

# CS 5/7320

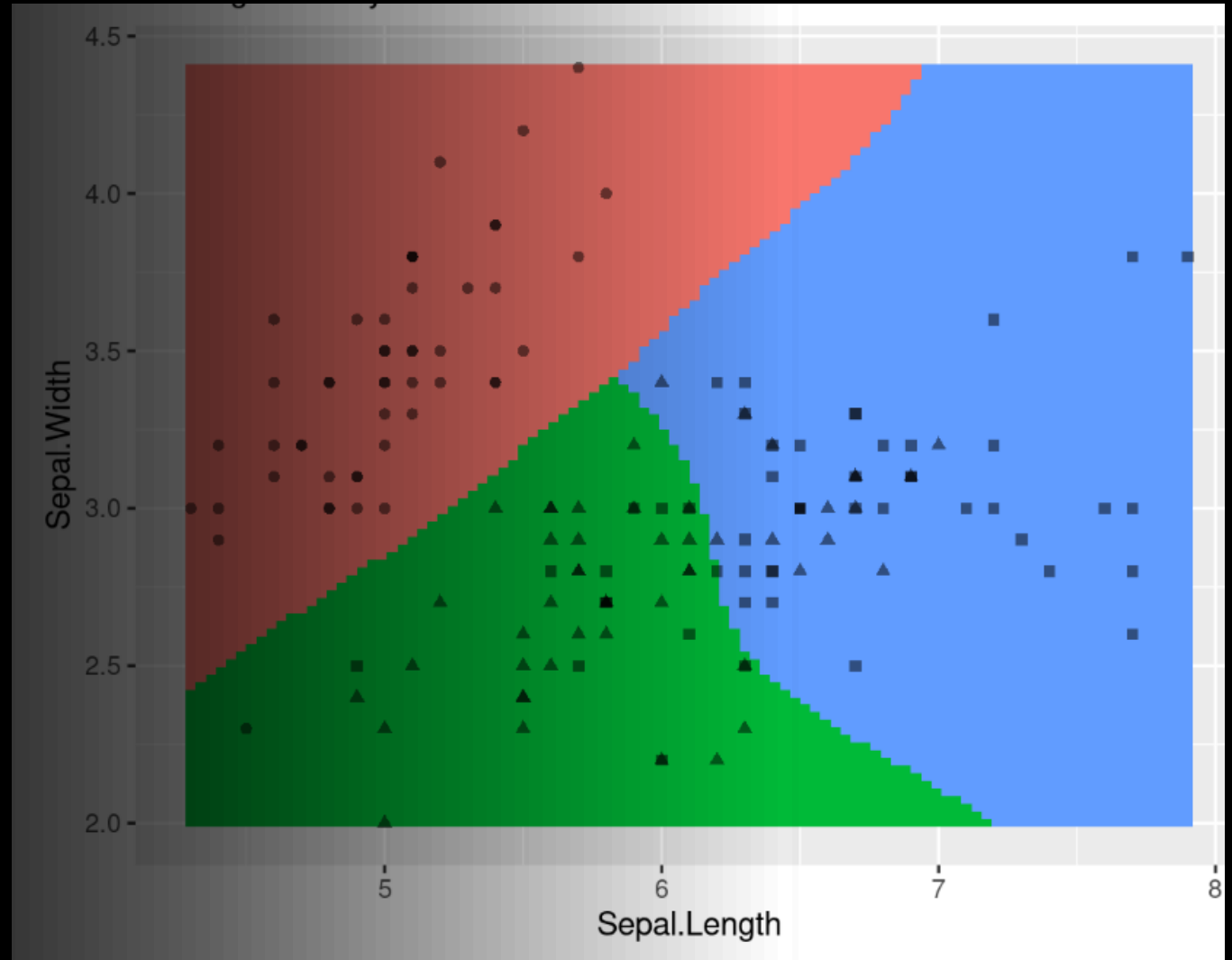
## Artificial Intelligence

### Learning from Examples

AIMA Chapter 19

Slides by Michael Hahsler

Based on slides by Dan Klein, Pieter Abbeel,  
Sergey Levine and A. Farhadi. All CS188  
materials are at <http://ai.berkeley.edu>  
with figures from the AIMA textbook.



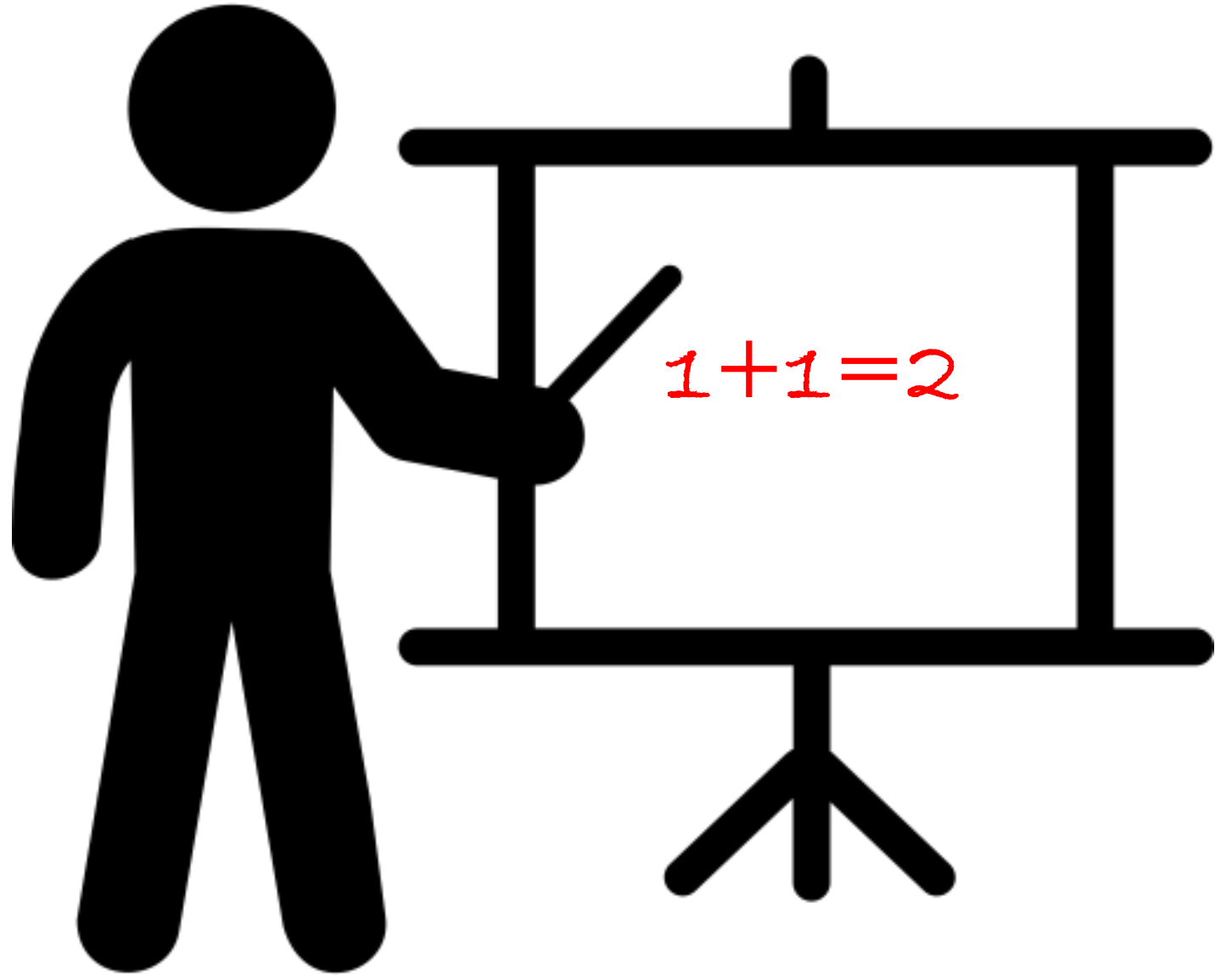
This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

# Learning from Examples: Machine Learning

- **Up until now:** hand-craft algorithms to make rational/optimal or at least good decisions. Examples: Search strategies, heuristics.
- **Learning:** Improve performance after making observations about the world. That is, learn what works and what doesn't.
- **Machine learning:** how to build a model from data/experience
  - Supervised Learning: Learn a function to map input to output. Examples:
    - Use a naïve Bayesian classifier to distinguish between spam/no spam
    - Learn a playout policy to simulate games (current board -> good move)
  - Reinforcement Learning: Learn from rewards/punishment (e.g., winning a game).
- **Learning vs. hard coding the agent function**
  - Designer cannot anticipate all possible future situations.
  - Designer may have examples but does not know how to program a solution.

# Supervised Learning

---



# Supervised Learning

- Examples

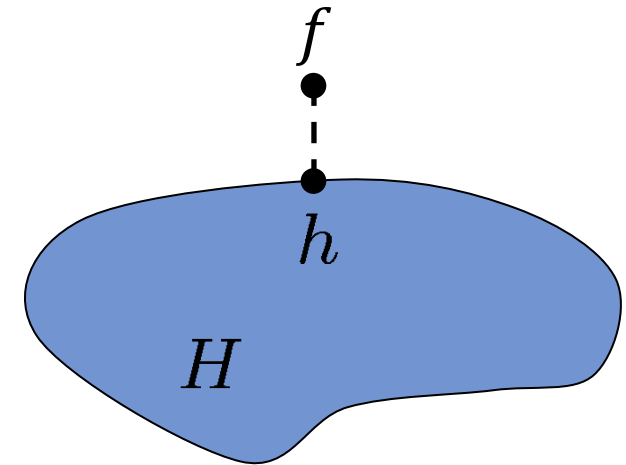
- Input-output pairs:  $E = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_N, y_N)$ .
- We assume that the examples are produced iid (with noise and errors) from a target function  $y = f(x)$ .

- Learning problem

- Given a hypothesis space  $H$
- Find a hypothesis  $h \in H$  such that  $\hat{y}_i = h(x_i) \approx y_i$
- That is, we want to approximate  $f$  by  $h$  using  $E$ .

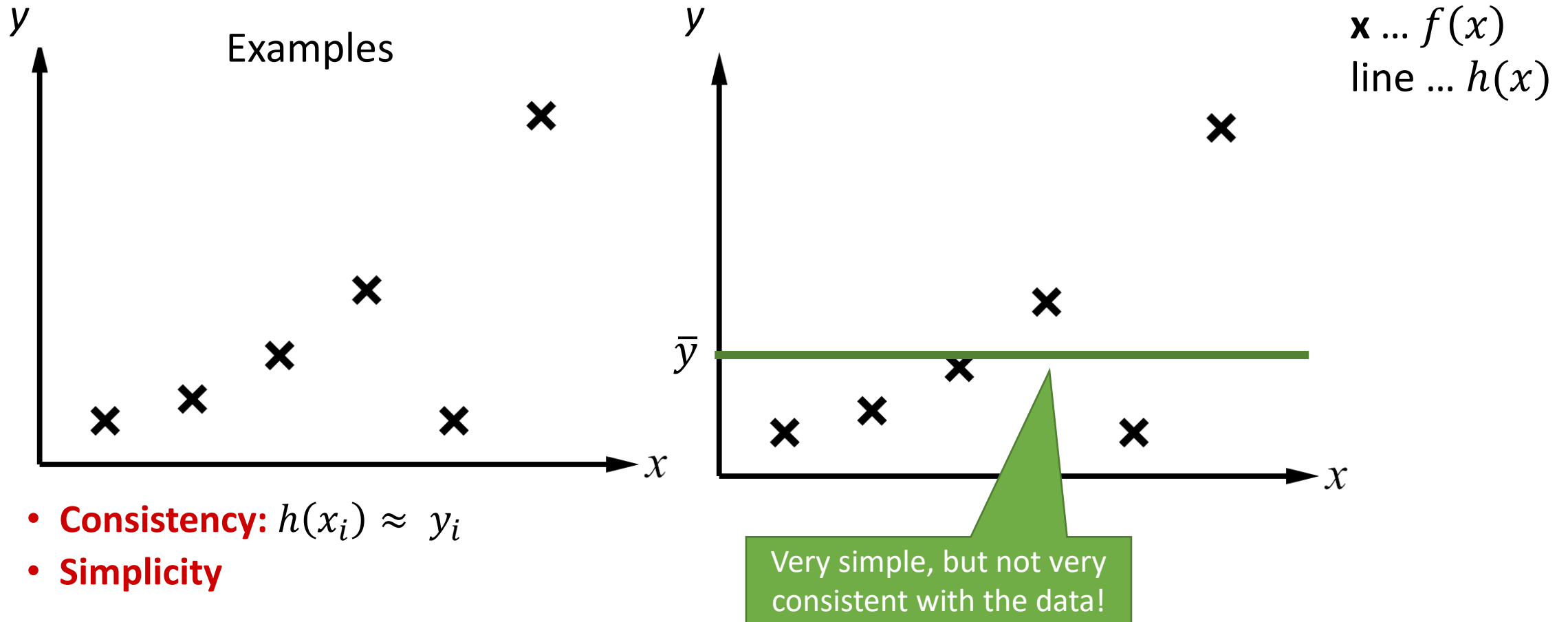
- Includes

- Classification (outputs = class labels). E.g.  $x$  is an email and  $f(x)$  is spam / ham
- Regression (outputs = real numbers). E.g.  $x$  is a house and  $f(x)$  is its selling price



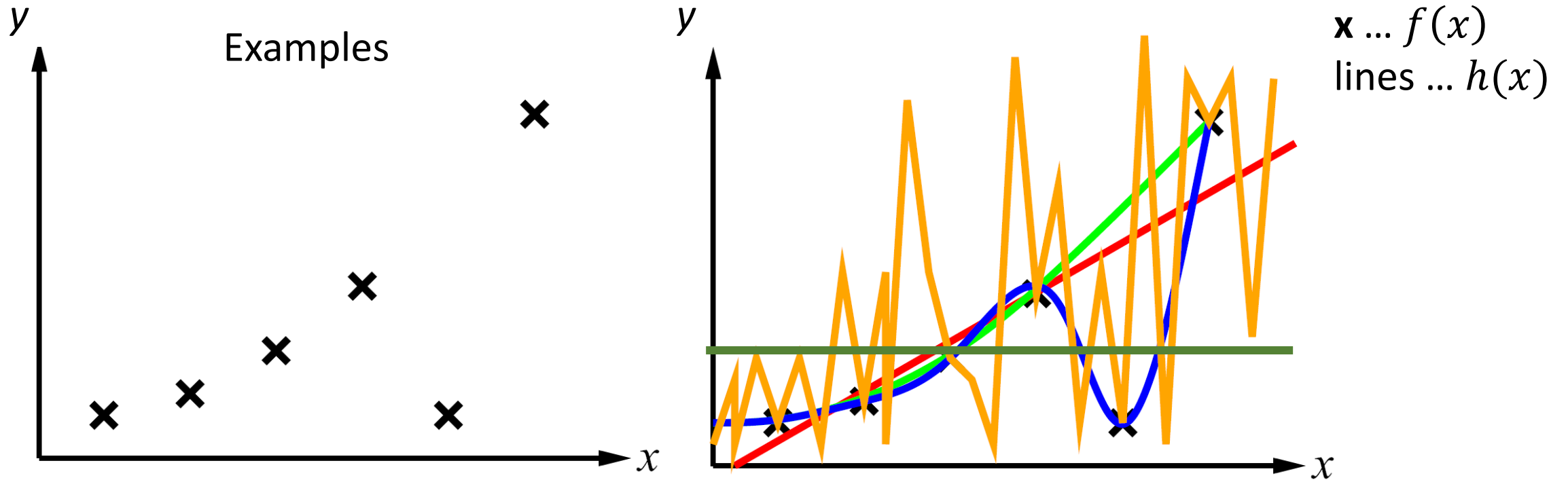
# Consistency vs. Simplicity

Example: Curve fitting (regression, function approximation)



# Consistency vs. Simplicity

Example: Curve fitting (regression, function approximation)



- **Consistency:**  $h(x_i) \approx y_i$
- **Simplicity**

# Consistency and Loss

**Goal of learning:** Find a hypothesis that makes good predictions that are consistent with the examples  $E = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_N, y_N)$ .

That is,  $\hat{y} = h(x) \approx y$ .

- **Measure mistakes:** Loss function  $L(y, \hat{y})$

- Absolute-value loss  $L_1(y, \hat{y}) = |y - \hat{y}|$
- Squared-error loss  $L_2(y, \hat{y}) = (y - \hat{y})^2$
- 0/1 loss  $L_{0/1}(y, \hat{y}) = 0 \text{ if } y = \hat{y}, \text{ else } 1$
- Log loss and others...

} For Regression

← For Classification

- **Empirical loss:** average loss over the N examples in the dataset

$$EmpLoss_{L,E}(h) = \frac{1}{|E|} \sum_{(x,y) \in E} L(y, h(x))$$

# Consistency and Loss

- Empirical loss

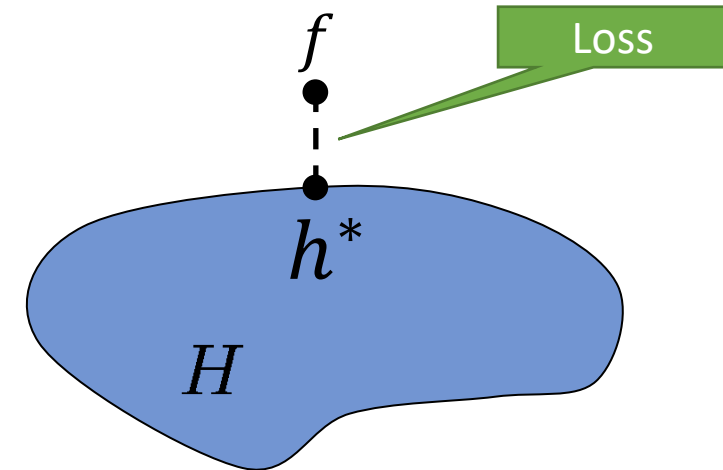
$$EmpLoss_{L,E}(h) = \frac{1}{|E|} \sum_{(x,y) \in E} L(y, h(x))$$

- Learning the best hypothesis (approximation)

$$h^* = \operatorname{argmin}_{h \in H} EmpLoss_{L,E}(h)$$

- Reasons for  $h^* \neq f$

- a) Realizability:  $f \notin H$
- b)  $f$  is nondeterministic or examples are noisy
- c) Computationally intractable to search all  $H$





# Example: Bayes Classifier

For 0/1 loss, the empirical loss is minimized by the model that predicts for each  $x$  the most likely class  $y$ .

$$h(x)^* = \operatorname{argmax}_y P(Y = y \mid X = x) = \operatorname{argmax}_y \frac{P(x \mid y) P(y)}{P(x)} = \operatorname{argmax}_y P(x \mid y) P(y)$$

**Optimality:** The **Bayes Classifier is optimal** and guarantees the lowest possible error rate called the **Bayes error rate**.

**Issue:**  $P(x \mid y) P(y) = P(x, y)$

Needs the complete joint probability which requires in the general case a probability table with one entry for each possible value for  $x$ . That is why we often use the naïve Bayes classifier, which is not optimal since it assumes conditional independence between features.

# Simplicity

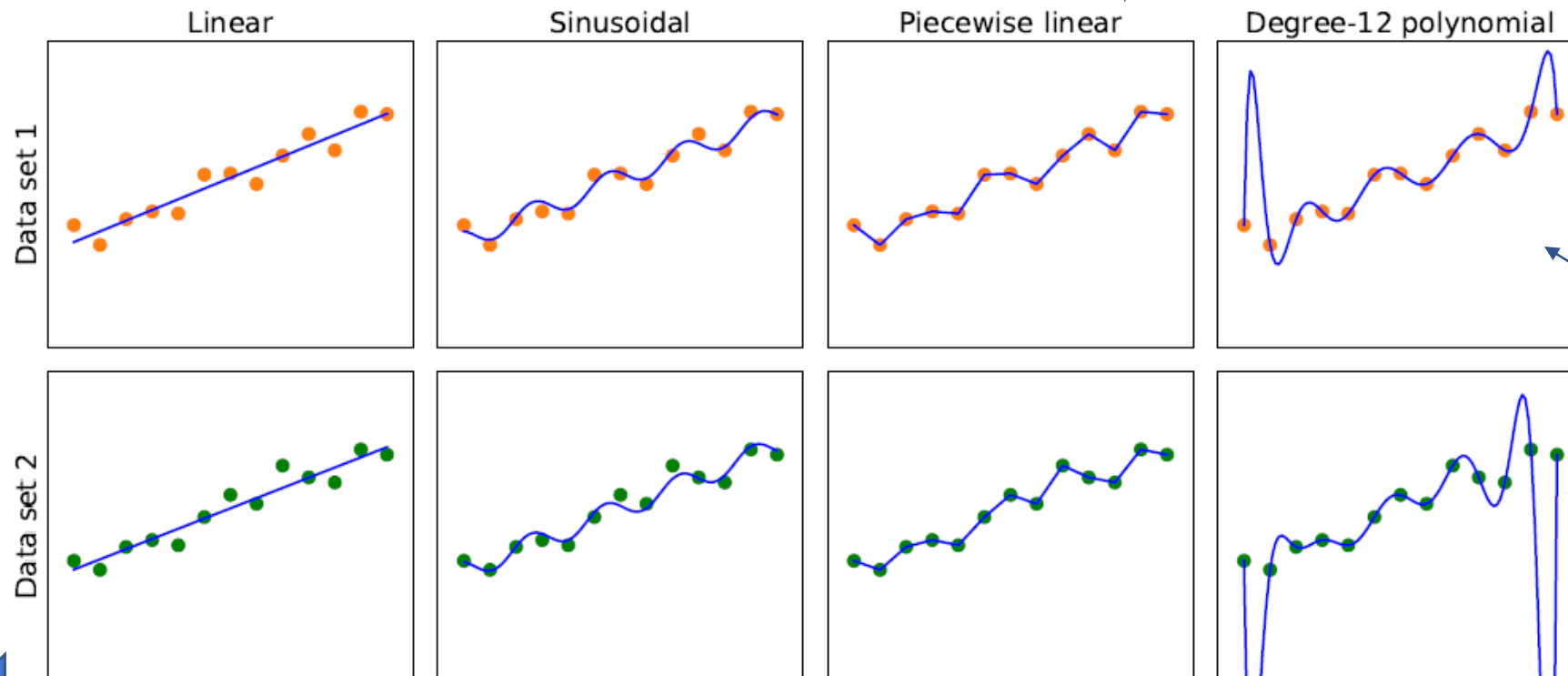
- **Ease of use:** Simpler hypotheses have fewer parameters and are easier to estimate.
- **Generalization:** How well does the hypothesis perform on new data?
  - We do not want the model to be too specific to the training examples (an issue called **overfitting**).
  - Simpler models typically generalize better to new examples.
- **How to achieve simplicity?**
  - a) Restrict  $H$  to simple models (e.g., independence assumption, linear models)
  - b) Feature selection (use fewer variables)
  - c) Regularization (penalize for complexity)

$$h^* = \operatorname{argmin}_{h \in H} \left[ \operatorname{EmpLoss}_{L,E}(h) + \underbrace{\lambda \operatorname{Complexity}(h)}_{\text{Penalty term}} \right]$$

Overfitting

# Model Selection: Bias vs. Variance

← Simpler      More consistent →



Two samples from the same function  $f$  (points) with the learned function  $h$  (lines).

High

**Bias:** restrictions by the model class

Low

Low **Variance:** difference in the model due to slightly different data. high



Data

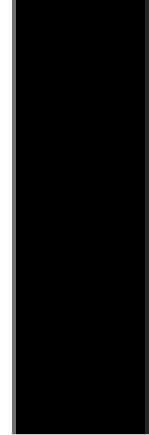
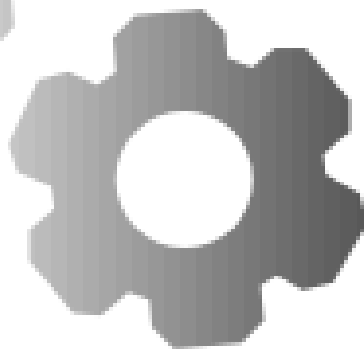
---

01

0110

0001

01101



# The Dataset

Features  
Variables  
Attributes

## Class Label

Examples  
Instances  
Observations

Example	Input Attributes										Output
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
<b>x<sub>1</sub></b>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>French</i>	<i>0–10</i>	<i>y<sub>1</sub> = Yes</i>
<b>x<sub>2</sub></b>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Thai</i>	<i>30–60</i>	<i>y<sub>2</sub> = No</i>
<b>x<sub>3</sub></b>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>Some</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Burger</i>	<i>0–10</i>	<i>y<sub>3</sub> = Yes</i>
<b>x<sub>4</sub></b>	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Thai</i>	<i>10–30</i>	<i>y<sub>4</sub> = Yes</i>
<b>x<sub>5</sub></b>	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Full</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>French</i>	<i>&gt;60</i>	<i>y<sub>5</sub> = No</i>
<b>x<sub>6</sub></b>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$</i>	<i>Yes</i>	<i>Yes</i>	<i>Italian</i>	<i>0–10</i>	<i>y<sub>6</sub> = Yes</i>
<b>x<sub>7</sub></b>	<i>No</i>	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>None</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Burger</i>	<i>0–10</i>	<i>y<sub>7</sub> = No</i>
<b>x<sub>8</sub></b>	<i>No</i>	<i>No</i>	<i>No</i>	<i>Yes</i>	<i>Some</i>	<i>\$\$</i>	<i>Yes</i>	<i>Yes</i>	<i>Thai</i>	<i>0–10</i>	<i>y<sub>8</sub> = Yes</i>
<b>x<sub>9</sub></b>	<i>No</i>	<i>Yes</i>	<i>Yes</i>	<i>No</i>	<i>Full</i>	<i>\$</i>	<i>Yes</i>	<i>No</i>	<i>Burger</i>	<i>&gt;60</i>	<i>y<sub>9</sub> = No</i>
<b>x<sub>10</sub></b>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$\$\$</i>	<i>No</i>	<i>Yes</i>	<i>Italian</i>	<i>10–30</i>	<i>y<sub>10</sub> = No</i>
<b>x<sub>11</sub></b>	<i>No</i>	<i>No</i>	<i>No</i>	<i>No</i>	<i>None</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Thai</i>	<i>0–10</i>	<i>y<sub>11</sub> = No</i>
<b>x<sub>12</sub></b>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Full</i>	<i>\$</i>	<i>No</i>	<i>No</i>	<i>Burger</i>	<i>30–60</i>	<i>y<sub>12</sub> = Yes</i>

Alternative

Hungry  
Patrons

Reservation

Wait time

Find a hypothesis (called “model”) to predict the class given the features.

# Feature Engineering

- Add information sources as new variables to the model.
- Add derived features that help the classifier (e.g.,  $x^2$ ).
- Example for Spam detection: In addition to words
  - Have you emailed the sender before?
  - Have 1000+ other people just gotten the same email?
  - Is the header information consistent?
  - Is the email in ALL CAPS?
  - Do inline URLs point where they say they point?
  - Does the email address you by (your) name?
- **Feature Selection:** Which features should be used in the model is a model selection problem (choose between models with different features).



# Training and Testing



# Model Evaluation

We want to test how well the model will perform on new data (i.e., how well it generalizes).

- **Testing loss**: Calculate the empirical loss for predictions on a testing data set  $T$  that is different from the data used for training.

$$EmpLoss_{L,T}(h) = \frac{1}{|T|} \sum_{(x,y) \in T} L(y, h(x))$$

- For classification we often use the **accuracy** measure, the proportion of correctly classified test examples.

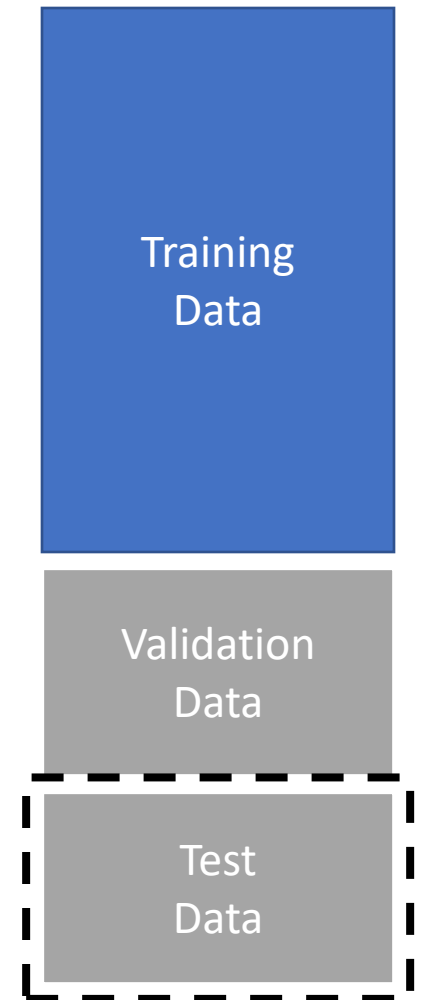
$$accuracy(h, T) = \frac{1}{|T|} \sum_{(x,y) \in T} [h(x) = y] = 1 - EmpLoss_{L_{0/1},T}(h)$$

$[c]$  is an indicator function returning 1 if  $c = True$  and otherwise 0



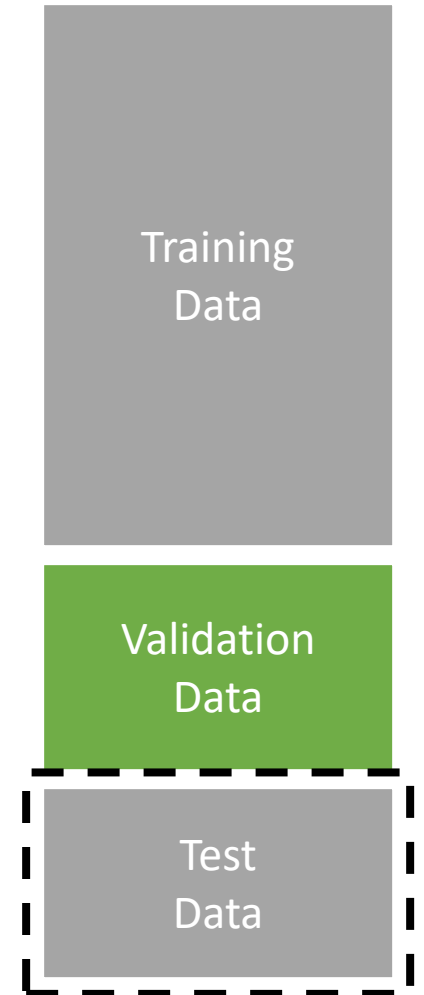
# Training a Model

- Models are “trained” (learned) on **the training data** (a part of the available data).
- Models have
  - Model parameters (the model): E.g., probabilities, weights, factors.
  - Hyperparameters: Choices for the algorithm used for learning. E.g., learning rate, regularization  $\lambda$ , maximal decision tree depth, selected features.
- The “Learner” (algorithm) tries to optimize the model parameters given user-specified hyperparameters.
- We can learn models with different hyperparameters, but how do we know which set of hyperparameters is best?



# Hyperparameter Tuning/Model Selection

- Learn models using the training set and different hyperparameters.
- Often a grid of possible hyperparameter combinations or some greedy search is used.
- Evaluate the models using the **validation data** and choose the model with the best accuracy. Selecting the right type of model, hyperparameters and features is called **model selection**.
- Learn the final model using all training and validation data.
- Notes:
  - The validation set was not used for training, so we get generalization accuracy.
  - If no model selection is necessary, then no validation set is used.



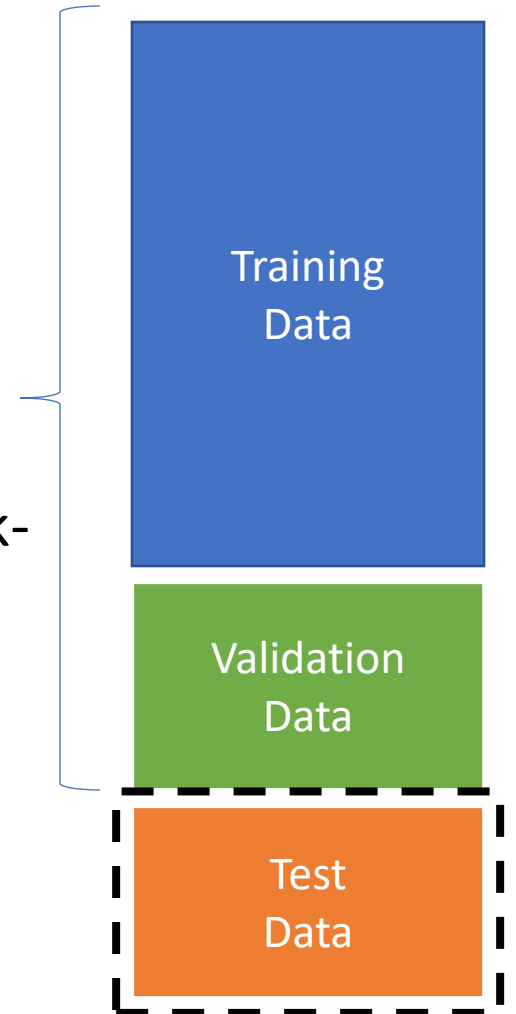
# Testing a Model

- After the hyper parameters are chosen, the final model is evaluated against the test set to estimate the model accuracy.
- Very important: never “peek” at the test set during learning!

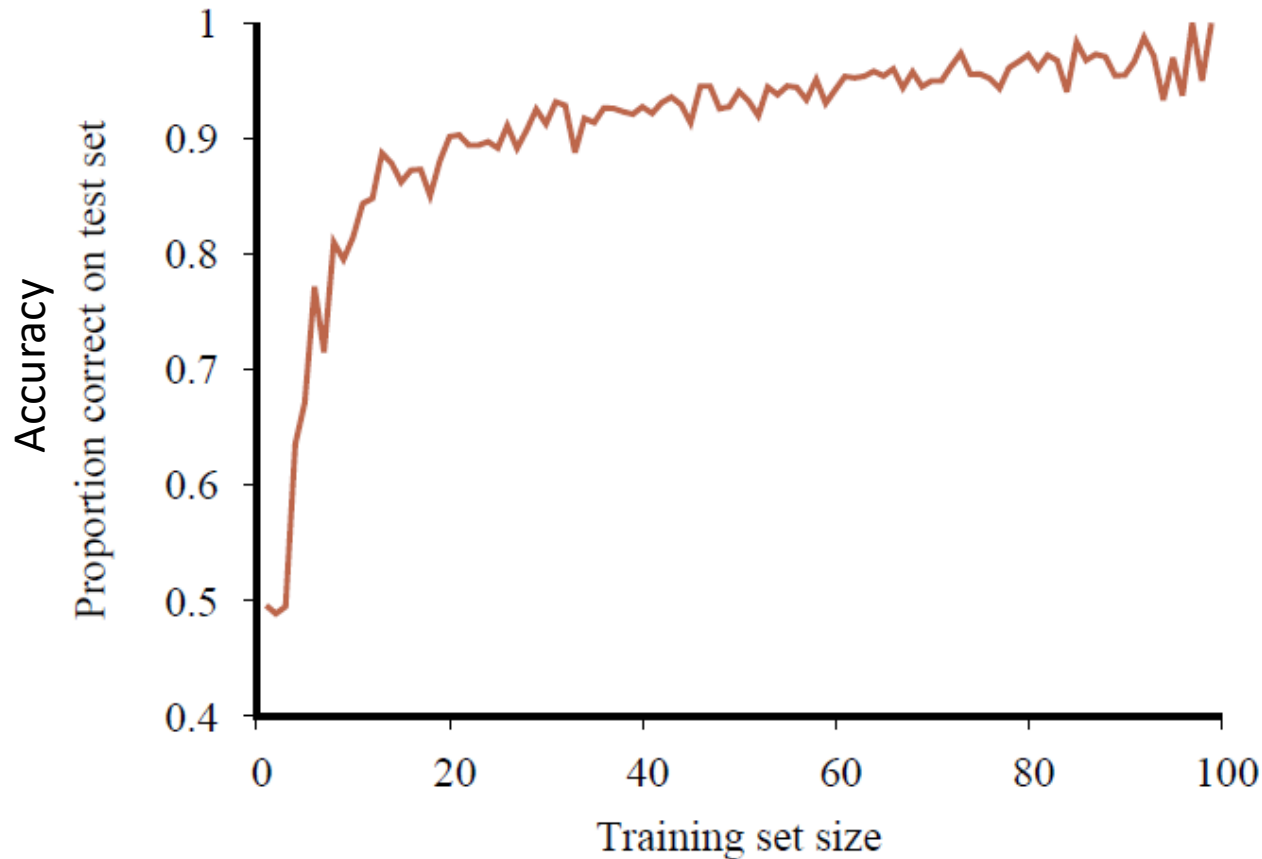


# How to Split the Dataset

- **Random splits:** Split the data randomly in, e.g., 60% training, 20% validation, and 20% testing.
- **k-fold cross validation:** Use training & validation data better
  - split the training & validation data randomly into k folds.
  - For k rounds hold 1 fold back for testing and use the remaining k-1 folds for training.
  - Use the average error/accuracy as a better estimate.
  - Some algorithms/tools do that internally.
- **LOOCV** (leave-one-out cross validation):  $k = n$  used if very little data is available.



# Learning Curve: The Effect the Training Data Size

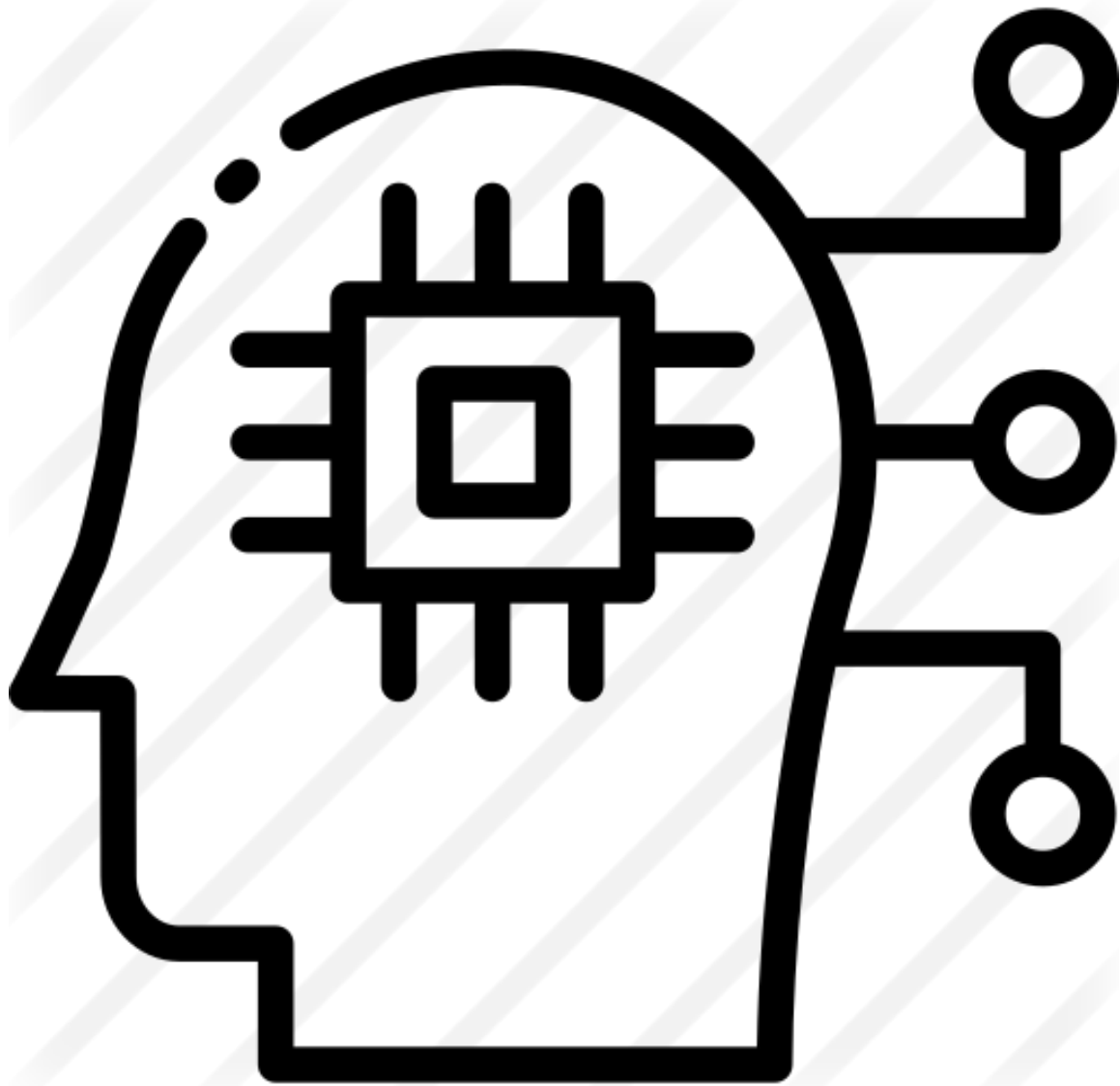


Accuracy of a classifier  
when the amount of  
available training data  
increases.

**More data is better!**

# Comparing to a Baselines

- First step: get a **baseline**
  - Baselines are very simple “straw man” model.
  - Helps to determine how hard the task is.
  - Helps to find out what a “good” accuracy is.
- **Weak baseline:** The most frequent label classifier
  - Gives all test instances whatever label was most common in the training set.
    - Example: For spam filtering, give every message the label “ham.”
  - Accuracy might be very high if the problem is skewed (called class imbalance).
    - Example: Calling everything “ham” gets already 66% right, so a classifier that gets 70% isn’t very good...
- **Strong baseline:** For research, we typically compare to previous work as a baseline.



# Types of Models

Regression: Predict a number

Classification: Predict a label



# Regression: Linear Regression

Model:  $h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1x_{j,1} + \dots + w_nx_{j,n} = \sum_i w_ix_{j,i} = \mathbf{w}^T \mathbf{x}_j$

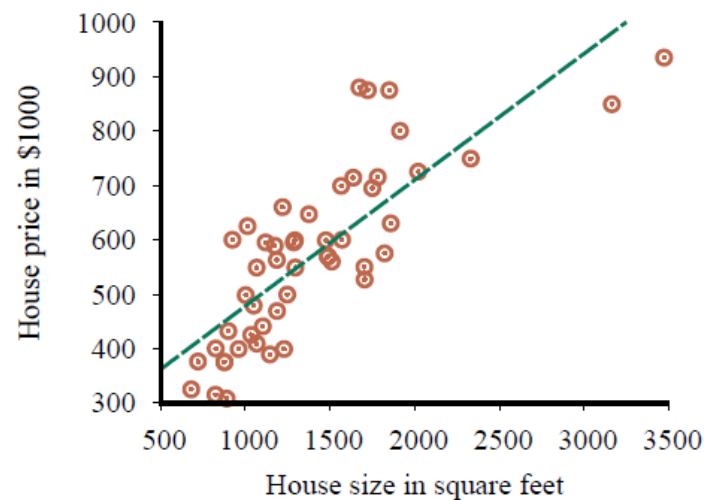
Loss function:  $L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$

Squared error loss over the whole data matrix  $\mathbf{X}$

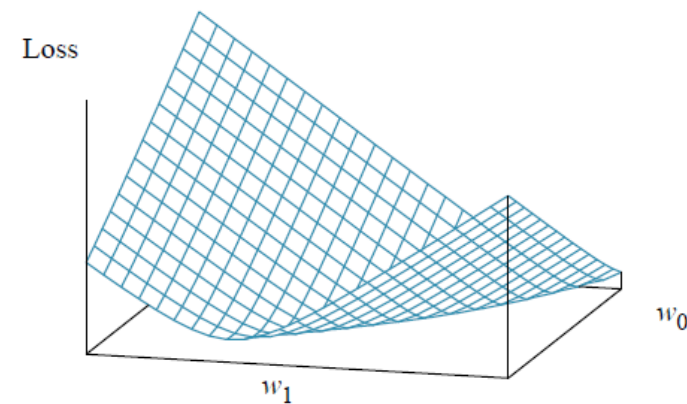
Gradient:  $\nabla_{\mathbf{w}}L(\mathbf{w}) = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$

The gradient  $\nabla_{\mathbf{w}}L(\mathbf{w})$  is a vector of partial derivatives

Solution:  $\mathbf{w}^* = \underbrace{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}_{\text{Pseudo inverse}} \mathbf{y}$



(a)

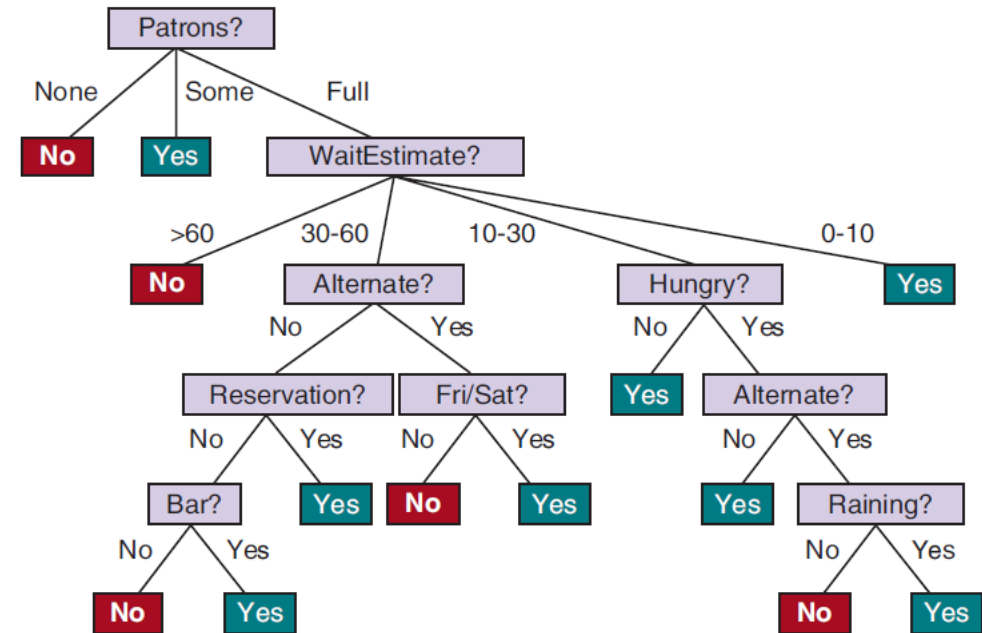


(b)



# Decision Trees

Example	Input Attributes										Output
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
x <sub>1</sub>	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	y <sub>1</sub> = Yes
x <sub>2</sub>	Yes	No	No	Yes	Full	\$	No	No	Thai	30-60	y <sub>2</sub> = No
x <sub>3</sub>	No	Yes	No	No	Some	\$	No	No	Burger	0-10	y <sub>3</sub> = Yes
x <sub>4</sub>	Yes	No	Yes	Yes	Full	\$	Yes	No	Thai	10-30	y <sub>4</sub> = Yes
x <sub>5</sub>	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	y <sub>5</sub> = No
x <sub>6</sub>	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0-10	y <sub>6</sub> = Yes
x <sub>7</sub>	No	Yes	No	No	None	\$	Yes	No	Burger	0-10	y <sub>7</sub> = No
x <sub>8</sub>	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0-10	y <sub>8</sub> = Yes
x <sub>9</sub>	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	y <sub>9</sub> = No
x <sub>10</sub>	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10-30	y <sub>10</sub> = No
x <sub>11</sub>	No	No	No	No	None	\$	No	No	Thai	0-10	y <sub>11</sub> = No
x <sub>12</sub>	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30-60	y <sub>12</sub> = Yes



- A sequence of decisions represented as a tree.
- Many implementations that differ by
  - How to select features to split?
  - When to stop splitting?
  - Is the tree pruned?

- Approximates a Bayesian classifier by

$$h(x) = \underset{y}{\operatorname{argmax}} P(Y = y \mid \text{leafNodeMatching}(x))$$

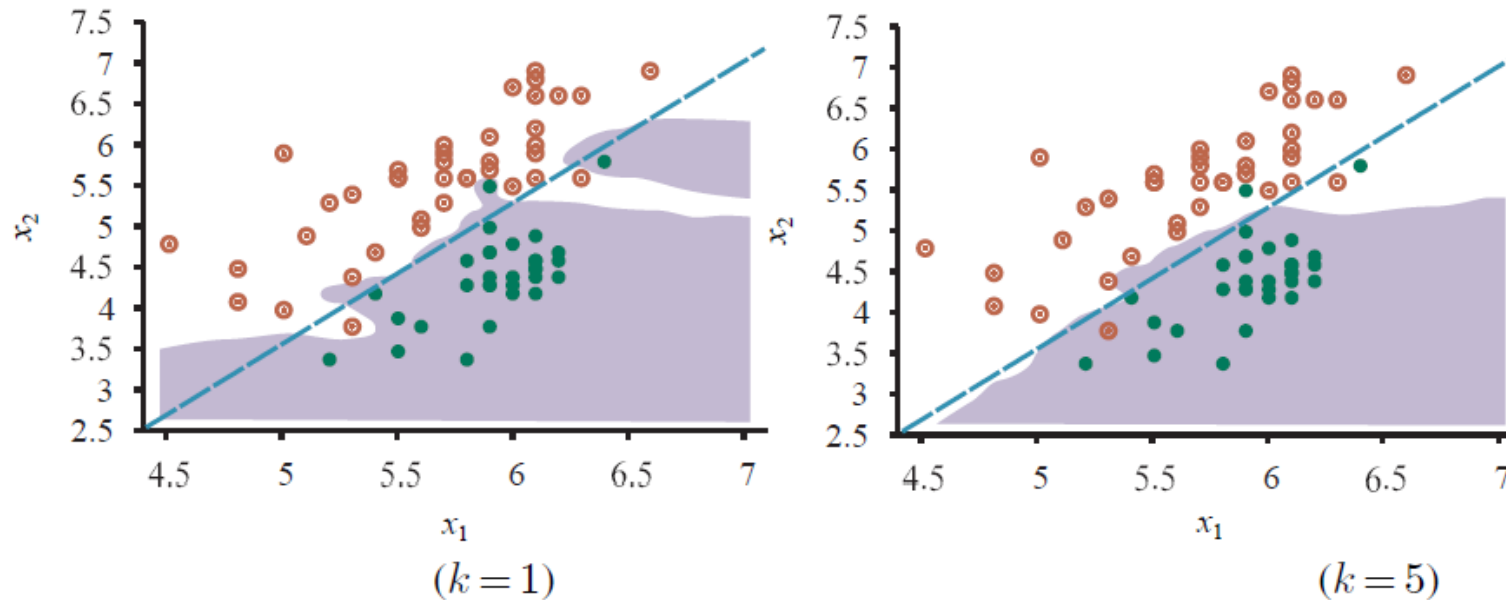
# Naïve Bayes Classifier

- Approximates a Bayes classifier with the naïve assumption that all  $n$  features are conditional independent given the class.

$$h(x) = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

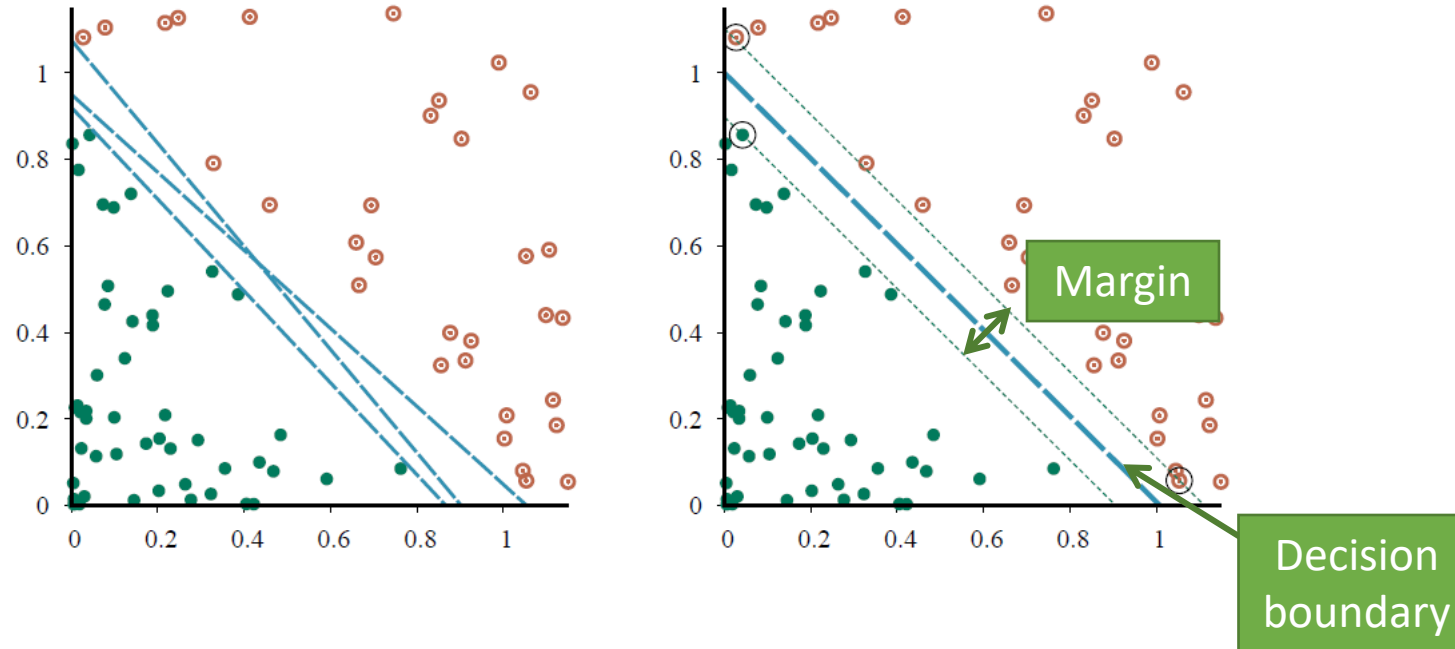
- We have only used discrete features so far, but it can be extended to continuous features.

# K-Nearest Neighbors Classifier



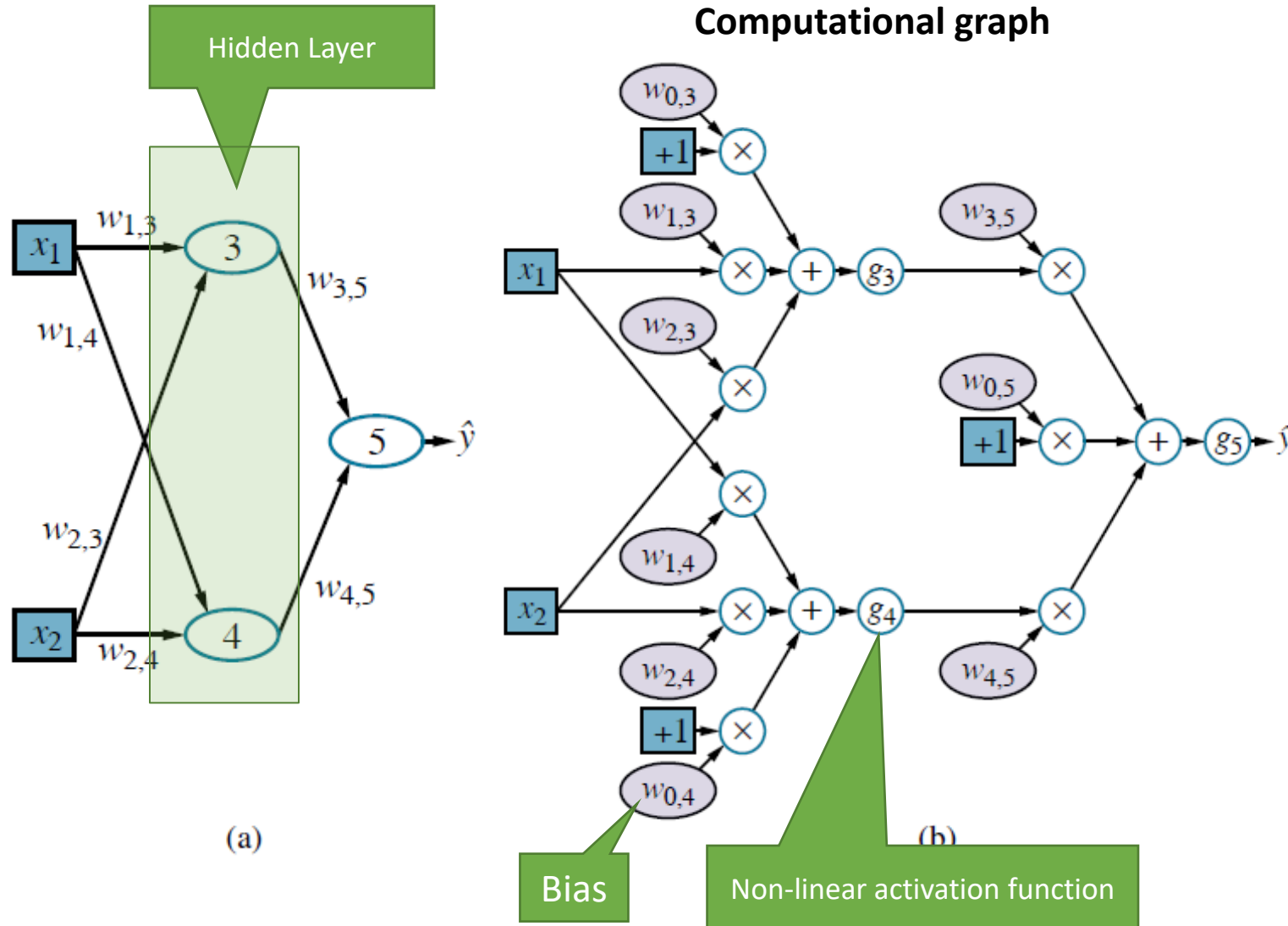
- Class is predicted by looking at the majority in the set of the  $k$  nearest neighbors.
- Neighbors are found using a distance measure (e.g., Euclidean distance between points).
- Approximates a Bayesian classifier by 
$$h(x) = \underset{y}{\operatorname{argmax}} P(Y = y \mid \text{neighborhood}(x))$$
- $k$  is a hyperparameter. Larger  $k$  smooth the decision boundary.

# Support Vector Machine (SVM)



- Linear classifier that finds the maximum margin separator using only the “support vectors” and quadratic optimization.
- The kernel trick can be used to learn non-linear decision boundaries.

# Artificial Neural Networks/Deep Learning



- Represent  $\hat{y} = f(x)$  as a network of weighted sums with non-linear activation functions (e.g., logistic, ReLU, tanh).
- Learn weights  $w$  from examples using **backpropagation** of prediction errors  $|\hat{y} - y|$  (gradient descend).
- ANNs can approximate any function (low bias). **Regularization** is typically used to avoid overfitting.
- **Deep learning** adds more layers (e.g., convolution layers)

# Other Models and Methods

- Many other models exist
  - **Generalized linear model (GLM)**: A model family that includes linear regression and the classification method **logistic regression**.
- Often used methods
  - **Ensemble Learning**: Use many models and combine the results (e.g., random forest, boosting).
  - **Regularization**: enforce simplicity by using a penalty for complexity.
  - **Stochastic Gradient Descent (SGD)**: to find model parameters that minimizes the loss.
  - **Kernel trick**: Let a linear classifier learn non-linear decision boundaries (= a linear boundary in a high dimensional space).