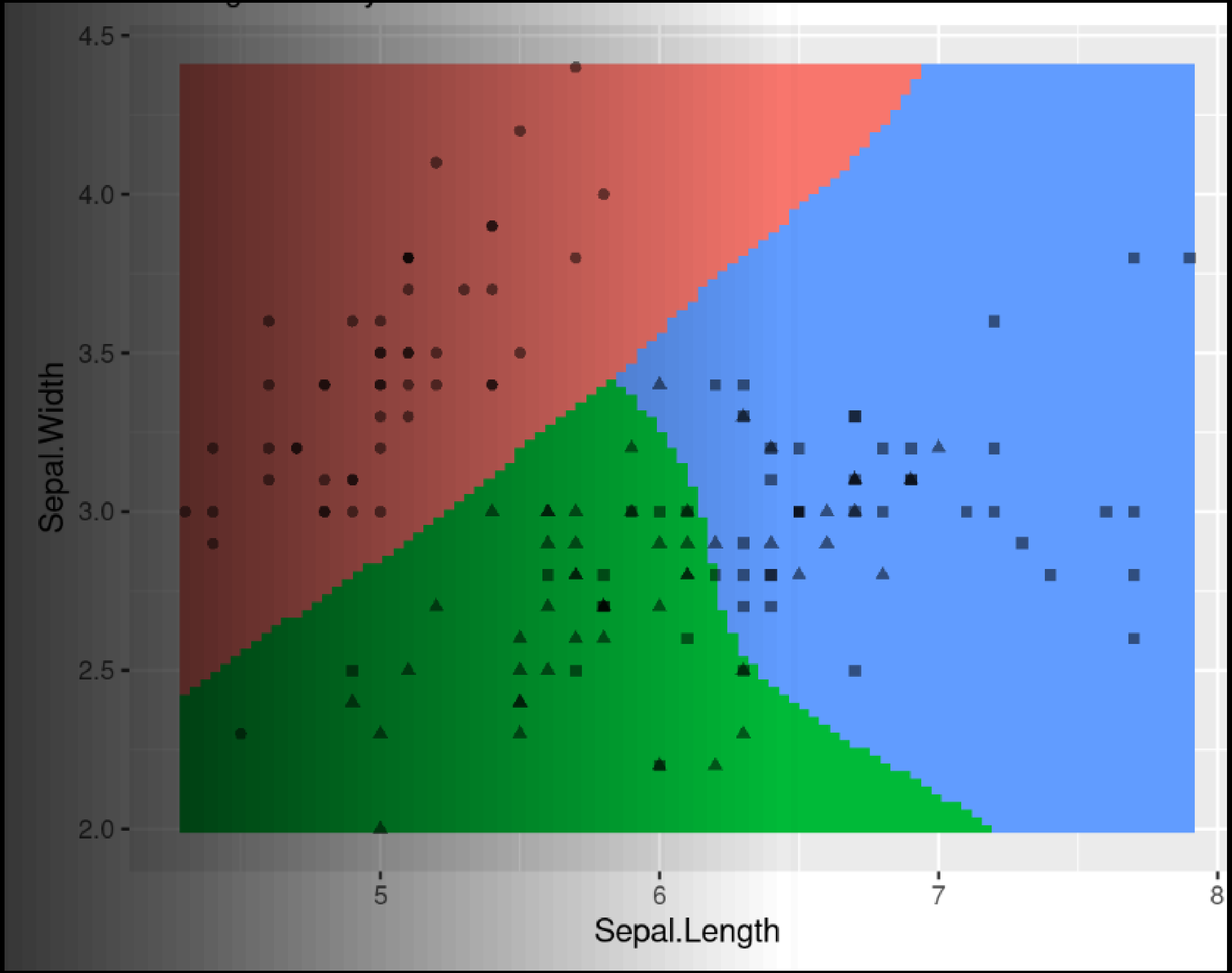
CS 5/7320 Artificial Intelligence

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.

Learning from Examples: Machine Learning

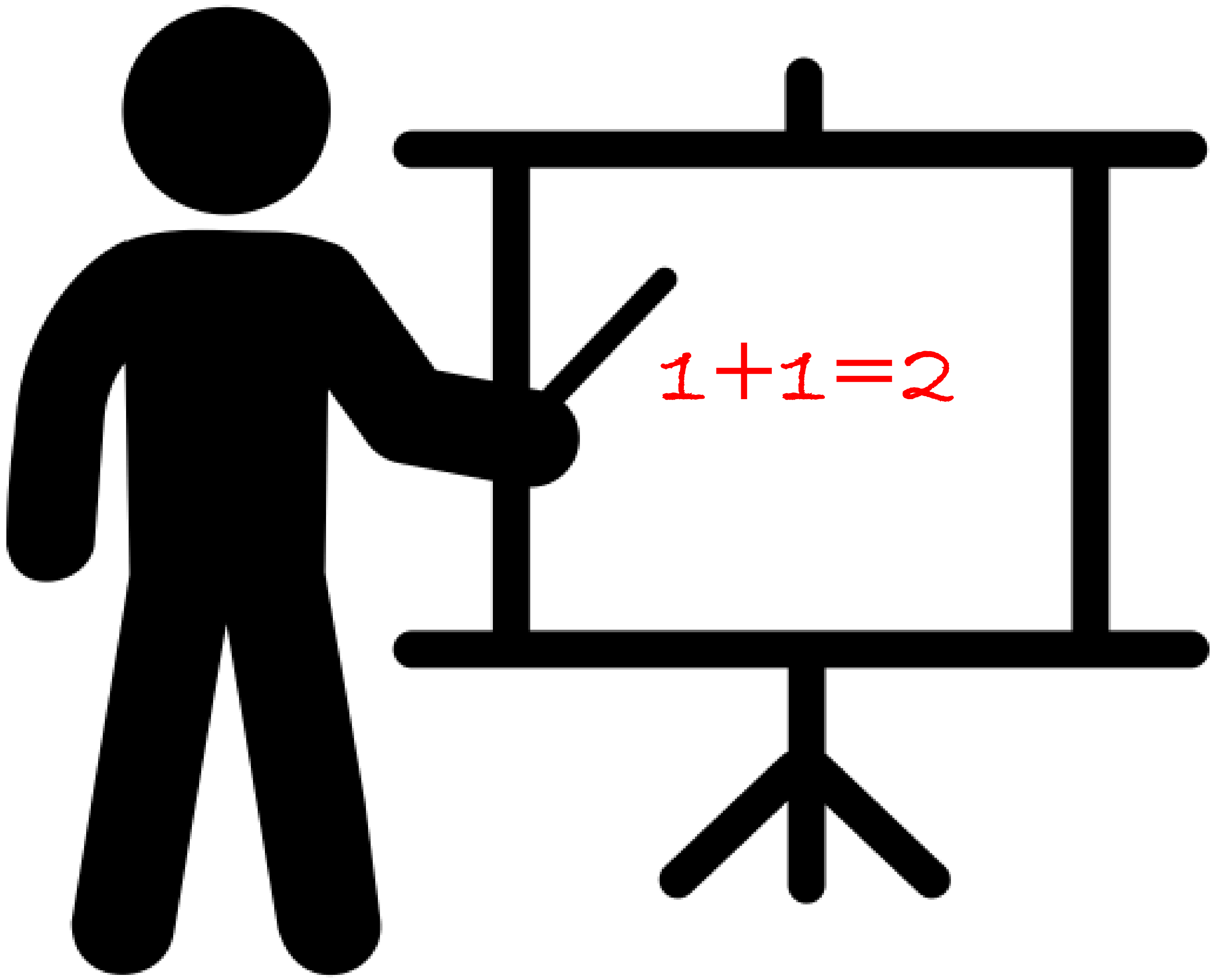
- **Up until now:** hand-craft algorithms to make rational/optimal or at least good decisions. Examples: Search strategies, heuristics.

- **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).

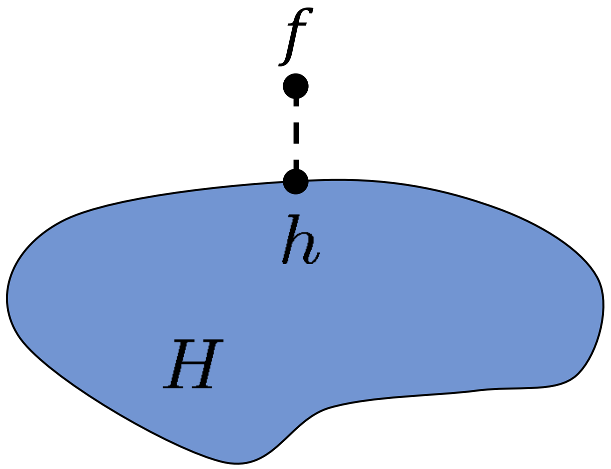
- **Learning vs. hard coding the agent function**
  - Designer cannot anticipate all possible future situations.
  - Designer may have examples but does not know how to program a solution.

Supervised Learning



Supervised Learning

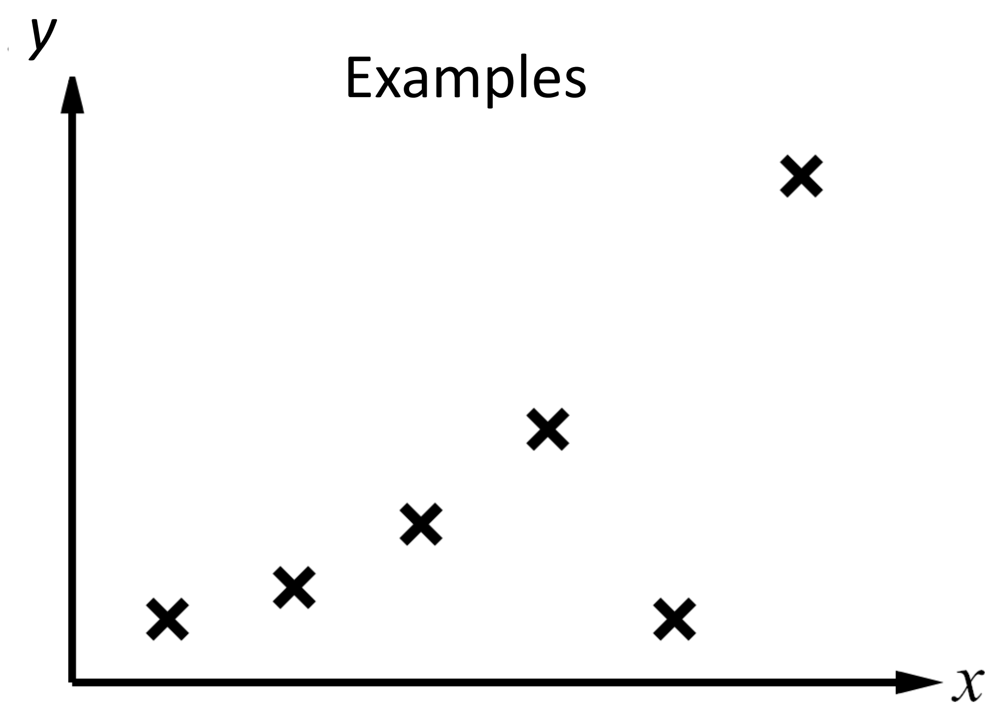
- Examples
  - Input-output pairs: $\mathrm{E} = (x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_N, y_N)$.
  - We assume that the examples are produced iid (with noise and errors) from a target function $y = f(x)$.
- Learning problem
  - Given a hypothesis space H
  - Find a hypothesis $h \in H$ such that $\hat{y}_i = h(x_i) \approx y_i$
  - That is, we want to approximate f by h using E.



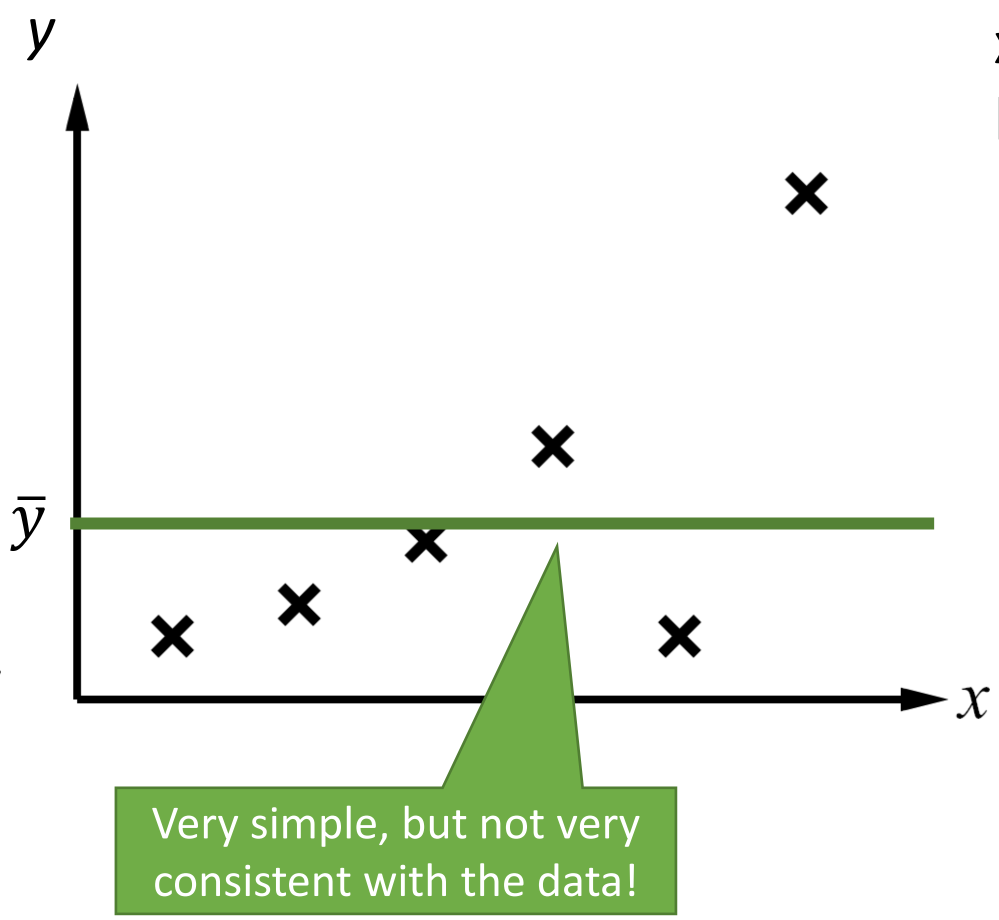
- Includes
  - Classification (outputs = class labels). E.g. x is an email and $f(x)$ is spam / ham
  - Regression (outputs = real numbers). E.g. x is a house and $f(x)$ is its selling price

Consistency vs. Simplicity

Example: Curve fitting (regression, function approximation)



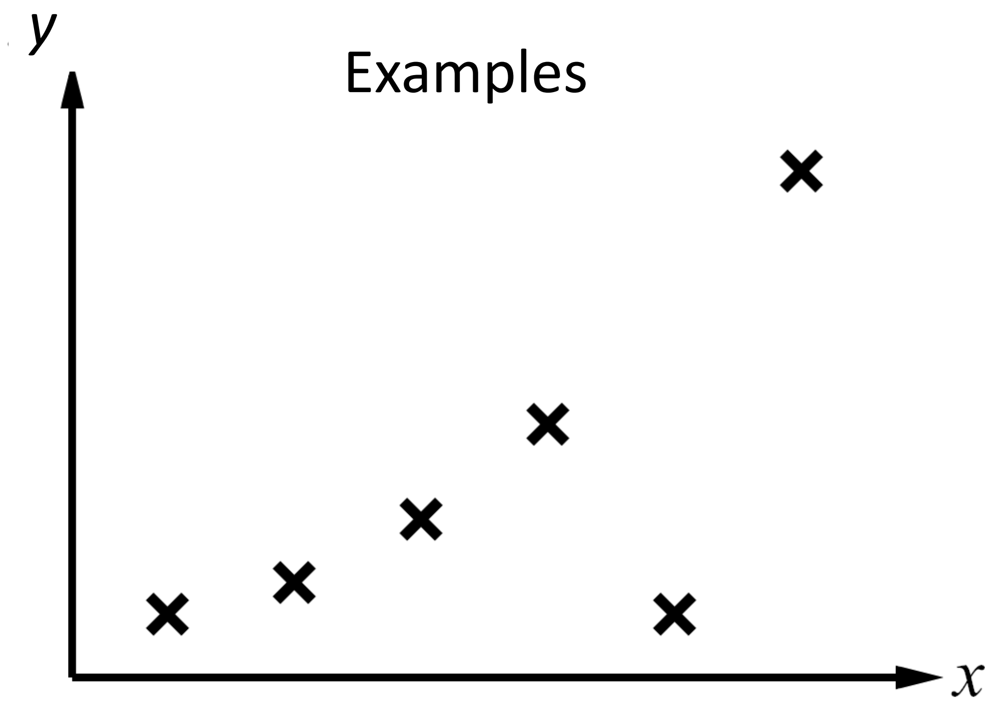
- **Consistency:** $h(x_i) \approx y_i$
- **Simplicity**

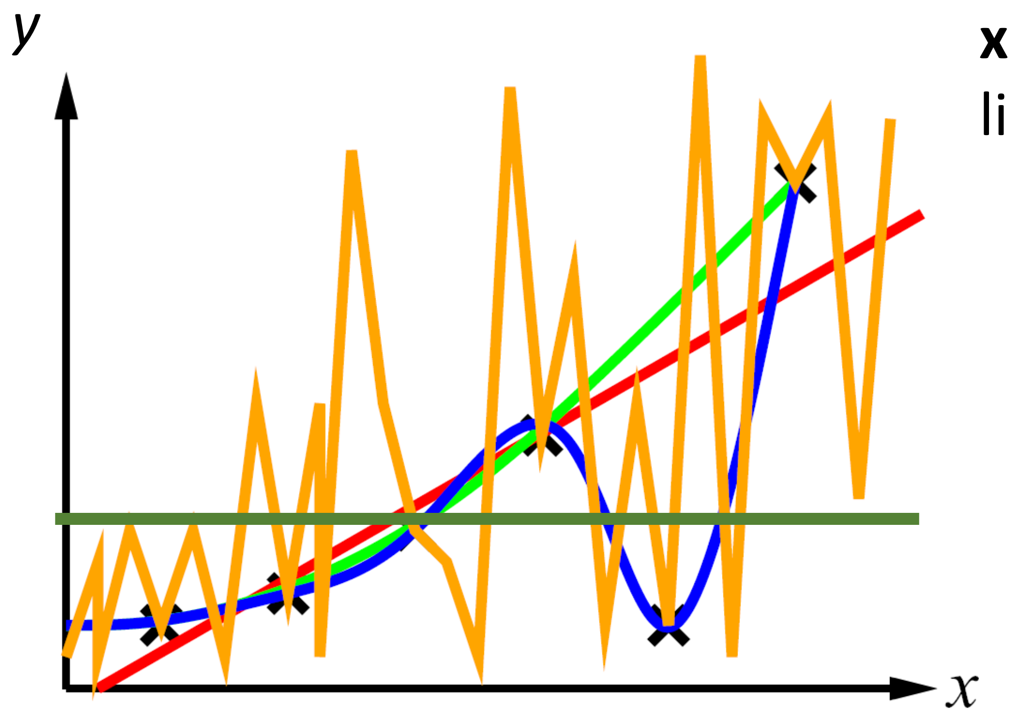


x … $f(x)$
line … $h(x)$

Consistency vs. Simplicity

Example: Curve fitting (regression, function approximation)





x … $f(x)$
lines … $h(x)$

- **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 $\mathrm{E} = (x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_N, y_N)$.

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…

$$\left.\begin{array}{l} L_1 \\ L_2 \end{array}\right\} \text{For Regression} \qquad L_{0/1} \longleftarrow \text{For Classification}$$

- **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

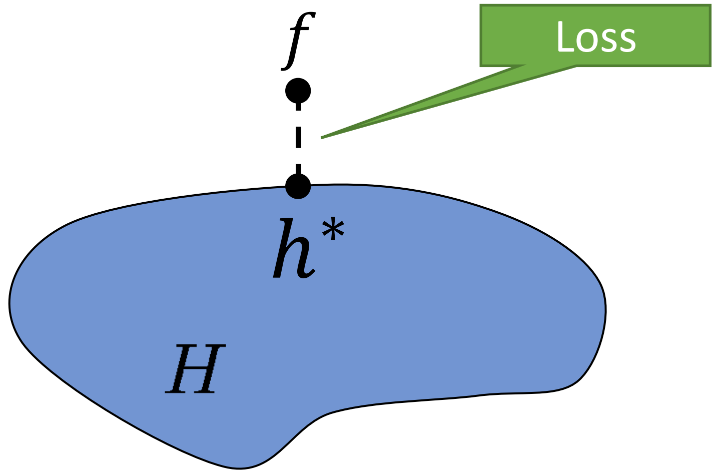
- Empirical loss

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

- Learning the best hypothesis (approximation)

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

- 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.

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

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)

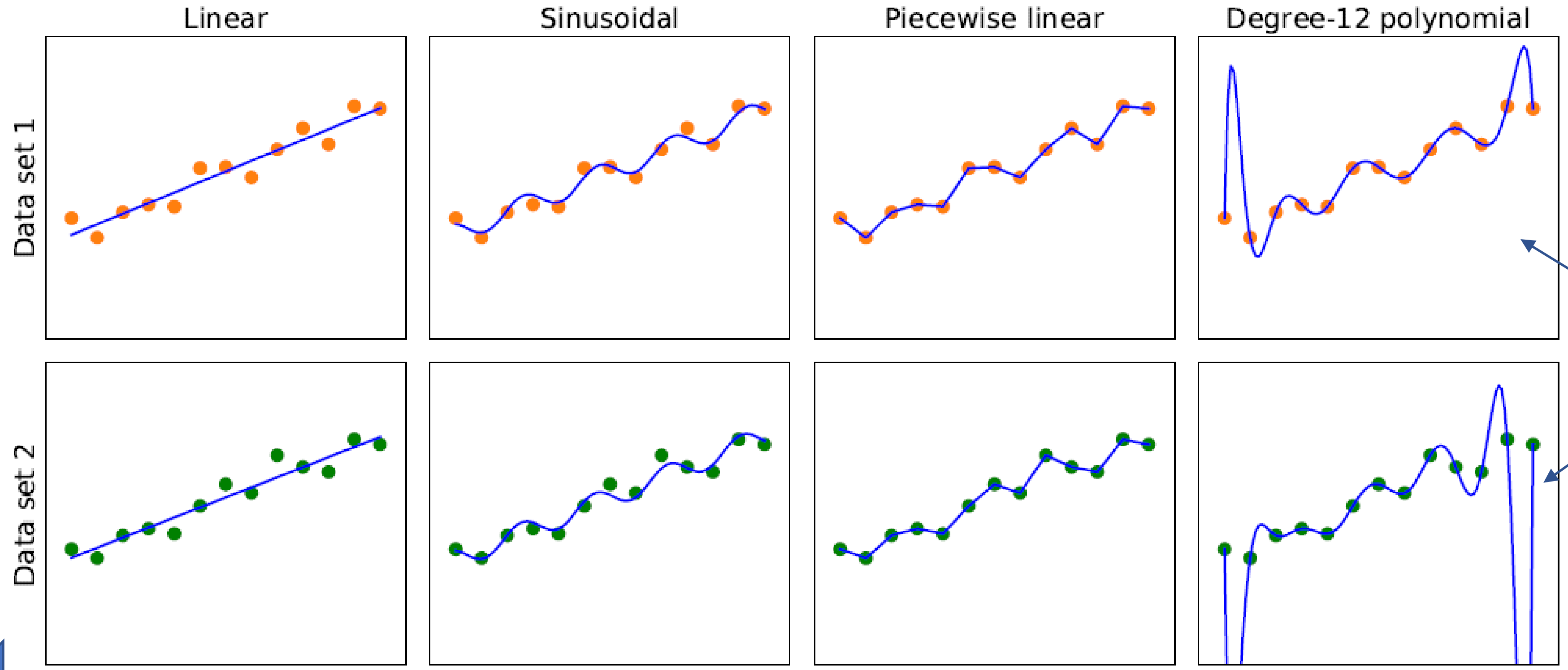
$$h^* = \underset{h \in H}{\operatorname{argmin}} \left[EmpLoss_{L,E}(h) + \underbrace{\lambda\, Complexity(h)}_{\text{Penalty term}} \right]$$

Model Selection: Bias vs. Variance

Simpler

More consistent

Overfitting

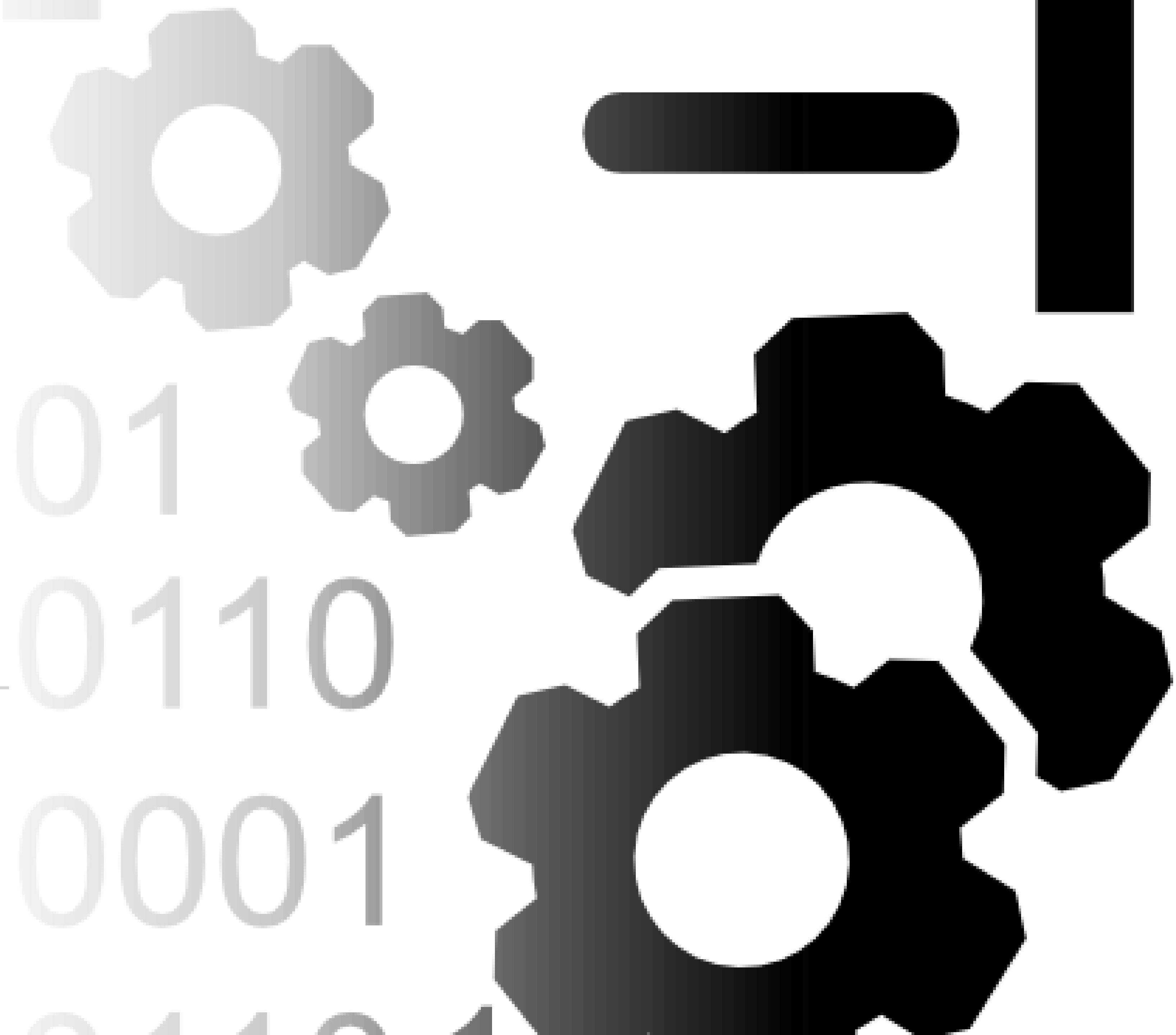


Two samples from the same function f (points) with the learned function h (lines).

High **Bias**: restrictions by the model class Low

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

Data

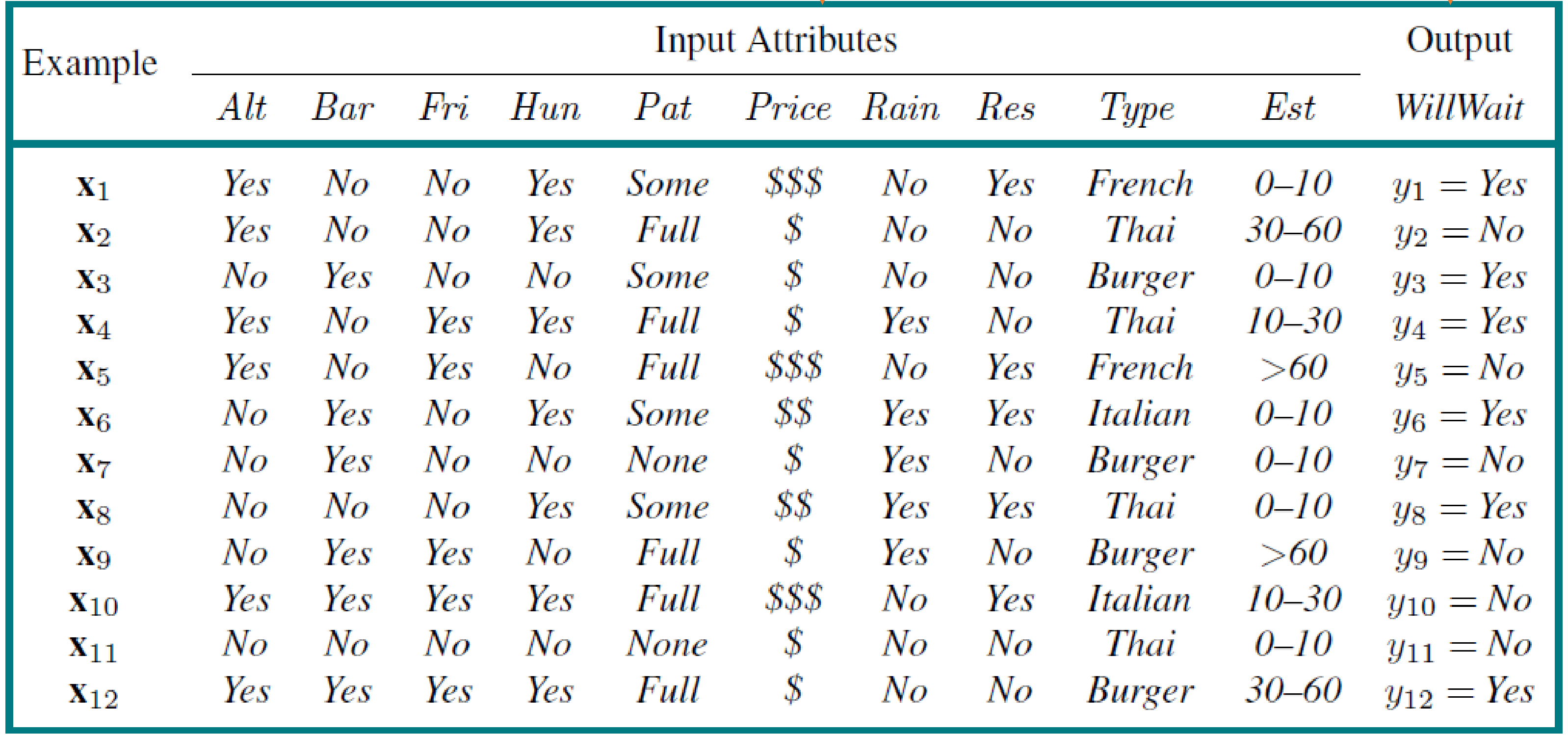


The Dataset

Features
Variables
Attributes

Class
Label

Examples
Instances
Observations



Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

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

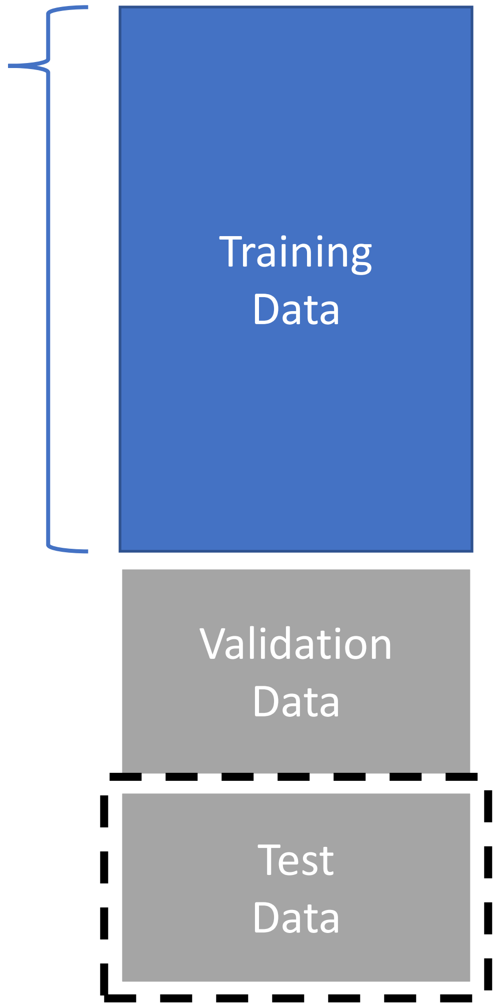
Training a Model

- 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 λ, maximal decision tree depth, selected features.

- The “Learner” (algorithm) tries to optimizes 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?

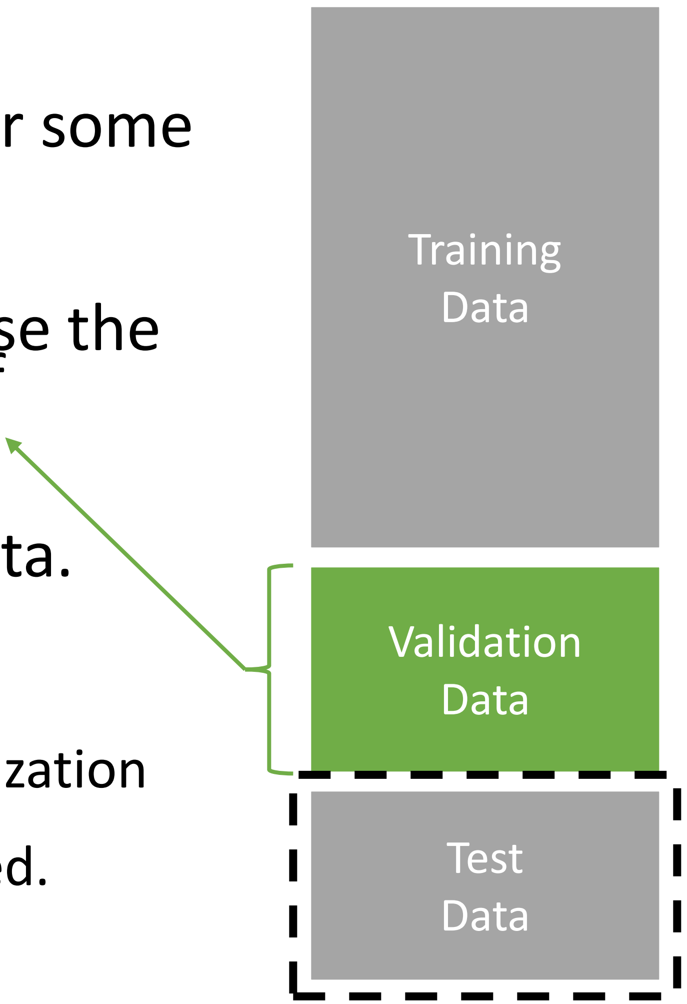


Hyperparameter Tuning/Model Selection

- 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.

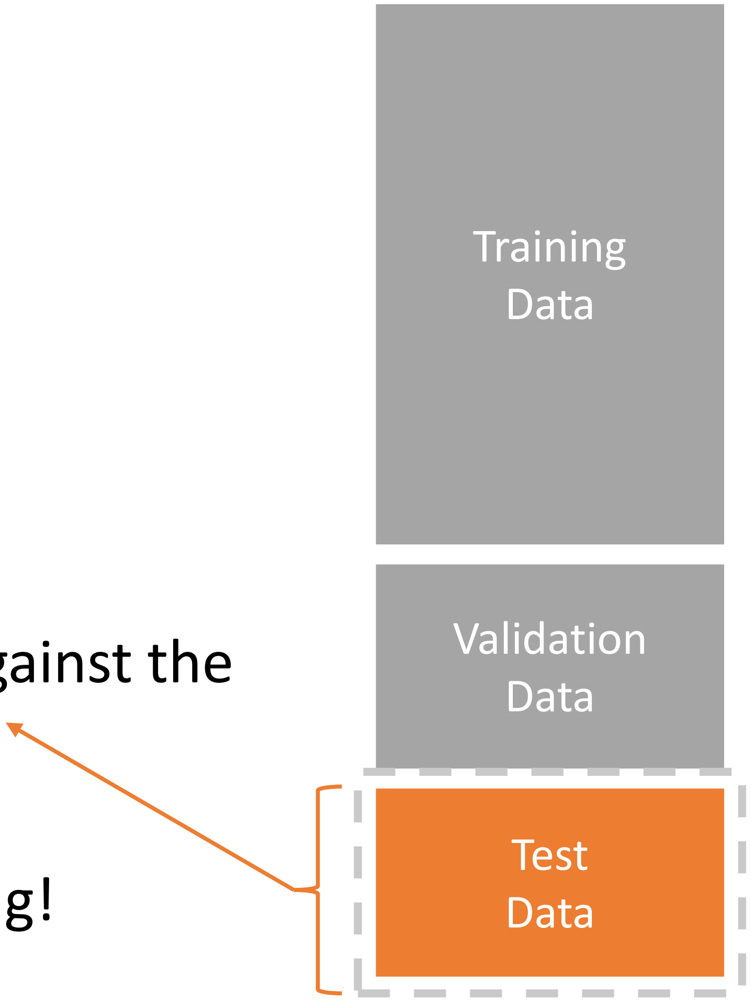
- Notes:
  - The validation set was not used for training, so we get generalization accuracy.
  - If no model selection is necessary, then no validation set is used.



Testing a Model

- After the model is selected, the final model is evaluated against the test set to **estimate the final model accuracy**.

- Very important: never "peek" at the test set during learning!

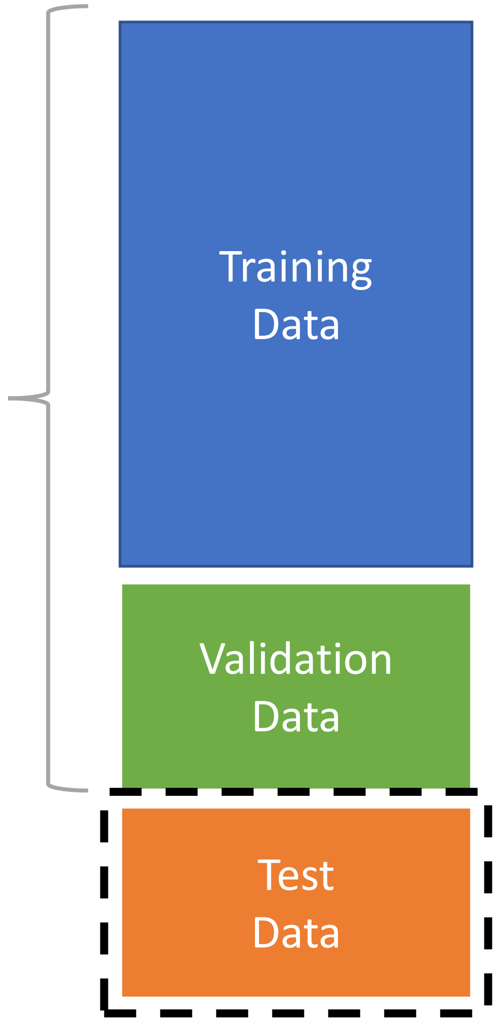


How to Split the Dataset

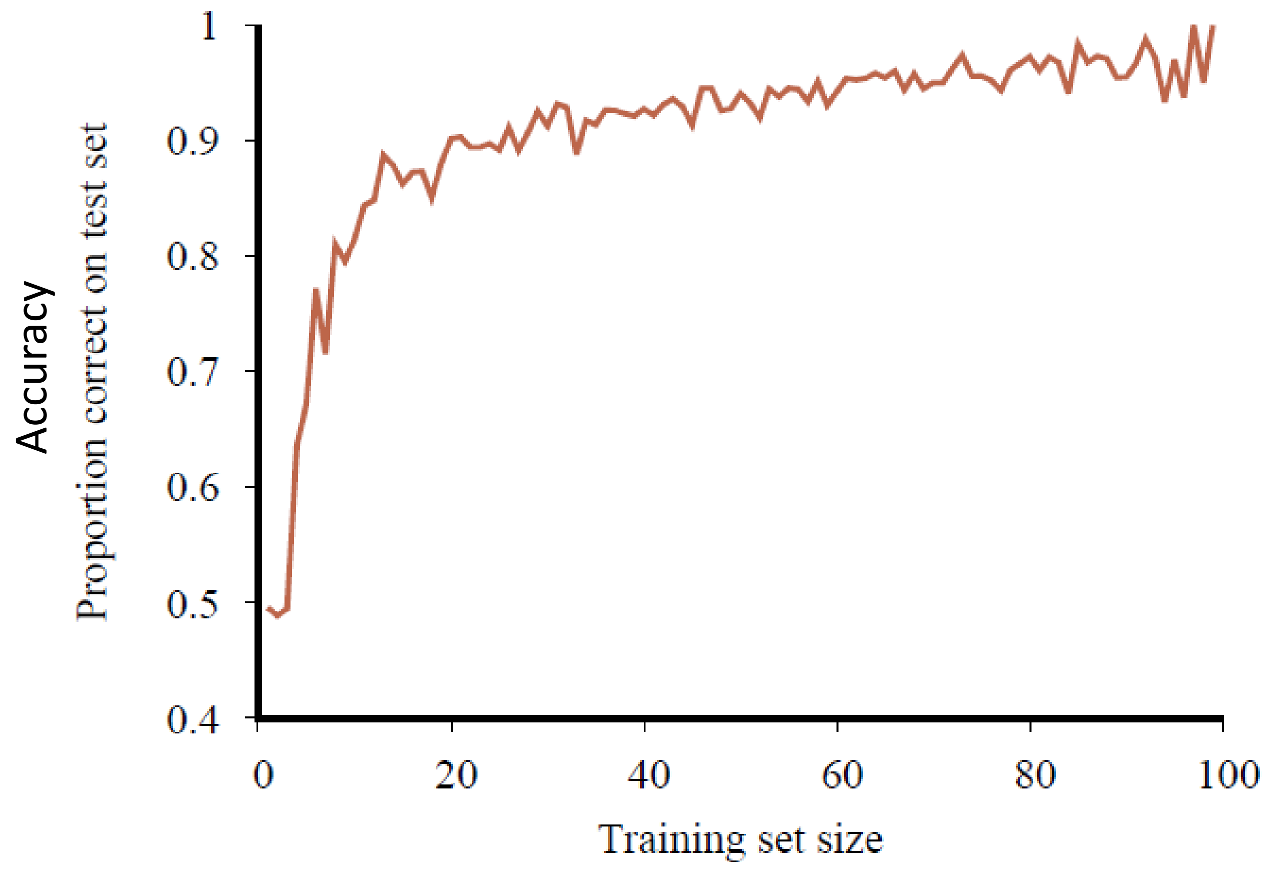
- **Random splits:** Split the data randomly in, e.g., 60% training, 20% validation, and 20% testing.

- **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.

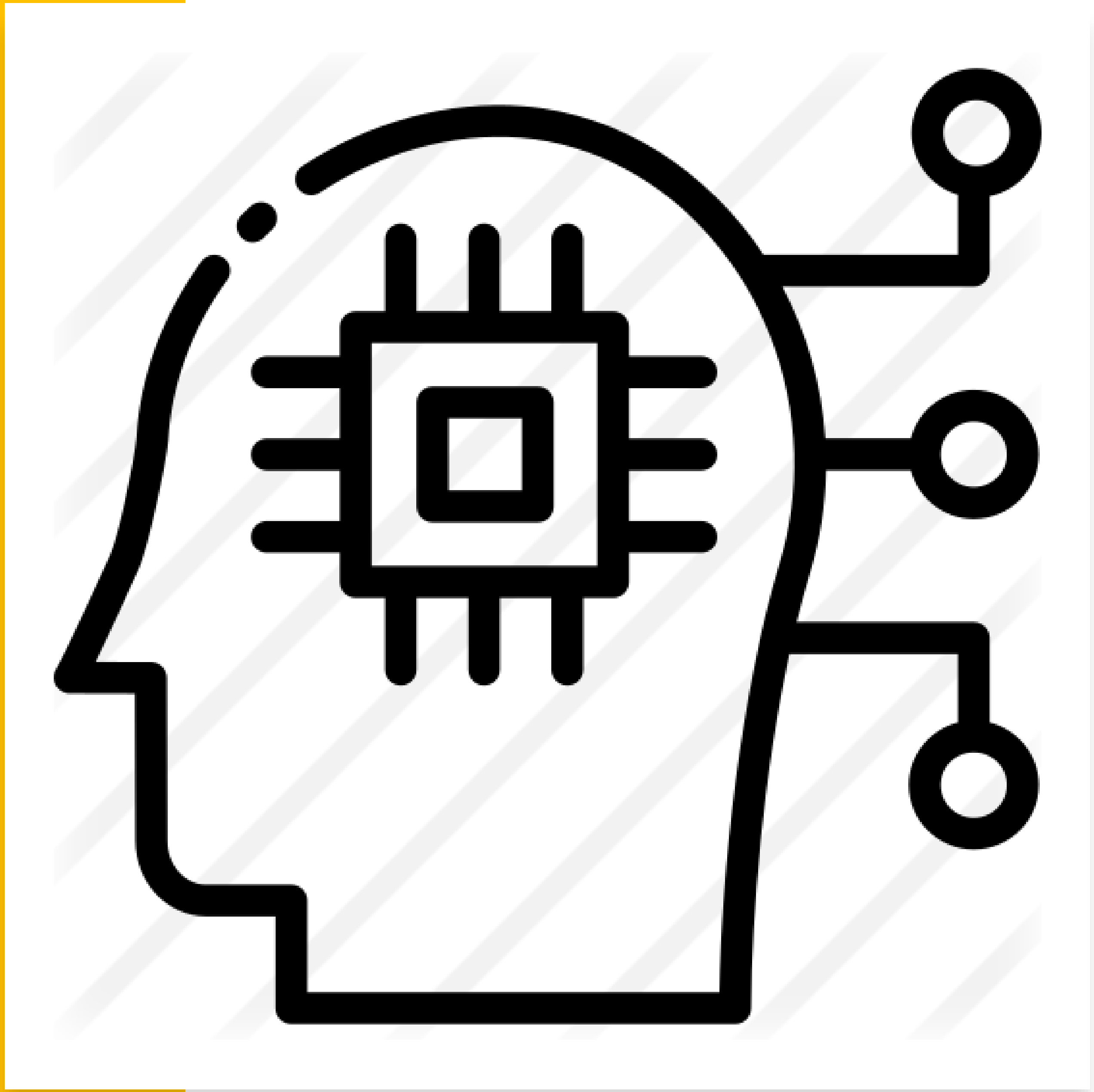
More data is better!

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: If 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: $h_{\boldsymbol{w}}(\boldsymbol{x}_j) = w_o + w_1 x_{j,1} + \cdots + w_n x_{j,n} = \sum_i w_i x_{j,i} = \boldsymbol{w}^T \boldsymbol{x}_j$

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

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

Gradient: $\nabla L(\boldsymbol{w}) = 2X^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$

The gradient is a vector of partial derivatives

$$\nabla L(\boldsymbol{w}) = \left[\frac{\partial L}{\partial w_1}(\boldsymbol{w}), \frac{\partial L}{\partial w_2}(\boldsymbol{w}), \dots, \frac{\partial L}{\partial w_n}(\boldsymbol{w})\right]^T$$

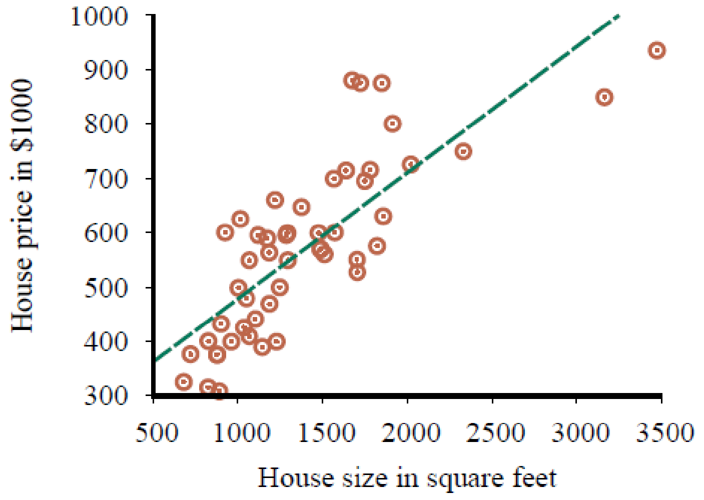
Find: $\nabla L(\boldsymbol{w}) = 0$

Gradient descend:

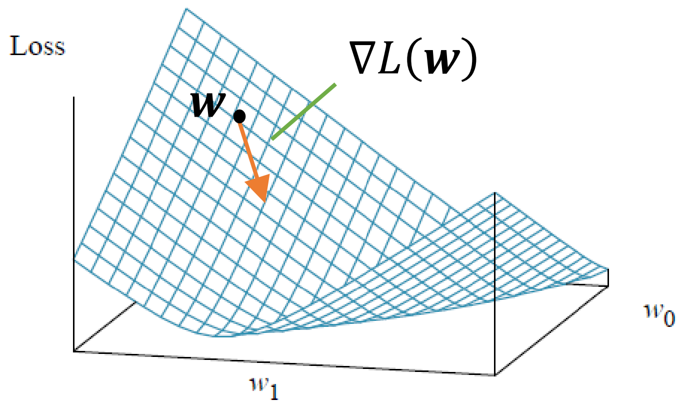
$$\boldsymbol{w} = \boldsymbol{w} - \alpha \nabla L(\boldsymbol{w})$$

Analytical solution:

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



(a)



(b)

Naïve Bayes Classifier

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

$$h(x) = \underset{y}{\operatorname{argmax}}\, P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

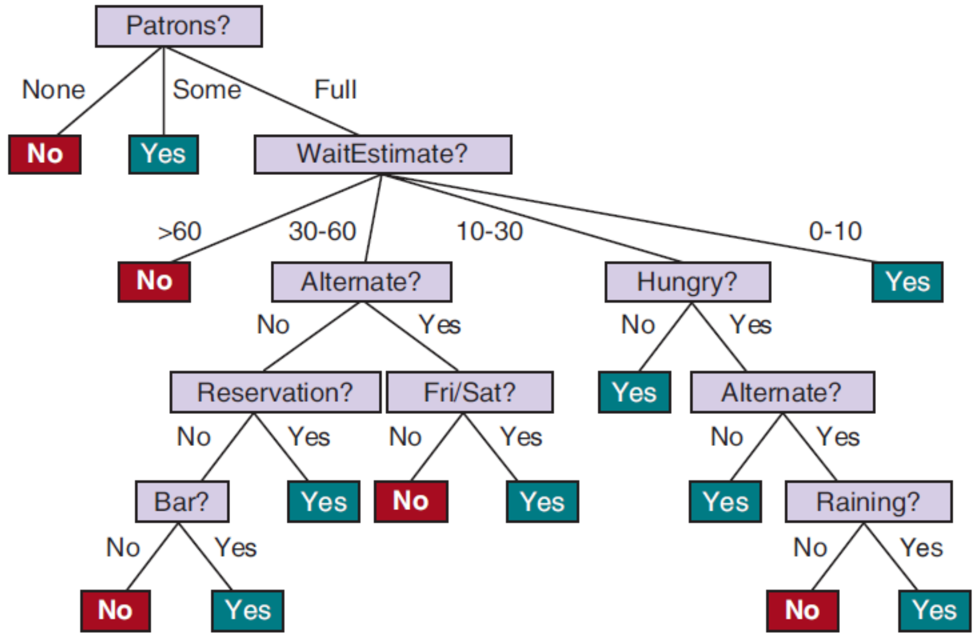
- We have only used discrete features so far, but it can be extended to continuous features. Gaussian Naïve Bayes Classifier assumes that continuous features have:

$$P(x_i \mid y) \sim N(\mu_y, \sigma_y)$$

The parameters for $N(\mu_y, \sigma_y)$ are estimated from data.

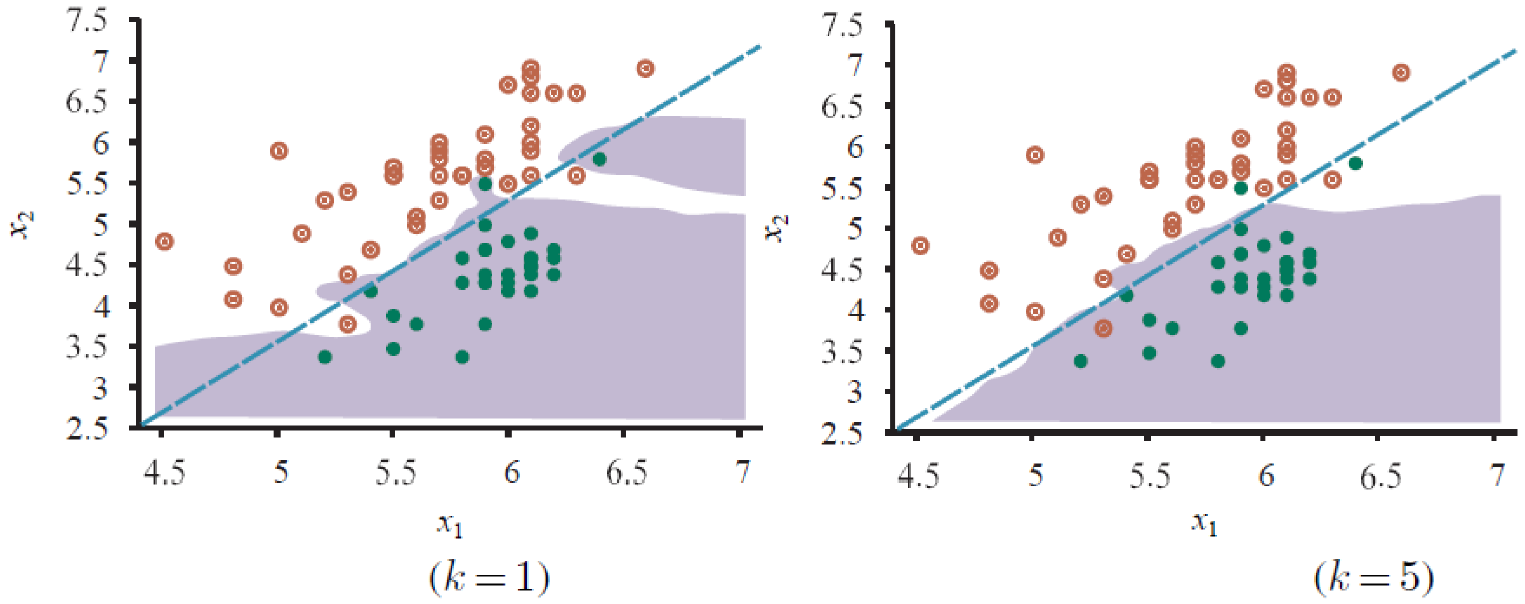
Decision Trees

Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$



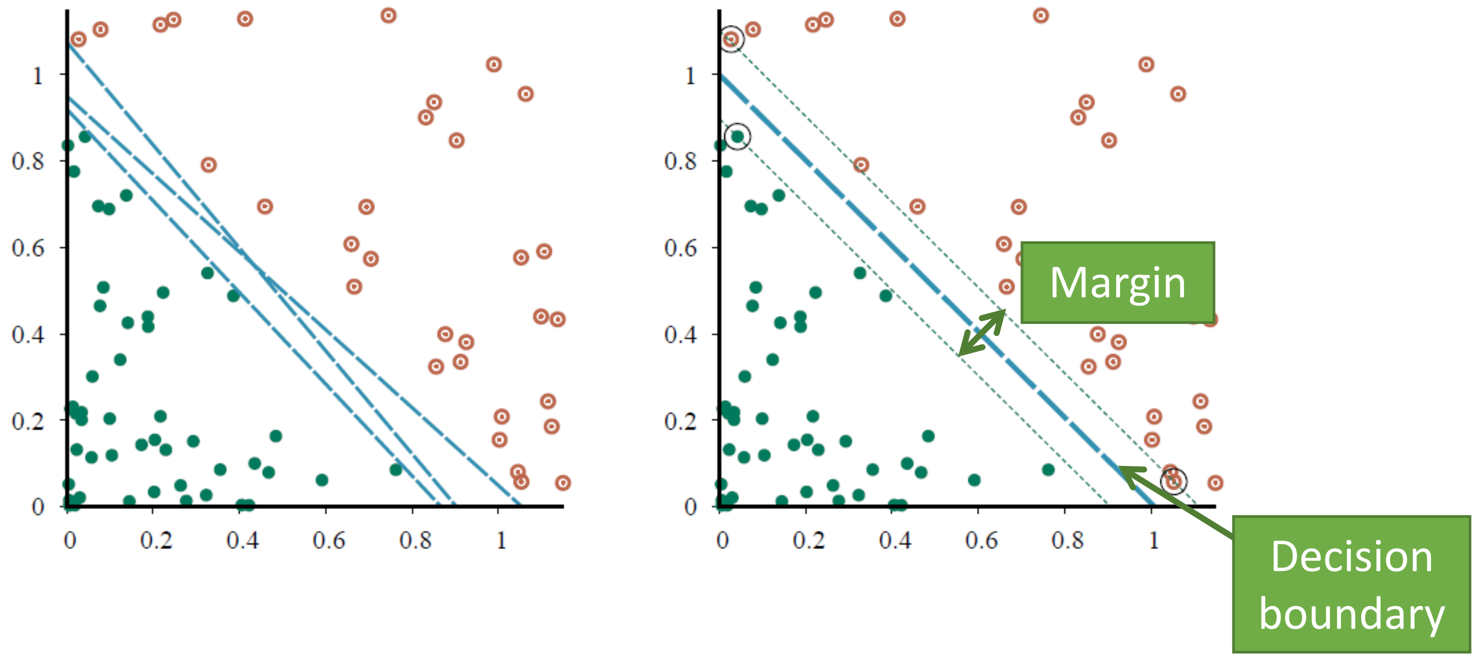
- A **sequence of decisions** represented as a tree.
- Many implementations that differ by
    - How to select features to split?
    - When to stop splitting?
    - Is the tree pruned?
- Approximates a Bayesian classifier by $h(x) = \underset{y}{\operatorname{argmax}}\, P(Y = y \mid \text{leafNodeMatching}(x))$

K-Nearest Neighbors Classifier



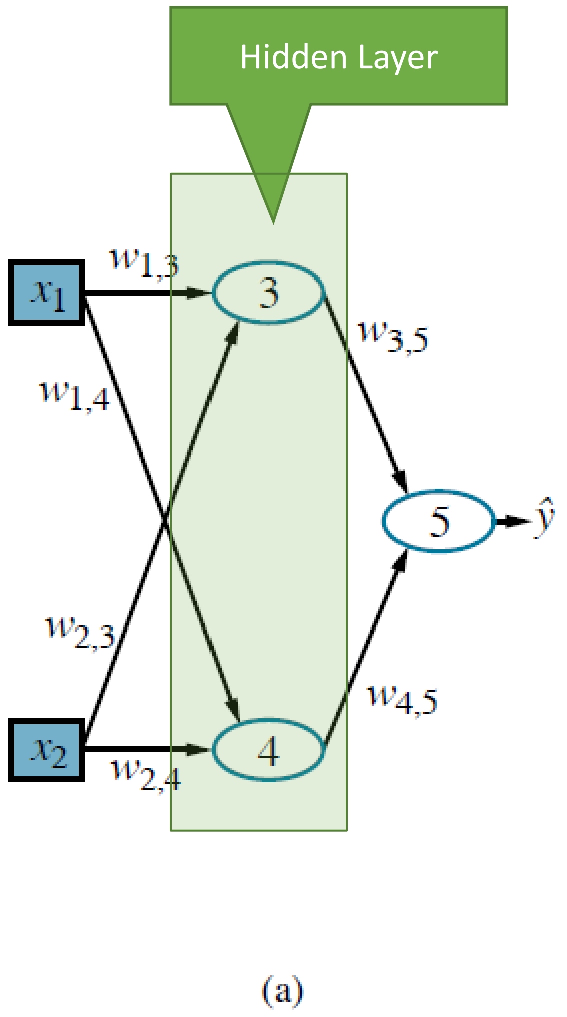
- Class is predicted by looking at the majority in the set of the k nearest **neighbors**. k is a hyperparameter. Larger k smooth the decision boundary.
- Neighbors are found using a distance measure (e.g., Euclidean distance between points).
- Approximates a Bayesian classifier by $h(x) = \underset{y}{\operatorname{argmax}}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

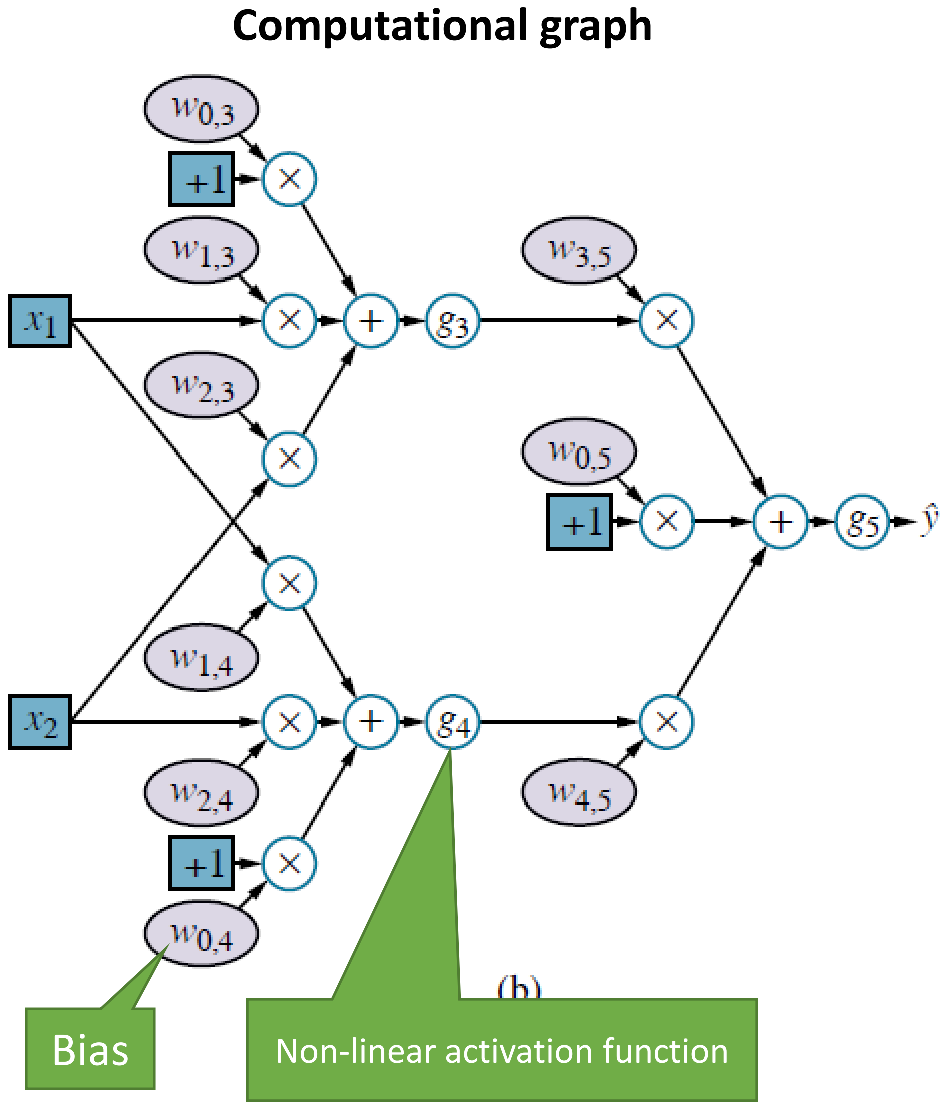
Support Vector Machine (SVM)



- 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 $\hat{y} = h(x)$ 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 (no bias). **Regularization** is typically used to avoid overfitting.
- **Deep learning** adds more layer types (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
  - **Regularization:** enforce simplicity by using a penalty for complexity.
  - **Kernel trick:** Let a linear classifier learn non-linear decision boundaries (= a linear boundary in a high dimensional space).
  - **Ensemble Learning:** Use many models and combine the results (e.g., random forest, boosting).