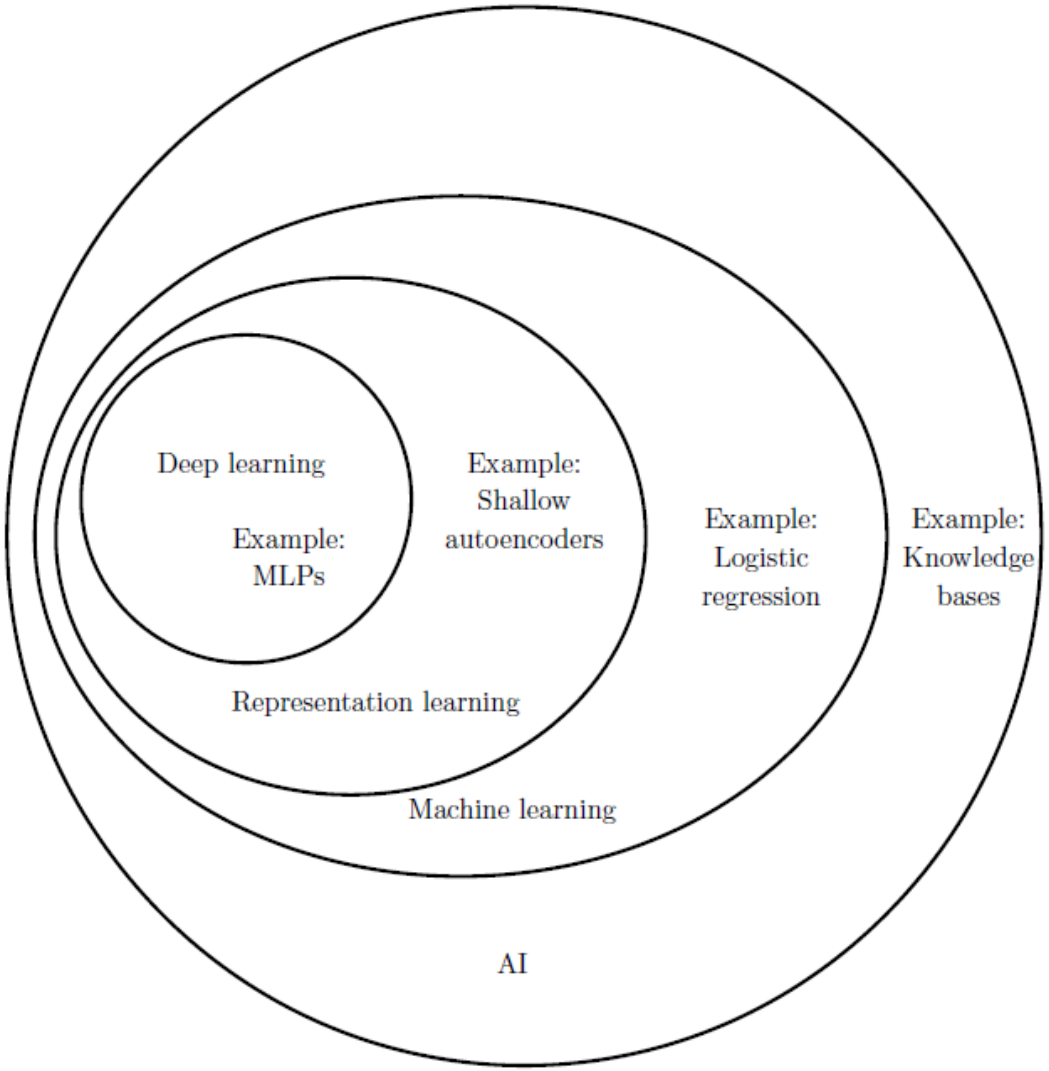
Machine Learning Basics

ESM5205 Learning from Big Data | Sep 11, 2019

Seokho Kang

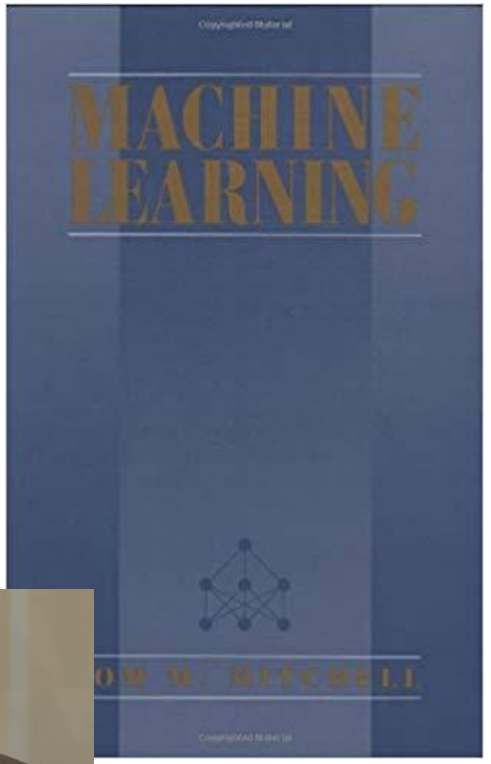
성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Review: AI, Machine Learning, and Deep Learning



Machine Learning

- **Tom Mitchell (Professor at CMU)'s definition of "learning"**
  - A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.



- A "**machine learning algorithm**" is an algorithm that is able to **learn from data**
  - Machine Learning = Learning from **E** in form of **Data**

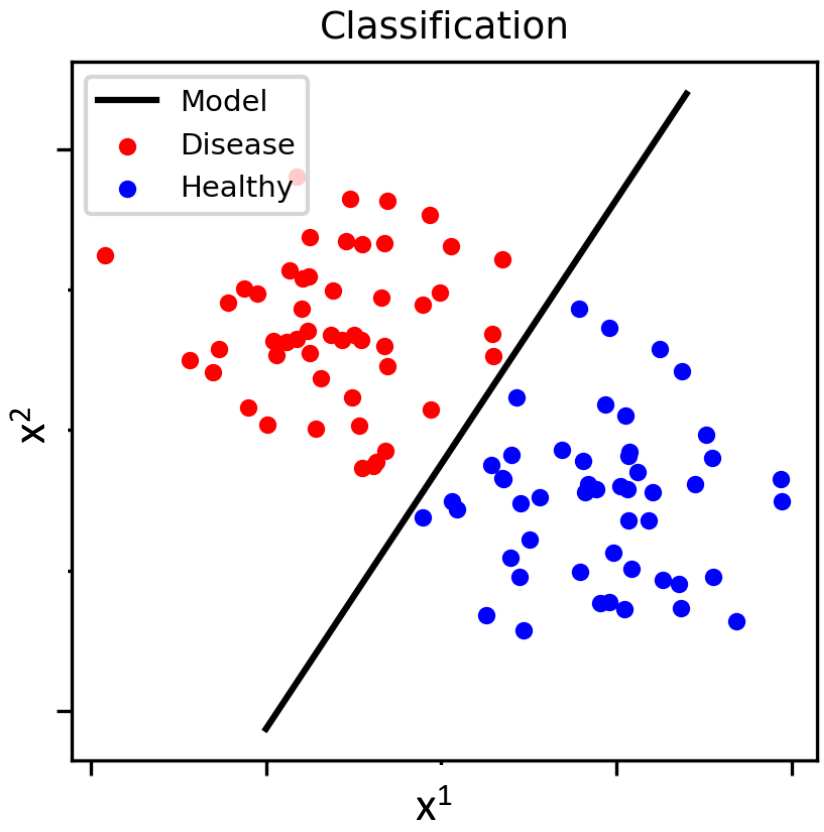
Machine Learning

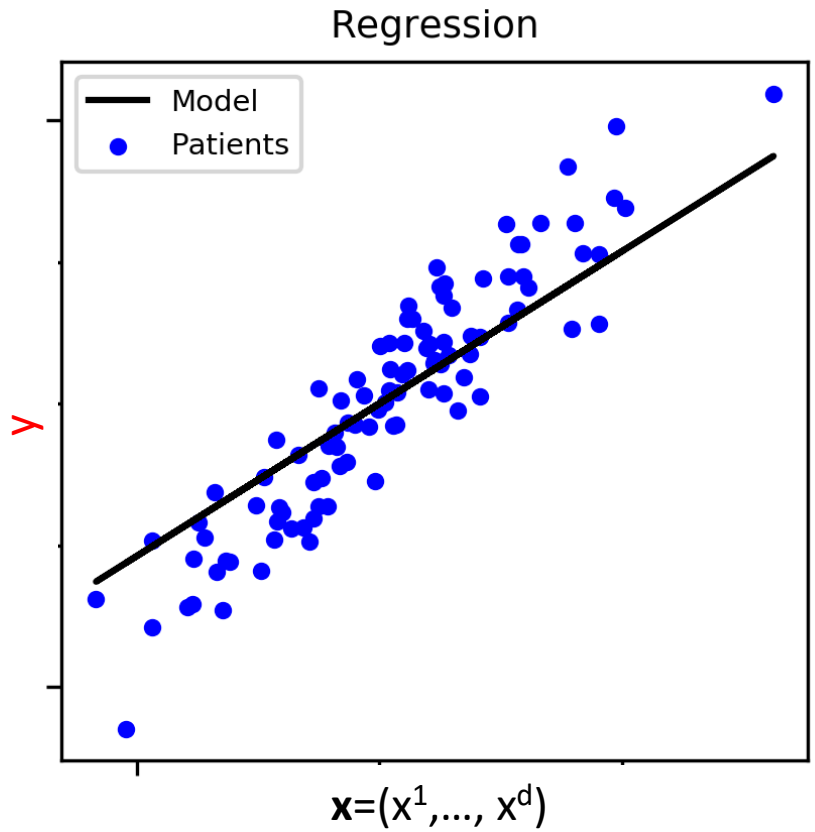
- **Unsupervised Learning**
  - *(in general)* **Unlabeled dataset** $\mathrm{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$,
    where each data point $\boldsymbol{x}_i \in R^d$ contains *d* features $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})$
  - To find useful properties/patterns of the structures of the dataset
  - *e.g.,* clustering, density estimation, one-class classification, association analysis, etc…

- **Supervised Learning**
  - *(in general)* **Labeled dataset** $\mathrm{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$,
    where each $\boldsymbol{x}_i$ is associated with a label y_i.
  - Input variables: $x_{i1}, \ldots, x_{id}$ (*d* features), output variable: y (label)
  - To find a functional relationship between input and output variable $\hat{y} = f(\boldsymbol{x})$ from the dataset
  - To use input variables to predict unknown or future values of output variable.
  - *e.g.,* classification (categorical label), regression (continuous label), etc…

Supervised Learning

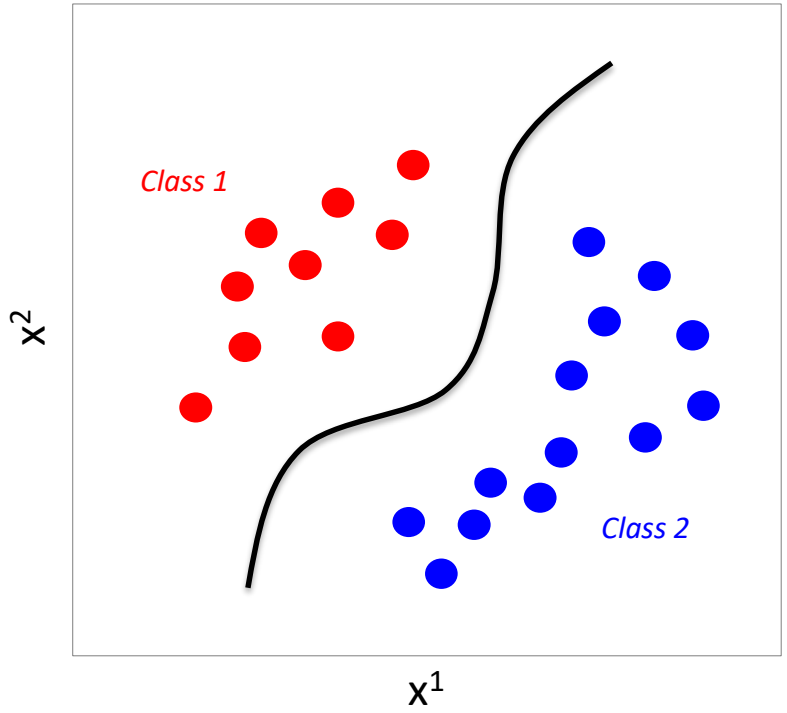
- **Classification vs Regression**

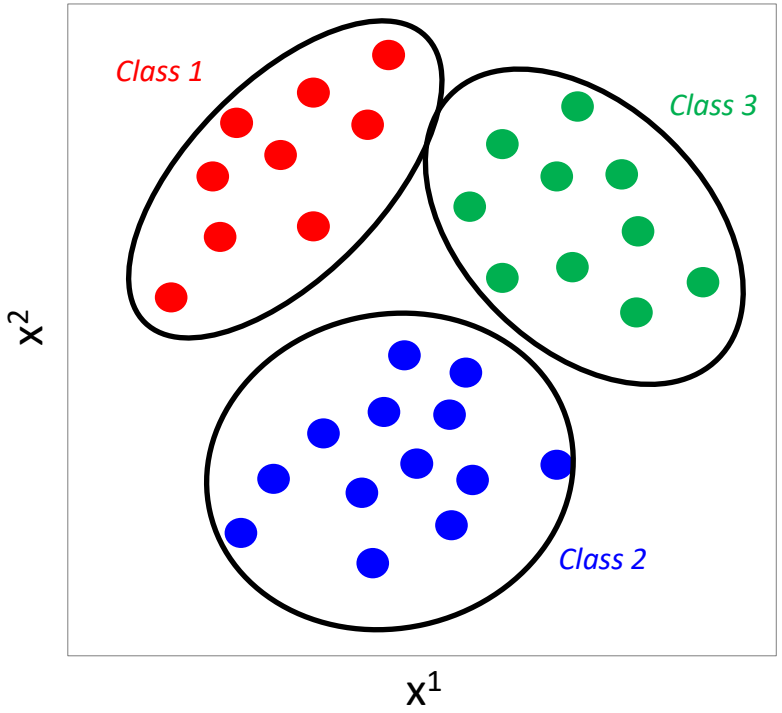




Supervised Learning

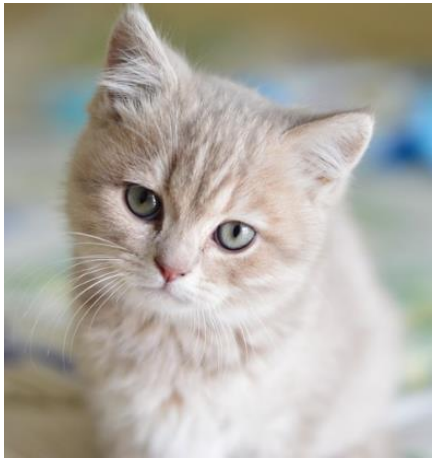
- **Binary vs Multi-class Classification**





Supervised Learning

- **Multi-label Classification**

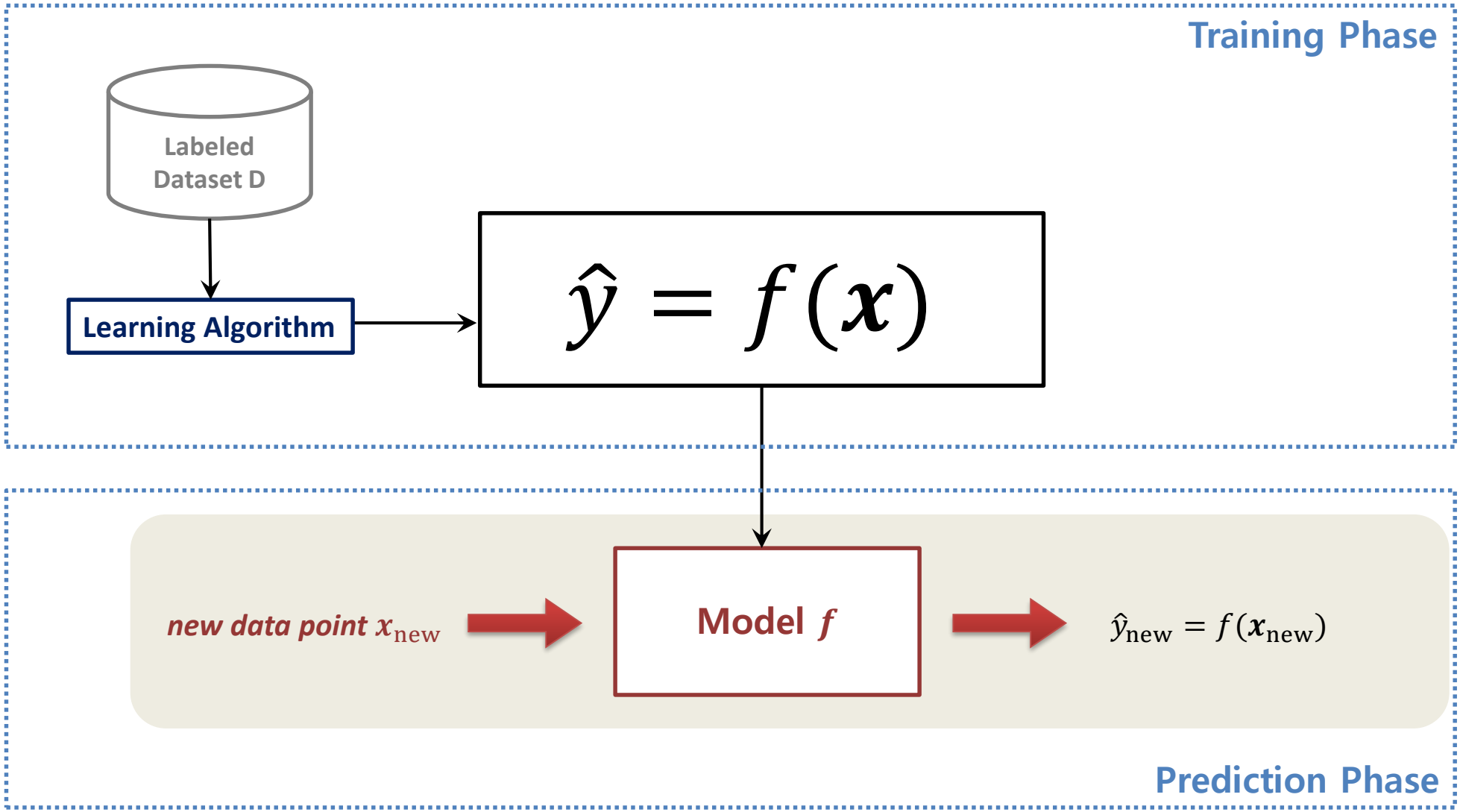


“Dog”

“Cat”

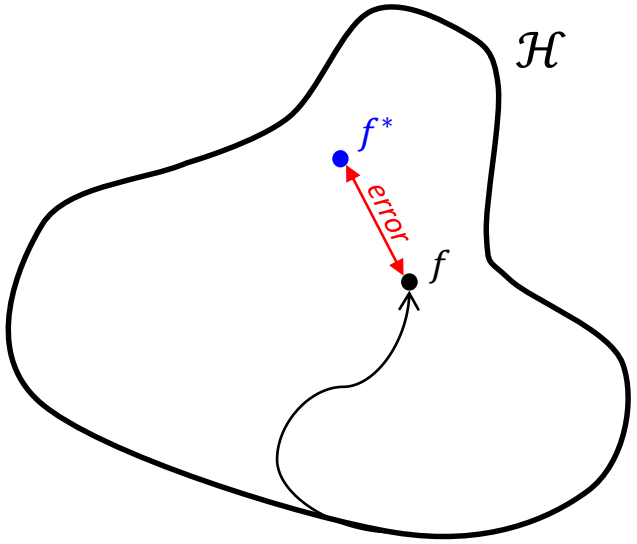
?

Supervised Learning



Supervised Learning

- **How to find the model *f*?**
  - A learning algorithm defines a hypothesis space $\mathcal{H}$, the set of all possible functions that the learning algorithm can generate.
  - Supervised learning is to find a hypothesis f in $\mathcal{H}$ that best approximates the unknown true function f^* given a finite number of training instances.



Formal Setup of Supervised Learning

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of the *d* input variables and y_i is the corresponding label of the output variable.

- We seek a function f that predicts the output y from the input $\boldsymbol{x}$: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the **parameter** of the model.
  - **Loss function** $L\big(\mathrm{y}, f(\boldsymbol{x}; \boldsymbol{\theta})\big)$: penalizing **errors in prediction** for a data point $(\boldsymbol{x}, y)$.
    - *e.g.,* squared loss $L\big(\mathrm{y}, f(\boldsymbol{x}; \boldsymbol{\theta})\big) = \big(y - f(\boldsymbol{x}; \boldsymbol{\theta})\big)^2$
  - **Cost function** $J(\boldsymbol{\theta})$: typically, a sum of **loss functions** L over the training set + some **regularizer** Ω

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{(\boldsymbol{x}_i, y_i) \in D} L\big(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})\big) + \Omega(\boldsymbol{\theta})$$

- **Training** is to find the optimal parameter $\boldsymbol{\theta}^*$ that minimizes the cost function $J(\boldsymbol{\theta})$

Supervised Learning Algorithms

- **Regression**
  - Linear Regression
  - K-Nearest Neighbors
  - Decision Tree
  - Neural Network
  - Support Vector Regression
  - …
- **Classification**
  - Logistic Regression
  - K-Nearest Neighbors
  - Decision Tree
  - Neural Network
  - Support Vector Machine
  - …

Linear Regression

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (1, x_{i1}, \dots, x_{id}) \in \mathbb{R}^{d+1}$ is the *i*-th input vector of the *d*+1 input variables and $y_i \in \mathbb{R}$ is the corresponding label of the output variable.

  - the first entry is always set to "1"

- The output of linear regression (prediction of y)
  : $\hat{y} = f(\boldsymbol{x}) = \mathbf{w}^T \boldsymbol{x}$, where $\mathbf{w} = (w_0, w_1, \dots, w_d)$ is a vector of parameters.

  - $w_1, \dots, w_d$ are called "coefficients" or "weights"
  - w_0 is called "intercept" or "bias"

- **Training**: To find the optimal parameter $\mathbf{w}^*$ that minimizes the training error (cost function)

  Here we use squared loss $L(y, \hat{y}) = (\hat{y} - y)^2$

$$J(\mathbf{w}) = \text{MSE}_{\text{train}} = \frac{1}{n} \sum_{(\boldsymbol{x}_i, y_i) \in D} (\hat{y_i} - y_i)^2 = \frac{1}{n} \|\mathbf{Xw} - \mathbf{y}\|_2^2$$

  - $\mathbf{X}$, $\mathbf{y}$ are matrix representation of D

Linear Regression

- **Training:** To find the optimal parameter $\mathbf{w}^*$ that minimizes the training error
  → **an optimization problem**

$$\text{MSE}_{\text{train}} = \frac{1}{n} \sum_{(x_i, y_i) \in D} (\hat{y}_i - y_i)^2 = \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

▶ how? set the gradient to 0 → a closed-form solution (normal equation)

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = \frac{1}{n} \nabla_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 = 0$$

...

...

...

$$\mathbf{w}^* = (\mathbf{X}^{\text{T}}\mathbf{X})^{-1}\mathbf{X}^{\text{T}}\mathbf{y}$$

- The trained model $f(\boldsymbol{x}) = \mathbf{w}^{*\text{T}}\boldsymbol{x}$

Linear Regression

- **Probabilistic Interpretation of Linear Regression**
  - Assume $y \sim \mathcal{N}(\hat{y}, \sigma^2), \hat{y} = \mathbf{w}^T \boldsymbol{x}$

$$p(y|\boldsymbol{x}; \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|y - \mathbf{w}^T \boldsymbol{x}\|^2}{2\sigma^2}\right)$$

p.d.f. of $N(\hat{y}, \sigma^2)$

- **Maximum Likelihood Estimation** (with respect to $\hat{y}$)

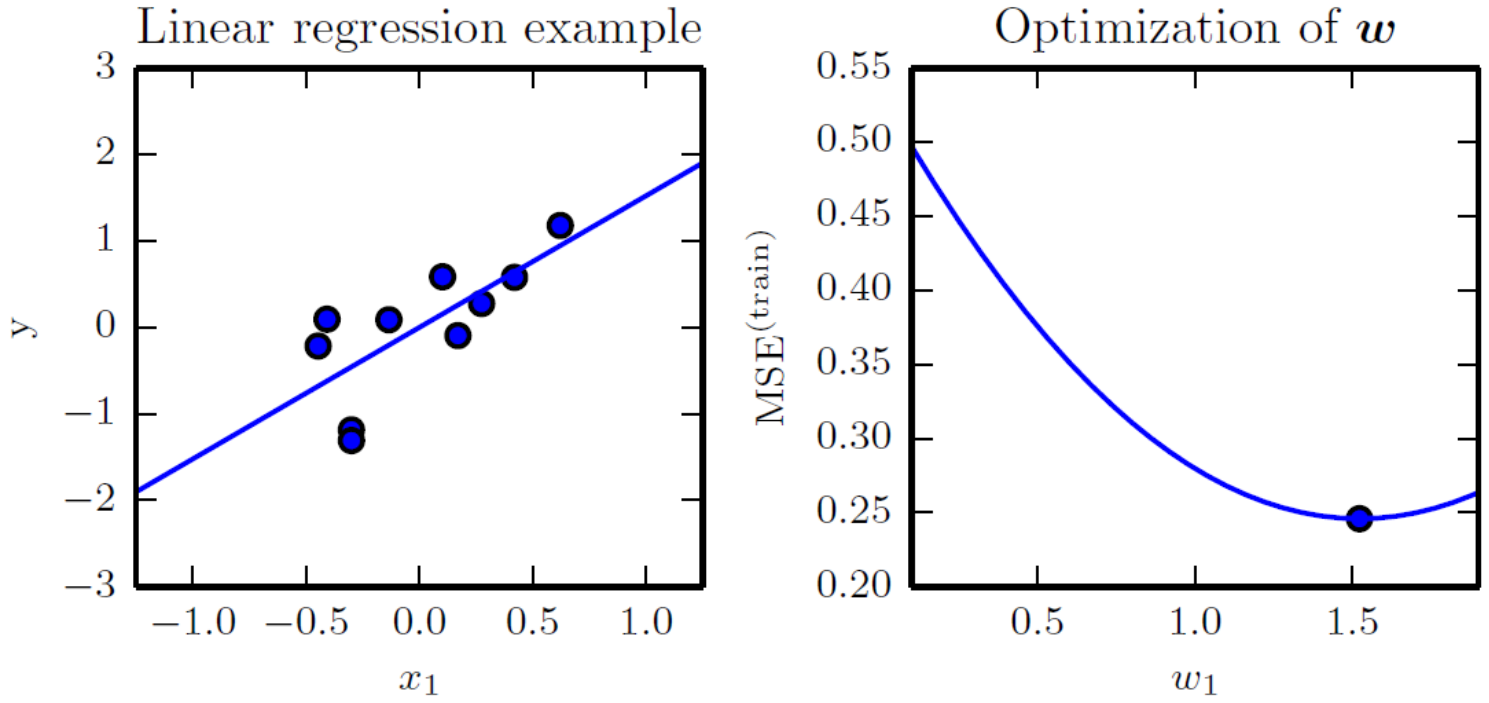
log-likelihood

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{(\boldsymbol{x}_i, y_i) \in D} p(y_i|\boldsymbol{x}_i; \mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{(\boldsymbol{x}_i, y_i) \in D} \log p(y_i|\boldsymbol{x}_i; \mathbf{w})$$

$$= \underset{\mathbf{w}}{\operatorname{argmax}} -\frac{n}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{(\boldsymbol{x}_i, y_i) \in D} \|y_i - \mathbf{w}^T \boldsymbol{x}_i\|^2$$

Linear Regression

- ***Example*** (univariate case): $\hat{y} = f(x_1) = w_1 \cdot x_1$
  - The training set consists of ten data points
  - A single parameter (w_1) is determined such that the line $y = w_1 \cdot x_1$ comes as close as possible to passing through all training data points
  - The trained model f minimizes the mean squared error on the training set



Generalization

- Using “Linear Regression”, we trained the model f by minimizing the **training error** (the error on the training set D)

- The model f must perform well on new, previously unseen data points, which is called **Generalization**.
- To evaluate generalization performance, we use a **test set** $D^{(\text{test})}$ consisting of $n^{(\text{test})}$ data points that were collected separately from the training set D.
- We want the **generalization error**, *a.k.a.* **test error** (the error on the test set) to be low as well.

$$\text{MSE}_{\text{test}} = \frac{1}{n^{(\text{test})}} \sum_{(\boldsymbol{x}_i, y_i) \in D^{(\text{test})}} (\hat{y}_i - y_i)^2 = \frac{1}{n^{(\text{test})}} \left\| \mathbf{X}^{(\text{test})} \mathbf{w} - \mathbf{y}^{(\text{test})} \right\|^2$$

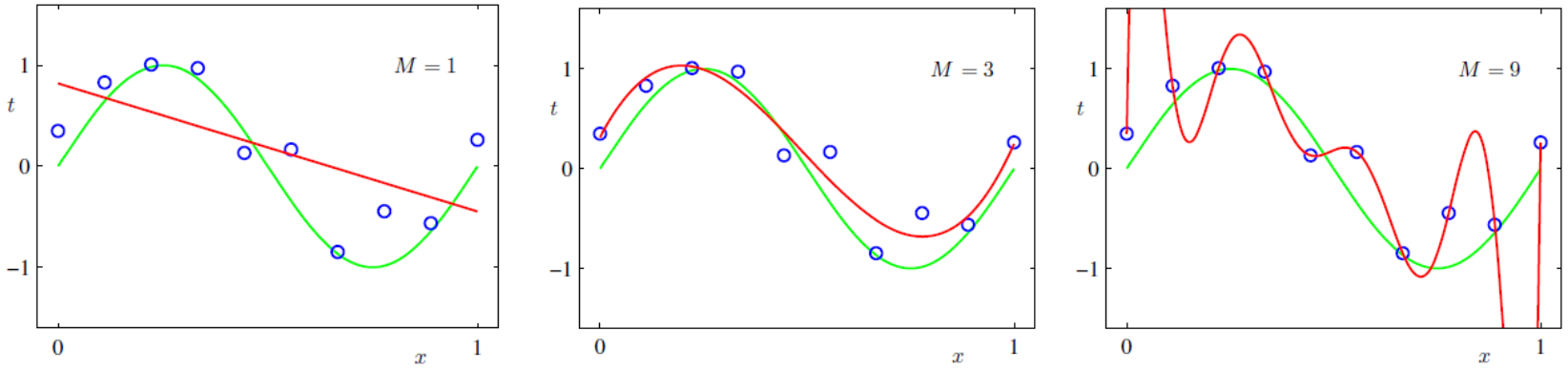
Capacity, Overfitting and Underfitting

- **The factors determining how well a machine learning algorithm will perform are its ability to:**
  1. Make the training error small.
  2. Make the gap between training and test error small

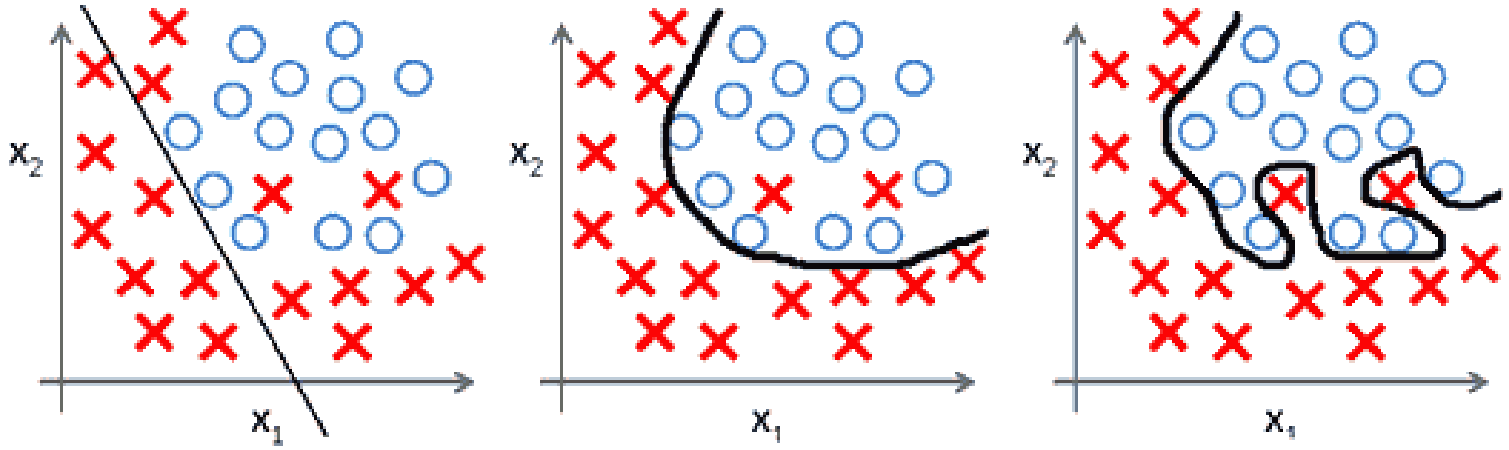
- **Overfitting:** the gap between the training error and test error is too large.
- **Underfitting:** the model is not able to obtain a sufficiently low error value on the training set.
- We can control whether a model is more likely to **overfit or underfit** by altering the **capacity**.

Capacity, Overfitting and Underfitting

- ***Example*: in Regression…**



- ***Example*: in Classification…**



Capacity, Overfitting and Underfitting

- How to control the **capacity** of a learning algorithm?
  : One way is to control the **hypothesis space** $\mathcal{H}$

- ***Example***: polynomial regression with different degree
  - An univariate linear regression model (polynomial degree=1)

$$\hat{y} = w_0 + w_1 x$$

  - A quadratic regression model (polynomial degree=2)

$$\hat{y} = w_0 + w_1 x + w_2 x^2$$

x^2: square of x

  - A polynomial regression model (polynomial degree=9)

$$\hat{y} = w_0 + \sum_{i=1}^{9} w_i x^i$$

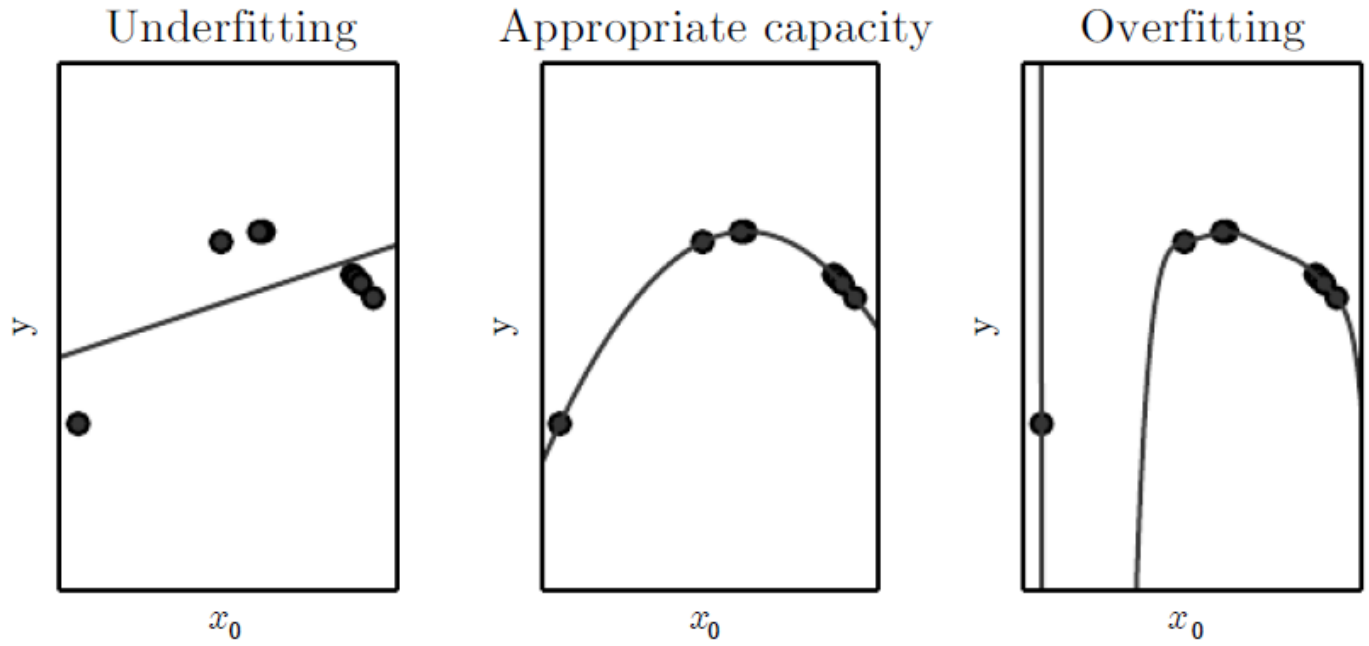
x^i: i-th power of x

Increase the polynomial degree → broaden the hypothesis space $\mathcal{H}$

→ increase the (representational) capacity

Capacity, Overfitting and Underfitting

- After training the models with **different capacities** (different polynomial degree) using the same training dataset.



(1) Polynomial degree=1

(low capacity)

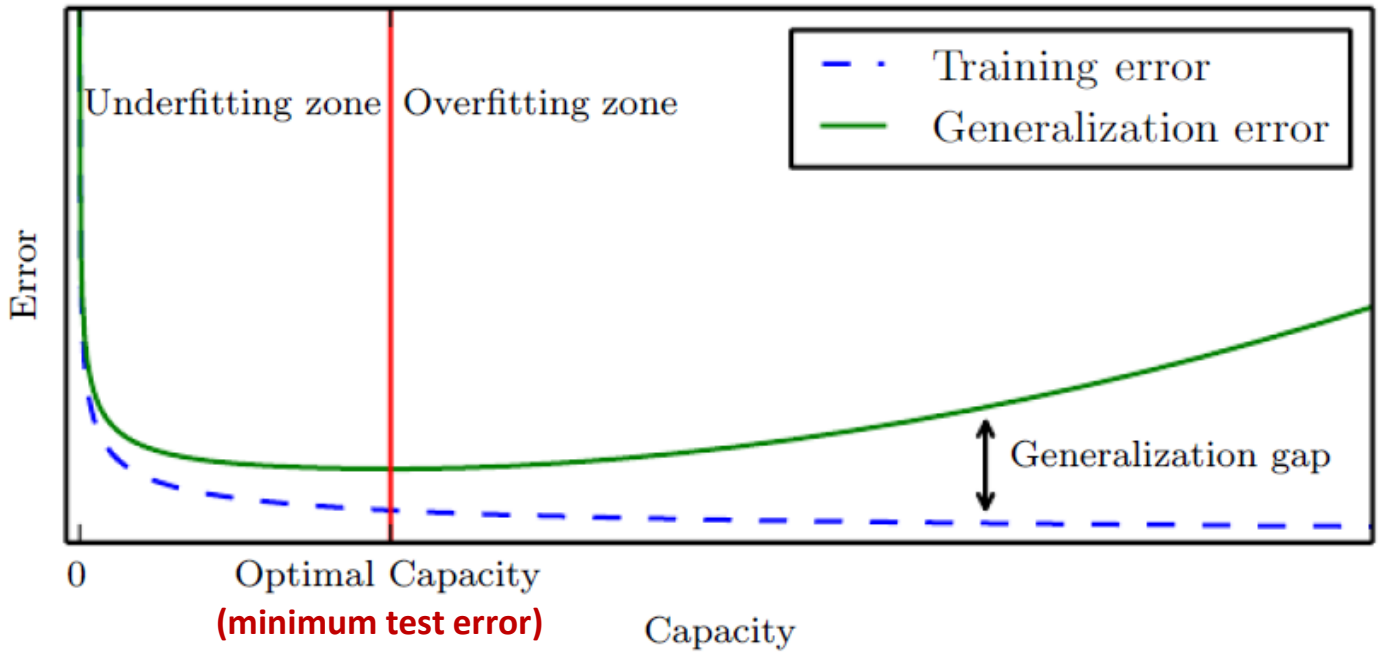
(2) degree=2

(3) degree=9

(high capacity)

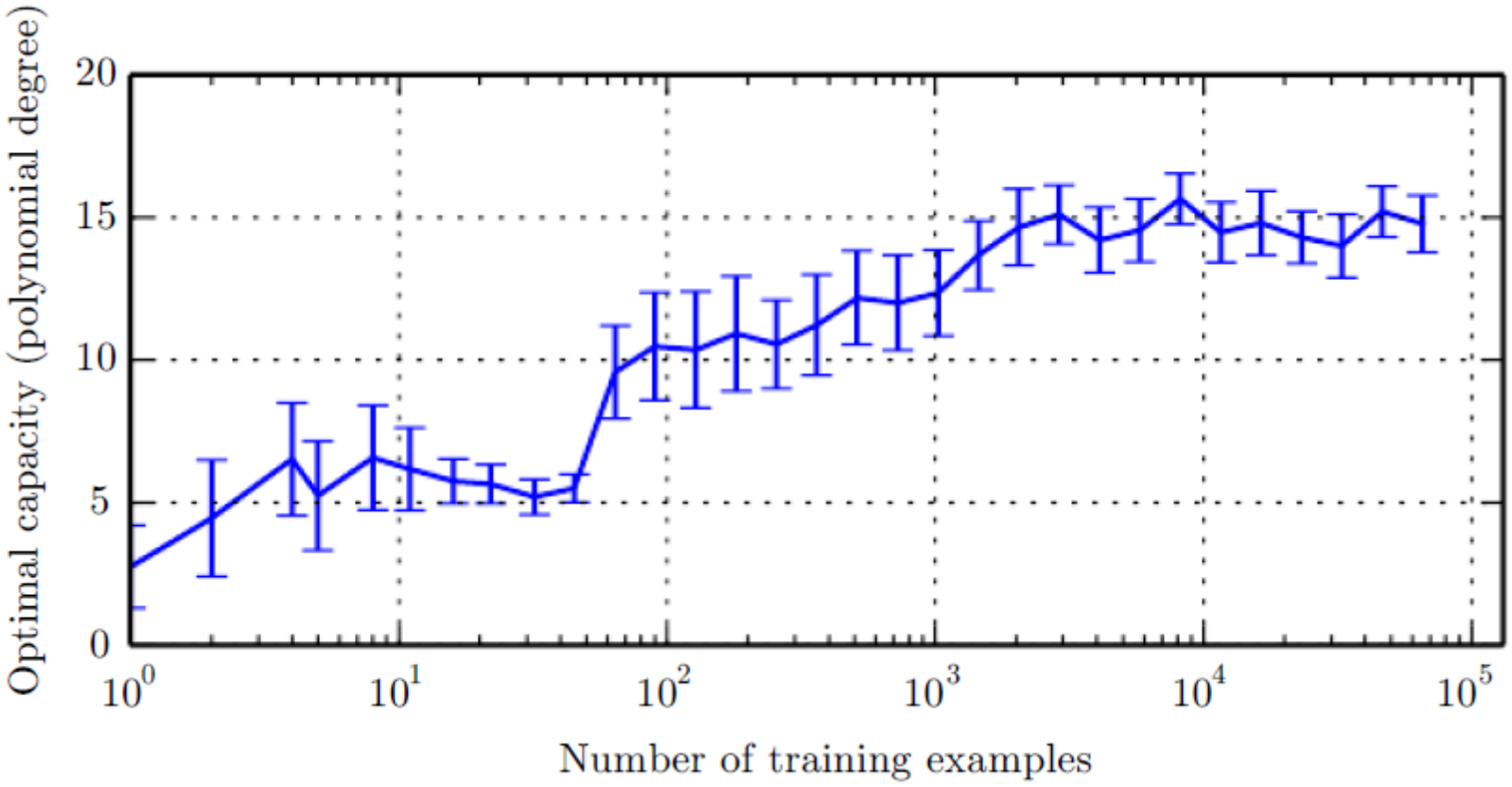
Capacity, Overfitting and Underfitting

- So, the typical relationship between **capacity** and **error**
  - **Underfitting** (training and test error are both high), when **capacity is too small**
  - **Overfitting** (the gap between training and test error is high), when **capacity is too large**



Capacity, Overfitting and Underfitting

- **The optimal capacity depends on data***
  - As the training data size increases, the optimal capacity increases.



Regularization

- **How to avoid overfitting when using a learning algorithm with large hypothesis space?**
  : One is to give a preference for one solution over another in its hypothesis space.

- **Regularization:** Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error
  - ***Example***: adding a penalty called a **regularizer** $\Omega(\mathbf{w})$ to the cost function

$$J(\mathbf{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda\Omega(\mathbf{w})$$

Regularization

- ***Example:*** **Weight decay** ($\Omega(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$**,** *a.k.a.* **L2 regularization)** for linear regression

  → *Ridge regression*

  - λ controls the strength of our preference for small parameters → reduced fluctuation

$$J(\mathbf{w}) = \mathrm{MSE}_{\mathrm{train}} + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

- ***Example:*** **L1 regularization** ($\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$) for linear regression

  → *Lasso regression*

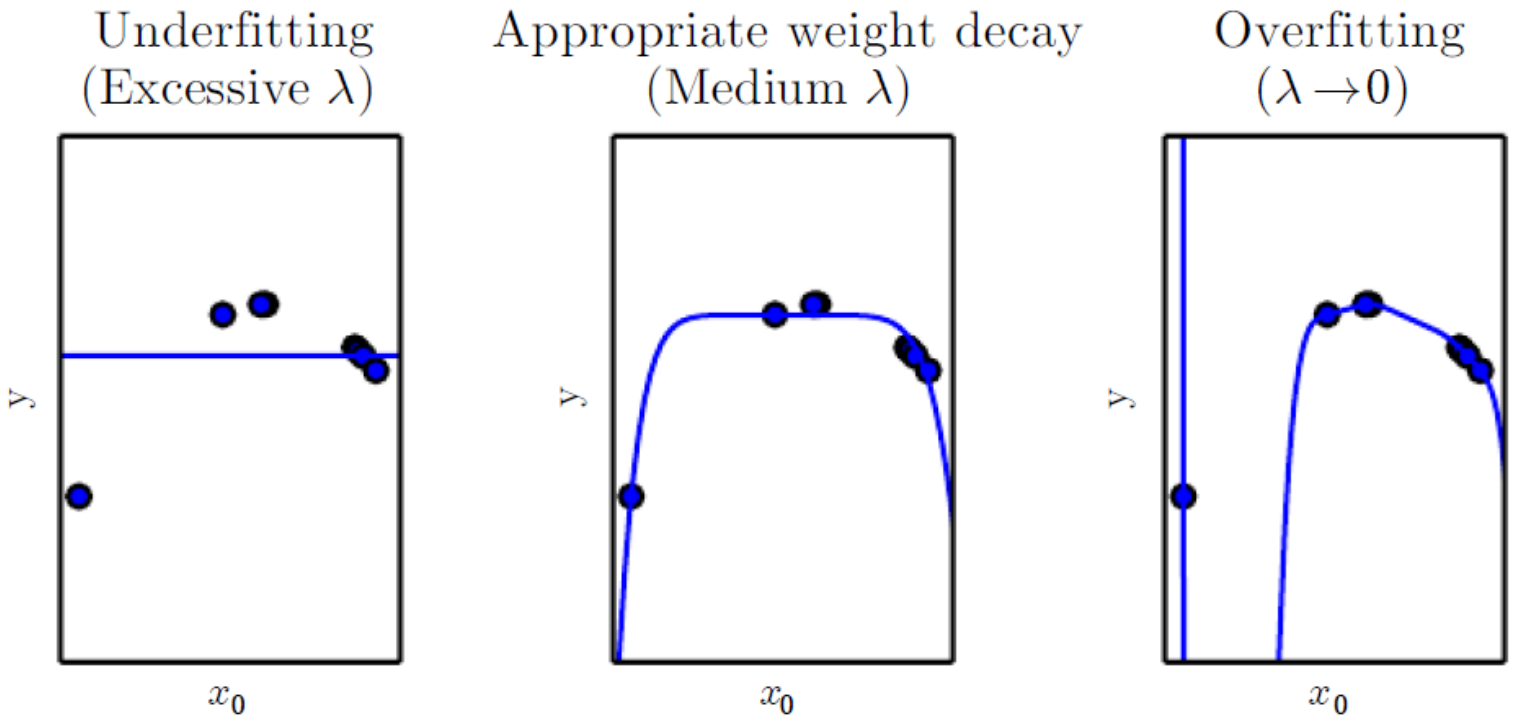
  - λ controls the strength of our preference for small parameters

    + sparsity → reduced fluctuation

$$J(\mathbf{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda\|\mathbf{w}\|_1 = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|_1$$

Regularization

- ***Example:*** **Weight decay** ($\Omega(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$) for polynomial regression with degree=9
  - $\lambda = 0$ → no preference (overfitting, as before)
  - larger λ → the weights $\mathbf{w}$ become smaller
  - Too large λ → $\mathbf{w} \cong \mathbf{0}$ (underfitting)



Hyperparameters and Validation Sets

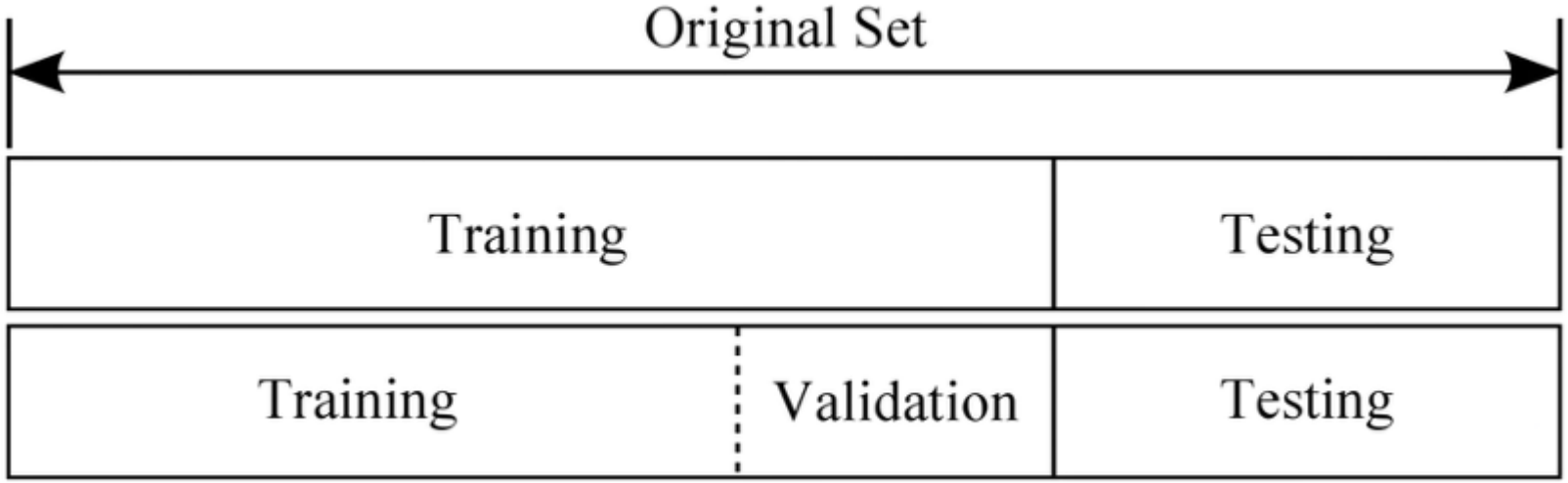
- Most machine learning algorithms have **hyperparameters** that control the algorithm's behavior (*e.g.*, capacity, preference, optimization, etc.).
  - Linear regression has no hyperparameter
  - In the previous example (Polynomial regression with weight decay), λ and the polynomial degree are hyperparameters

- It is not appropriate to choose **hyperparameters** based on the training set
  - If learned on the training set, they choose the maximum possible model capacity, resulting in overfitting.
  - *Hyperparameters* are set before training begins, whereas *parameters* are derived via training.

- So, we need a **validation set**, which consists of data points that were not observed during training → **validation error** (the error on the validation set)

Training, Validation, and Test

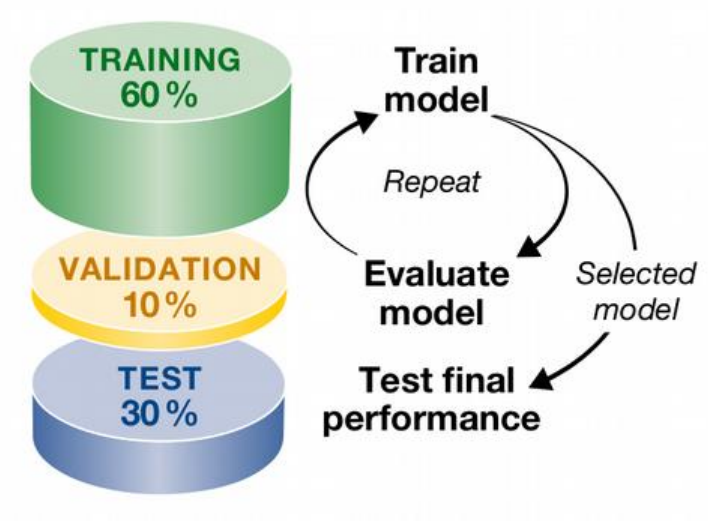
- **Training set:** to learn the parameters of the model
- **Validation set:** to choose the hyperparameters of the model
- **Test set:** for final evaluation of the **generalization error** of the model
  (How well will our model perform with new data that were not observed during training and validation?)



"must be disjoint!"

Training, Validation, and Test

- **Example of a Modeling Procedure:**
  - Suppose that we have a dataset of 10,000 data points, and we consider 100 hyperparameter candidates.
  - Split the data into training, validation, and test sets
    - 60% → training set
    - 10% → validation set
    - 30% → test set
  - For each hyperparameter candidate,
    - Train a model on the training set.
    - Evaluate a model on the validation set.
  - Choose the best one with the minimum validation error.
  - Finally, evaluate the best one on the test set.



Cross-Validation

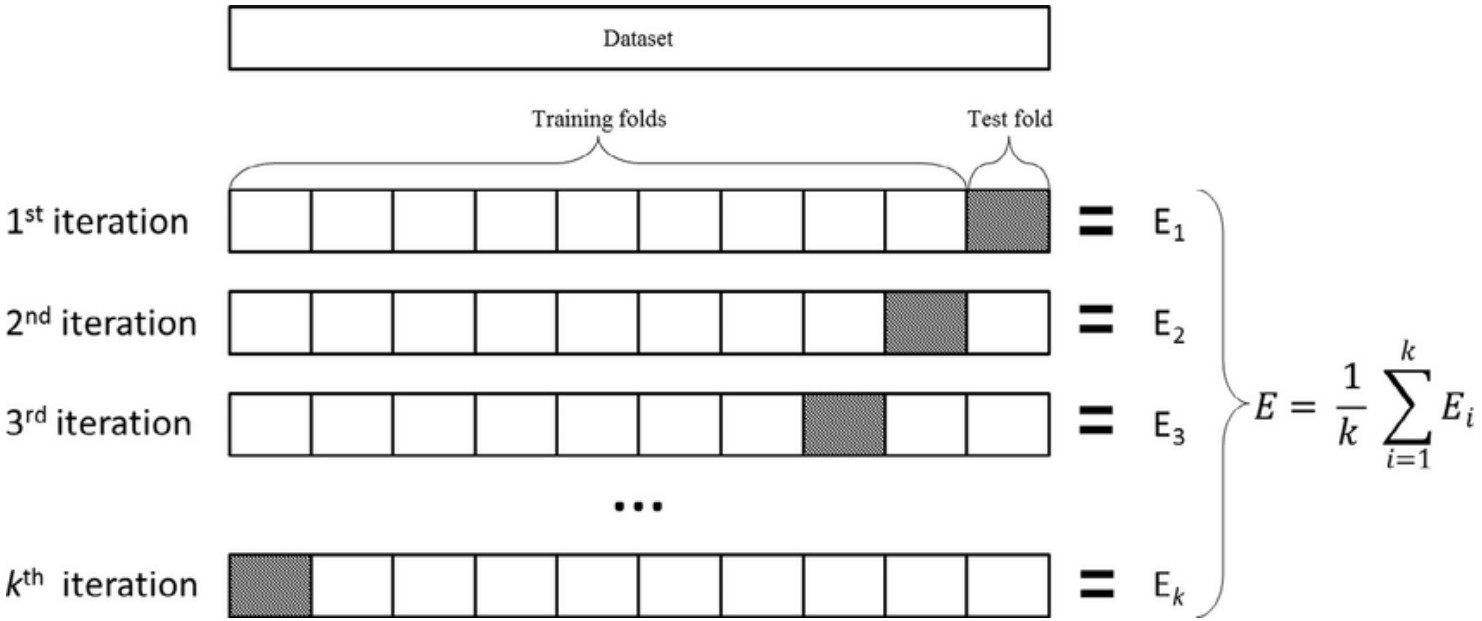
- **But, what if the original dataset is too small?**

- ***K*-fold Cross-Validation**

  1. Split the dataset into *K* nonoverlapping subsets

  2. (from *i*=1 to *K*) On the *i*-th run, the *i*-th subset is used as the test set, the rest of the dataset is used as the training/validation set

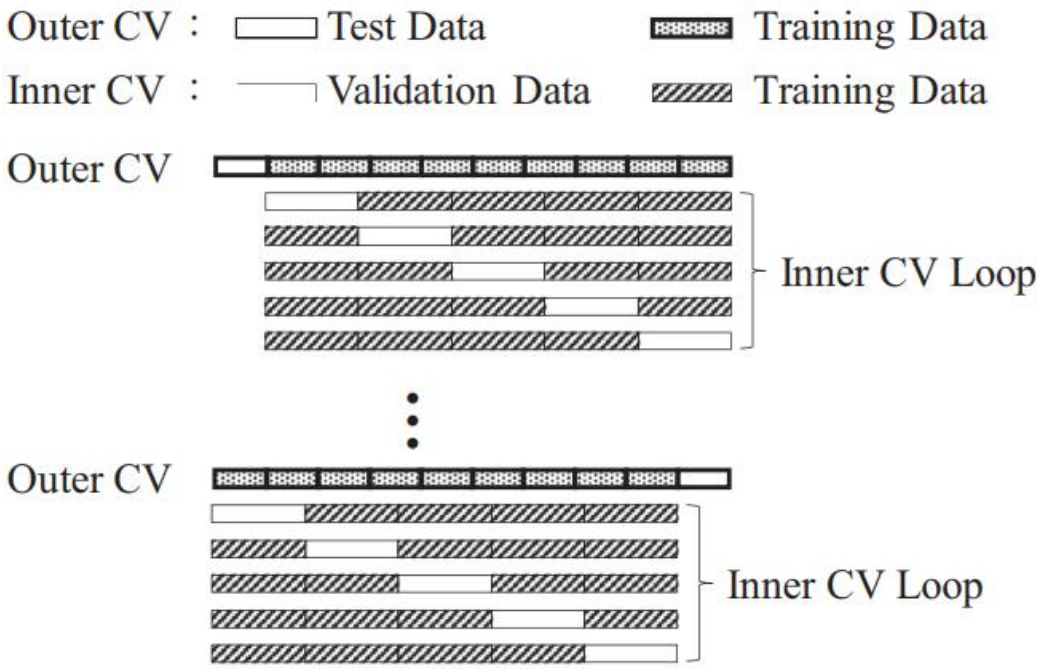
  3. The test error is estimated by averaging test error across the *K* runs



Cross-Validation

- **Leave-One-Out Cross-Validation**
  - *K*-fold Cross-Validation with *K*=the number of data points

- **Nested Cross-Validation (Double Cross-Validation)**
  - **Outer CV:** (Training/Validation) vs Test
  - **Inner CV:** Training vs Validation



Supervised Learning Algorithms

- **Regression**
  - **Linear Regression**
  - **K-Nearest Neighbors**
  - Decision Tree
  - Neural Network
  - Support Vector Regression
  - …
- **Classification**
  - **Logistic Regression**
  - **K-Nearest Neighbors**
  - Decision Tree
  - Neural Network
  - **Support Vector Machine**
  - …

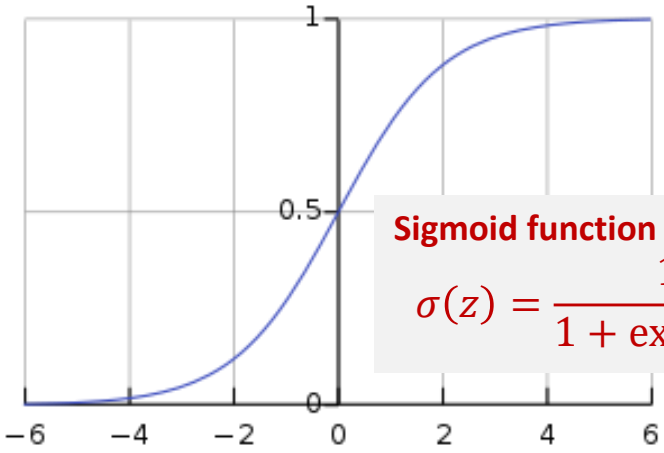
Logistic Regression

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (1, x_{i1}, \dots, x_{id}) \in \mathbb{R}^{d+1}$ is the *i*-th input vector of the *d*+1 input variables and $y_i \in \{0,1\}$ is the corresponding label of the output variable.

- **Logistic Regression**: Extends the idea of linear regression to situation where **the output variable is binary** (y = 0 or 1)

- The output of logistic regression (prediction of y) : $\hat{y} = f(\boldsymbol{x}) = \sigma(\mathbf{w}^T\boldsymbol{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\boldsymbol{x})}, \hat{y} \in [0,1]$

What are parameters?
What are hyperparameters?



Sigmoid function

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

Logistic Regression

- **Training**: To find the optimal parameter $\mathbf{w}^*$ that minimizes the training error (cost function) → here we use "cross-entropy" loss

$$J(\mathbf{w}) = \frac{1}{n} \sum_{(\boldsymbol{x}_i, y_i) \in D} L(y_i, \hat{y}_i) = \frac{1}{n} \sum_{(\boldsymbol{x}_i, y_i) \in D} [-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)]$$

what if $y_i = 0?\ y_i = 1?$

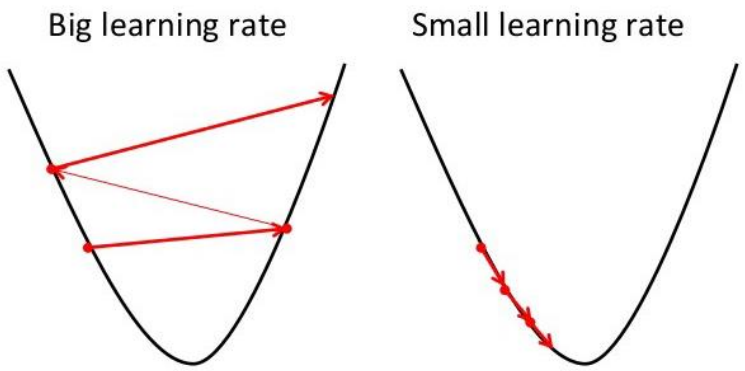
▶ how? gradient descent! (no closed-from solution)

Repeat the following until convergence

$$\mathbf{w} \coloneqq \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w})$$

$$\rightarrow w_j \coloneqq w_j - \epsilon \frac{\partial}{\partial w_j} J(\boldsymbol{w}), \forall w_j \in \boldsymbol{w}$$

ϵ is the learning rate



- The trained model $f(\boldsymbol{x}) = \sigma(\mathbf{w}^{*\mathrm{T}} \boldsymbol{x})$

Logistic Regression

$$L(y, \hat{y}) = -y \log \sigma(z) - (1 - y) \log\big(1 - \sigma(z)\big),$$

where $\hat{y} = \sigma(z), z = \mathbf{w}^T \boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_d x_d$

$$\frac{\partial L(\boldsymbol{w})}{\partial w_j} = \frac{\partial L(\boldsymbol{w})}{\partial z} \frac{\partial z}{\partial w_j} = (\hat{y} - y) x_j$$

$$\begin{aligned}
\frac{\partial L(\boldsymbol{w})}{\partial z} &= -y \frac{\partial \log \sigma(z)}{\partial z} - (1 - \mathrm{y}) \frac{\partial \log\big(1 - \sigma(z)\big)}{\partial z} \\
&= -y \frac{1}{\sigma(z)} \frac{\partial \sigma(z)}{\partial z} - (1 - \mathrm{y}) \frac{-1}{1 - \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \\
&= -y \frac{1}{\sigma(z)} \sigma(z)\big(1 - \sigma(z)\big) - (1 - \mathrm{y}) \frac{-1}{1 - \sigma(z)} \sigma(z)\big(1 - \sigma(z)\big) \\
&= -y + \sigma(z) = \hat{y} - y
\end{aligned}$$

$$\frac{\partial z}{\partial w_j} = \frac{\partial (w_0 + w_1 x_1 + \cdots + w_d x_d)}{\partial w_j} = x_j$$

Logistic Regression

- **Probabilistic Interpretation of Logistic Regression**
  - Assume $y \sim Bernoulli(\hat{y})$, $\hat{y} = \sigma(\mathbf{w}^T \boldsymbol{x})$

$$p(y = 1|x; \mathbf{w}) = \sigma(\mathbf{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \boldsymbol{x})}$$

$$p(y = 0|x; \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \boldsymbol{x}) = \frac{\exp(-\mathbf{w}^T \boldsymbol{x})}{1 + \exp(-\mathbf{w}^T \boldsymbol{x})}$$

▼

$$p(y|x; \mathbf{w}) = \big(\sigma(\mathbf{w}^T \boldsymbol{x})\big)^y \big(1 - \sigma(\mathbf{w}^T \boldsymbol{x})\big)^{1-y}$$

p.f. of *Bernoulli*($\hat{y}$)

- **Maximum Likelihood Estimation** (with respect to $\hat{y}$)

log-likelihood

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} \log \prod_{(\boldsymbol{x}_i, y_i) \in D} p(y_i|\boldsymbol{x}_i; \mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{(\boldsymbol{x}_i, y_i) \in D} \log p(y_i|\boldsymbol{x}_i; \mathbf{w})$$

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{(\boldsymbol{x}_i, y_i) \in D} [y_i \log \sigma(\mathbf{w}^T \boldsymbol{x}) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \boldsymbol{x}))]$$

k-Nearest Neighbors

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of the *d* input variables and y_i is the corresponding label of the output variable.

- ***k*-Nearest Neighbors** is an instance-based learning algorithm **that does not require any training of models.**

- For a test data point $\boldsymbol{x}_{\text{new}}$
  1. Compute distance from $\boldsymbol{x}_{\text{new}}$ to each data point in D (distance measure)
  2. Identify *k* nearest neighbors of $\boldsymbol{x}_{\text{new}}$, $kNN(\boldsymbol{x}_{\text{new}}) = \{(\boldsymbol{x}_{(1)}, y_{(1)}), \dots, (\boldsymbol{x}_{(k)}, y_{(k)})\} \subset D$
  3. Use labels of the nearest neighbors to predict $\hat{y}_{\text{new}}$ (weighting scheme)
     - e.g) voting or weighted voting for classification, averaging or weighted averaging for regression.

What are parameters?
What are hyperparameters?

k-Nearest Neighbors

- **For Classification,**

  Voting: $\hat{y} = \underset{j}{\operatorname{argmax}} \sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} I(y_{(i)} = j)$

  Weighted Voting: $\hat{y} = \underset{j}{\operatorname{argmax}} \sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} w(\boldsymbol{x}_{(i)}, \boldsymbol{x}) I(y_{(i)} = j)$

- **For Regression,**

  Averaging: $\hat{y} = \frac{1}{k} \sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} y_{(i)}$

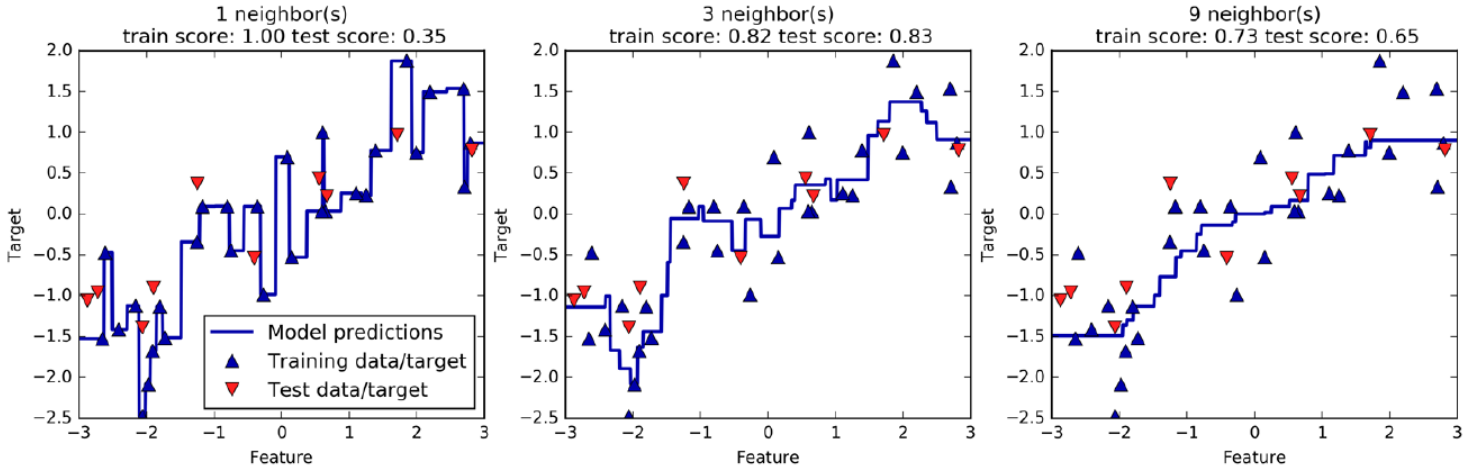
  Weighted Averaging: $\hat{y} = \frac{1}{\sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} w(\boldsymbol{x}_{(i)}, \boldsymbol{x})} \sum_{(\boldsymbol{x}_{(i)}, y_{(i)}) \in kNN(\boldsymbol{x})} w(\boldsymbol{x}_{(i)}, \boldsymbol{x}) y_{(i)}$

$w(\boldsymbol{x}_{(i)}, \boldsymbol{x})$ is a weight function *(hyperparameter, not learned)*

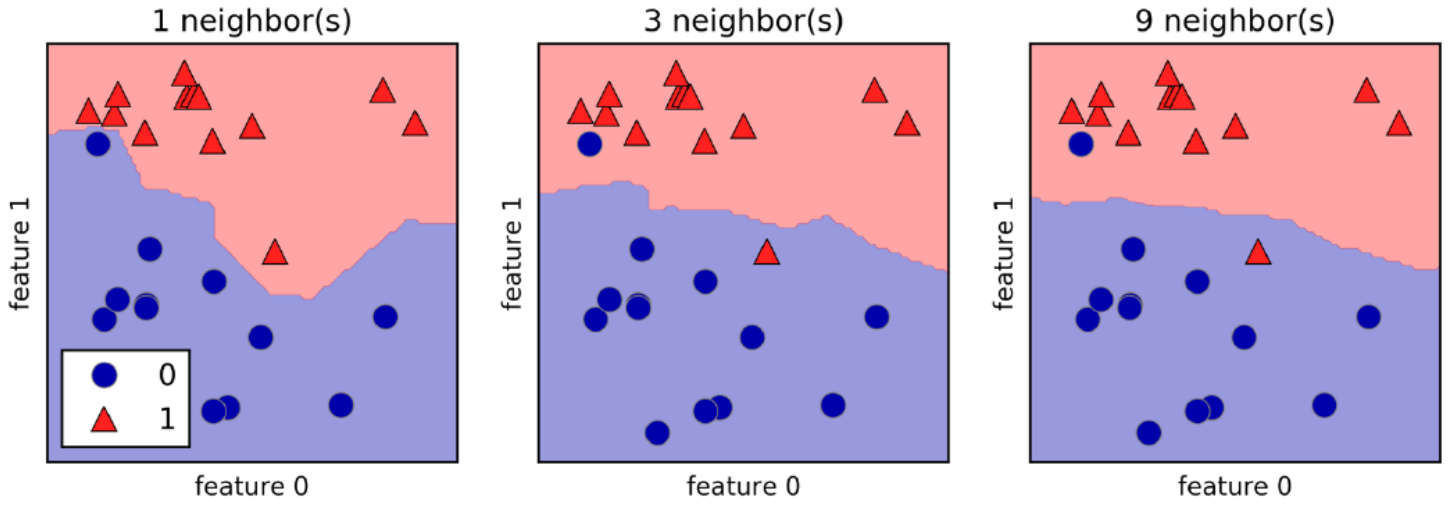
e.g., inverse of Euclidean distance $\frac{1}{\|\boldsymbol{x}_{(i)} - \boldsymbol{x}\|_2}$

k-Nearest Neighbors

- **In Regression…**

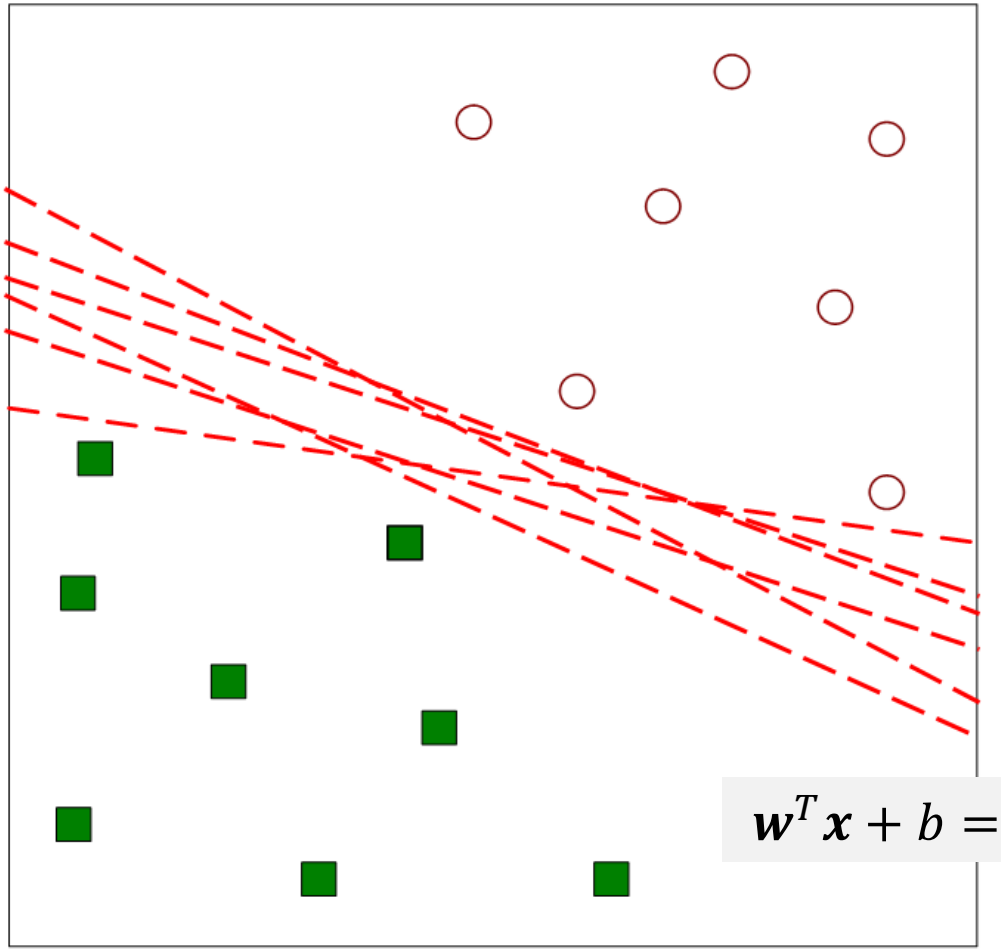


- **In Classification…**



(Linear) Support Vector Machine

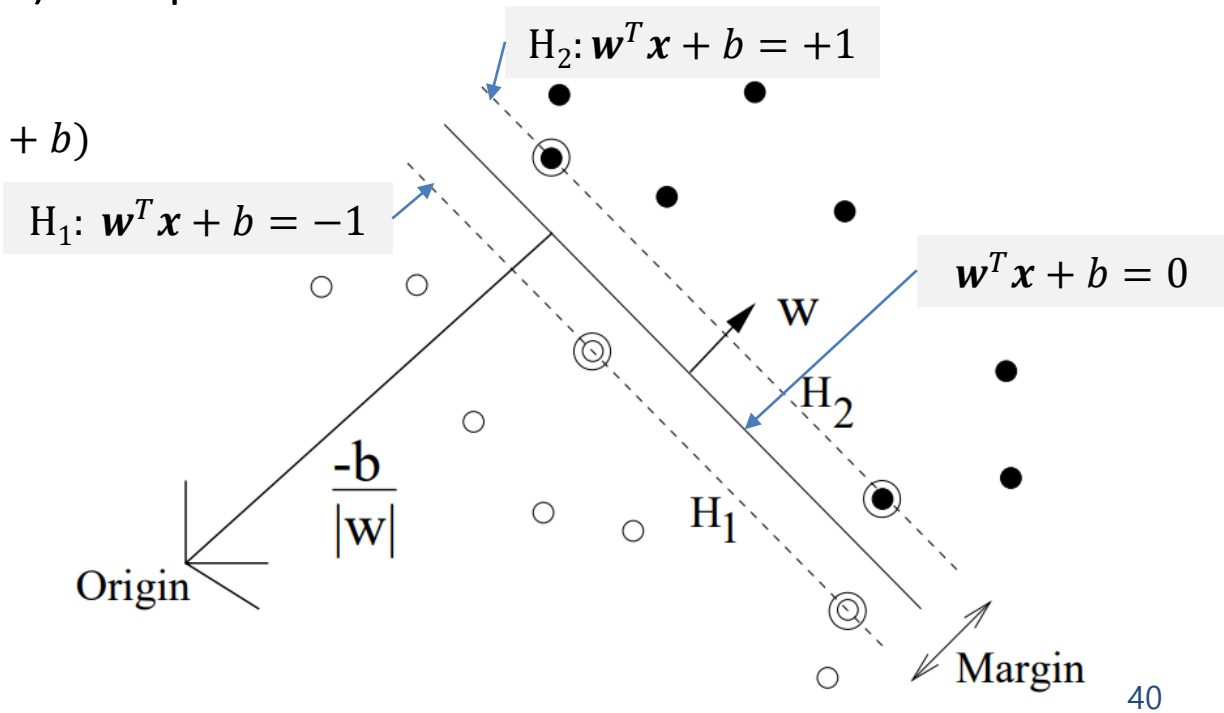
- **Binary Classification Problem**
  - Many possible hyperplanes that separates the training data



(Linear) Support Vector Machine

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of the *d* input variables and $y_i \in \{-1, +1\}$ is the corresponding label of the output variable.

- SVM looks for the **maximum-margin hyperplane** $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ between positive ($y_i = +1$) and negative ($y_i = -1$) data points
  - Margin: $2/||\boldsymbol{w}||$
  - Prediction $\hat{y} = f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$



(Linear) Support Vector Machine

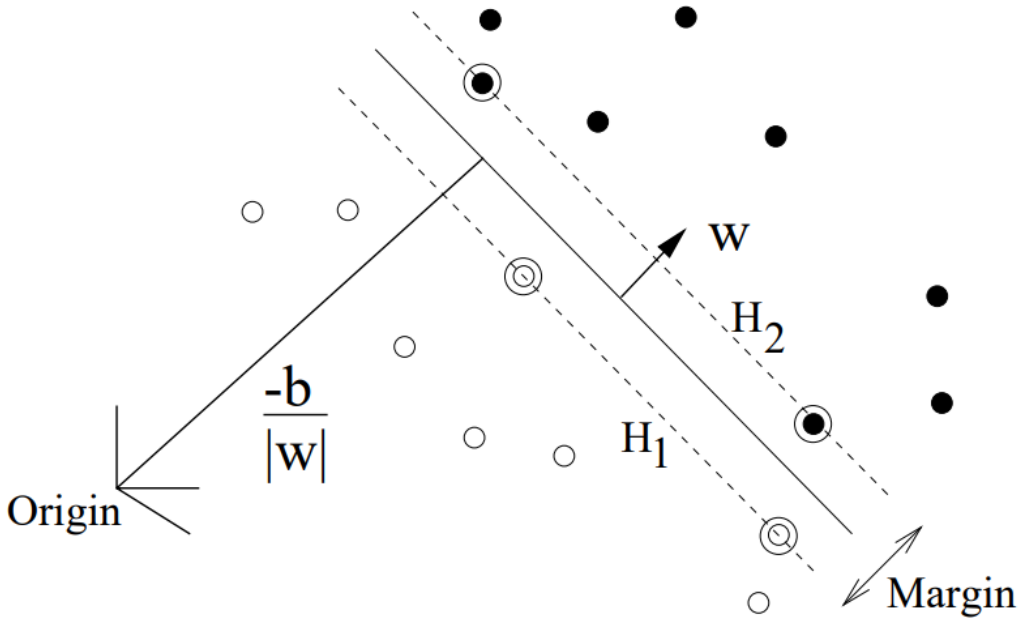
- **Hard-margin formulation**

  : Do not allow any errors, no training points fall between H_1 and H_2

$$\min J(\boldsymbol{w}, b) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$ ← *maximize the margin*

$$\text{subject to } y_i(\boldsymbol{w}^T\boldsymbol{x}_\text{i} + \text{b}) \geq 1, \forall i$$ ← *all training data points are outside the margin*

What are parameters?
What are hyperparameters?



(Linear) Support Vector Machine

- **Soft-margin formulation**

  : Allow some errors by introducing slack variables $\xi_i \geq 0$

maximize the margin

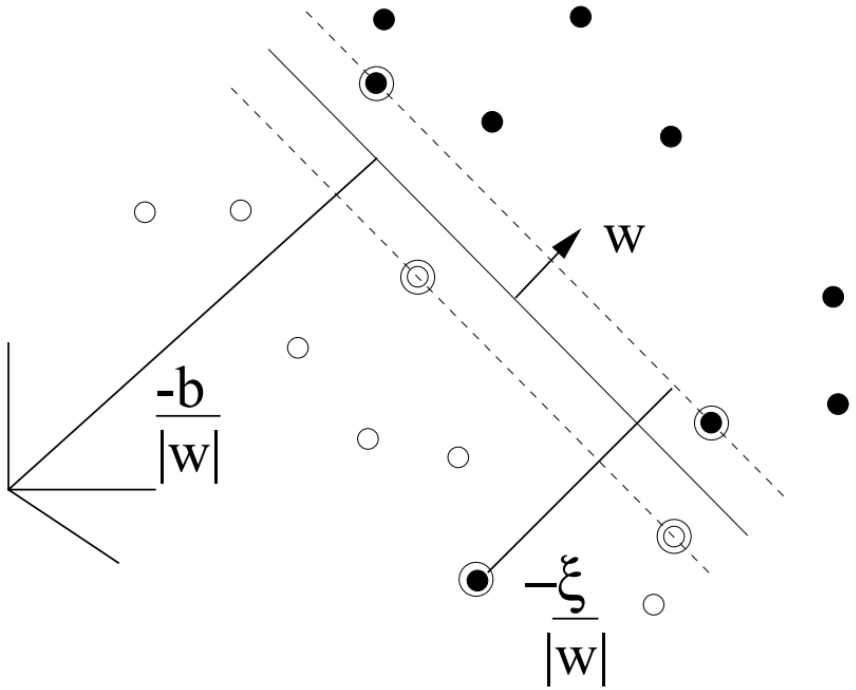
$$\min J(\boldsymbol{w}, b, \xi_i) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \mathrm{C}\sum_i \xi_i$$

minimize empirical risk (hinge loss)
trade-off hyperparameter C

$$L(\mathrm{y}, \boldsymbol{w}^T\mathbf{x} + b) = \max(0, 1 - y(\boldsymbol{w}^T\mathbf{x} + b))$$

$$\text{subject to } y_i\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{x_i} + \mathrm{b}\right) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, \forall i$$

most training data points are outside the margin, but some are not



What are parameters?
What are hyperparameters?

(Linear) Support Vector Machine

- **Soft-margin formulation**

  : Introducing Lagrangian multiplier $\alpha \geq 0$, $\mu \geq 0$ (Wolfe duality)

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} - \sum_i \mu_i \xi_i$$

  - **Karush-Kuhn-Tucker(KKT) conditions**

$$\frac{\partial L_P}{\partial w_\nu} = w_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0$$
$$\frac{\partial L_P}{\partial b} = -\sum_i \alpha_i y_i = 0$$
$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0$$

Stationary conditions

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0$$
$$\xi_i \geq 0$$

Primal feasibility conditions

$$\alpha_i \geq 0$$
$$\mu_i \geq 0$$

Dual feasibility conditions

$$\alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} = 0$$
$$\mu_i \xi_i = 0$$

Complementary slackness conditions

(Linear) Support Vector Machine

- **Soft-margin formulation**

  : Dual Problem (**Quadratic Programming**) → Use a QP Solver!

usually, $O(n^3)$ *complexity*
what if n is very large?

$$\max L(\alpha_i) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$\text{subject to} \quad \sum_i \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \forall i$$

Convex optimization
→ Global optimum is guaranteed

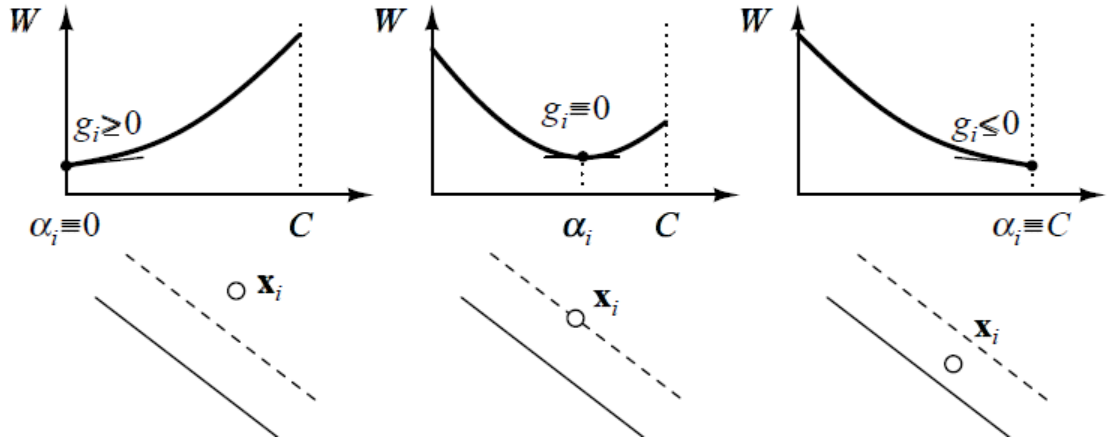
(Linear) Support Vector Machine

- **Soft-margin formulation**

  : After obtaining the optimal parameters $\mathbf{w}^*$, b^*, *how?*

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \qquad b^* = \frac{1}{y_{sv}} - \mathbf{w}^{*T} \boldsymbol{x}_{sv} = \frac{1}{y_{sv}} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_{sv}$$

- **The trained model** , where $(\boldsymbol{x}_{sv}, y_{sv}) \in \{(\boldsymbol{x}_i, y_i) | 0 < \alpha_i < C\}$
  - $f(\boldsymbol{x}) = \text{sign}\left(\mathbf{w}^{*T} \boldsymbol{x} + b^*\right) = \text{sign}\left(\sum_{(\boldsymbol{x}_i, y_i) \in D} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + b^*\right)$
  - Let $D_{SV} = \{(\boldsymbol{x}_i, y_i) \in D | \alpha_i > 0\}$, then $f(\boldsymbol{x}) = \text{sign}\left(\sum_{(\boldsymbol{x}_i, y_i) \in D_{SV}} \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + b^*\right)$ (sparse solution)



non-support vectors
$\{(\boldsymbol{x}_i, y_i) | \alpha_i = 0\}$

margin support vectors
$\{(\boldsymbol{x}_i, y_i) | 0 < \alpha_i < C\}$

error support vectors
$\{(\boldsymbol{x}_i, y_i) | \alpha_i = C\}$

what are support vectors?

(Nonlinear) Support Vector Machine

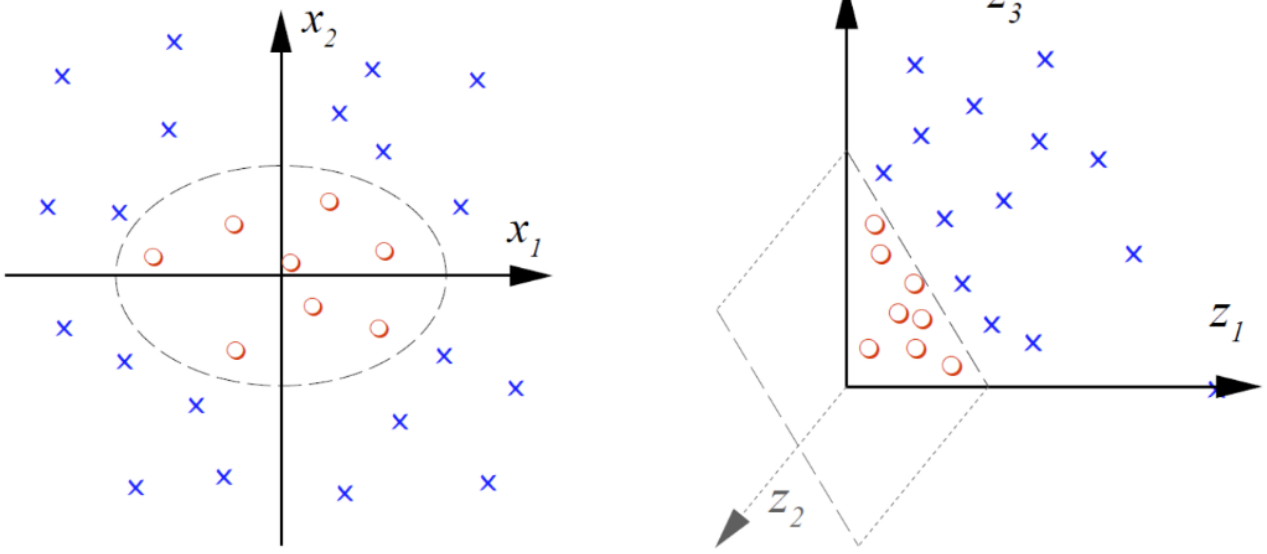
- **SVM for Non-linear Classification: Kernel Trick**

  : Use a function φ that maps the data into a higher dimensional space.

  - Replace $\boldsymbol{x}_i$ by $\varphi(\boldsymbol{x}_i)$

  - **Example**: $\varphi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1x_2, x_2^2)$$



(Nonlinear) Support Vector Machine

- **SVM for Non-linear Classification: Kernel Trick**

  : If there is a "kernel function" *k* that defines inner products in the transformed space, such that $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$, then **we don't have to know φ at all, but use k instead.**

  - Replace $\boldsymbol{x}_i^T \boldsymbol{x}_j$ by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - Not all functions can be kernels (Mercer's theorem)

  why do we need to solve a dual problem instead of a primal problem? (e.g., the number of parameters when using the RBF kernel.)

- **Examples of Kernel functions**
  - **Linear Kernel** $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$
  - **Polynomial Kernel** $k(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^p$
  - **Tanh Kernel** $k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a + b\boldsymbol{x}^T \boldsymbol{x}')$
  - **RBF Kernel** $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma(\boldsymbol{x} - \boldsymbol{x}')^2)$ ← most popular, maps the data into infinite dimensional space

    * Automatic Relevance Determination

    * Multiple Kernel Learning

If we use a kernel, what are parameters and hyperparameters?

(Nonlinear) Support Vector Machine

- **Soft-margin formulation**

  : Dual Problem (**Quadratic Programming**) → Use a QP Solver!

usually, O(n^3) *complexity*
what if n is very large?

$$\max L(\alpha_i) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$\text{subject to} \quad \sum_i \alpha_i y_i = 0$$
$$0 \le \alpha_i \le C, \forall i$$

Convex optimization
→ Global optimum is guaranteed

(Nonlinear) Support Vector Machine

- **Soft-margin formulation**

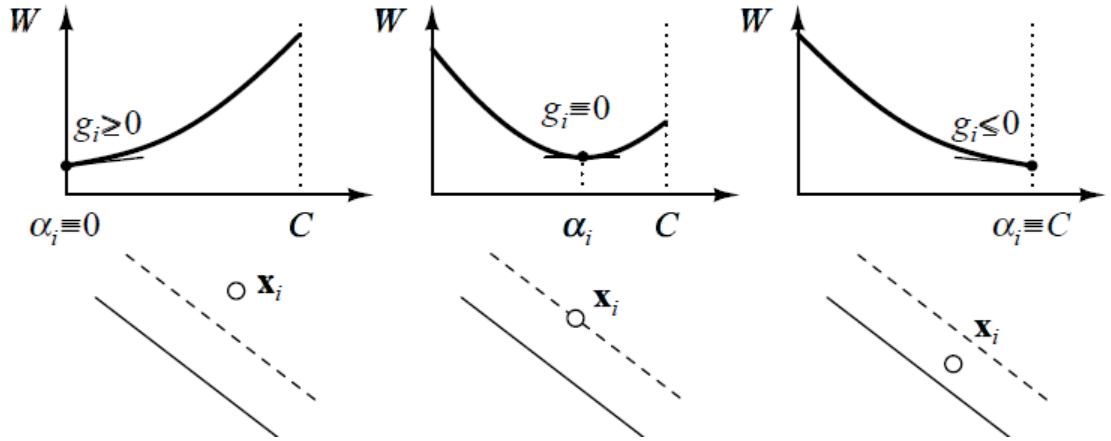
  : After obtaining the optimal parameters $\mathbf{w}^*$, b^*,

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i y_i \varphi(\boldsymbol{x}_i) \qquad b^* = \frac{1}{y_{sv}} - \mathbf{w}^{*T} \boldsymbol{x}_{sv} = \frac{1}{y_{sv}} - \sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}_{sv})$$

- **The trained model**

  , where $(\boldsymbol{x}_{sv}, y_{sv}) \in \{(\boldsymbol{x}_i, y_i) | 0 < \alpha_i < C\}$

  - $f(\boldsymbol{x}) = \text{sign}\left(\mathbf{w}^{*T} \varphi(\boldsymbol{x}) + b^*\right) = \text{sign}\left(\sum_{(\boldsymbol{x}_i, y_i) \in D} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right)$
  - Let $D_{SV} = \{(\boldsymbol{x}_i, y_i) \in D | \alpha_i > 0\}$, then $f(\boldsymbol{x}) = \text{sign}\left(\sum_{(\boldsymbol{x}_i, y_i) \in D_{SV}} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right)$ (sparse solution)



non-support vectors
$\{(\boldsymbol{x}_i, y_i) | \alpha_i = 0\}$

margin support vectors
$\{(\boldsymbol{x}_i, y_i) | 0 < \alpha_i < C\}$

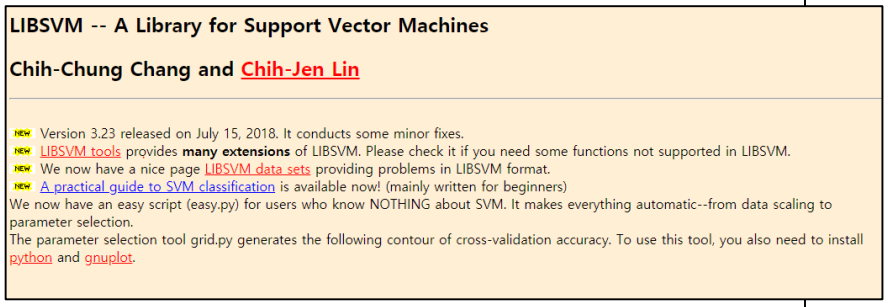
error support vectors
$\{(\boldsymbol{x}_i, y_i) | \alpha_i = C\}$

what are support vectors?

Hyperparameters of SVM

- **Practical Guideline when using SVM with RBF Kernel**

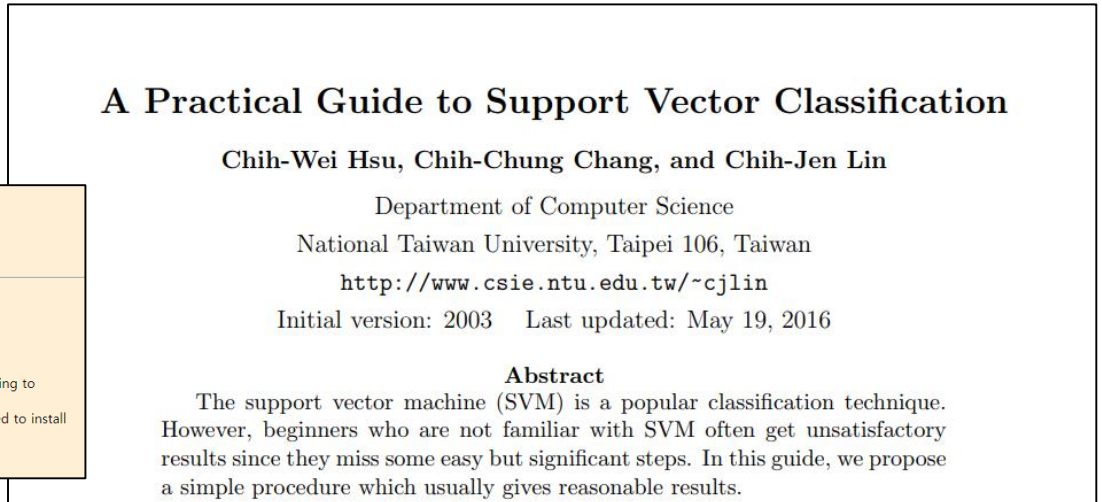
We recommend a "grid-search" on C and γ using cross-validation. Various pairs of (C, γ) values are tried and the one with the best cross-validation accuracy is picked. We found that trying exponentially growing sequences of C and γ is a practical method to identify good parameters (for example, $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$, $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$).



LIBSVM -- A Library for Support Vector Machines

Chih-Chung Chang and Chih-Jen Lin

NEW Version 3.23 released on July 15, 2018. It conducts some minor fixes.
NEW LIBSVM tools provides **many extensions** of LIBSVM. Please check it if you need some functions not supported in LIBSVM.
NEW We now have a nice page LIBSVM data sets providing problems in LIBSVM format.
NEW A practical guide to SVM classification is available now! (mainly written for beginners)
We now have an easy script (easy.py) for users who know NOTHING about SVM. It makes everything automatic--from data scaling to parameter selection.
The parameter selection tool grid.py generates the following contour of cross-validation accuracy. To use this tool, you also need to install python and gnuplot.



A Practical Guide to Support Vector Classification

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin

Department of Computer Science
National Taiwan University, Taipei 106, Taiwan
http://www.csie.ntu.edu.tw/~cjlin

Initial version: 2003 Last updated: May 19, 2016

Abstract

The support vector machine (SVM) is a popular classification technique. However, beginners who are not familiar with SVM often get unsatisfactory results since they miss some easy but significant steps. In this guide, we propose a simple procedure which usually gives reasonable results.

Fast Training of SVM

- **Main weakness of SVM: its high training complexity**
  - Time complexity: $O(n^3)$
  - Space complexity: $O(n^2)$, where n is the number of training data points

- **Reducing the complexity**
  - Strategy 1: Improving the efficiency of QP solving
    - Efficient gradient descent, Chunking, Sequential Minimal Optimization (SMO), …
  - Strategy 2: Reducing the number of training data points: Shrinking
    - Eliminating non-support vectors early during or before the optimization process
    - Clustering technique, Filtering, Expected margin, …
  - Strategy 3: Closed-form solution (without QP)
    - Least-Squares Support Vector Machine (lacks sparseness → sparse approximation)
  - Strategy 4: Use something else
    - Tree ensembles, Neural network with SGD, …

SVM and Neural Network

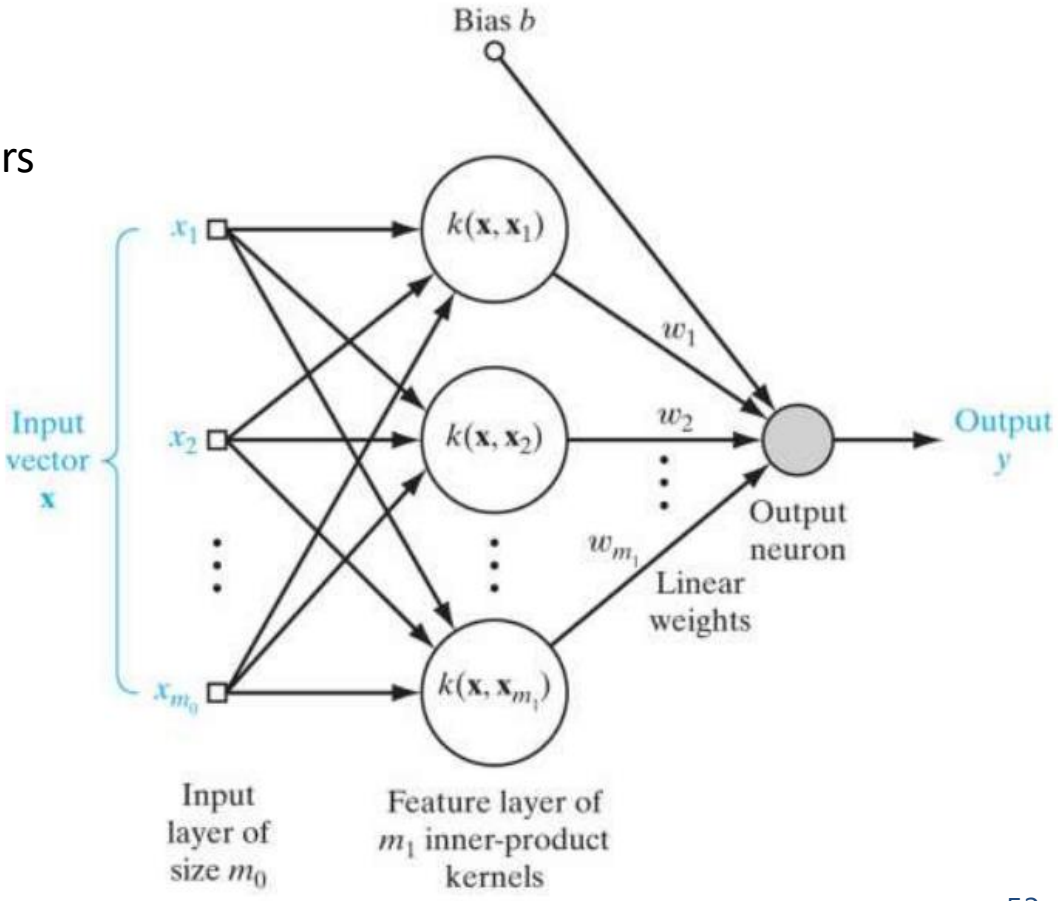
- **SVM**
  - Deterministic (Global optimum)
  - Good Generalization Properties based on Structural Risk Minimization
  - Hard to learn – learned in batch mode using QP
  - Non-linearity by kernel functions
- **Neural Network**
  - Non-deterministic (Local optima)
  - Good Generalization but doesn't have strong mathematical foundation (Empirical Risk Minimization)
  - Can easily be learned in incremental fashion
  - Non-linearity by multiple layers of non-linear activation functions

SVM and Neural Network

- **SVM**

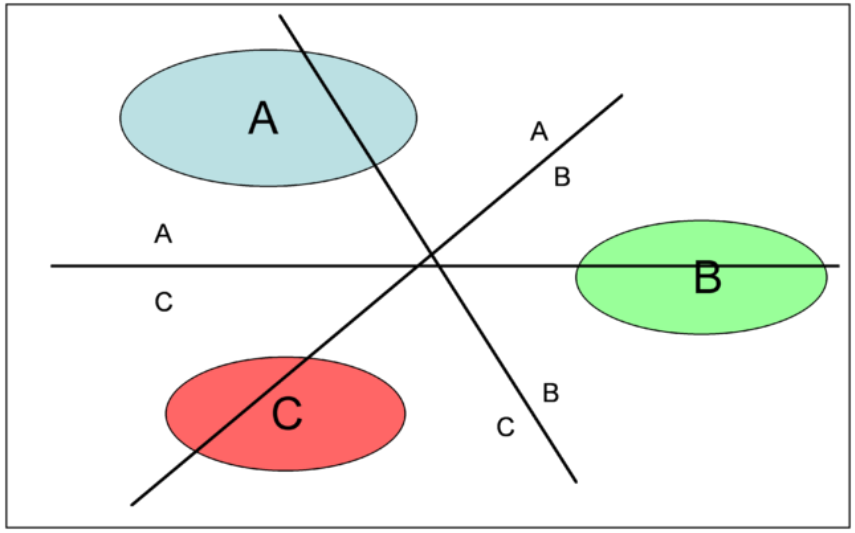
  : $f(\boldsymbol{x}) = \text{sign}\left(\sum_{(\boldsymbol{x}_i, y_i) \in D_{SV}} \alpha_i y_i \, \boldsymbol{x}_i^T \boldsymbol{x} + b^*\right)$

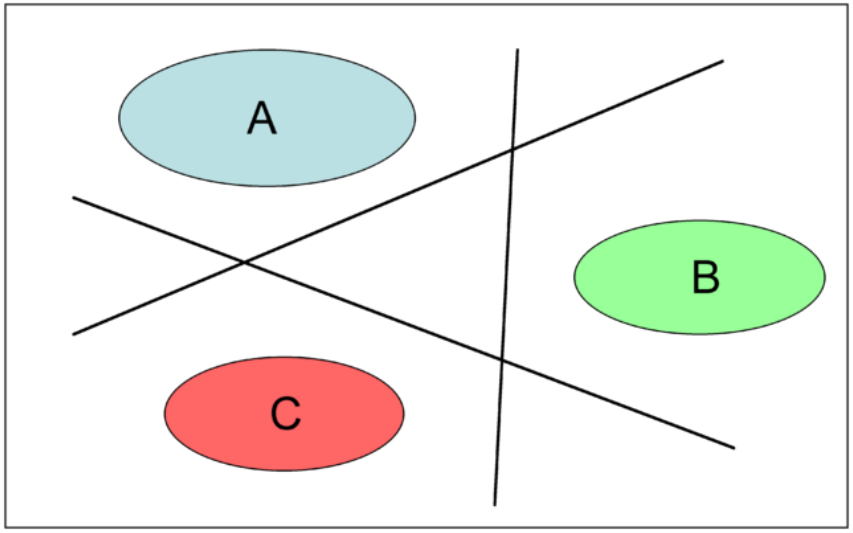
- **Neural Network**
  - No. hidden units ∽ No. support vectors
  - Activation function ∽ Kernel function
  - No. hidden layers = 1
  - But, different training procedure



Multi-class Classification with SVM

- **For a *c*-class classification problem,**
  - **One-Against-One Approach**
    : $c(c-1)/2$ SVMs are trained for all possible class pairs.
    : A test data point is classified based on majority voting by $c(c-1)/2$ SVMs
  - **One-Against-Rest Approach**
    : *c* base classifiers are trained, each of which separates one class from the remaining classes.
    : A test data point is classified as the class whose corresponding SVM gives the highest score.





* No-free-lunch Theorem

- **The "no-free-lunch theorem" for machine learning** (Wolpert, 1996)
  : No machine learning algorithm is universally better than any other on every task.

- The goal of machine learning research is **not to seek a universal learning algorithm or the absolute best learning algorithm.**
- We must design our machine learning algorithms **to perform well on a specific task**.