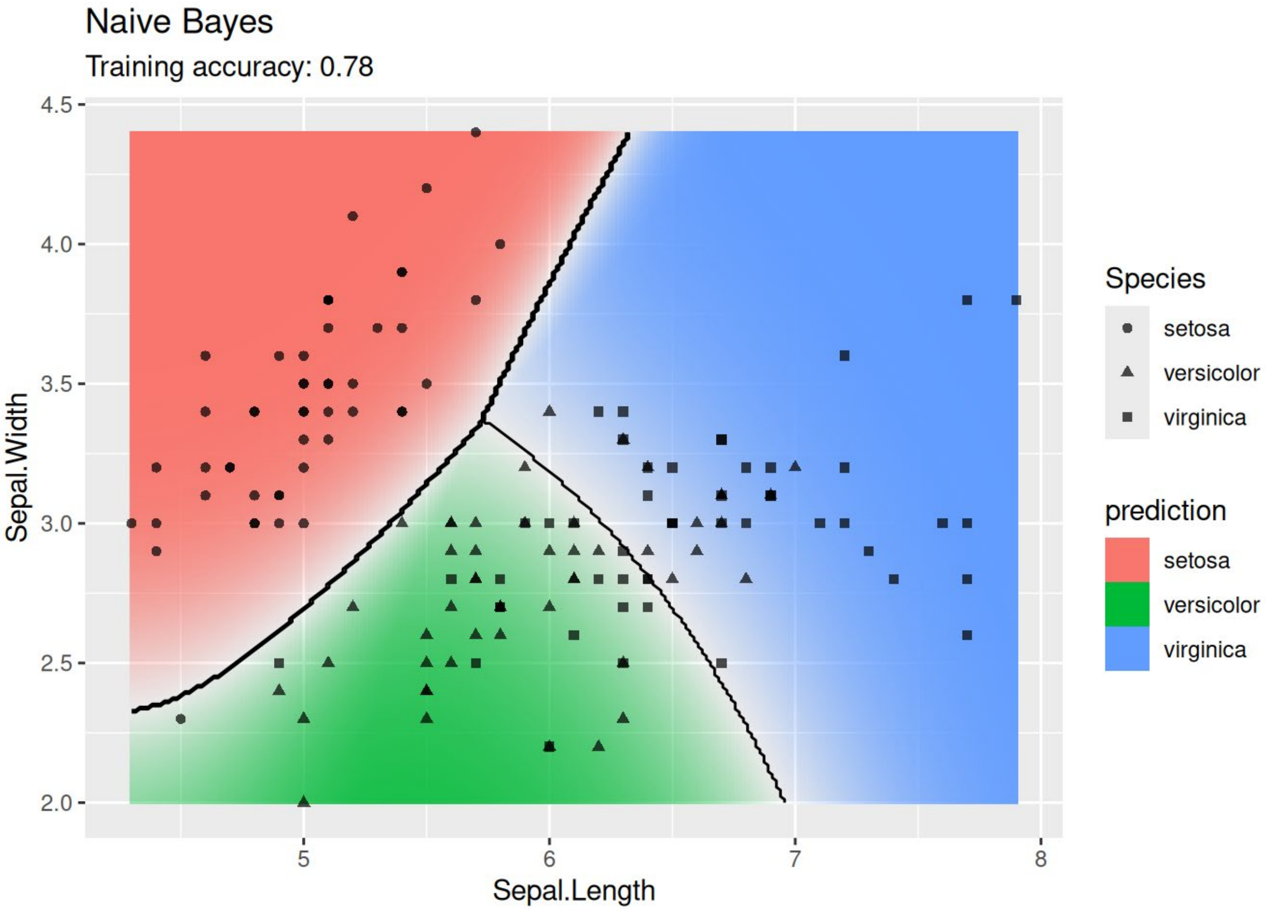
CS 5/7320
Artificial Intelligence

Learning from Examples: Supervised Machine Learning

AIMA Chapter 19

Slides by Michael Hahsler

Some slides are based on Dan Klein's slides (http://ai.berkeley.edu); with figures from the AIMA textbook.



Online Material

Topics

- Agents & ML
- Supervised Learning
- Training Data
- Training & Testing
- Types of supervised ML Models
- Use in AI

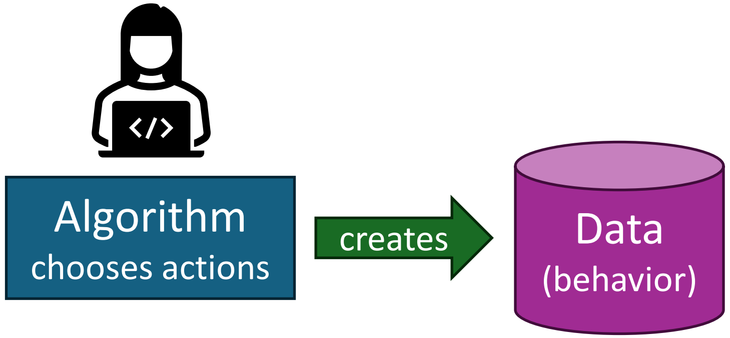
Agents and ML

DeepAi.org with prompt: “A happy cartoon robot with an artificial neural network for a brain on white background learning to play chess”

Learning from Examples

Up until now in this course:

- **Hand-craft algorithms** to make rational/optimal or at least good decisions.
  Examples: Search strategies, heuristics, and constructing Bayesian networks.

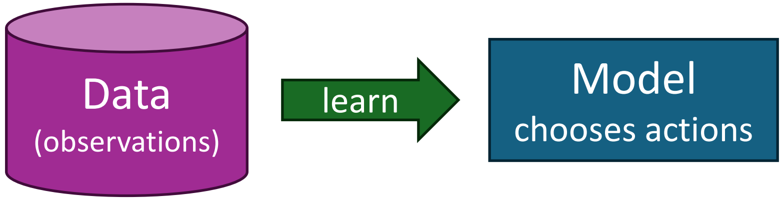


Issues

- We may not be able to anticipate all possible future scenarios.
- We may have examples, but we do not know how to implement a solution.

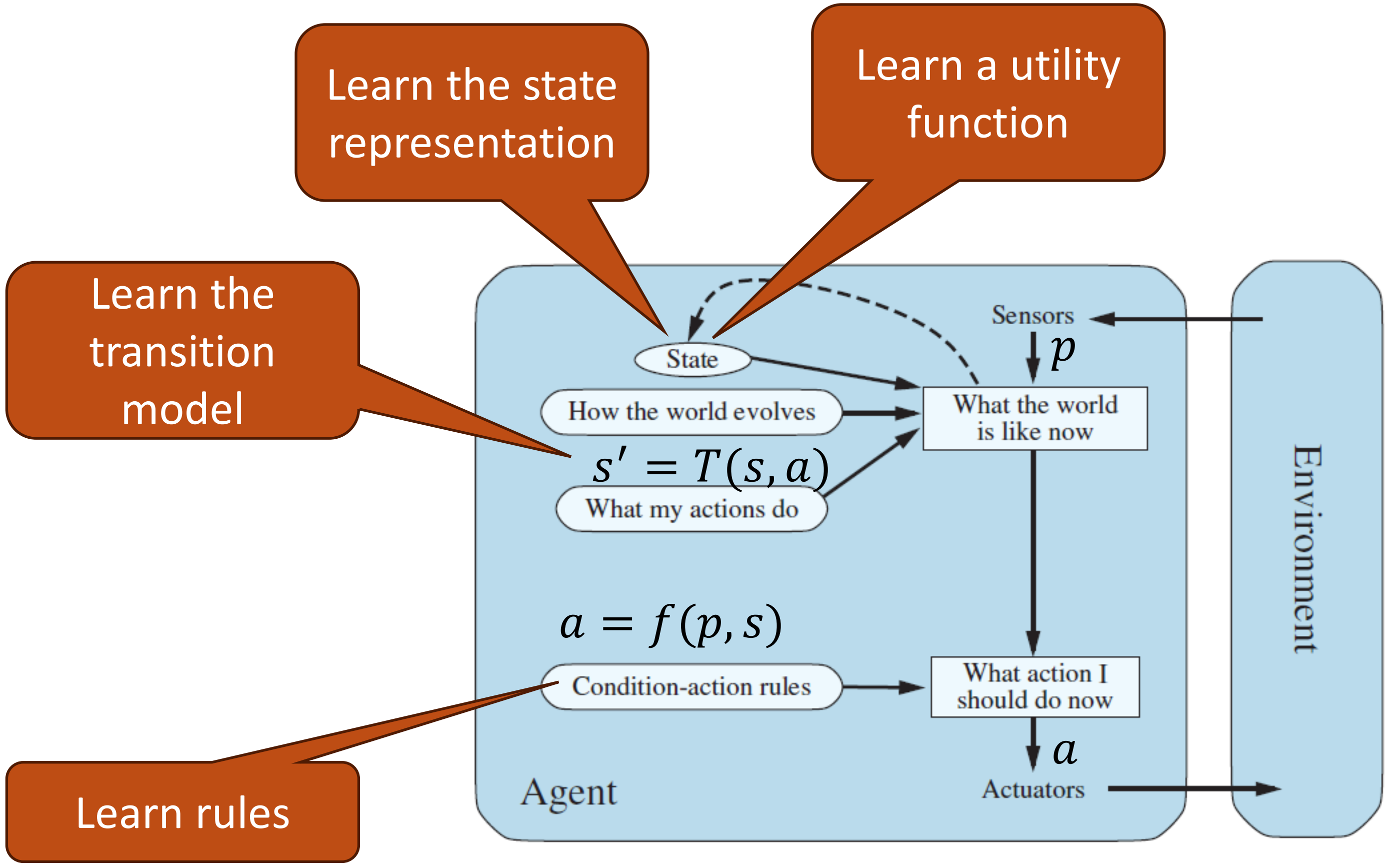
Supervised Machine Learning

- Uses observations: training data with the correct answers.
- Learn a function (model) to map an input (e.g., state) to an output (e.g., action) representing the desired behavior.
- Examples:
  - Use a naïve Bayesian classifier to distinguish between spam/non-spam.
  - Learn a playout policy to simulate games (current board -> good move)



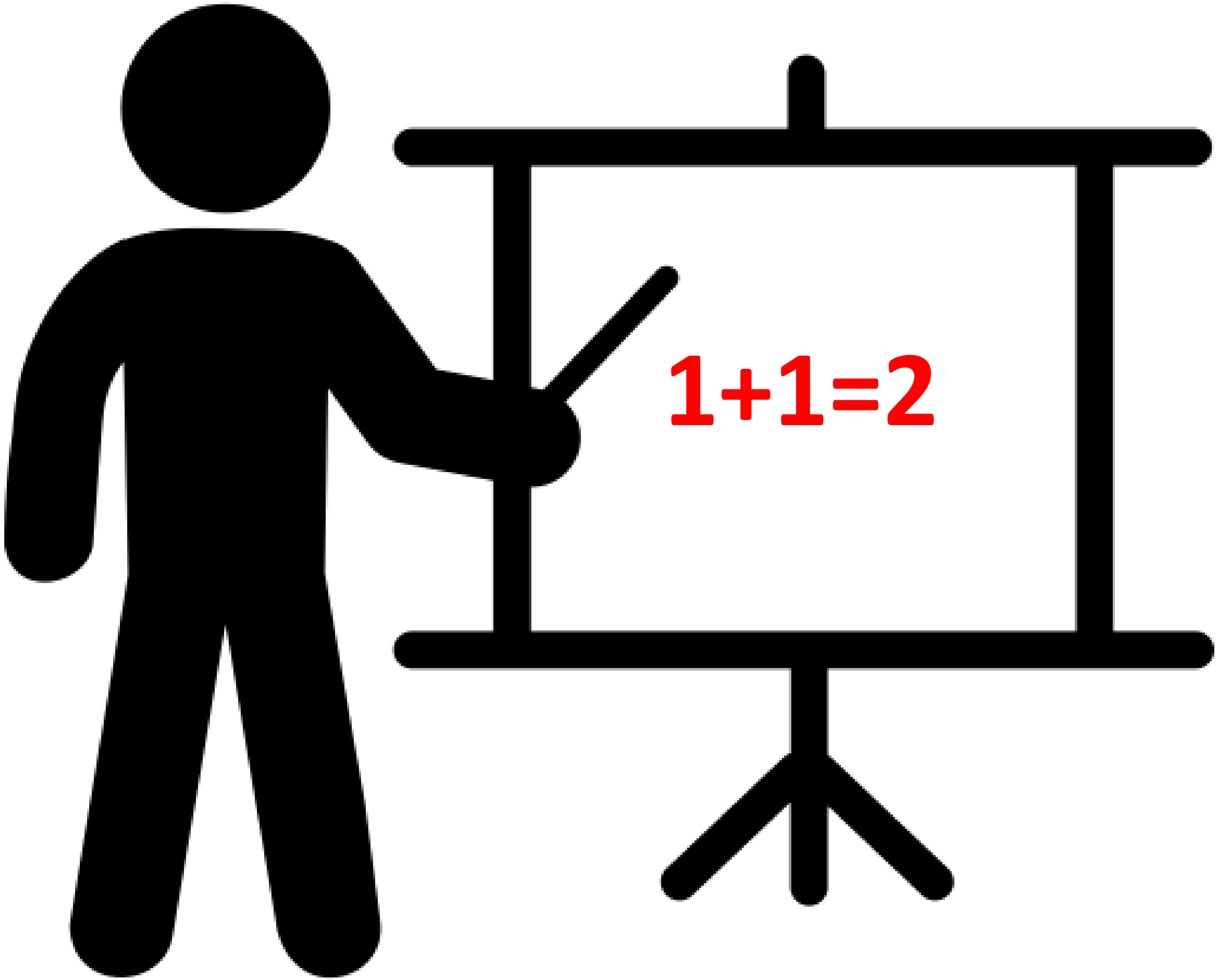
Learning Components of an Agent

- We can learn many different components of an agent from examples
- **Example**: Learning components of a model-based reflex agent



Supervised Learning

Teaching a model to answer correctly



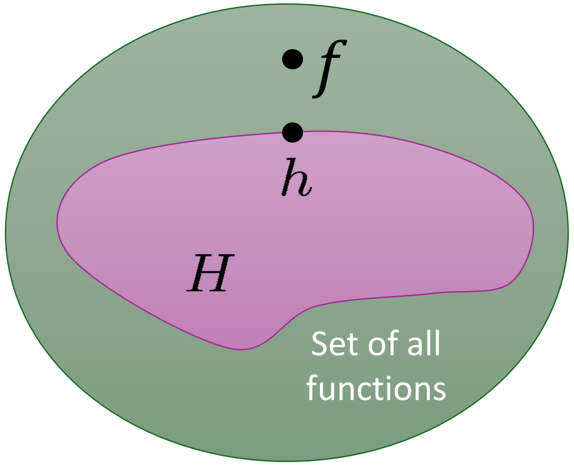
Supervised Learning As Function Approximation

Examples

- We assume there exists an unknown target function $y = f(\boldsymbol{x})$ that produces iid (independent and identically distributed) examples, possibly with noise and errors.
- Examples are observed input-output pairs $\mathrm{E} = (\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_i, y_i), \dots, (\boldsymbol{x}_N, y_N)$, where $\boldsymbol{x}_i$ is a vectors called the feature vector.

Learning problem

- Given a hypothesis space H of representable models.
- Find a hypothesis $h \in H$ such that $\hat{y}_i = h(\boldsymbol{x}_i) \approx y_i \; \forall i$
- That is, we want to approximate f by h using E.



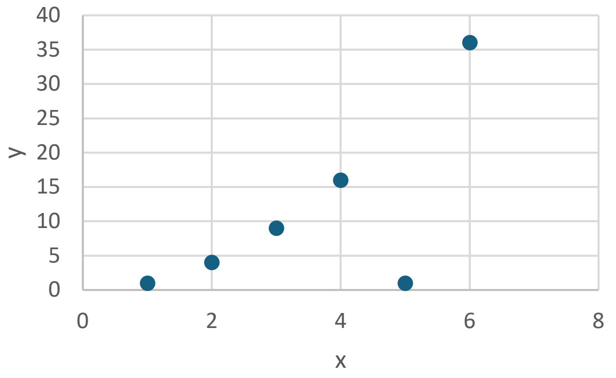
Supervised learning includes

- Classification: y is a class labels. E.g., $\boldsymbol{x}$ are percepts and y is the chosen action.
- Regression: y is a real number. E.g., $\boldsymbol{x}$ are state features and y is the state utility.

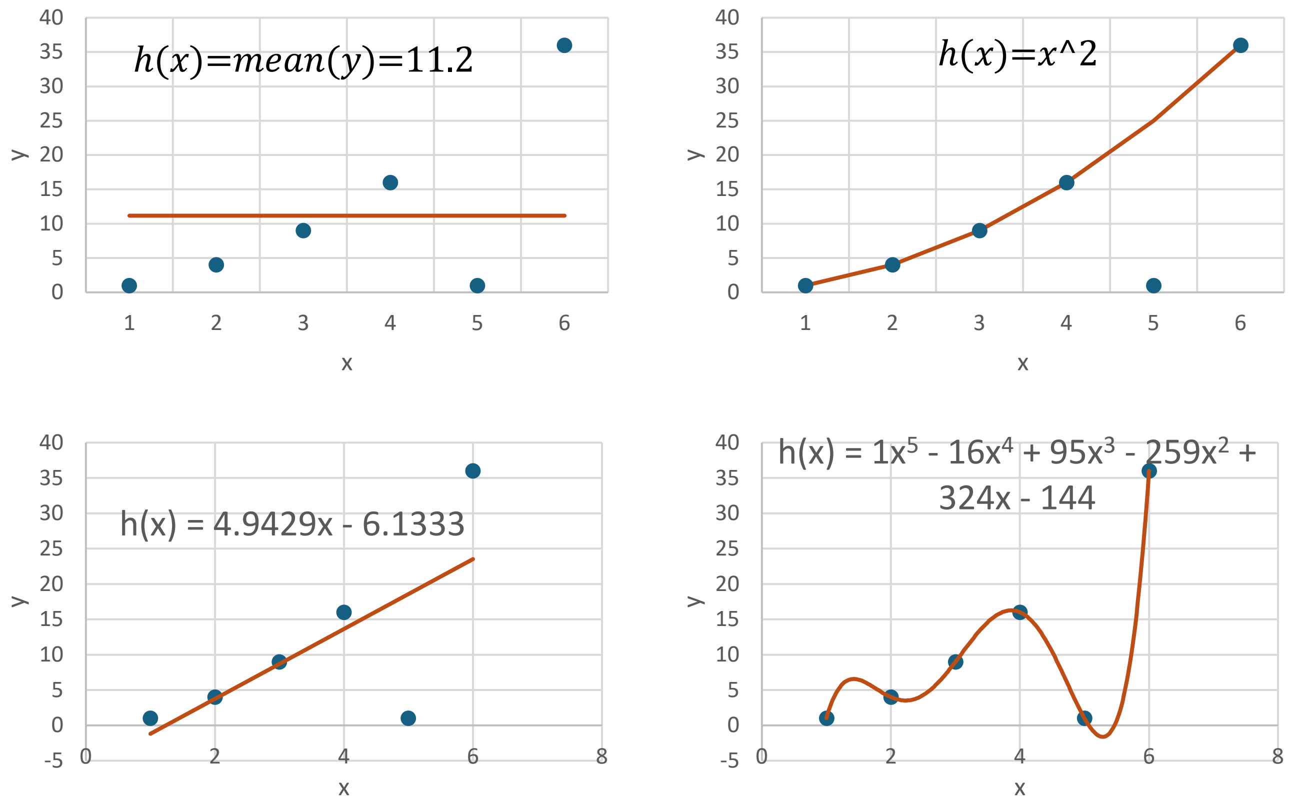
Consistency vs. Simplicity

Example: Univariate curve fitting (regression, function approximation)

Examples $f(x)$



Learned Models



Consistency: $h(x_i) \approx y_i$ (minimize the error)
Simplicity: small number of model parameters.

Measuring Consistency using Loss

Goal of learning: Find a hypothesis that makes predictions that are consistent with the examples $\mathrm{E} = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_i, y_i), \ldots, (\boldsymbol{x}_N, y_N)$.

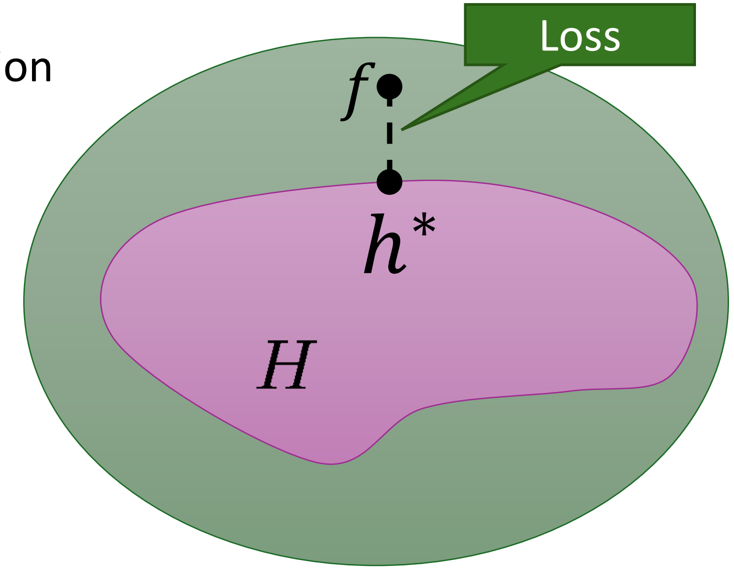
That is,
$$\hat{y} = h(\boldsymbol{x}) \approx y.$$

Measure mistakes: Loss function $L(y, \hat{y}) = L(f(\boldsymbol{x}), h(\boldsymbol{x}))$

- Absolute-value loss $L_1(y, \hat{y}) = |y - \hat{y}|$
- Squared-error loss $L_2(y, \hat{y}) = (y - \hat{y})^2$

For Regression

- 0/1 loss $L_{0/1}(y, \hat{y}) = 0 \text{ if } y = \hat{y}, \text{else } 1$ ← For Classification
- Cross-entropy loss and many others…



Learning Consistent Approximations

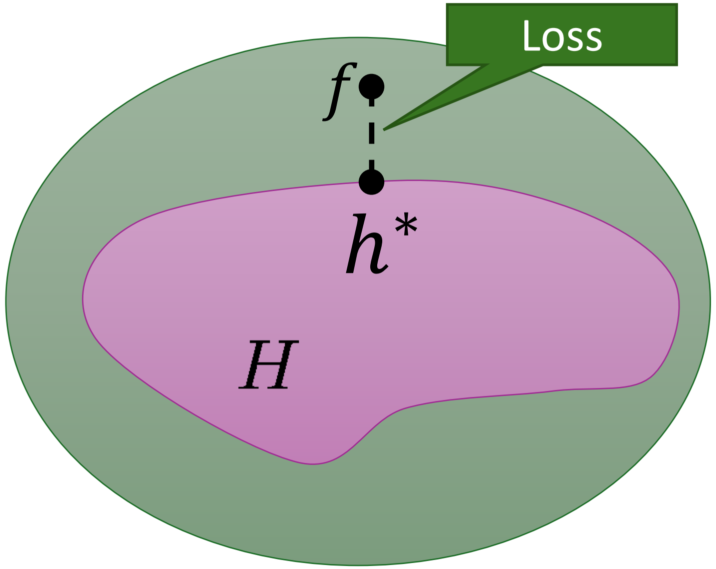
- Empirical loss

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

- Find the best hypothesis that minimizes the loss

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

- Reasons for $h^* \neq f$
  a) Realizability: $f \notin H$
  b) f is nondeterministic.
  c) It is computationally intractable to search all of H, so we use a non-optimal heuristic like local search.



The Most Consistent Classifier

For 0/1 loss, the empirical loss is minimized by the model that predicts for each $\boldsymbol{x}$ the most likely class y using MAP (Maximum a posteriori) estimates. This is called the Bayes classifier.

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

Optimality: The **Bayes classifier is optimal for 0/1 loss.** It is the most consistent classifier possible with the lowest possible error called the **Bayes error rate**. No better classifier exists for 0/1 loss!

Issue: The classifier requires to learn $P(\boldsymbol{x} \mid y)\, P(y) = P(\boldsymbol{x}, y)$ from the examples.

- It **needs the complete joint probability** which requires in the general case a probability table with one entry for each possible feature combination in vector $\boldsymbol{x}$.
- This is impractical (unless a simple Bayesian network exists).
  Most classifiers try to approximate the Bayes classifier using a **simpler model** with fewer parameters.

Simplicity

Ease of use

- Simpler hypotheses have fewer model parameters to estimate and store. Also makes prediction faster.

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) **Model bias:** Restrict H to simpler models (e.g., assumptions like independence, or only consider linear models).

b) **Feature selection:** use fewer variables from the feature vector $\boldsymbol{x}$.

c) **Regularization:** directly penalize the model for its complexity (e.g., number of parameters)

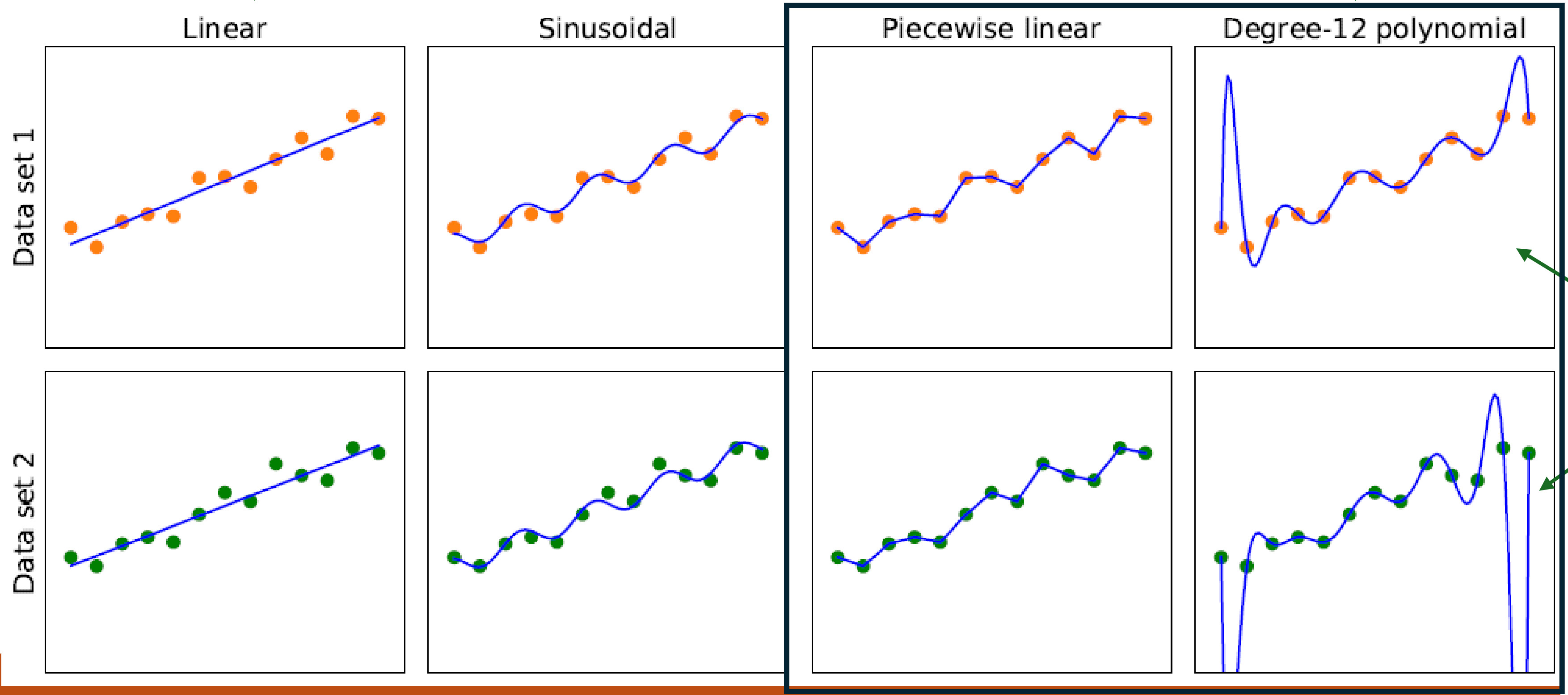
$$h^* = \underset{h \in H}{\operatorname{argmin}} \left[EmpLoss_{L,E}(h) + \underbrace{\lambda\, Complexity(h)}_{\textbf{Penalty term}} \right]$$

Model Selection: Bias vs. Variance

Overfitting

Simpler

More consistent



Points: Two samples from the same generating function f.

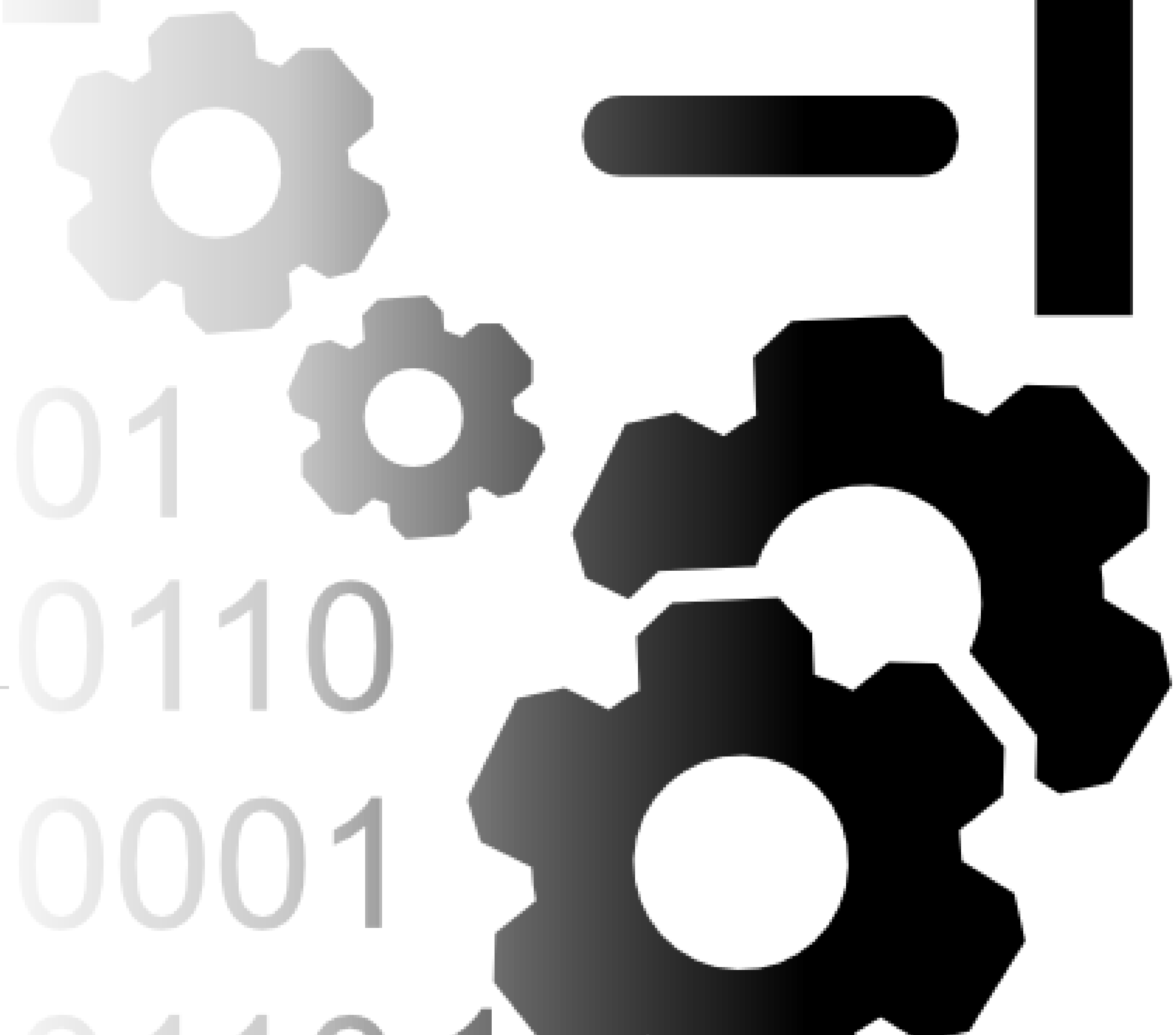
Lines: the learned model function h^*.

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

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

This is a tradeoff
The right choice depends on the application.

Training Data



The Dataset

Feature vector $\boldsymbol{x}$
(Features, Variables, Attributes)

Class Label y

Examples (Instances, Observation)

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

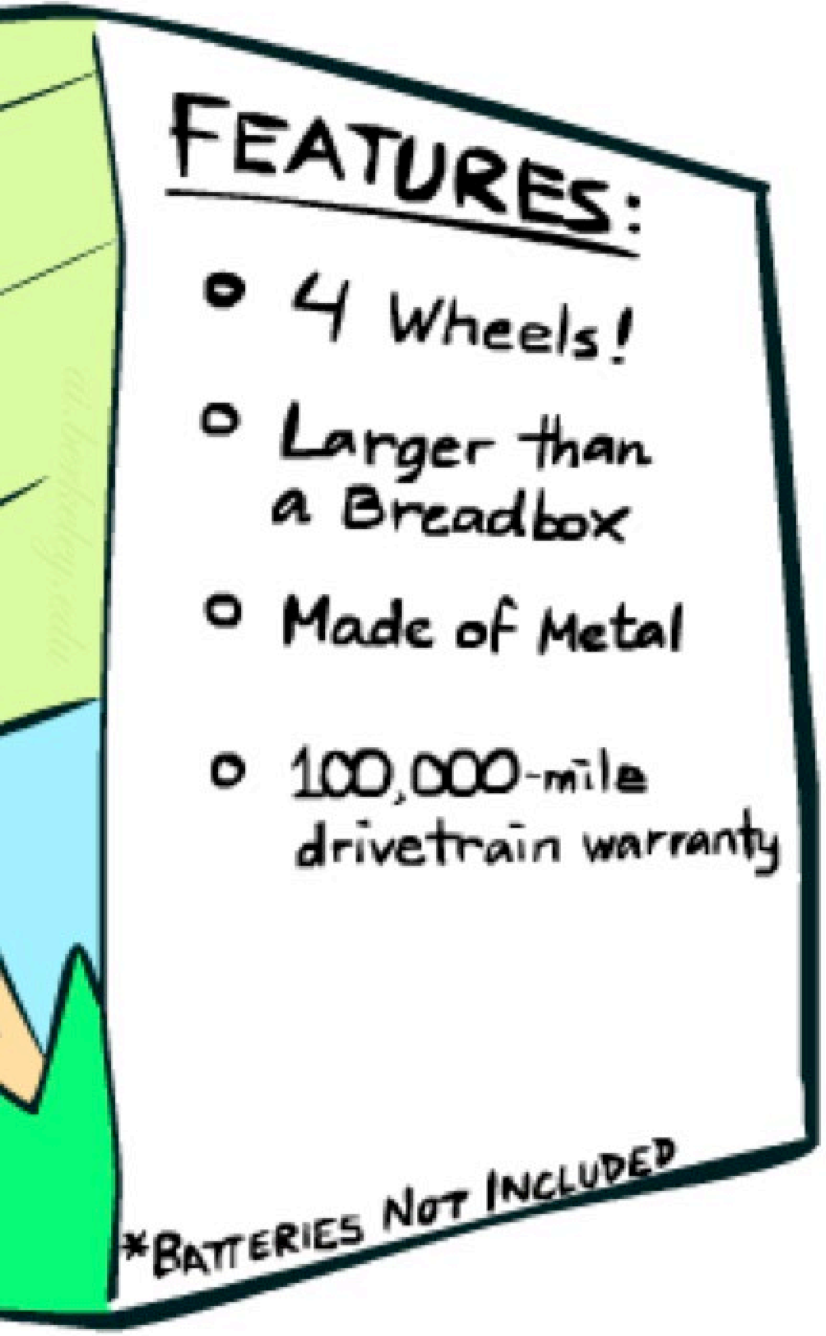
Hungry

Patrons

Reservation

Wait time

Task: Find a hypothesis (called "model") to predict the class given the features.



Feature Engineering

- Add more information sources as new variables to the model.
- Add **derived features** that help the classifier (e.g., $x_1 x_2$, x_1^2, $\ln(x_1)$).
- **Embedding**: E.g., convert words to vectors where vector similarity between vectors reflects semantic similarity.
- **Feature Selection**: Which features should be used in the model is a model selection problem (choose between models with different features).
- (Deep) neural networks can perform "automatic" feature engineering called **end-to-end machine learning**.

- Example for Spam detection: In addition to words, add features for:
  - Have you emailed the sender before?
  - Have 1,000+ other people received the same email?
  - Is the email in ALL CAPS?

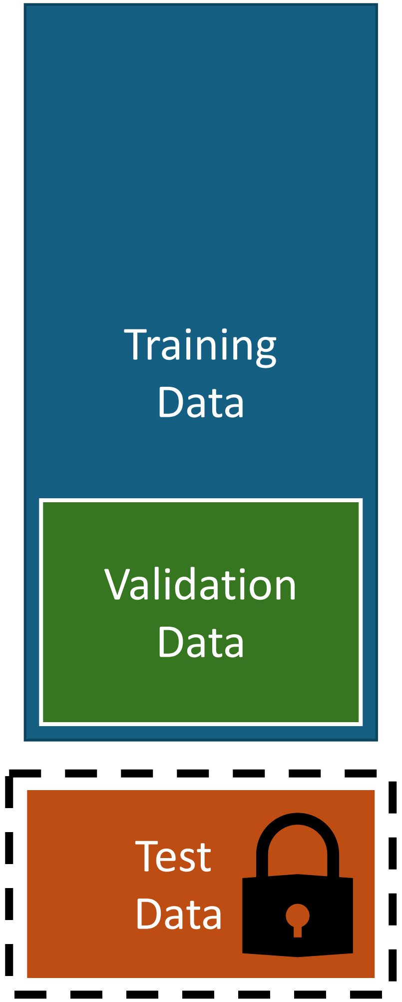
Training Data in AI

- Training Data in AI can come from many sources
  - **Existing Data**: Download documents from the internet to train Large Language Models.
  - **Observation**: Record video of a task being performed (e.g., for self-driving cars).
  - **Simulation**: E.g., simulated games using a playout strategy.
  - **Expert feedback** on how well a task was performed.

Training and Testing

Training a Model

1. Hold a **test set** back to estimate the generalization error (often 20%).
2. Hold a **validation data** set back from the training data (often 20%).
3. **Learn different models** using the training set with different **hyperparameters** (learning rate, regularization λ, maximal decision tree depth, selected features,…). Often, a grid of possible hyperparameter combinations or some greedy search is used.
4. **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/hyper parameter tuning**.
5. **Learn the final model** with the chosen hyperparameters using all training (including validation data).



- Notes:
  - The validation set was not used for training with different hyperparameters, so we get an estimate of the generalization error for comparing different hyperparameter settings.
  - If no model selection is necessary, then no validation set is used.

Testing: Evaluating the Final Model

The final model was trained on the training examples E. We want to test how well the model will perform on new examples T (i.e., how well it **generalizes to new data**). We use the held-back test data.

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

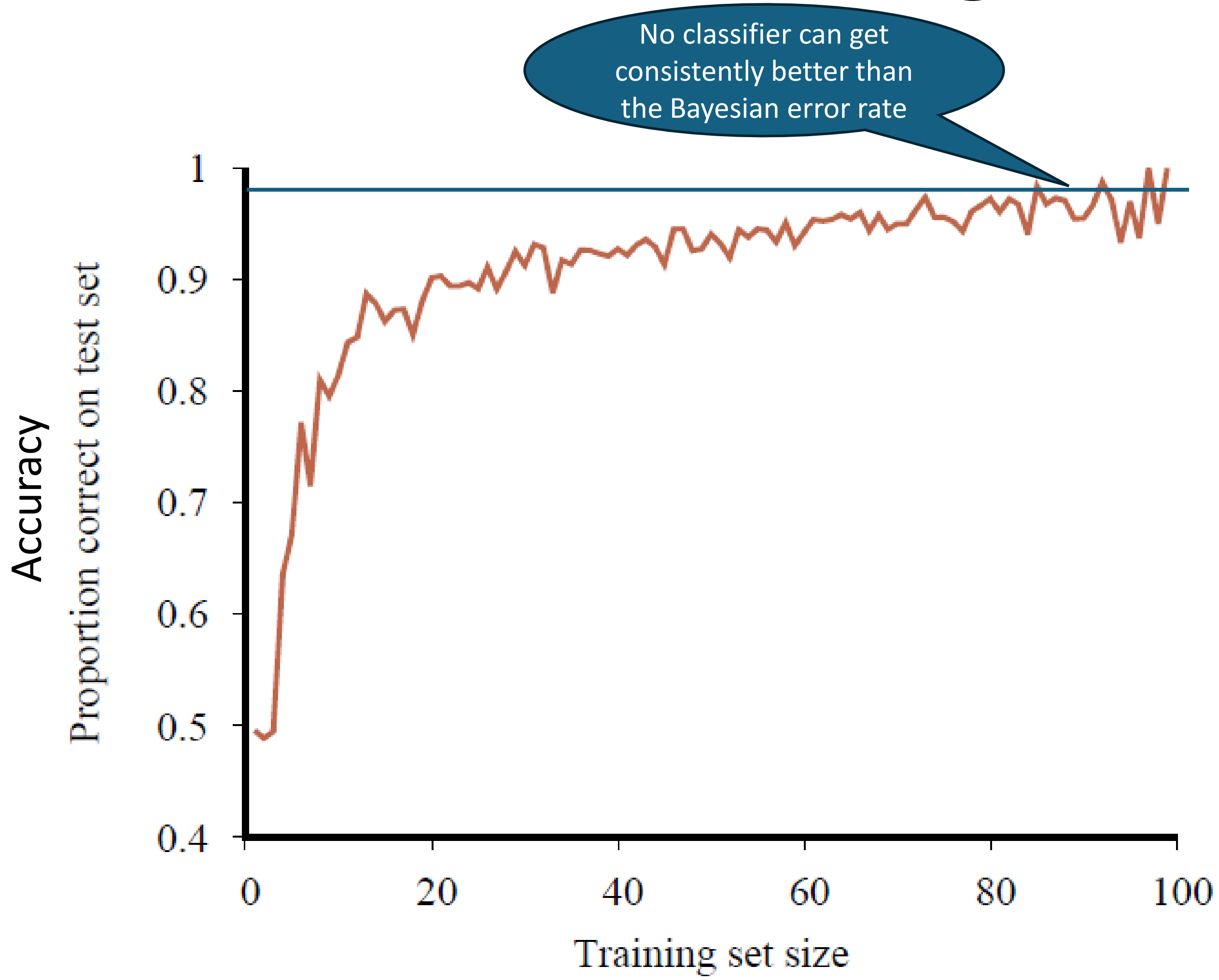
$$EmpLoss_{L,T}(h) = \frac{1}{|T|} \sum_{(\boldsymbol{x},y)\in T} L(y, h(\boldsymbol{x}))$$

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

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

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

Learning Curve:
The Effect the Training Data Size



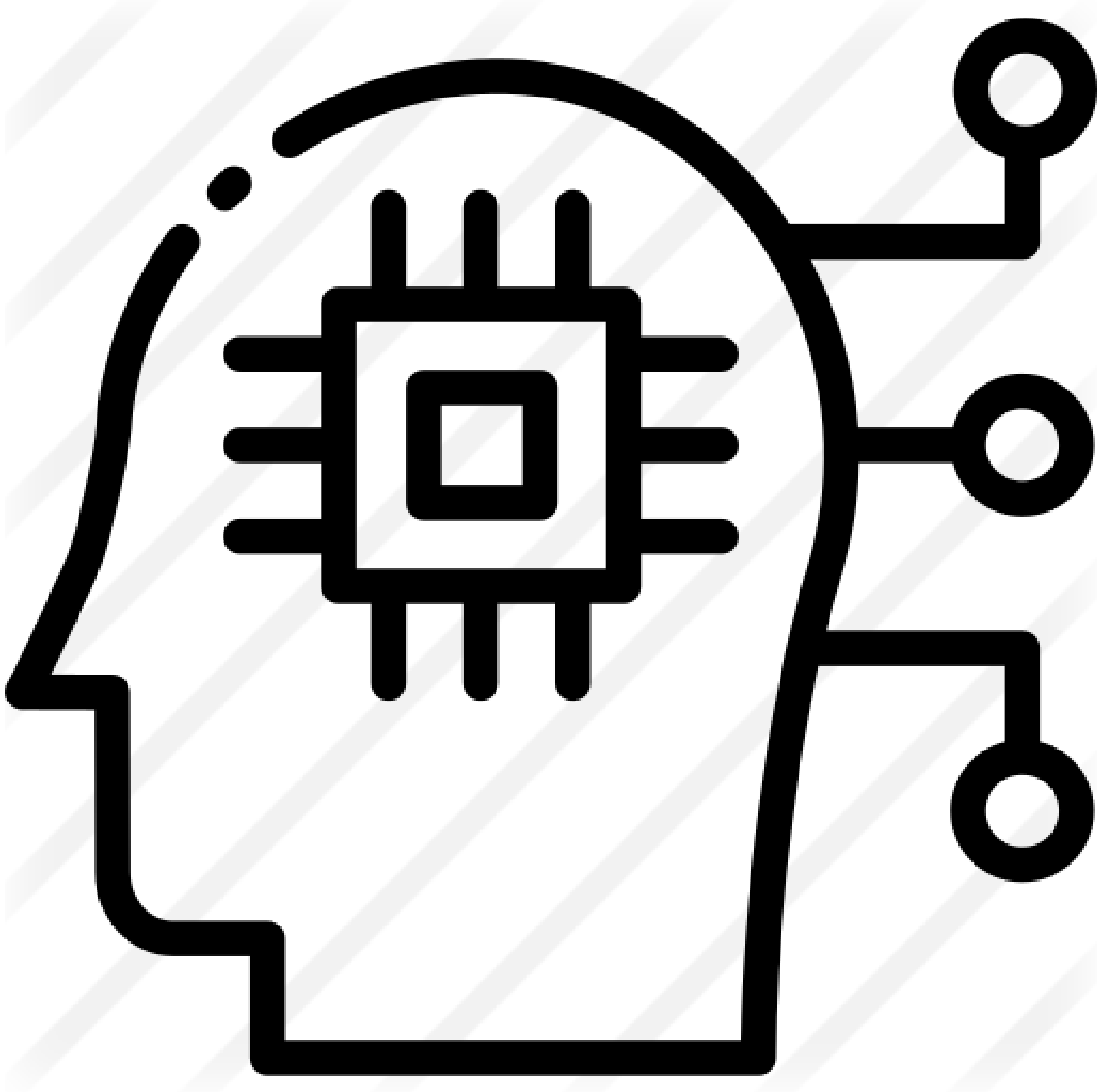
Accuracy increases when the amount of available training data increases.

More data is better!

At some point, the learning curve flattens out, and more data does not contribute much!

Comparing to a Baselines

- First step: get a **baseline**
  - The baseline is often a simple model.
  - Helps to determine how hard the task is.
  - Helps determine what a good accuracy is.
- **Weak baseline**: E.g., the most frequent label classifier
  - Gives all test instances the same label which is the most common label in the training set.
    - Example: For spam filtering, give every message the label “ham” since most messages are ham.
  - Accuracy might be very high if the problem is skewed (called class imbalance).
    - Example: If calling everything “ham” gets already 66% right, then a classifier that gets 68% isn’t a big improvement!
- **Strong baseline:** For research, we typically compare our model with the performance of previously published state-of-the-art methods.



Types of Supervised ML 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: $L(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$
- Gradient: $\nabla L(\boldsymbol{w}) = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$
- Minimum loss: $\nabla L(\boldsymbol{w}) = 0$

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

The gradient is a vector of partial derivatives

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

- **Solution**: Gradient descent

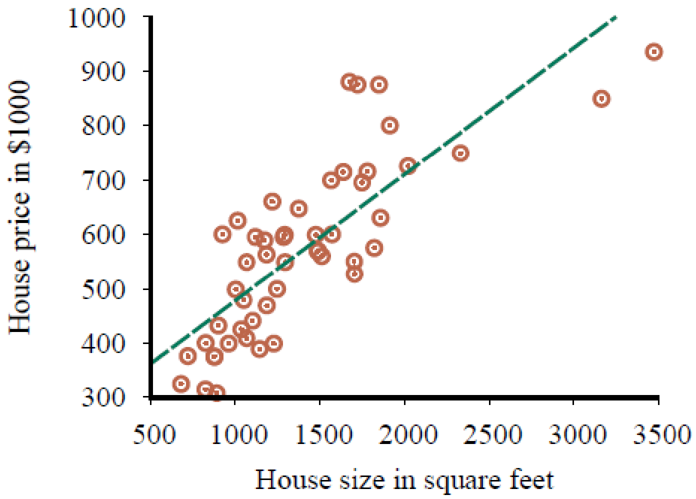
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla L(\boldsymbol{w})$$

- **Alternative**: Analytical solution

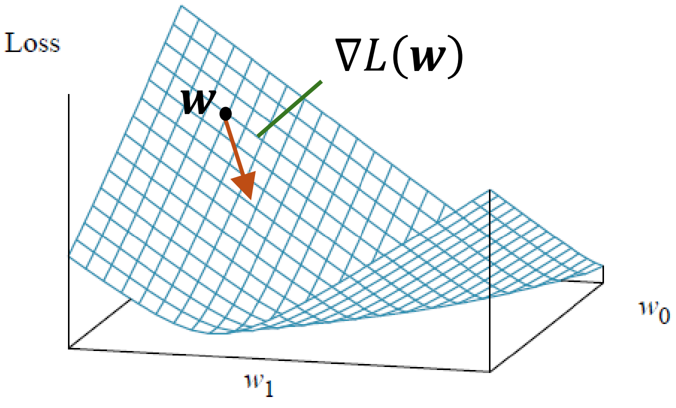
$2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = \boldsymbol{0}$ (solve for w)

$\boldsymbol{w}^* = (\boldsymbol{X}$

Pseudo inverse



(a)



Convex loss function

(b)

Naïve Bayes Classifier

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

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

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

  The priors $P(y)$s and the likelihoods $P(\boldsymbol{x}_i \mid y)$s are estimated from the data by (smoothed) counting.

- **Gaussian Naïve Bayes Classifiers** extend the approach to **continuous features** by modeling the feature likelihood for each class y as a Gaussian probability density:

$$P(\boldsymbol{x}_i \mid y) \sim N\left(\mu_y, \sigma_y\right)$$

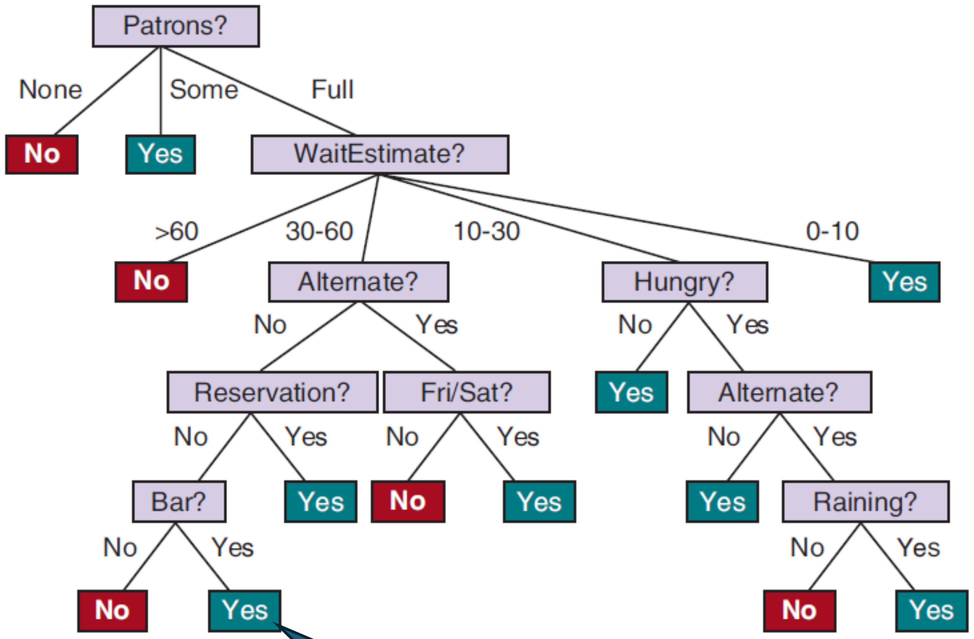
  The parameters for the normal distribution $N\left(\mu_y, \sigma_y\right)$ are estimated from data.

Decision Trees

Bayes Classifier

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

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	$y_{12} = Yes$



Class labels for leaf nodes are decided by the majority of the training data ending up in the leaf node. The probability can be estimated as:

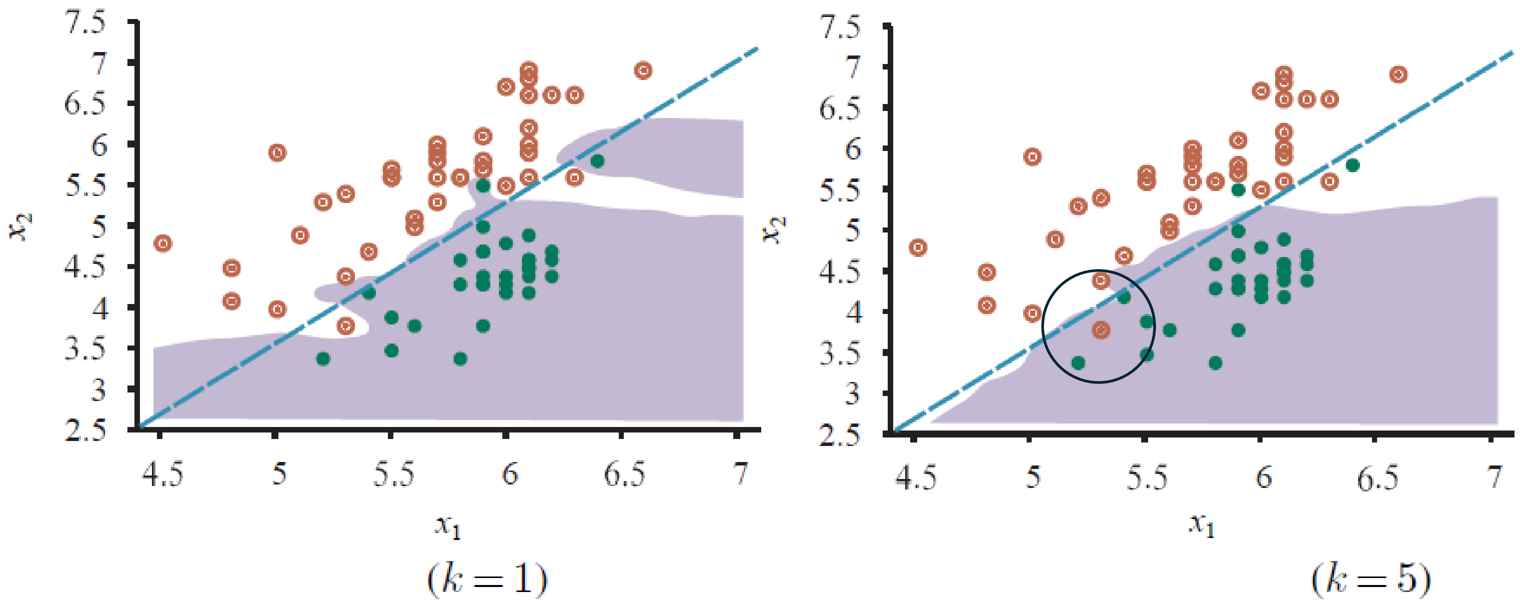
$$\hat{P}(\text{Yes}|\text{node N}) = \frac{N_{Yes}}{N_{Yes} + N_{No}}$$

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

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

K-Nearest Neighbors Classifier

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

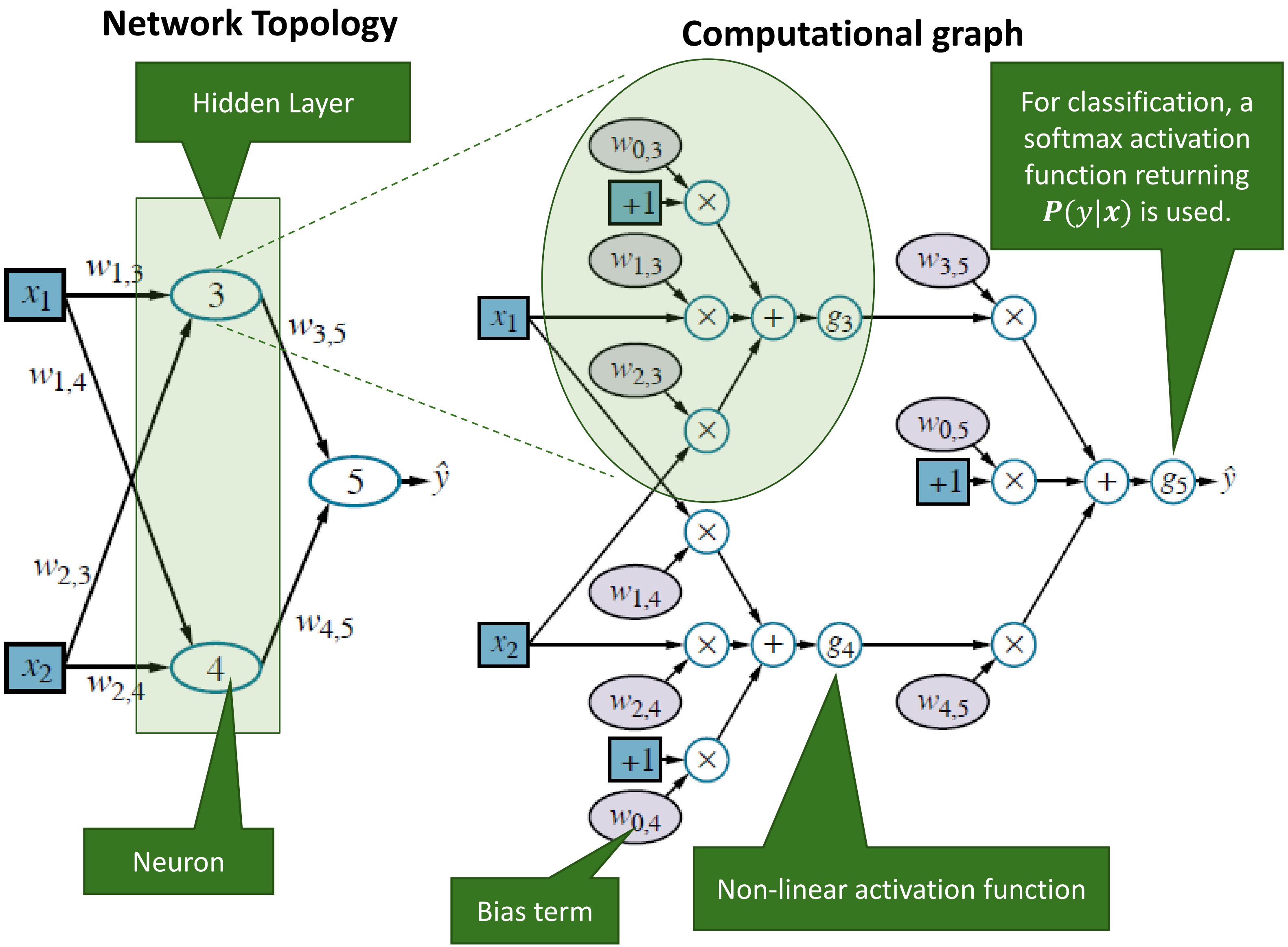


- Class is predicted by looking at the majority in the set of the k nearest **neighbors**. k is a hyperparameter. Larger k smooths the decision boundary.
- Neighbors are found using a distance measure (e.g., Euclidean distance between points).
- Approximates a Bayesian classifier by

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

Most Used in AI: Artificial Neural Networks

Superscript $[n]$ means the layer. Layer weights are collected in a matrix.



- Represent
  $$\hat{y} = h(\boldsymbol{x}) = g^{[2]}\left(\boldsymbol{W}^{[2]}\, g^{[1]}(\boldsymbol{W}^{[1]}\boldsymbol{x})\right)$$
  as a network of weighted sums with non-linear **activation functions** $g(\cdot)$ (e.g., sigmoid, ReLU).
- Learn weight matrices $\boldsymbol{W}$ from examples using gradient descent with **backpropagation** of prediction errors $L(\hat{y}, y)$.
- ANNs are **universal approximators**. Large networks can approximate any function (has no bias). **Regularization** is typically needed to avoid overfitting.
- **End-to-end learning**: The hidden layer performs “automatic feature engineering
- **Deep learning** adds more hidden layers and layer types (e.g., convolution layers) for more efficient learning and transfer learning.

Typical Use of Supervised ML for Intelligent Agents

Learn a Policy

- Classification: Directly learn the best action for each state from examples.

  $a = h(state\ features)$

- This model can also be used as a **playout policy** for Monte Carlo tree search with data from self-play.

Learn Evaluation Functions

- Regression: Learn evaluation functions to estimate state utilities.

  $eval = h(state\ features)$

- Can learn a **heuristic** for heuristic alpha-beta search.
- For reinforcement learning we can learn action values $q(state, action)$.

Learn Perception and Actuation

- **Natural language processing:** Use deep learning / word embeddings / language models to understand concepts, translate between languages, or generate text.
- **Speech recognition**: Identify the most likely sequence of words.
- **Vision**: Object recognition in images/videos. Generate images/video.
- **Robotics**: Learn how to move safely.

Compressing Tables

- Neural networks can be used as a compact representation of tables that do not fit in memory. E.g.,
  - Joint and conditional probability tables
  - State utility tables (i.e., an evaluation function)
  - Q-Value tables in reinforcement learning

Bottom line: Learning a function is often more effective than hard-coding it. However, we do not always know how it performs for rare and edge cases!