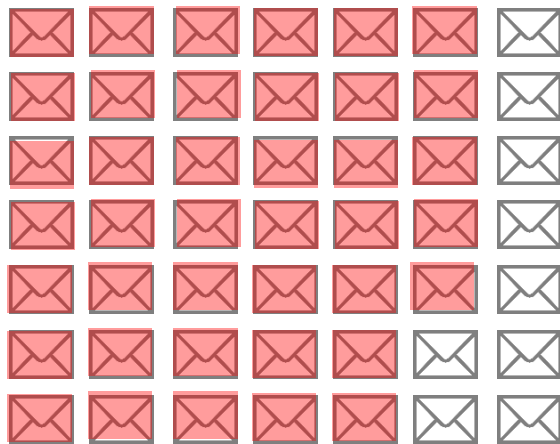
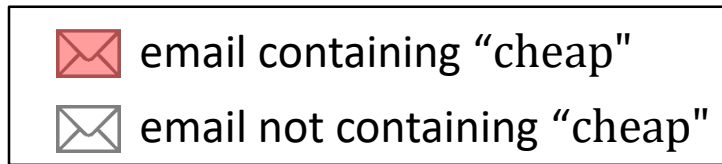


- Input: x = email message
- Output $y \in \{\text{Spam}, \text{non-spam}\}$
- Objective: Obtain a classifier f

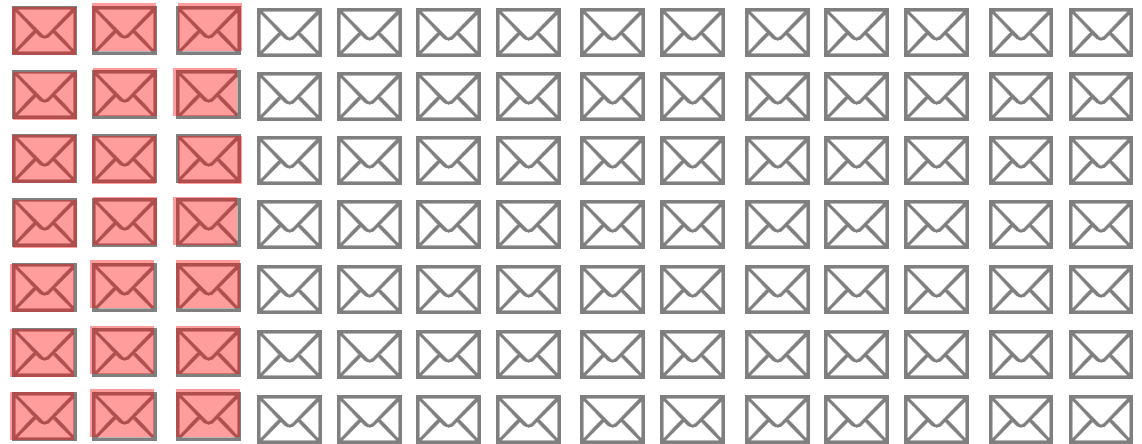


Naïve Bayes Classification

Data



Spam emails



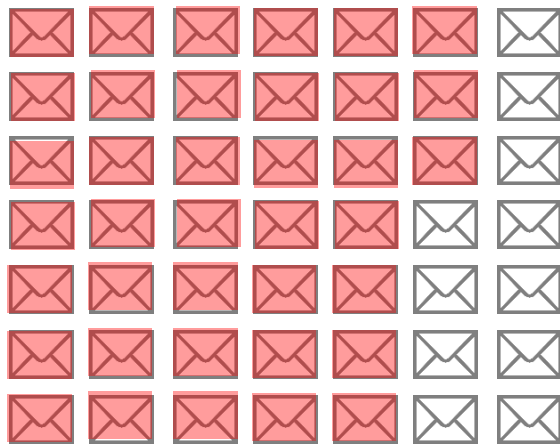
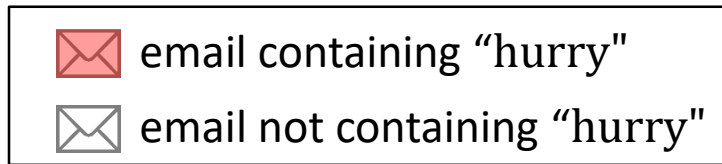
Non-spam emails

$$P(\text{cheap} | y = \text{spam}) = \frac{40}{49}$$

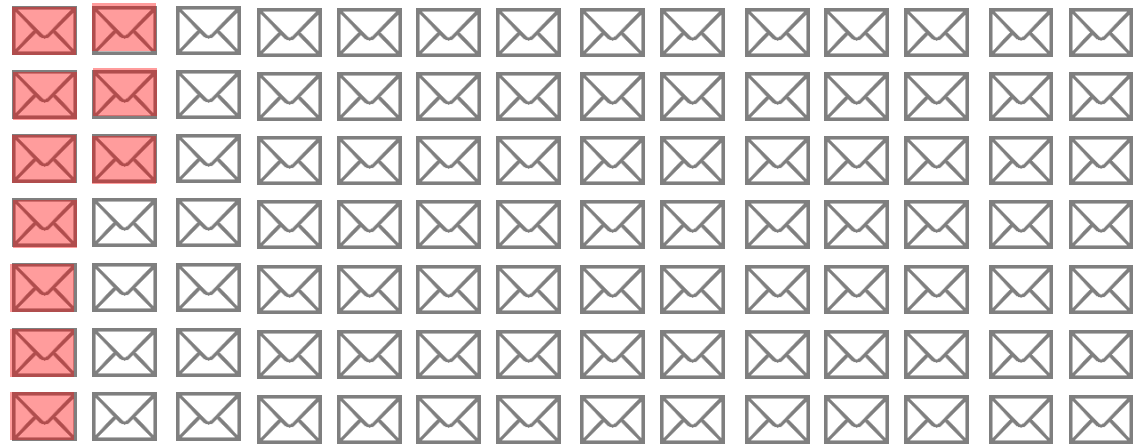
$$P(\text{cheap} | y = \text{nospam}) = \frac{21}{98}$$

Naïve Bayes Classification

Data



Spam emails



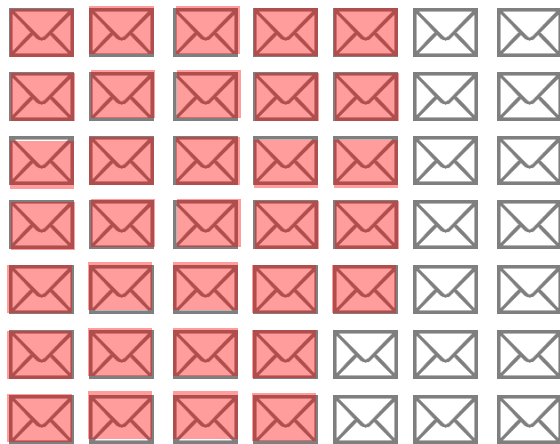
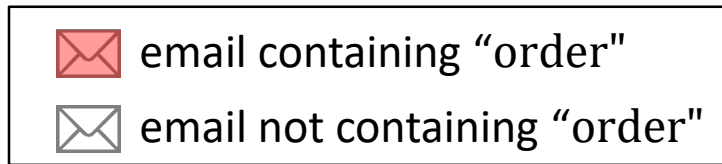
Non-spam emails

$$P(\text{"hurry"}|y = \text{spam}) = \frac{38}{49}$$

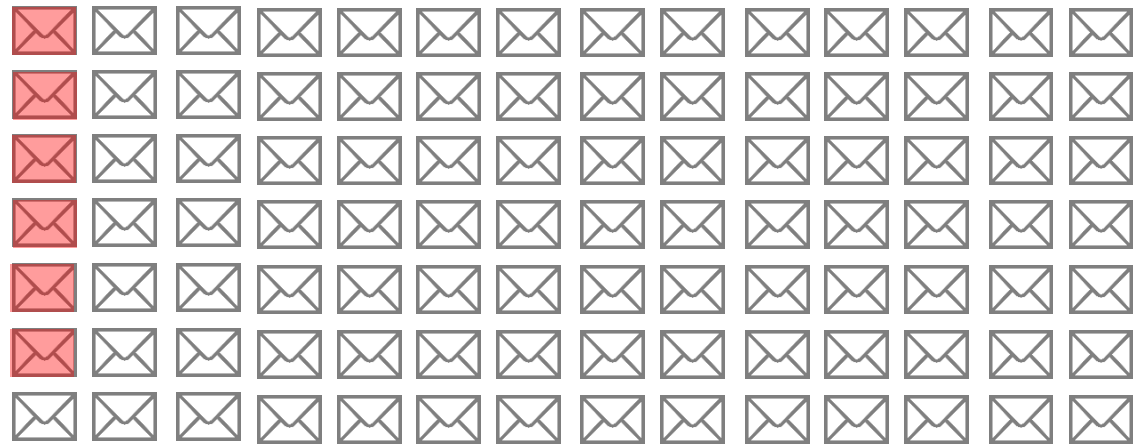
$$P(\text{"hurry"}|y = \text{nospam}) = \frac{10}{98}$$

Naïve Bayes Classification

Data



Spam emails



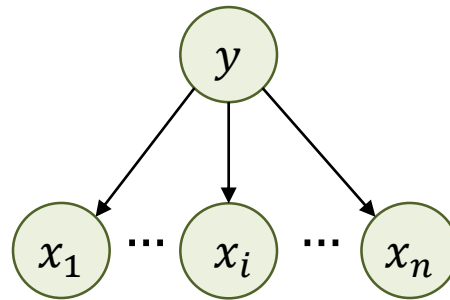
Non-spam emails

$$P(\text{"order"}|y = \text{spam}) = \frac{33}{49}$$

$$P(\text{"order"}|y = \text{nospam}) = \frac{6}{99}$$

Naïve Bayes Classification

- Now it's a time to build a model for spam classifier

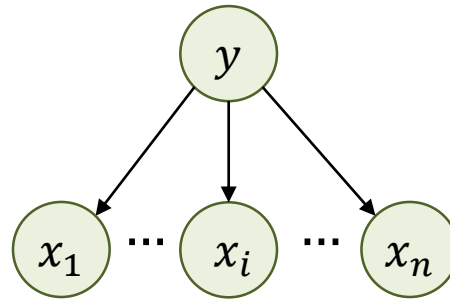


- Input $x = \{x_1, x_2, \dots, x_i, \dots, x_n\}$ $x_i = \begin{cases} 1 & \text{if } i\text{th word is in the email} \\ 0 & \text{otherwise} \end{cases}$
- Output $y = \begin{cases} 1 & \text{if Spam} \\ 0 & \text{otherwise} \end{cases}$
- $p(y)$ is a prior on class
- Naïve Bayes Model assumes x_i (attributes) are conditionally independent given y model. Thus, the likelihood is

$$P(x|y) = \prod_{i=1}^m P(x_i|y)$$

Naïve Bayes Classification

- Now it's a time to build a model for spam classifier



- Posterior :
$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y \in Y} P(x|y)P(y)}$$
- Class prediction :
$$\begin{aligned} \hat{y} &= \operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y \frac{P(x|y)P(y)}{\sum_y P(x|y)P(y)} \\ &= \operatorname{argmax}_y \prod_{i=1}^m P(x_i|y) P(y) \\ &= \operatorname{argmax}_y \prod_{i=1}^m P(x_i|y) P(y) \end{aligned}$$

Naïve Bayes Classification

- Training the model

$$D = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_m, y_m) \qquad x_i = (x_{i1}, \dots, x_{in})$$

- Parameterization

$$\phi_{j|y=1} = p(x_j = 1|y = 1) \qquad 1 - \phi_{j|y=1} = p(x_j = 0|y = 1)$$

$$\phi_{j|y=0} = p(x_j = 1|y = 0) \qquad 1 - \phi_{j|y=0} = p(x_j = 0|y = 0)$$

$$\phi_y = p(y = 1) \qquad 1 - \phi_y = p(y = 0)$$

- Cost function = log-likelihood

$$L(\phi_{j|y=1}, \phi_{j|y=0}, \phi_y) = \prod_{i=1}^m P(y_i, x_i | \phi) = \prod_{i=1}^m P(x_i | y_i, \phi) p(y_i | \phi) \prod_{i=1}^m \prod_{j=1}^n P(x_{ij} | y_i, \phi) p(y_i | \phi)$$

- Maximizing log likelihood with respect to the parameters leads

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x_{ij} = 1 \cap y_i = 1\}}{\sum_{i=1}^m 1\{y_i = 1\}} \qquad \phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x_{ij} = 1 \cap y_i = 0\}}{\sum_{i=1}^m 1\{y_i = 0\}} \qquad \phi_{y=0} = \frac{\sum_{i=1}^m 1\{y_i = 1\}}{m}$$

Naïve Bayes Classification

- Training the model

$$D = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_m, y_m) \qquad x_i = (x_{i1}, \dots, x_{in})$$

- Parameterization

$$\phi_{j|y=1} = p(x_j = 1|y = 1) \qquad 1 - \phi_{j|y=1} = p(x_j = 0|y = 1)$$

$$\phi_{j|y=0} = p(x_j = 1|y = 0) \qquad 1 - \phi_{j|y=0} = p(x_j = 0|y = 0)$$

$$\phi_y = p(y = 1) \qquad 1 - \phi_y = p(y = 0)$$

- Cost function = loglikelihood

$$\begin{aligned} L(\phi_{j|y=1}, \phi_{j|y=0}, \phi_y) &= \log \prod_{i=1}^m P(y_i, x_i | \phi) \\ &= \log \prod_{i=1}^m P(x_i | y_i, \phi) p(y_i | \phi) = \log \prod_{i=1}^m \prod_{j=1}^n P(x_{ij} | y_i, \phi) p(y_i | \phi) \end{aligned}$$

- Maximizing log likelihood with respect to the parameters leads

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x_{ij} = 1 \cap y_i = 1\}}{\sum_{i=1}^m 1\{y_i = 1\}} \qquad \phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x_{ij} = 1 \cap y_i = 0\}}{\sum_{i=1}^m 1\{y_i = 0\}} \qquad \phi_{y=0} = \frac{\sum_{i=1}^m 1\{y_i = 1\}}{m}$$

Naïve Bayes Classification

- Example

$$P(\text{cheap}|y = \text{spam}) = \frac{40}{49}$$

$$P(\text{cheap}|y = \text{nospam}) = \frac{21}{98}$$

$$P(\text{"hurry"}|y = \text{spam}) = \frac{38}{49}$$

$$P(\text{"hurry"}|y = \text{nospam}) = \frac{10}{98}$$

$$P(\text{"order"}|y = \text{spam}) = \frac{33}{49}$$

$$P(\text{"order"}|y = \text{nospam}) = \frac{6}{98}$$

$x = (\text{if cheap, if hurry, if order})$

$p(y = \text{spam}) = p(y = \text{non-spam}) = 0.5$

- The new email has been arrived with $x = (1, 1, 0)$

$$p\{y = 1|x = (1, 1, 0)\} \propto P\{x = (1, 1, 0) | y = 1\}P(y = 1) = \frac{40}{49} \frac{38}{49} \left(1 - \frac{33}{49}\right) = 0.206$$

$$p\{y = 0|x = (1, 1, 0)\} \propto P\{x = (1, 1, 0) | y = 0\}P(y = 0) = \frac{21}{49} \frac{10}{49} \left(1 - \frac{6}{49}\right) = 0.077$$

$$p\{y = 0|x = (1, 1, 0)\} = \frac{0.206}{0.206 + 0.077} = 0.730, \quad p\{y = 1|x = (1, 1, 0)\} = 0.270$$

Unsupervised Learning

K-means algorithm



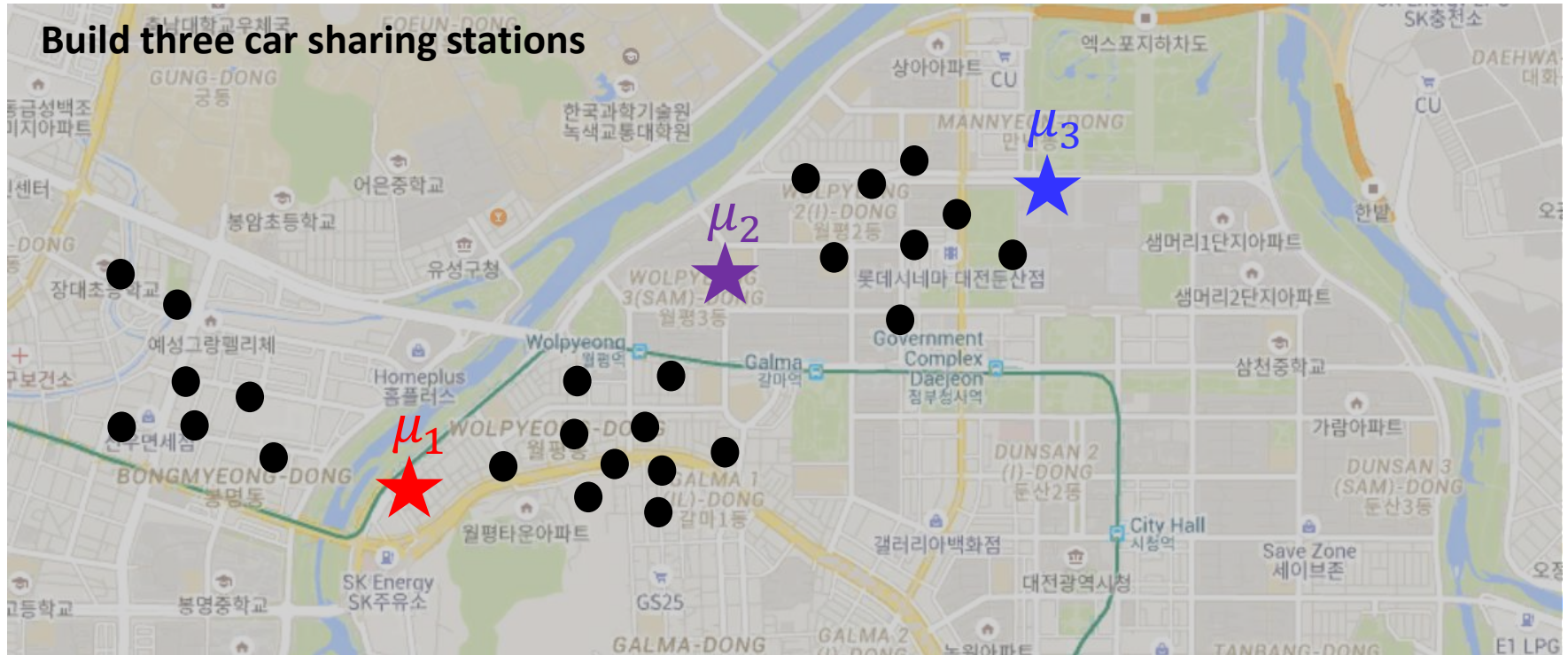
Potential demands for car sharing service

Build three car sharing stations



K-means algorithm

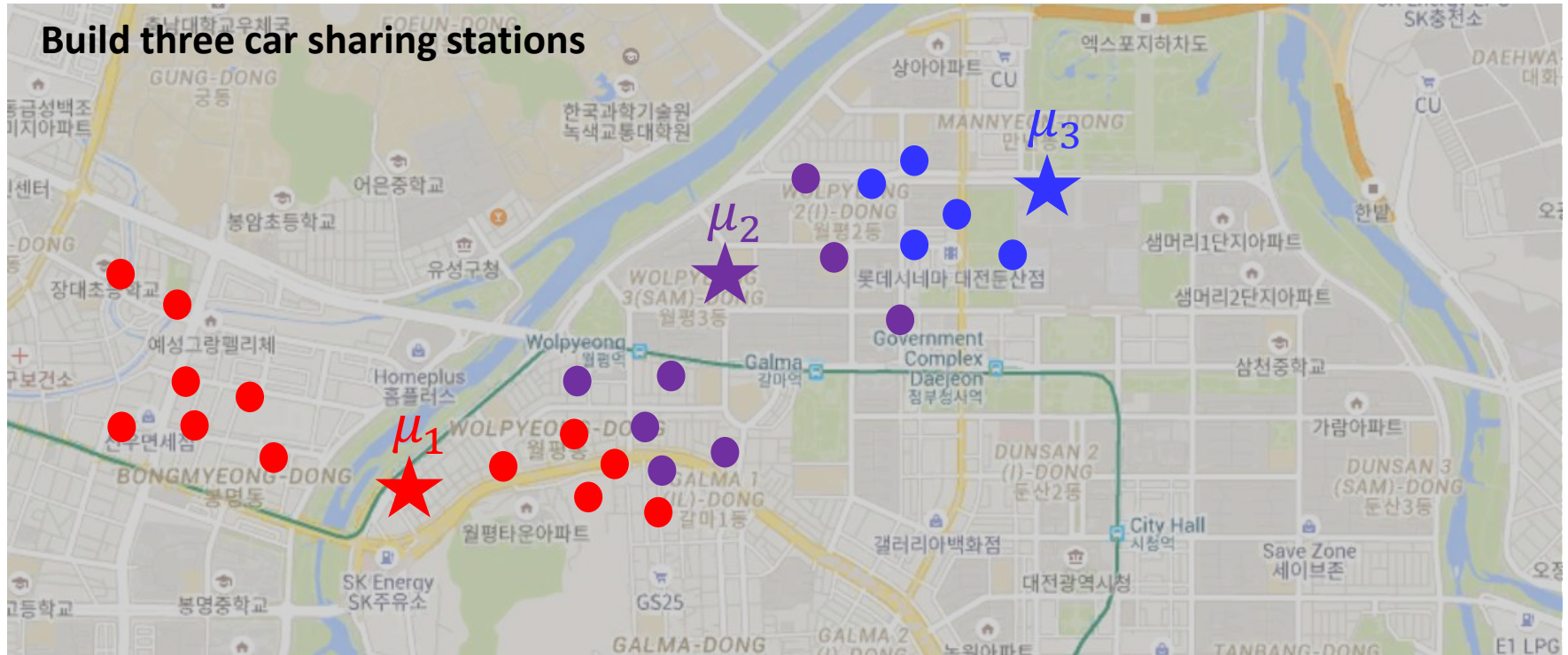
Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

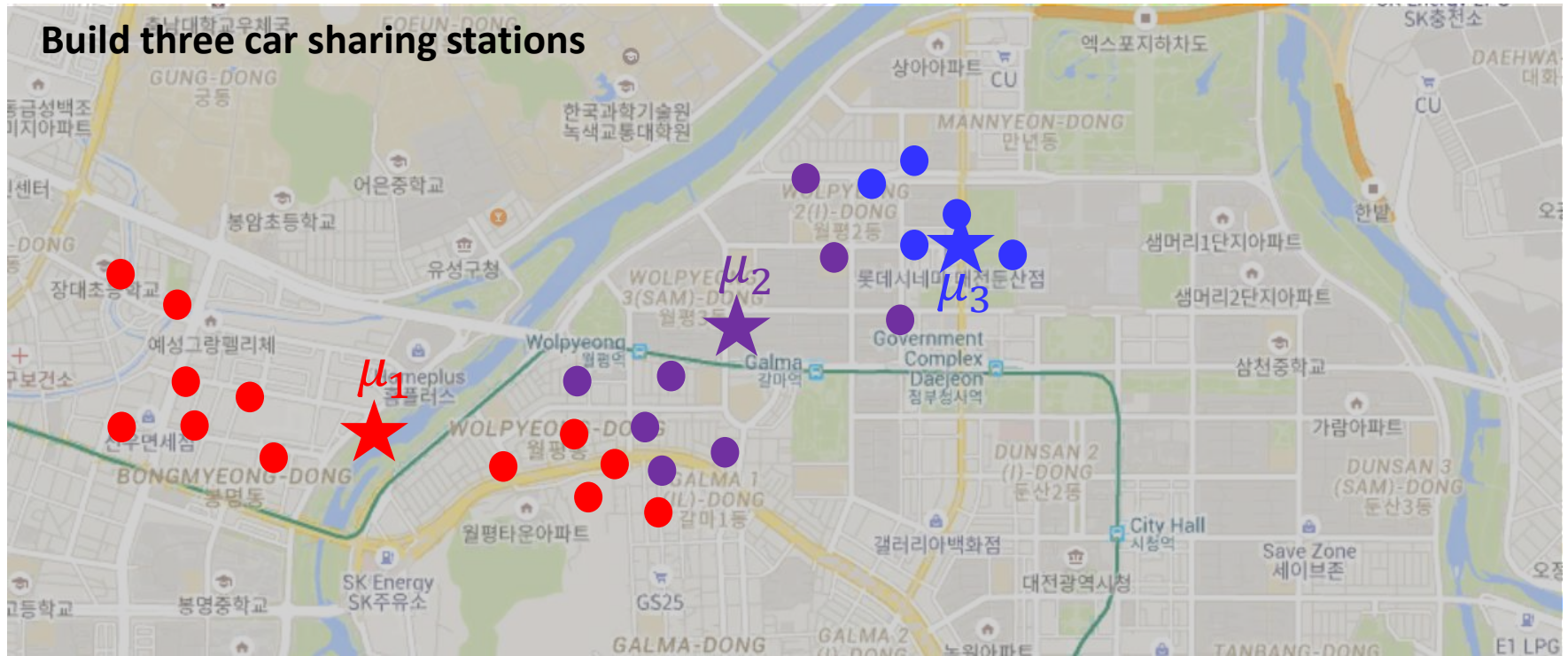
2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

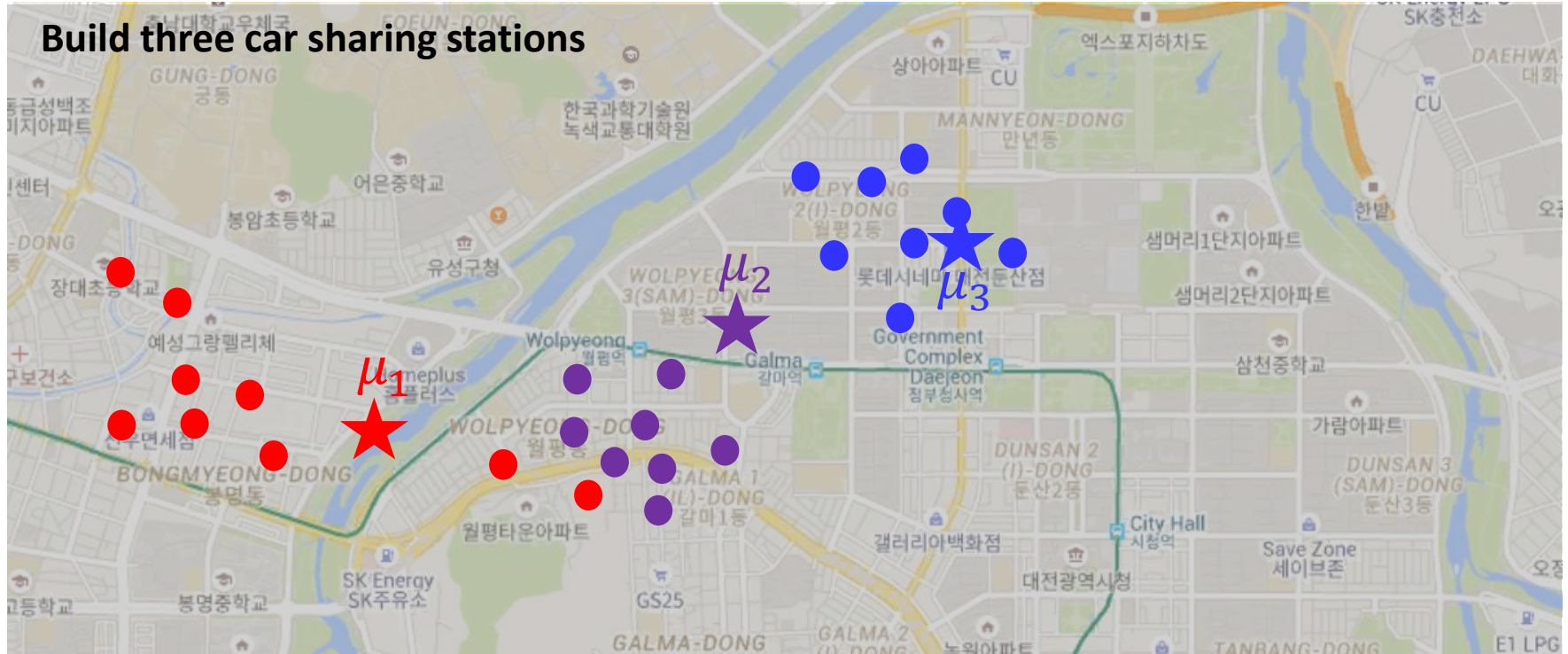
For every j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

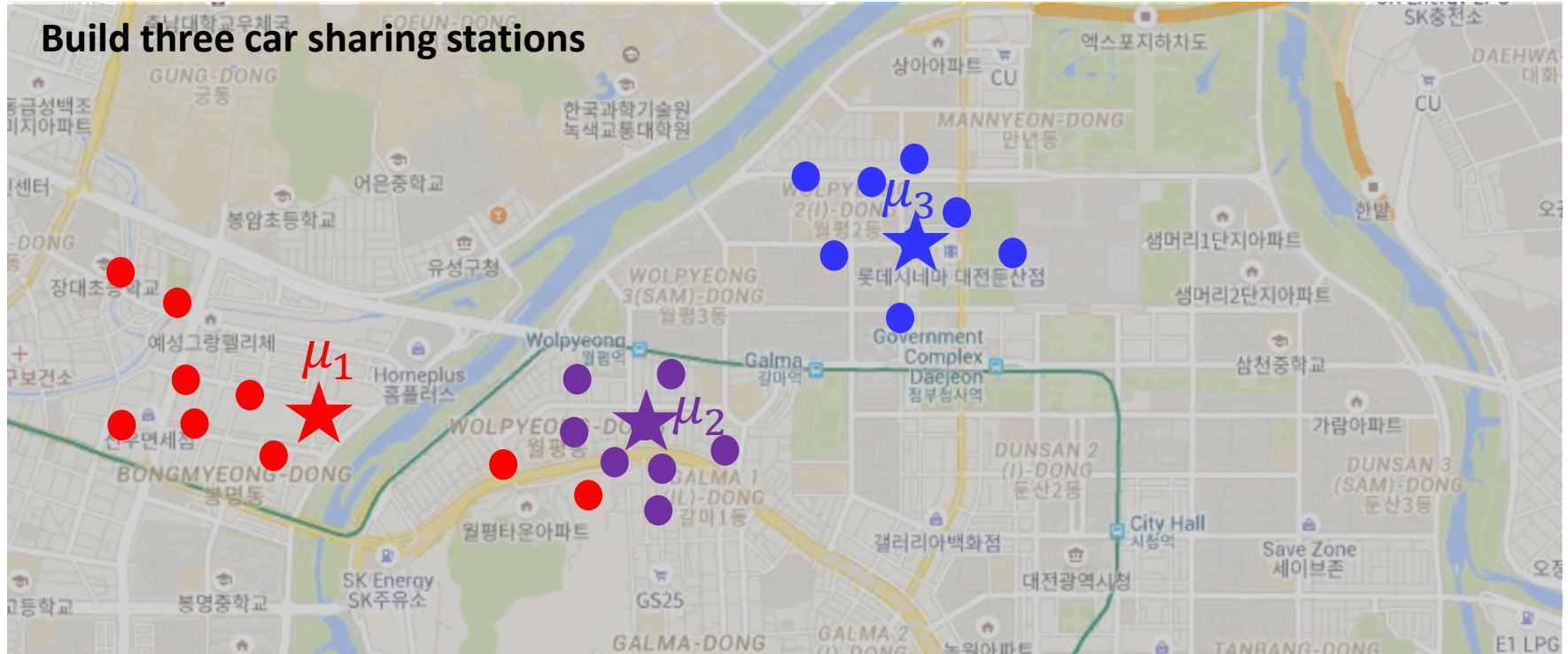
For every j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

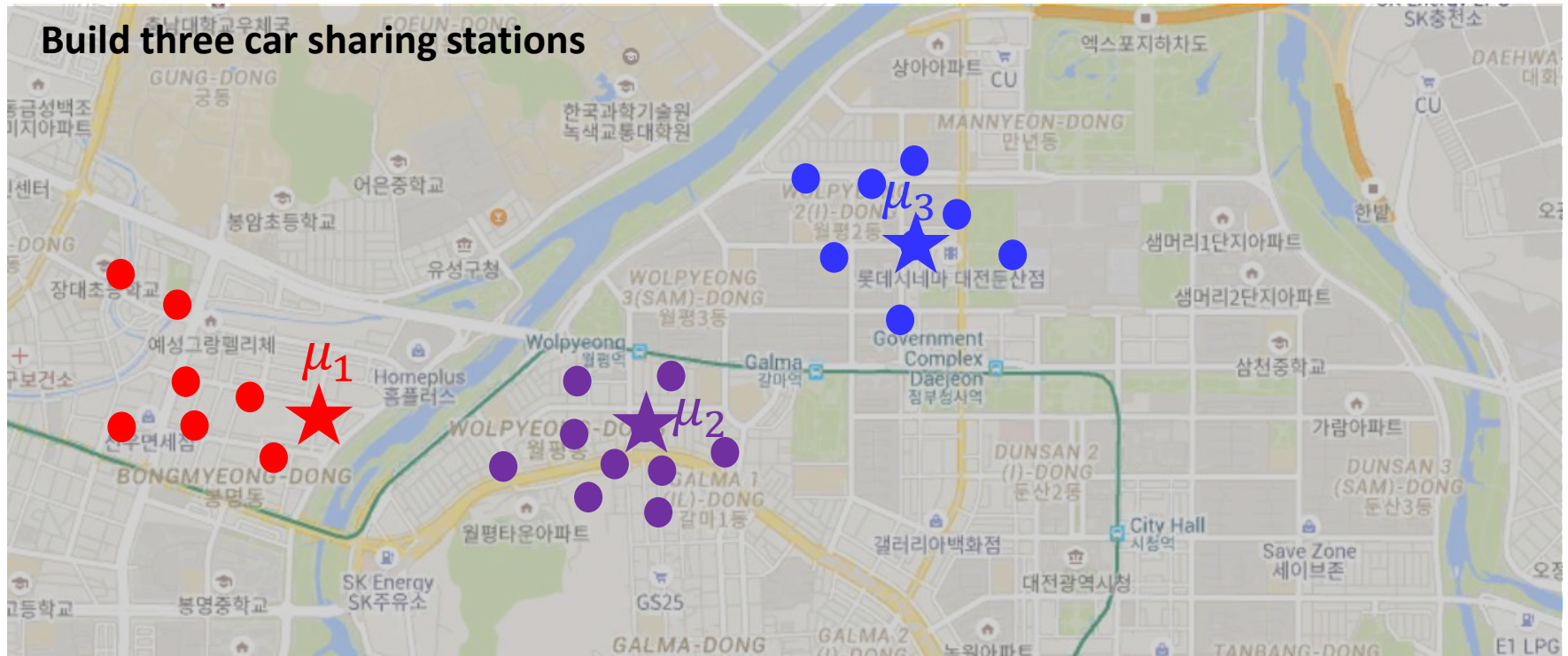
For every j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

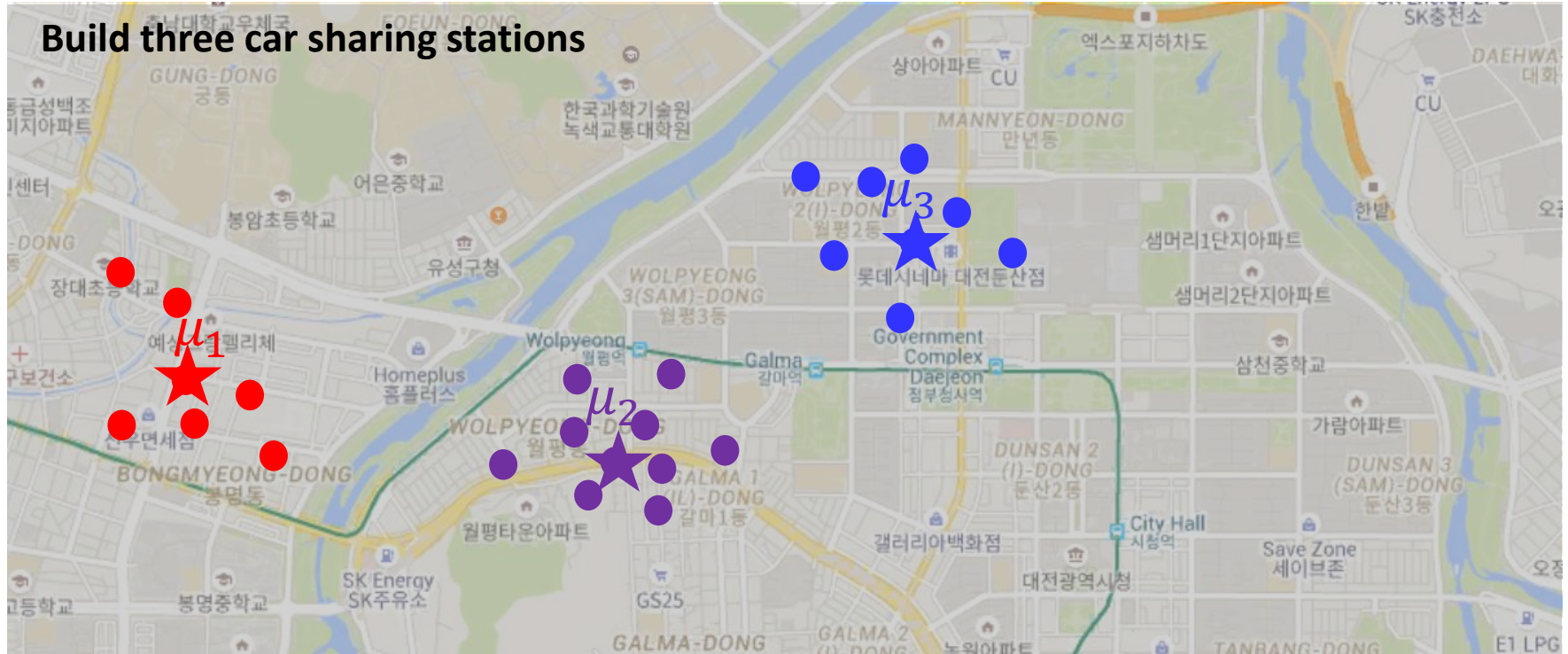
For every j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

K-means algorithm

Build three car sharing stations



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly

2. Repeat until convergence: {

For every i , set

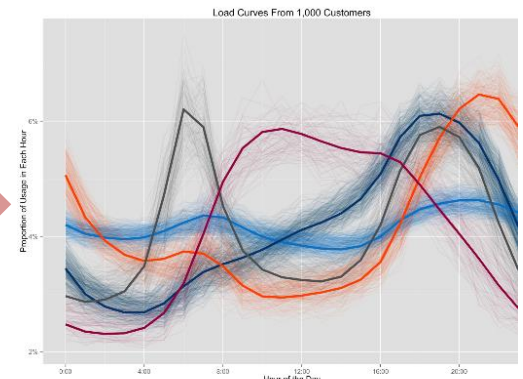
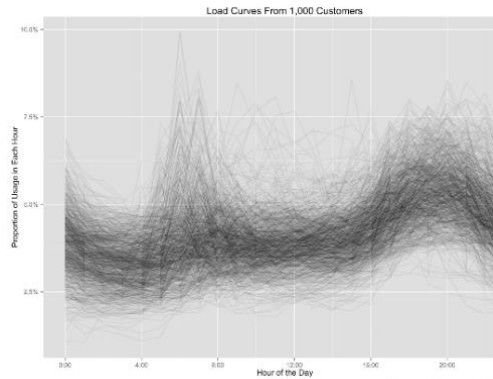
$$c^{(i)} := \underset{j}{\operatorname{argmin}} \|x^{(i)} - \mu_j\|^2$$

For every j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}$$

}

K-means algorithm : applications

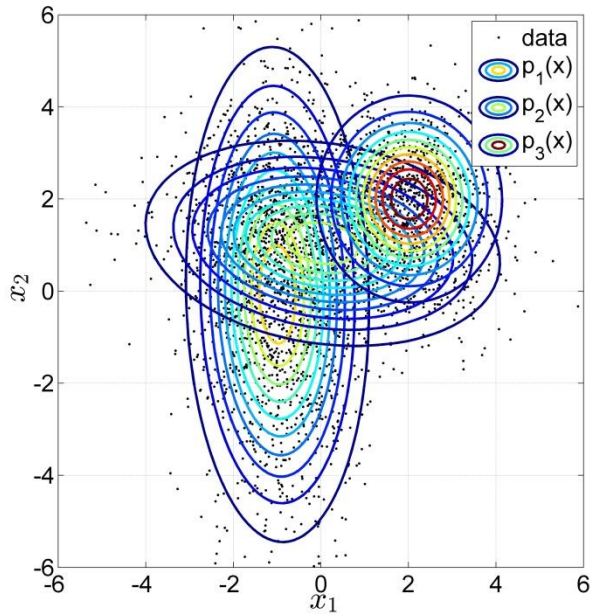


Acquire energy consumption data

Cluster users depending on their energy consumption patterns

Gaussian Mixture Models: a density estimation method

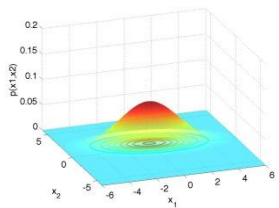
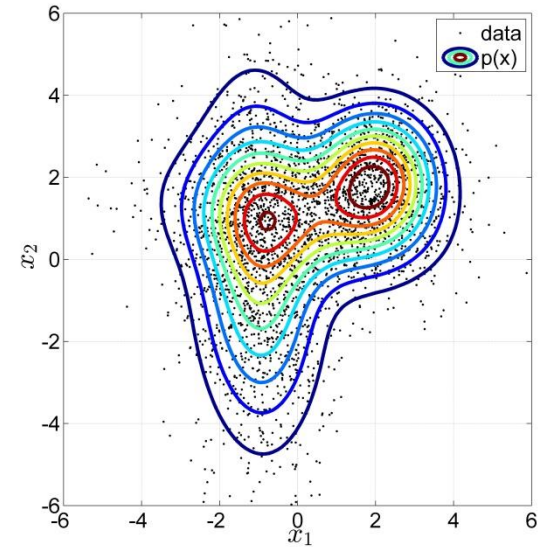
Modeling a **probability density** as a combination of K Gaussian components



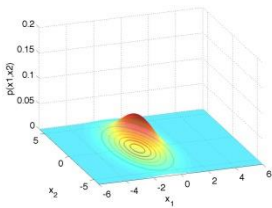
$$p(x) = \sum_{k=1}^K p_k(x) \varphi_k$$



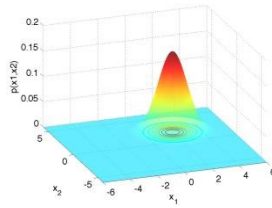
Weighted sum of
Gaussian PDFs



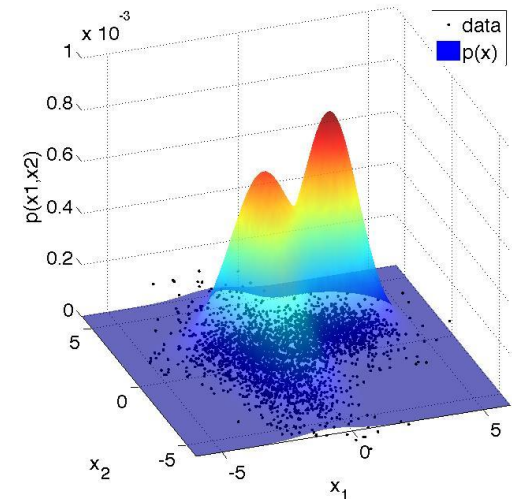
$p_1(x)$



$p_2(x)$



$p_3(x)$



Gaussian Mixture Models

- A probability density for input $\mathbf{x} = (x_1, x_2, \dots, x_n)$, is modeled as a weighed sum of K Gaussian distribution

$$p(\mathbf{x}; \boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{k=1}^K p_k(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \varphi_k$$

- the k th component density is of a form of Gaussian

$$p_k(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = N(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

- φ_k : (mixture) weight for k th Gaussian component ($\sum_{k=1}^K \varphi_k = 1$)
- $\boldsymbol{\mu}_k$: mean vector for the k th Gaussian component
- $\boldsymbol{\Sigma}_k$: covariance matrix for the k th PDF

Gaussian Mixture Models

- The parameters, $\boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$, for GMM

K : number of GPDFs

$\boldsymbol{\varphi} = \{\varphi_1, \dots, \varphi_K\}$: set of weights

$\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K\}$: set of mean vectors

$\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K\}$: set of covariance matrices

are found as ones maximizing the log-likelihood of data

$$l(\boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^m \log p(\mathbf{x}^{(i)}; \boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^m \log \sum_{z^{(i)}=1}^K p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z^{(i)}; \boldsymbol{\varphi})$$

- Due to the latent variable $z^{(i)}$ representing the Gaussian PDF from which the data $\mathbf{x}^{(i)}$ is drawn, the log likelihood is not explicitly defined → **Difficult to optimize the GMM parameters $\boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$.**

Gaussian Mixture Models

$$\begin{aligned}
 l(\boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \sum_{i=1}^m \log p(\mathbf{x}^{(i)}; \boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\
 &= \sum_{i=1}^m \log \sum_{z^{(i)}=1}^K p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z^{(i)}; \boldsymbol{\varphi}) \\
 &= \sum_{i=1}^m \log \sum_{z^{(i)}=1}^K Q_i(z^{(i)}) \frac{p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z^{(i)}; \boldsymbol{\varphi})}{Q_i(z^{(i)})} \\
 &\geq \sum_{i=1}^m \sum_{z^{(i)}=1}^K Q_i(z^{(i)}) \log \frac{p(\mathbf{x}^{(i)} | z^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z^{(i)}; \boldsymbol{\varphi})}{Q_i(z^{(i)})}
 \end{aligned}$$

Jensen's inequality:
 $f(E[X]) \geq E[f(X)]$ if f is concave

Expected Maximization (EM) algorithm (Ref: Dempster, et.al., 1977)

Repeat until convergence {

E-Step: for each i , set

$$Q_i(z^{(i)}) = p(z^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\varphi})$$

← Estimated parameters in the previous step

(soft estimation of $z^{(i)}$)

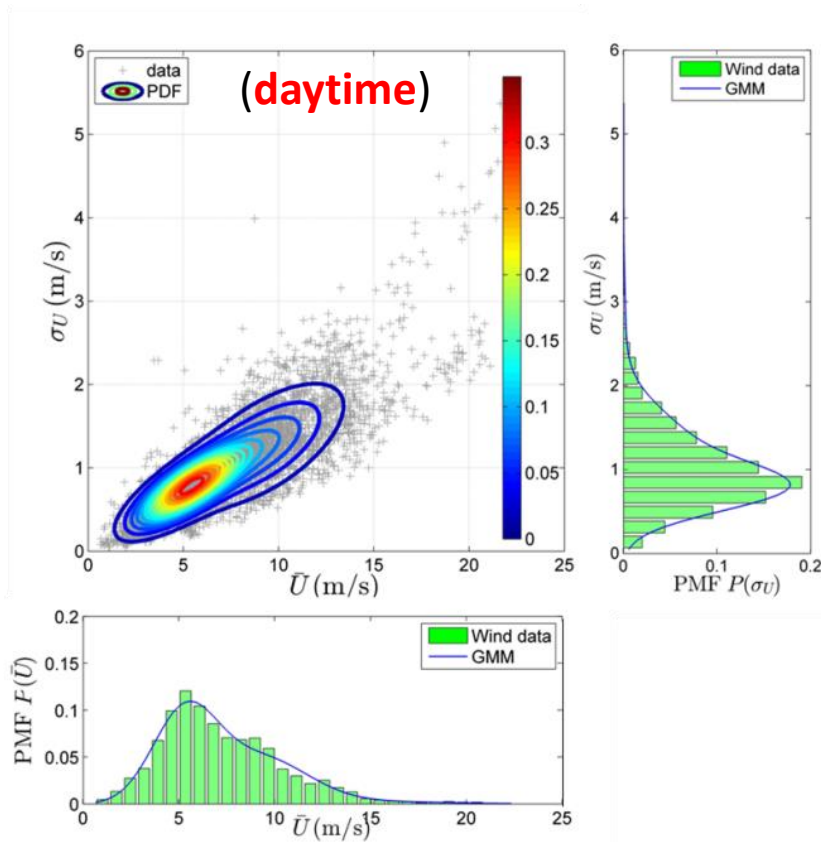
M-Step: maximize the following the log-likelihood with respect to $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\varphi}$

$$\sum_{i=1}^m \sum_{j=1}^K Q_i(z^{(i)} = j) \log \frac{p(\mathbf{x}^{(i)} | z^{(i)} = j; \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z^{(i)} = j; \boldsymbol{\varphi})}{Q_i(z^{(i)} = j)}$$

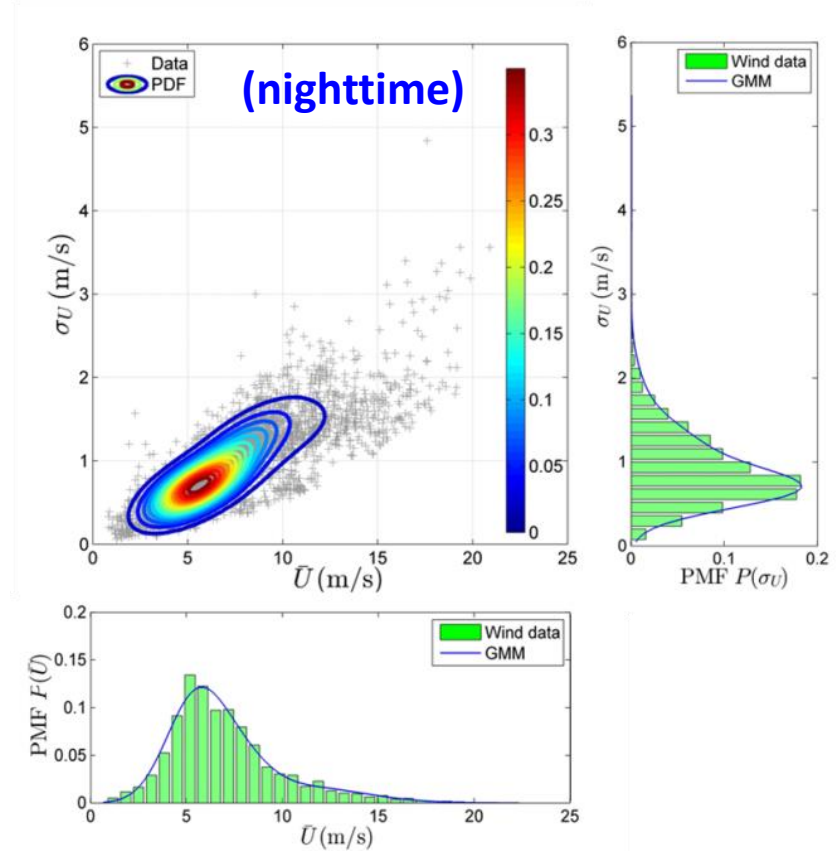
}

 Concave function → can be easily maximized

Application to wind monitoring data



$$p(x|t = \text{daytime})$$



$$P(x|t = \text{nighttime})$$

- The wind field characteristics are represented 2-dimensional PDF.
- The differences between the daytime and nighttime wind fields can be studied by comparing the two PDFs.