Machine Learning 1:
Linear Models

유환조, POSTECH

http://di.postech.ac.kr/hwanjoyu

Linear classification

Application: Spam Classification

- Input: $\mathbf{x} = \mathit{email\ message}$

> From: hwanjoyu@postech.ac.kr
> Date: September 24, 2022
> Subject: CSE342 announcement
>
> Hello students,
> I've attached the answers to homework 1...

> From: a9k62n@hotmail.com
> Date: September 24, 2022
> Subject: URGENT
>
> Dear Sir or maDam:
> my friend left sum of 10m dollars...

- Output: $y \in \{\mathit{spam}, \mathit{not\_spam}\}$

- Objective: obtain a predictor f

$$\mathbf{x} \longrightarrow \boxed{f} \longrightarrow y$$

Types of prediction tasks

- Binary classification (e.g., email => spam / not spam):

$$\mathbf{x} \longrightarrow \boxed{f} \longrightarrow y \in \{+1, -1\}$$

- Regression (e.g., location, year => housing price):

$$\mathbf{x} \longrightarrow \boxed{f} \longrightarrow y \in \mathbb{R}$$

Types of prediction tasks

- Multiclass classification: y is a category

 $\longrightarrow \boxed{f} \longrightarrow$ cat

- Ranking: y is an ordering

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \longrightarrow \boxed{f} \longrightarrow \mathbf{x}_3 \succ \mathbf{x}_2 \succ \mathbf{x}_4 \succ \mathbf{x}_1$$

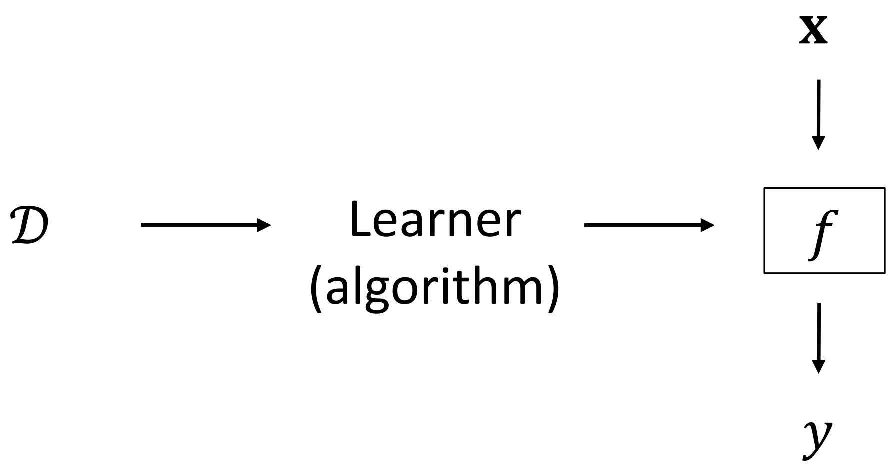
Data (supervised learning)

- An ***example*** is an input-output pair $(\mathbf{x}, y)$, which specifies that y is the **label** (ground-truth output) for $\mathbf{x}$.

- Training data: a ***set*** of examples $\mathcal{D}$

$\mathcal{D}$ = {
("… CSE441 students …", -1),
("… 10m dollars …", +1),
…
}

ML framework



Feature extraction

- Example task: predict y, whether a string $\mathbf{x}$ is an email address
- Question: what properties of $\mathbf{x}$ might be relevant for predicting y ?
- Feature extractor: Given input $\mathbf{x}$, output a set of (feature name, feature value) pairs.

"abc@gmail.com" $\xrightarrow{\text{feature extractor}}$

Length<10	: 0
fracOfAlphabet	: 0.85
Contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

Feature vector

- Mathematically, feature vector doesn't need feature names:

Length<10	: 0
fracOfAlphabet	: 0.85
Contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

$$\longrightarrow \begin{bmatrix} 0 \\ 0.85 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Definition: feature vector

- For an input $\mathbf{x}$, its feature vector is:

$$\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \dots, \phi_d(\mathbf{x})]$$

- Think of $\phi(\mathbf{x}) \in \mathbb{R}^d$ as a point in a d-dimensional space.

Weight vector

- Weight vector $\mathbf{w} \in \mathbb{R}^d$: for each feature j, have real numbers w_j representing contribution of feature to prediction

Length<10	: -1.2
fracOfAlphabet	: 0.6
Contains_@	: 3
endsWith_.com	: 2.2
endsWith_.org	: 1.4

- Weight vector is automatically computed from D_{train} by a learner (algorithm).

Linear predictor

Weight vector $\mathbf{w} \in \mathbb{R}^d$	
Length<10	: -1.2
fracOfAlphabet	: 0.6
Contains_@	: 3
endsWith_.com	: 2.2
endsWith_.org	: 1.4

Feature vector $\phi(\mathbf{x}) \in \mathbb{R}^d$	
Length<10	: 0
fracOfAlphabet	: 0.85
Contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

- Score: weighted combination of features

$$\mathbf{w} \cdot \phi(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$$

Example: $-1.2(0) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 5.71$

Linear (binary) classifier

Weight vector $\mathbf{w} \in \mathbb{R}^d$

Feature vector $\phi(\mathbf{x}) \in \mathbb{R}^d$

For binary classification (two-class classification):

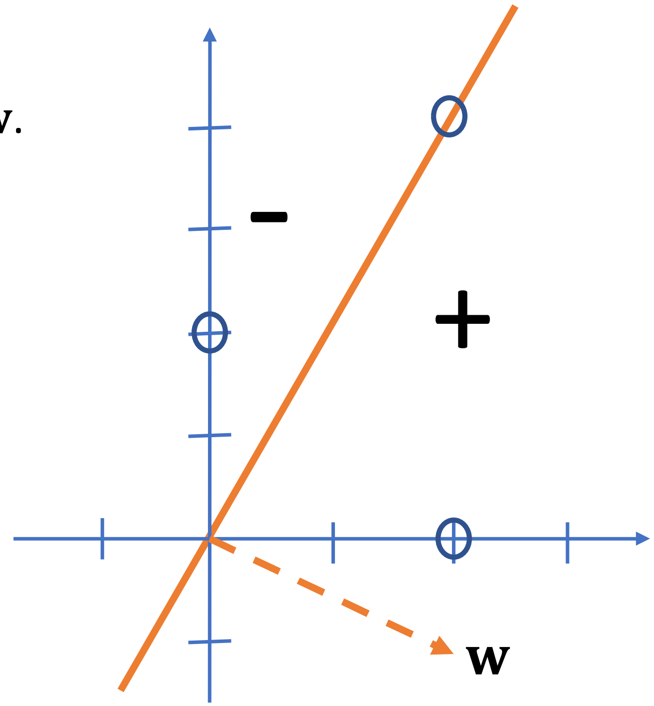
Definition: linear (binary) classifier

$$f_{\mathbf{w}}(x) = \operatorname{sign}\big(\mathbf{w} \cdot \phi(\mathbf{x})\big) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(\mathbf{x}) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(\mathbf{x}) < 0 \end{cases}$$

Linear classifier: geometric intuition

$$f_{\mathbf{w}}(x) = \mathrm{sign}\big(\mathbf{w}\cdot\phi(\mathbf{x})\big) = \begin{cases} +1 & \text{if } \mathbf{w}\cdot\phi(\mathbf{x}) > 0 \\ -1 & \text{if } \mathbf{w}\cdot\phi(\mathbf{x}) < 0 \end{cases}$$

- A binary classifier $f_{\mathbf{w}}$ defines a hyperplane with normal vector $\mathbf{w}$.
- Example:
  - $\mathbf{w} = [2, -1]$
  - $\phi(\mathbf{x}) \in \{[2,0], [0,2], [2,4]\}$



- $\mathbb{R}^2$ => line
- $\mathbb{R}^3$ => plane
- $\mathbb{R}^{4 \text{ or higher}}$ => hyperplane

Score and margin

Correct label: y

Predicted label: $\hat{y} = f_{\mathbf{w}}(x) = \text{sign}\big(\mathbf{w} \cdot \phi(x)\big)$

Definition: score

- The **score** on an example (x, y) is $\mathbf{w} \cdot \phi(x)$, how **confident** we are in predicting +1.

Definition: margin

- The **margin** on an example (x, y) is $\big(\mathbf{w} \cdot \phi(x)\big)y$, how **correct** we are.
- Geometrically, if $||\mathbf{w}|| = 1$, the margin of an input x is exactly the distance from its feature vector $\phi(x)$ to the decision boundary.

Loss function for linear classifier

Loss function:

- Design a loss function $l(\mathbf{w}, \mathbf{x}, y)$ that quantifies

  "how much loss we would get if we use $\mathbf{w}$ to make prediction on $\mathbf{x}$ when the correct output is y".

Loss function for linear (binary) classifier:

- We want to find a $\mathbf{w}$ such that

$$\begin{cases} \mathbf{w} \cdot \phi(\mathbf{x}) > 0 & \text{if } y = +1 \\ \mathbf{w} \cdot \phi(\mathbf{x}) < 0 & \text{if } y = -1 \end{cases}, \qquad \forall (\mathbf{x}, y) \in \mathcal{D}$$

- which is identical to

$$\big(\mathbf{w} \cdot \phi(\mathbf{x})\big) y > 0, \qquad \forall (\mathbf{x}, y) \in \mathcal{D}$$

- Thus, loss function:

$$l(\mathbf{w}, \mathbf{x}, y) =$$

Loss minimization

Loss function:

$$l(\mathbf{w}, \mathbf{x}, y) = \max\{-(\mathbf{w} \cdot \phi(\mathbf{x}))y, 0\}$$

[draw graph where x-axis is margin and y-axis is loss]

Loss function $\mathcal{J}(\mathbf{w})$ on training data $\mathcal{D}$:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} l(\mathbf{w}, \mathbf{x}, y)$$

Loss minimization

- Find $\mathbf{w}$ that minimizes $\mathcal{J}(\mathbf{w})$:

$$\mathbf{w}^* = arg \min_{\mathbf{w}} \mathcal{J}(\mathbf{w})$$

Gradient Descent

$$\mathcal{J}(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \max\{-(\mathbf{w} \cdot \phi(\mathbf{x}))y, 0\}$$

$$\mathbf{w}^* = arg\,\min_{\mathbf{w}} \mathcal{J}(\mathbf{w})$$

- How to find $\mathbf{w}^*$? => Gradient Descent

Gradient Descent: compute $\mathbf{w}$ where $\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} = 0$

- Initialize $\mathbf{w}$ randomly.
- Repeat until convergence
  - Compute the **gradient** of $\mathcal{J}(\mathbf{w})$, i.e., the vector increasing $\mathcal{J}(\mathbf{w})$ the most.
  - Move $\mathbf{w}$ to the opposite direction of the gradient.

Gradient of $\mathcal{J}(\mathbf{w})$:

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} =$$

Gradient Descent

Gradient of $\mathcal{J}(\mathbf{w})$:

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} -\phi(\mathbf{x})y\big[\big(\mathbf{w}\cdot\phi(\mathbf{x})\big)y < 0\big]$$

Gradient Descent:

- Initialize $\mathbf{w}$ randomly.
- Repeat until convergence:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}}$$

- *Step size* (or *learning rate*) α is a hyperparameter.

Gradient Descent (GD) is slow!

Gradient Descent:

- Initialize $\mathbf{w}$ randomly.
- Repeat until convergence:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} -\phi(\mathbf{x})y\big[\big(\mathbf{w}\cdot\phi(\mathbf{x})\big)y < 0\big]$$

Computing gradient requires reading all the training data!

Stochastic Gradient Descent (SGD)

Gradient Descent:

- Initialize $\mathbf{w}$ randomly.
- Repeat until convergence:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} -\phi(\mathbf{x})y\big[\big(\mathbf{w} \cdot \phi(\mathbf{x})\big)y < 0\big]$$

Stochastic Gradient Descent (SGD):

- Initialize $\mathbf{w}$ randomly.
- Repeat for each $\mathcal{D}$ (**epoch**):
  - Iterate for each batch $\boldsymbol{\mathcal{B}}$ $(\subset \mathcal{D})$ (**iteration**):

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{1}{|\boldsymbol{\mathcal{B}}|} \sum_{(\mathbf{x},y)\in\boldsymbol{\mathcal{B}}} -\phi(\mathbf{x})y\big[\big(\mathbf{w} \cdot \phi(\mathbf{x})\big)y < 0\big]$$

- (Mini-)batch $\boldsymbol{\mathcal{B}}$ is a random sample from $\mathcal{D}$, and (mini-)batch size $|\boldsymbol{\mathcal{B}}|$ is a hyperparameter.

SGD and Perceptron algorithm

Stochastic Gradient Descent (SGD):

- Initialize $\mathbf{w}$ randomly.
- Repeat for each $\mathcal{D}$ (**epoch**):
  - Iterate for each batch $\boldsymbol{\mathcal{B}}$ $(\subset \mathcal{D})$ (**iteration**):

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{1}{|\boldsymbol{\mathcal{B}}|} \sum_{(\mathbf{x},y)\in\boldsymbol{\mathcal{B}}} -\phi(\mathbf{x})y\left[\left(\mathbf{w}\cdot\phi(\mathbf{x})\right)y < 0\right]$$

- (Mini-)batch $\boldsymbol{\mathcal{B}}$ is a random sample from $\mathcal{D}$, and (mini-)batch size $|\boldsymbol{\mathcal{B}}|$ is a hyperparameter.

Perceptron algorithm ($|\boldsymbol{\mathcal{B}}| = 1$):

- Initialize $\mathbf{w}$ randomly.
- Repeat for each $\mathcal{D}$:
  - Iterate for each $(\mathbf{x}, y)$:
    - If $\left(\mathbf{w}\cdot\phi(\mathbf{x})\right)y < 0$ (misclassified), then

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} + \alpha\phi(\mathbf{x})y$$

Hinge loss (for linear classifier)

Hinge loss function:

$$l(\mathbf{w}, \mathbf{x}, y) = \max\{1 - (\mathbf{w} \cdot \phi(\mathbf{x}))y, 0\}$$

[draw graph where x-axis is margin and y-axis is loss]

- Intuition: increase loss if margin is less than 1.
- SVM (Support vector machine) uses hinge loss with L2 regularization.

Gradient of hinge loss $l(\mathbf{w}, \mathbf{x}, y)$:

$$\frac{\partial l(\mathbf{w}, \mathbf{x}, y)}{\partial \mathbf{w}} = -\phi(\mathbf{x})y[(\mathbf{w} \cdot \phi(\mathbf{x}))y < 1]$$

Linear regression

Regression: problem setup

Given a set of N labeled examples, $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ ($\mathbf{x}_n \in X \subset \mathbb{R}^d$ and $y_n \in Y \subset \mathbb{R}$), the goal is to learn a mapping

$$f(\mathbf{x}): X \rightarrow Y,$$

which associates $\mathbf{x}$ with y, such that we can make prediction about y^*, when a new input $\mathbf{x}^* \notin \mathcal{D}$ is provided.

- $\mathbf{x}$: input, independent variable, predictor, regressor, covariate
- y: output, dependent variable, response

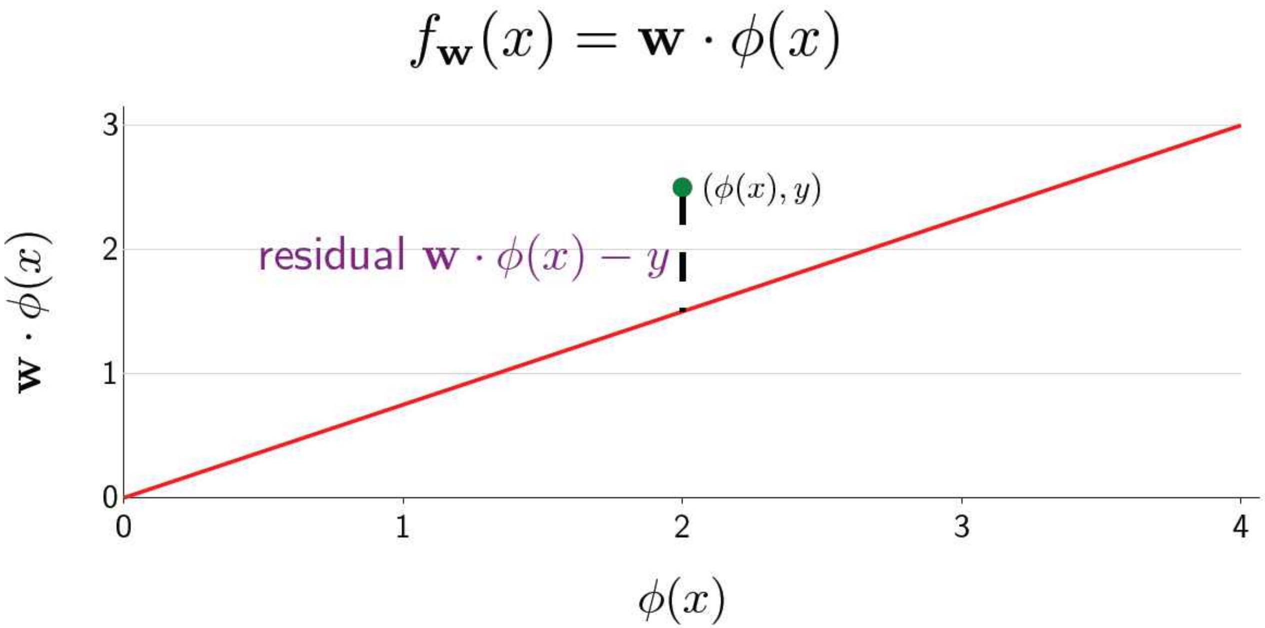
Linear regression

Linear regression:

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathrm{x}) = \mathbf{w} \cdot \phi(x) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x}) + w_0 \phi_0(\mathbf{x})$$

- $\mathbf{w} = [w_0, w_1, \dots, w_M]^{\mathrm{T}} \in \mathbb{R}^{d+1}$ (w_i: weight, learning parameter)
- $\phi(\mathrm{x}) = [\phi_0(\mathbf{x})(= 1), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})]^{\mathrm{T}} \in \mathbb{R}^{d+1}$ ($\phi_i(\mathbf{x})$: basis function, feature)

Regression loss



Definition: residual

- The **residual** is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots the target y.

Regression loss

Definition: squared loss (L_2 loss)

$$\text{loss}_{\text{squared}}(\mathbf{w}, \mathbf{x}, y) = (\underbrace{f_{\mathbf{w}}(x) - y}_{\text{residual}})^2$$

Loss function $\mathcal{J}(\mathbf{w})$ on training data $\mathcal{D}$:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} (\mathbf{w} \cdot \phi(x) - y)^2$$

Gradient of $\mathcal{J}(\mathbf{w})$:

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} 2(\underbrace{\mathbf{w} \cdot \phi(x) - y}_{\text{residual}})\phi(x) = \begin{bmatrix} \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} 2(\mathbf{w} \cdot \phi(x) - y)\phi_0(x) \\ \cdots \\ \cdots \\ \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} 2(\mathbf{w} \cdot \phi(x) - y)\phi_d(x) \end{bmatrix}$$

SGD for linear regression

Stochastic Gradient Descent (SGD):

- Initialize $\mathbf{w}$ randomly.
- Repeat for each $\mathcal{D}$ (**epoch**):
    - Iterate for each batch $\boldsymbol{\mathcal{B}}$ $(\subset \mathcal{D})$ (**iteration**):

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \alpha \frac{1}{|\boldsymbol{\mathcal{B}}|} \sum_{(\mathbf{x},y)\in\boldsymbol{\mathcal{B}}} (\underbrace{\mathbf{w} \cdot \phi(x) - y}_{\text{residual}})\phi(x)$$

- (Mini-)batch $\boldsymbol{\mathcal{B}}$ is a random sample from $\mathcal{D}$, and (mini-)batch size $|\boldsymbol{\mathcal{B}}|$ is a hyperparameter.

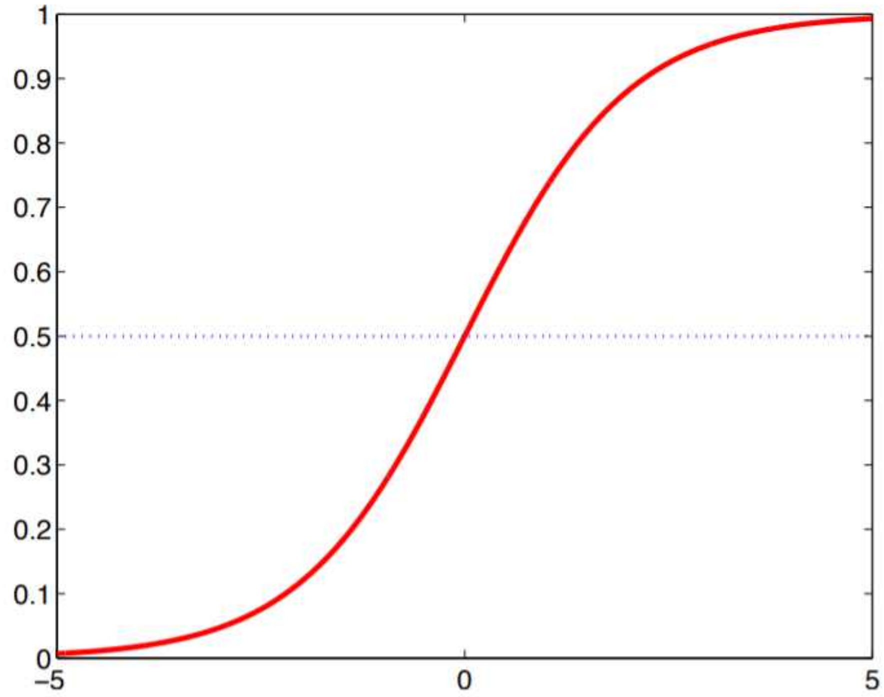
Logistic Regression

Logistic function (or sigmoid function)

Logistic function (or sigmoid function) $\sigma(\xi)$:

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}.$$

- $\sigma(\xi) \to 0$ as $\xi \to -\infty$.
- $\sigma(\xi) \to 1$ as $\xi \to \infty$.
- $\sigma(-\xi) = 1 - \sigma(\xi)$.
- $\frac{d}{d\xi}[\sigma(\xi)] = \sigma(\xi)\sigma(-\xi) = \sigma(\xi)(1 - \sigma(\xi))$.



Logistic regression

Bernoulli distribution:

- A random variable $y = 1$ with probability p and $y = 0$ with probability $1 - p$.
- E.g., The likelihood $p(Y = \{1,0,1,0,0\}) = p(1-p)p(1-p)(1-p) = p^2(1-p)^3$

Logistic regression:

- Predicts a binary output $y_n \in \{0,1\}$ from an input $\mathbf{x}_n$
- Models the input-output by a conditional Bernoulli distribution:

$$\mathbb{E}[y_n|\mathbf{x}_n] = p(y_n = 1|\mathbf{x}_n) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n),$$

- using sigmoid function

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

Logistic regression: MLE (Maximum Likelihood Estimation)

Given $\{(\mathbf{x}_n, y_n) | n = 1, \dots, N\}$, the likelihood is given by

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n = 1|\mathbf{x}_n)^{y_n} \big(1 - p(y_n = 1|\mathbf{x}_n)\big)^{1-y_n} = \prod_{n=1}^{N} \sigma\big(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n\big)^{y_n} \Big(1 - \sigma\big(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n\big)\Big)^{1-y_n}$$

Then, log-likelihood function is given by

$$\mathcal{L} = \sum_{n=1}^{N} \log p(y_n|\mathbf{x}_n) = \sum_{n=1}^{N} \{y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)\},$$

where $\hat{y}_n = \sigma\big(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n\big)$.

(Binary) cross entropy loss

- For binary classification where $p \in \{y, 1-y\}, q \in \{\hat{y}, 1-\hat{y}\}$, **cross entropy loss** is:

$$\mathcal{J} = \sum_{n=1}^{N} [-y_n \log \hat{y}_n - (1-y_n)\log(1-\hat{y}_n)].$$

- Note, the cross entropy loss $\mathcal{J}$ is equal to the negative log-likelihood $-\mathcal{L}(\mathbf{w})$.

Gradient Descent/Ascent

- The gradient descent/ascent learning is a first-order iterative method for minimization/maximization.
- Gradient descent: iterative minimization

$$\mathbf{w}^{(\boldsymbol{k}+\mathbf{1})} = \mathbf{w}^{(\boldsymbol{k})} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} \right).$$

- Gradient ascent: iterative maximization

$$\mathbf{w}^{(\boldsymbol{k}+\mathbf{1})} = \mathbf{w}^{(\boldsymbol{k})} + \alpha \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}} \right).$$

- Learning rate (or step size): $\alpha > 0$