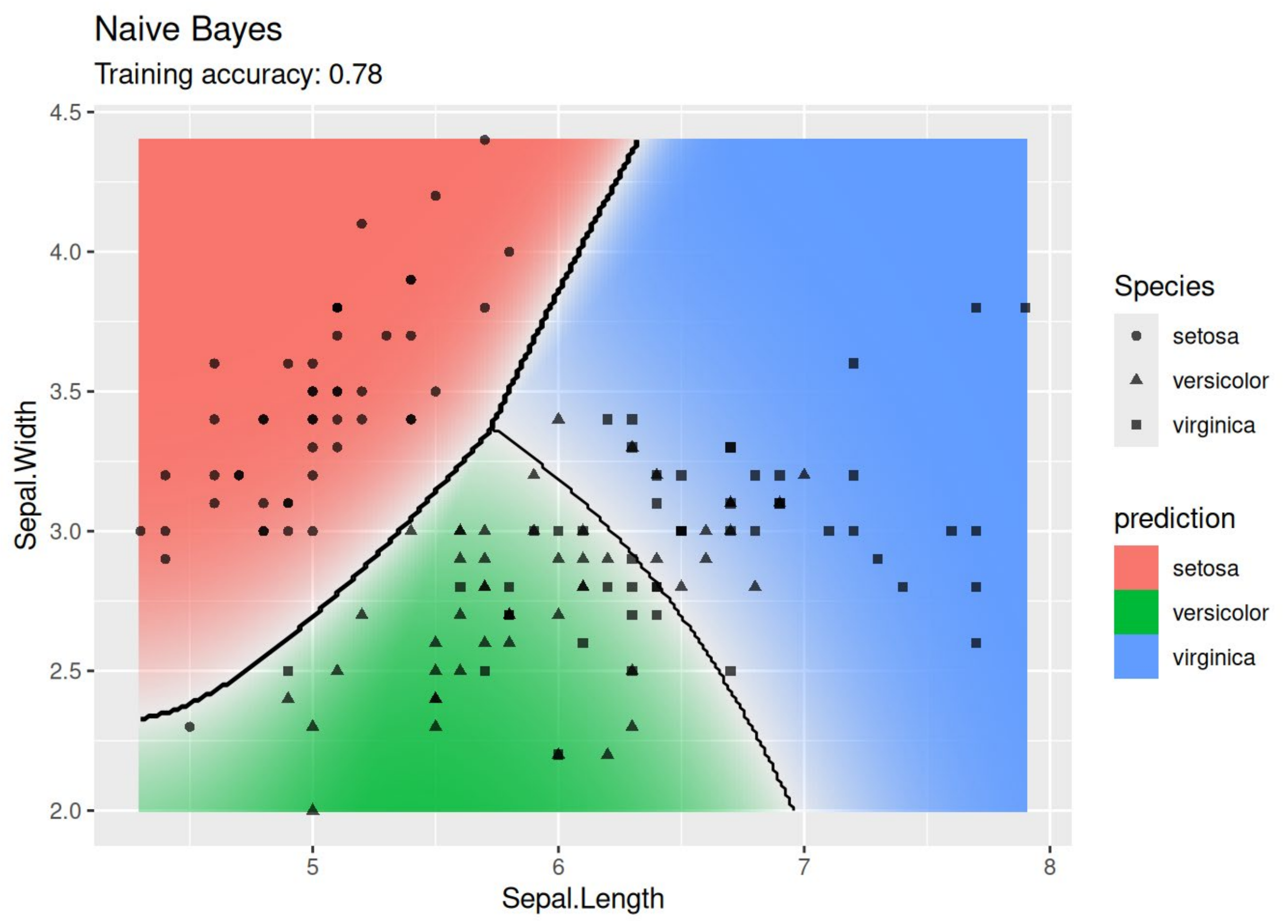
CS 5/7320
Artificial Intelligence

Learning from Examples: Machine Learning

AIMA Chapter 19

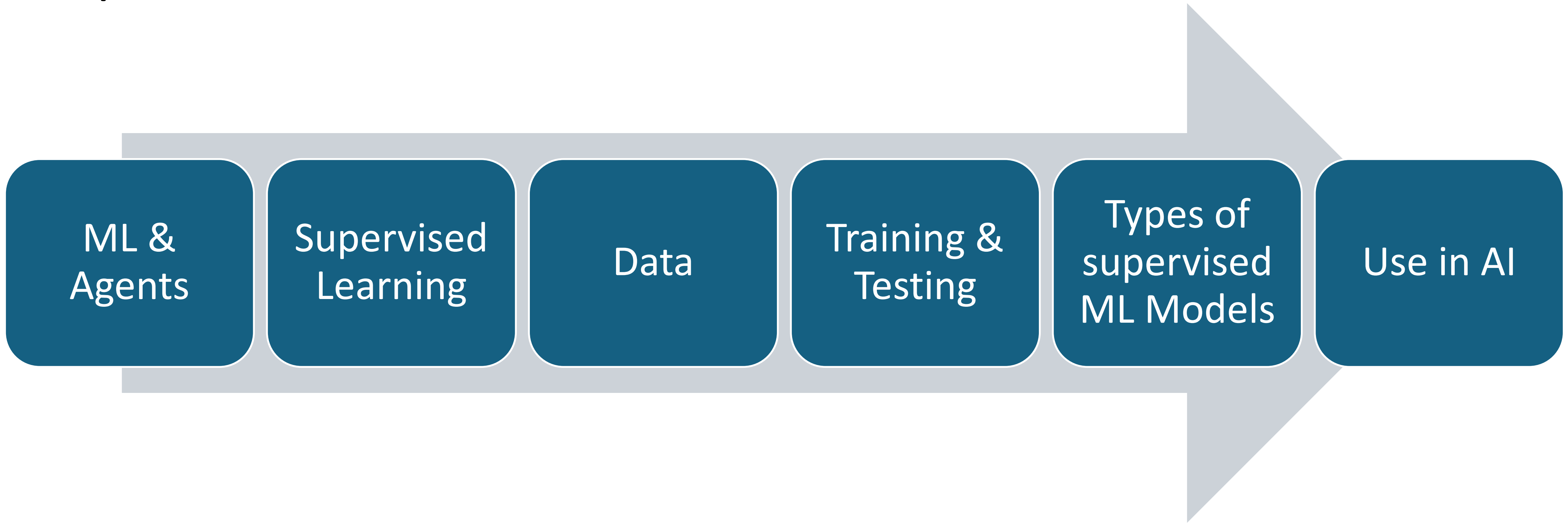
Slides by Michael Hahsler

Based on slides by Dan Klein, Pieter Abbeel, Sergey Levine, and A. Farhadi (http://ai.berkeley.edu) with figures from the AIMA textbook.



Online Material

Topics


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

ML and Agents

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: Machine Learning

Up until now in this course:

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

Issues

- Designer cannot anticipate all possible future situations.
- Designer may have examples but does not know how to program a solution.

Machine Learning

- Learning = Improving performance after making observations about the world. That is, learn what works and what does not.
- We learn a model that decides on the actions to take. This is called the "performance element."
- The goal is to get closer to optimal decisions. I.e., it is an optimization problem.

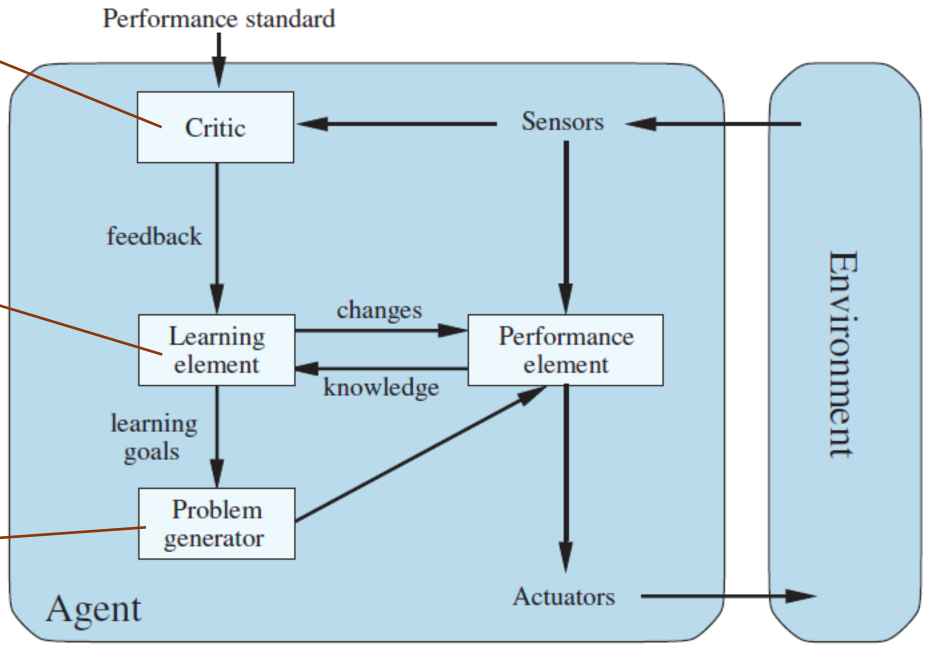
From Chapter 2: Agents that Learn

The **learning element** modifies the performance element to improve its performance.

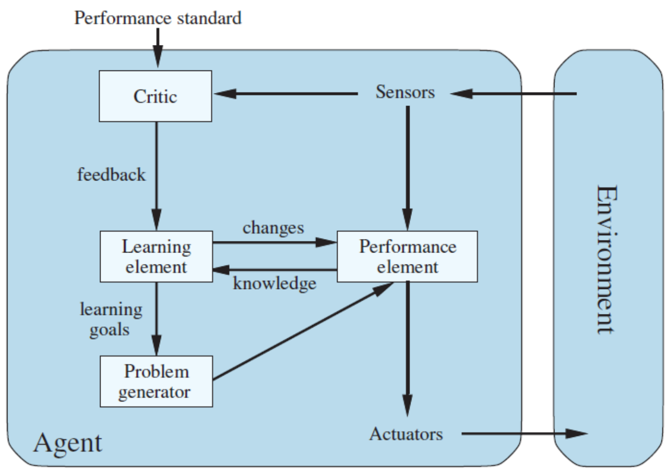
Critic: How is the agent currently performing?

Learning Element: Update the performance element and changes how it selects actions.
E.g., adding rules, changing weights

Problem generators: Explore actions that lead to better information



Considerations for Using Machine Learning in Agents



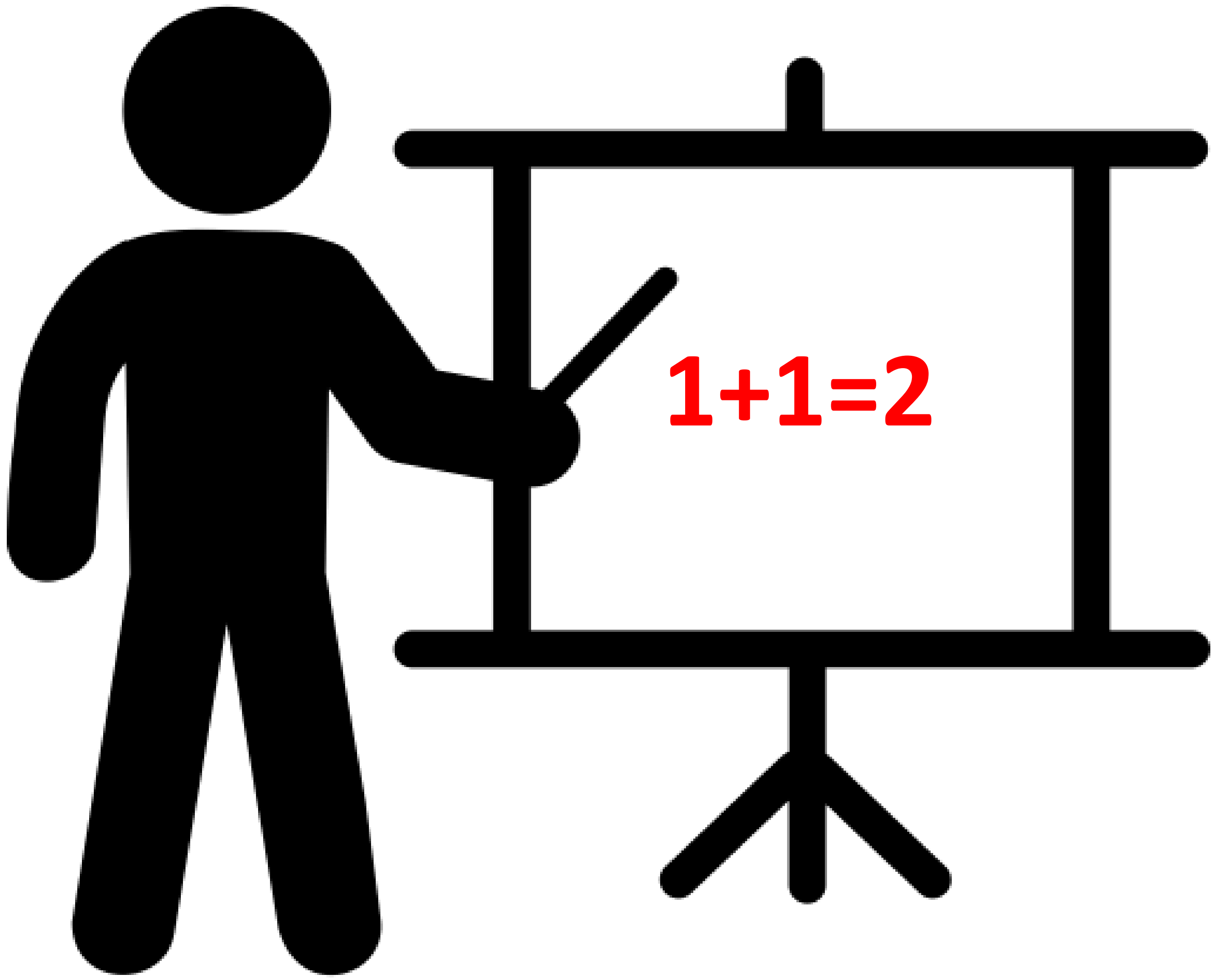
1. What **component** of the performance element is learned?
   E.g., how to select an action, estimate the utility of a state, …

2. What **representation** (model) is used in the component?
   Linear regression, rules, trees, neural nets,…

3. What **feedback** is available for learning?
   - **Unsupervised Learning**: No feedback, just organize data (e.g., clustering, embedding)
   - **Supervised Learning**: Uses a dataset with correct answers. Learn a function (model) to map an input (e.g., state) to an output (e.g., action or utility). Examples:
     - Use a naïve Bayesian classifier to distinguish between spam/non-spam
     - Learn a playout policy to simulate games (current board -> good move)
   - **Reinforcement Learning**: Learn from rewards/punishment (e.g., winning a game) obtained via interaction with the environment over time.

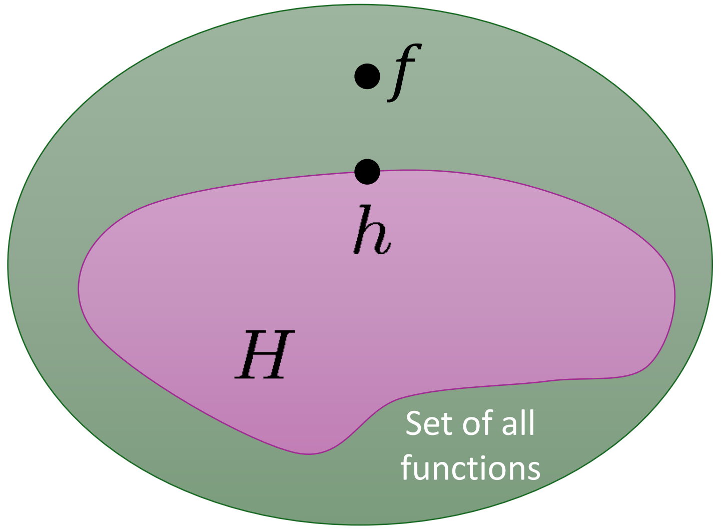
We focus on supervised learning

Supervised Learning



Supervised Learning As Function Approximation

- Examples
  - We assume there exists a 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.



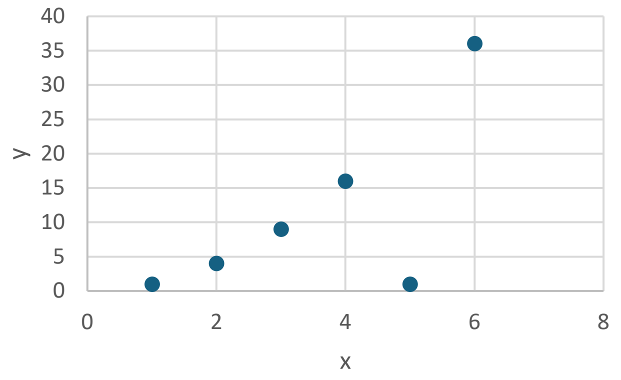
- Learning problem
  - Given a hypothesis space H of representable models.
  - Find a hypothesis $h \in H$ such that $\hat{y}_i = h(\boldsymbol{x}_i) \approx y_i \ \forall i$
  - That is, we want to approximate f by h using E.

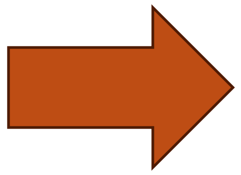
- Supervised learning includes
  - Classification (outputs = class labels). E.g., $\boldsymbol{x}$ is an email and $f(\boldsymbol{x})$ is spam / ham.
  - Regression (outputs = real numbers). E.g., $\boldsymbol{x}$ is a house and $f(\boldsymbol{x})$ is its selling price.

Consistency vs. Simplicity

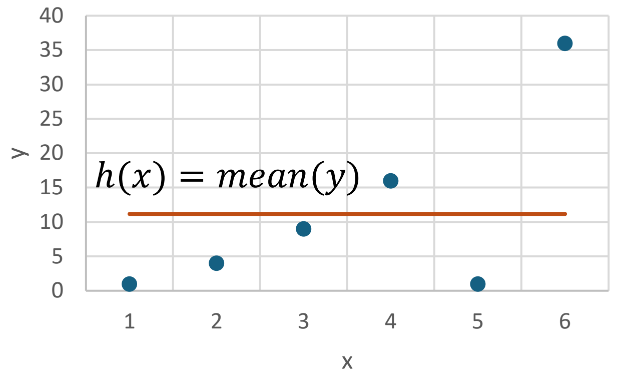
Example: Univariate curve fitting (regression, function approximation)

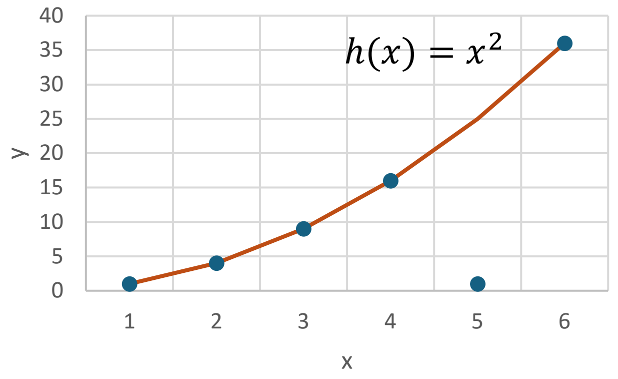
Examples $f(x)$

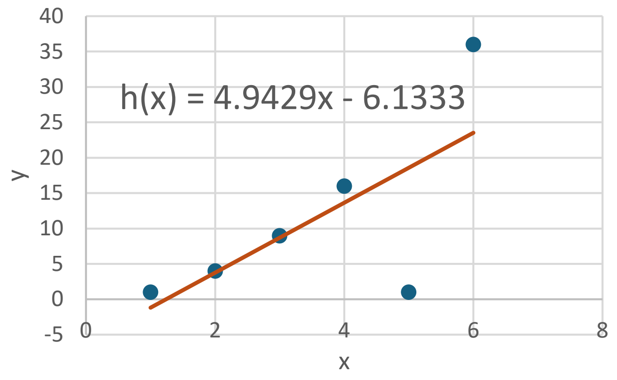


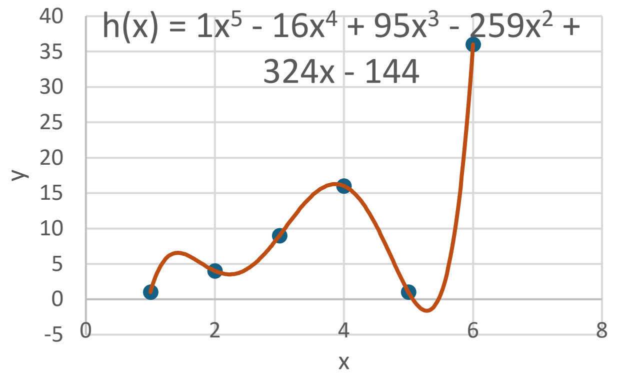


Learned Models









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

Measuring Consistency using Loss

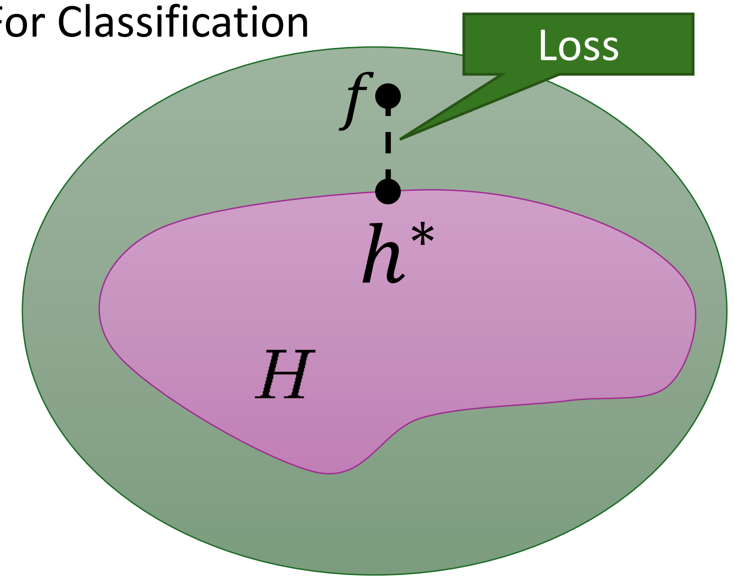
Goal of learning: Find a hypothesis that makes predictions that are consistent with the examples $\mathrm{E} = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_i, y_i), \ldots, (\boldsymbol{x}_N, y_N)$.

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
  - Log loss, cross-entropy loss and many others…



Learning Consistent h by Minimizing the Loss

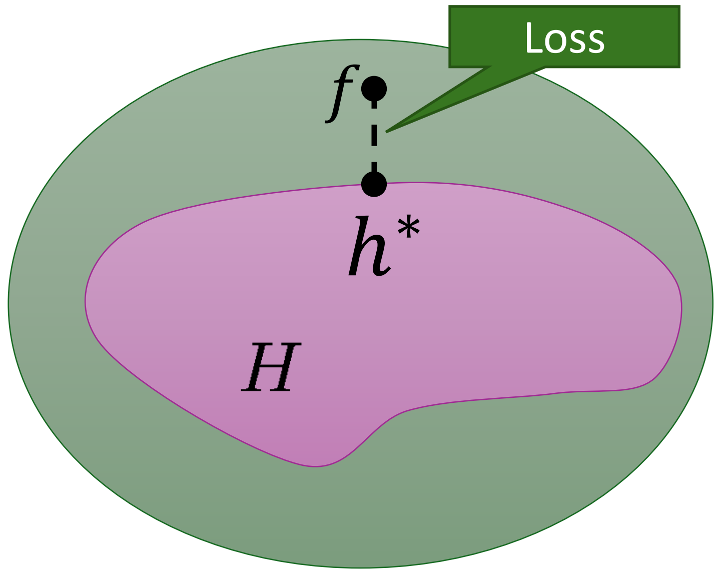
- Empirical loss

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

- Find the best hypothesis that minimizes the loss

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

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



The Most Consistent Classifier
The Bayes 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 value for the feature vector $\boldsymbol{x}$.
- This is impractical (unless a simple Bayes 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.

Generalization: How well does the hypothesis perform on new data?

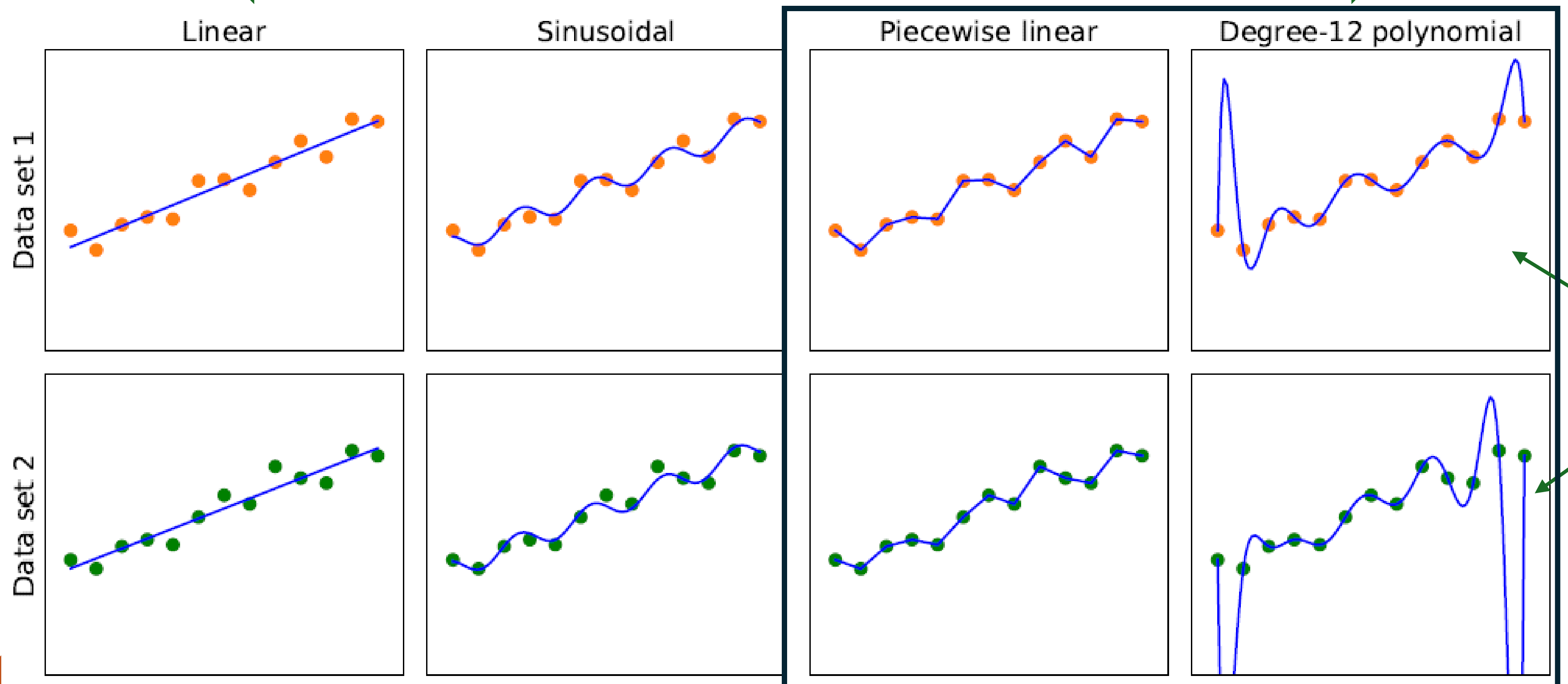
- We do not want the model to be too specific to the training examples (an issue called **overfitting**).
- Simpler models typically generalize better to new examples.

How to achieve simplicity?

a) **Model bias:** Restrict H to simpler models (e.g., assumptions like independence, only consider linear models).
b) **Feature selection:** use fewer variables from the feature vector $\boldsymbol{x}$.
c) **Regularization:** penalize model for its complexity (e.g., number of parameters)

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

Model Selection: Bias vs. Variance



Points: Two samples from the same generating function f.

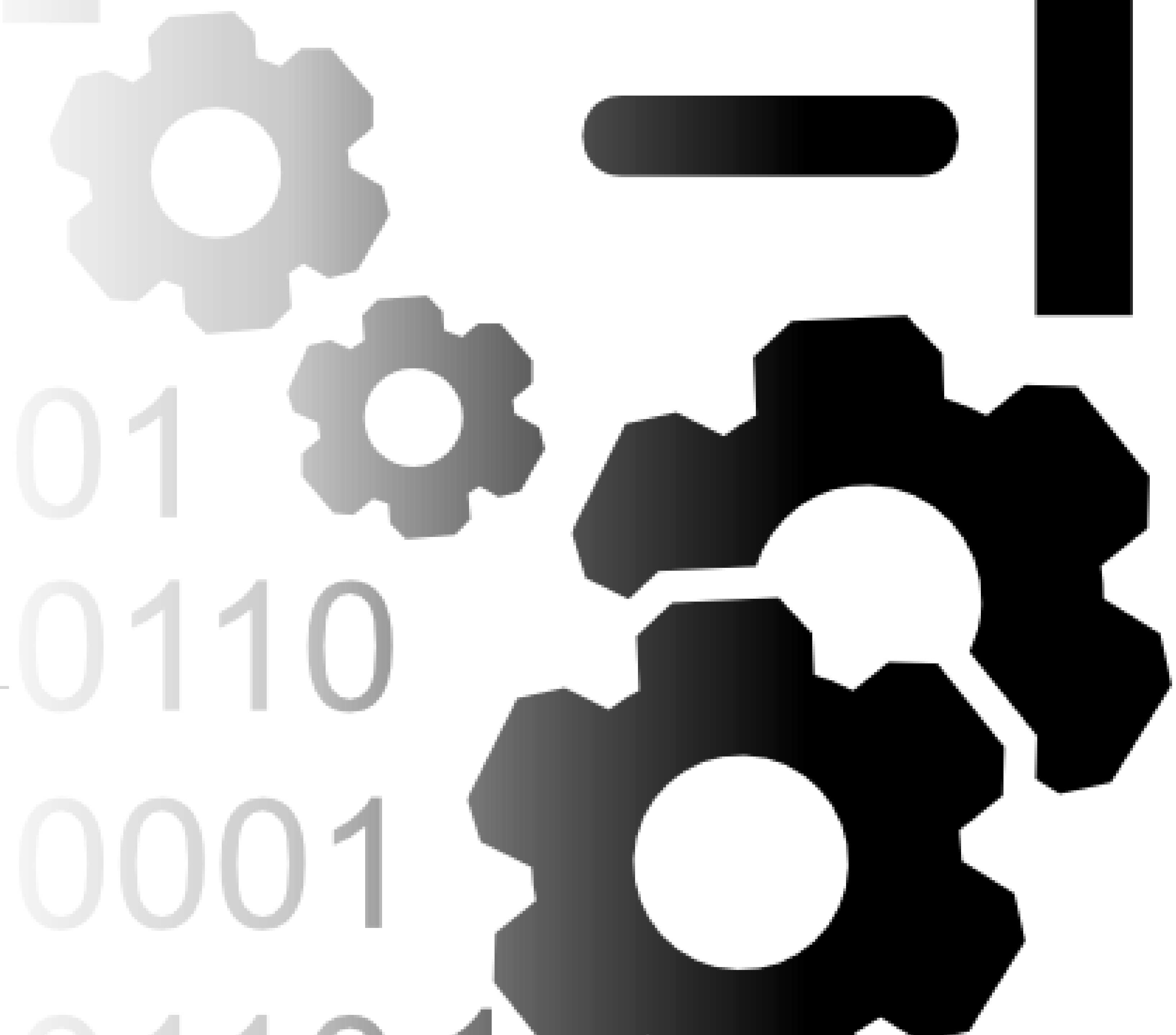
Lines: the learned model function h^*.

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

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

This is a tradeoff

Data

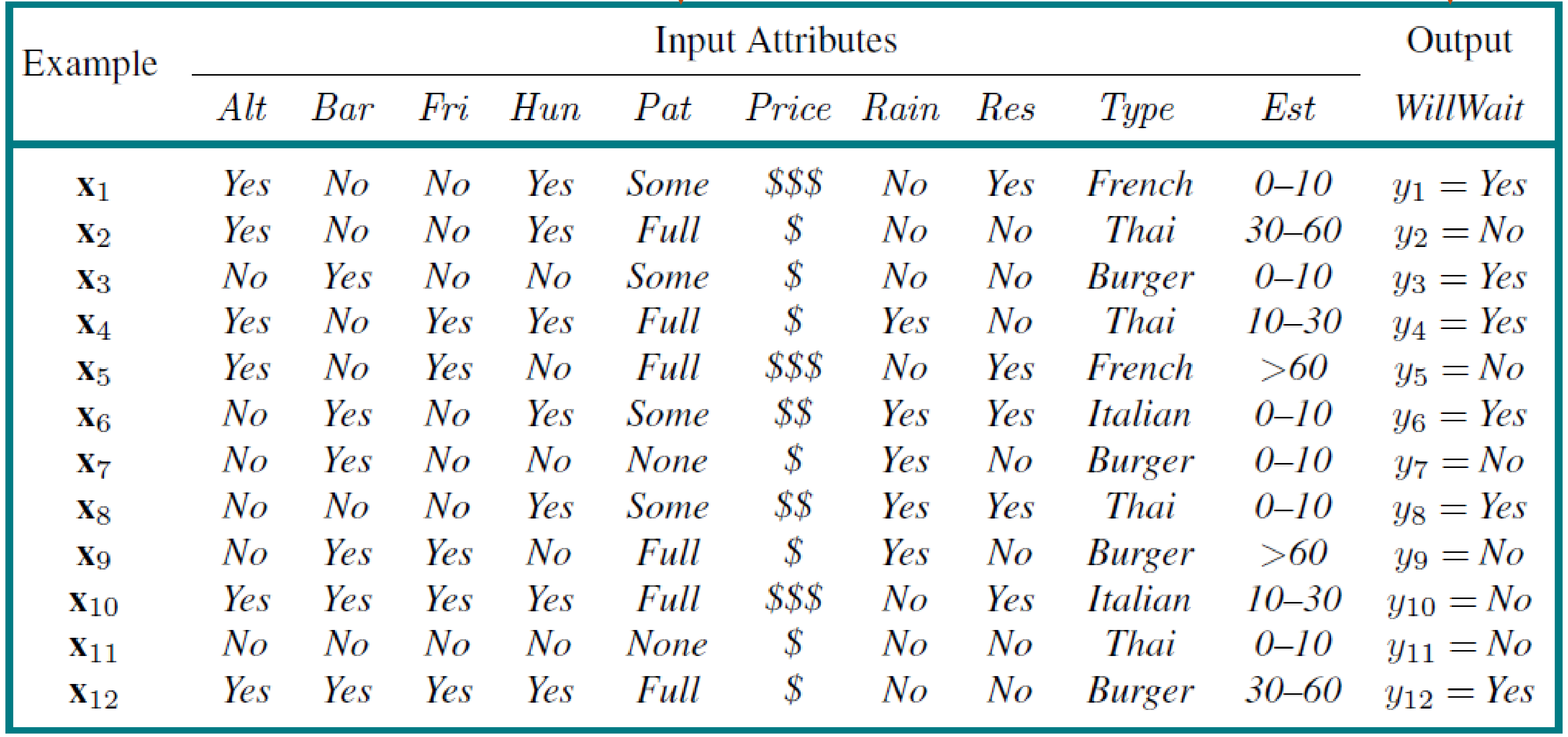


The Dataset

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

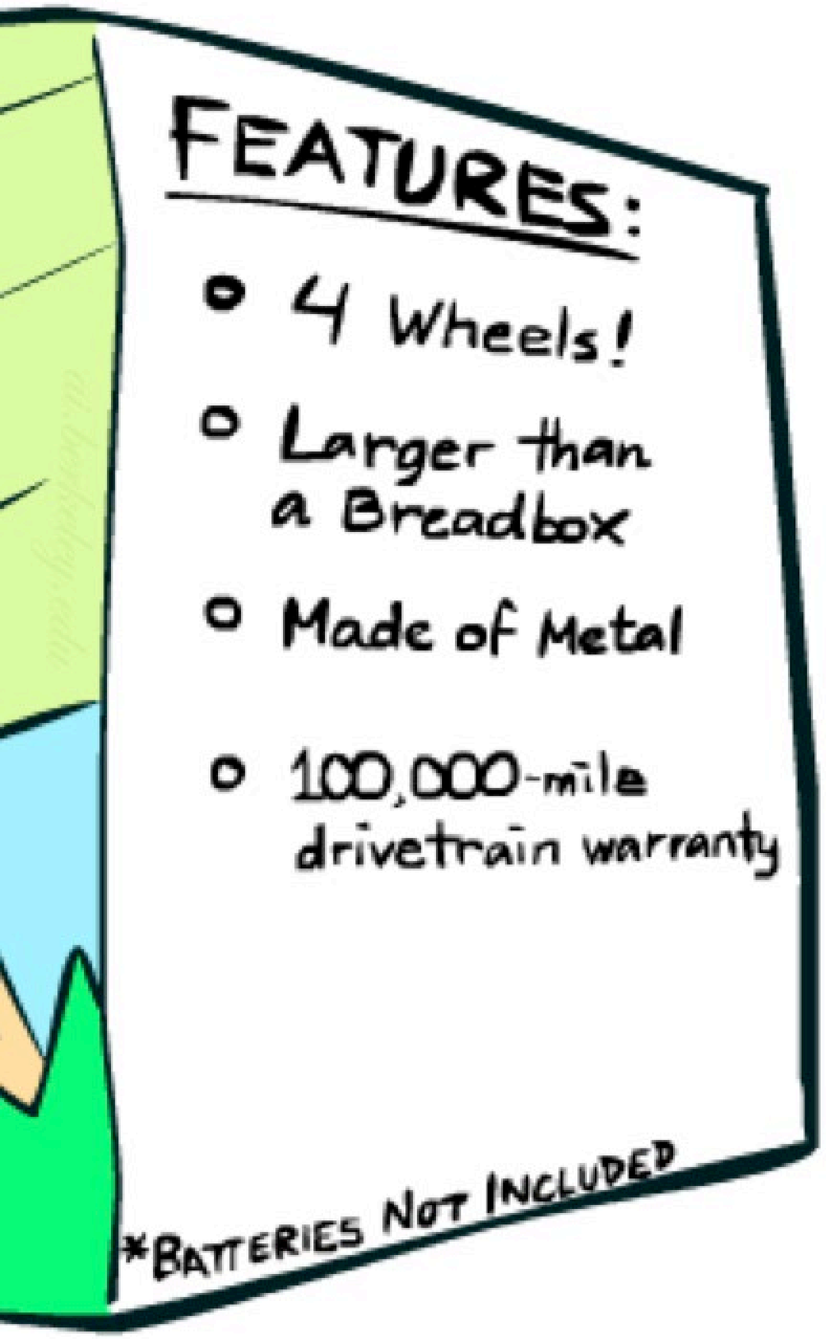
Hungry

Patrons

Reservation

Wait time

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



Feature Engineering

- Add information sources as new variables to the model.
- Add **derived features** that help the classifier (e.g., x_1x_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 1000+ other people just gotten the same email?
  - Is the email in ALL CAPS?

Data in AI

- 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.

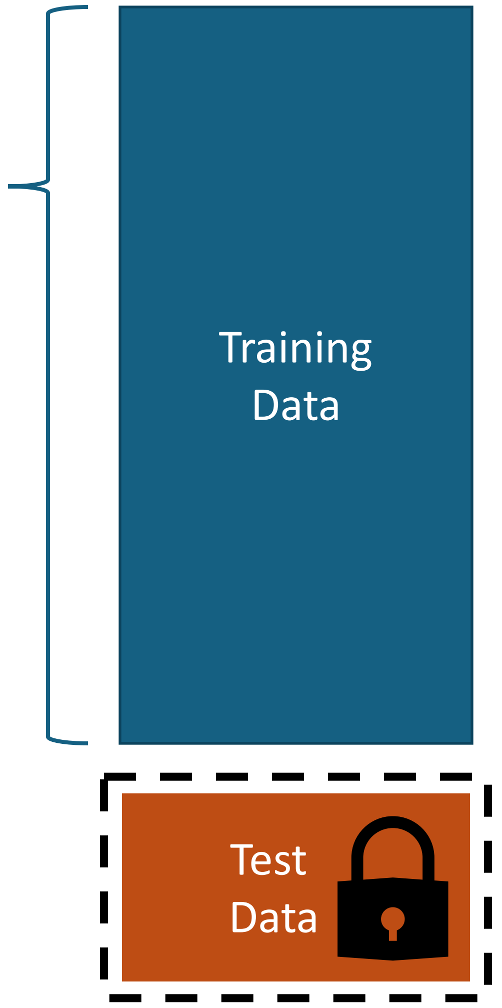
Training and Testing

Training a Model

- A test set is held back to estimate the generalization error.
- Models are "trained" (learned) on **the training data.** This involved estimating:

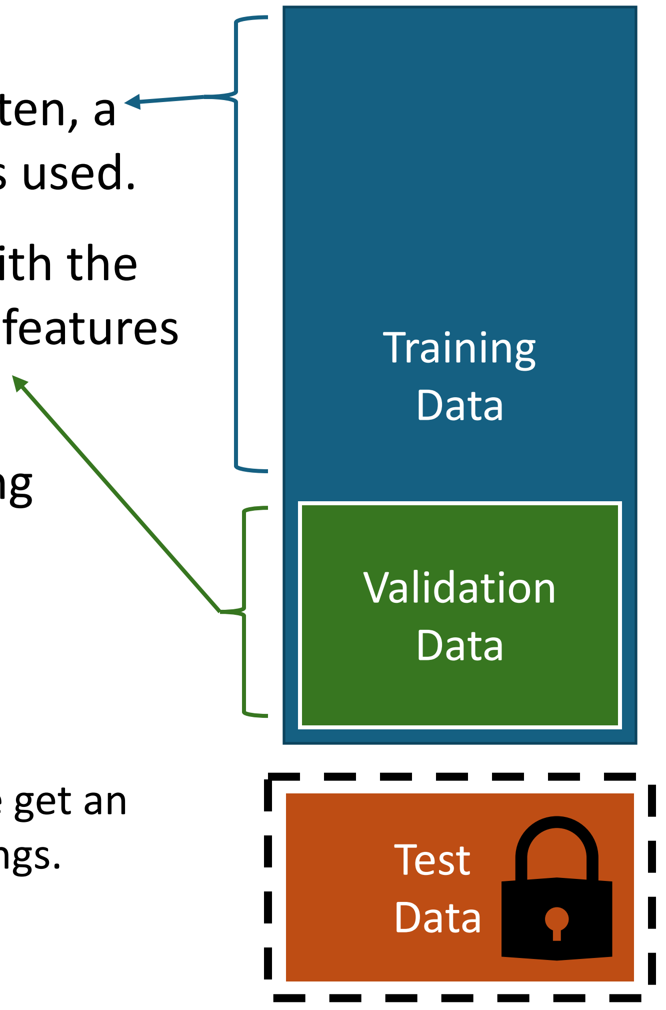
  1. **Model parameters** (the model): E.g., probabilities, weights, ...
  2. **Hyperparameters**: Many learning algorithms have choices for learning rate, regularization λ, maximal decision tree depth, selected features,... The algorithm tries to optimize the model parameters given user-specified hyperparameters.

- We need to select the type of algorithm and the hyperparameters. This is called **model selection**.



Hyperparameter Tuning/Model Selection

1. Hold a validation data set back from the training data.
2. **Learn models** using the training set with different hyperparameters. Often, a grid of possible hyperparameter combinations or some greedy search is used.
3. **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**.
4. 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.

Model Evaluation (Testing)

The 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.

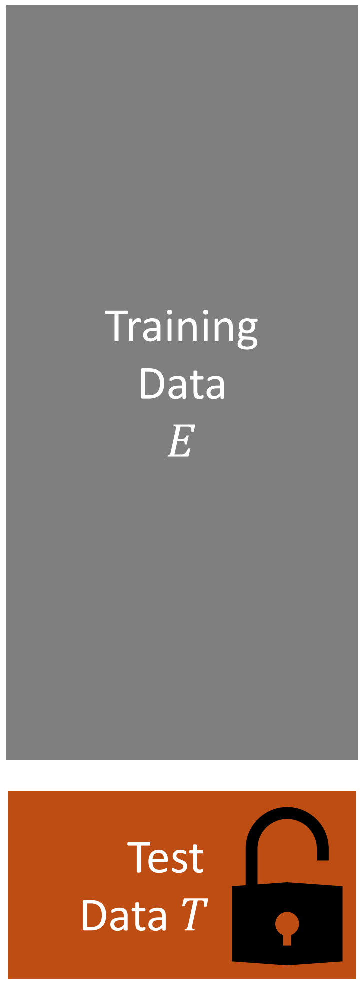
- **Testing loss**: Calculate the empirical loss for predictions on a testing data set T that is different from the data used for training.

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

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

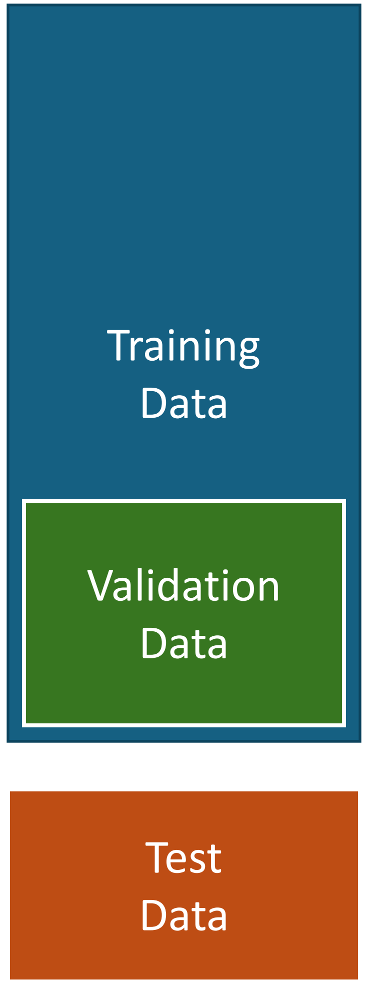
$$accuracy(h,T) = \frac{1}{|T|} \sum_{(\boldsymbol{x},y)\in T} [h(\boldsymbol{x}) = y] = 1 - EmpLoss_{L_{0/1},T}(h)$$

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

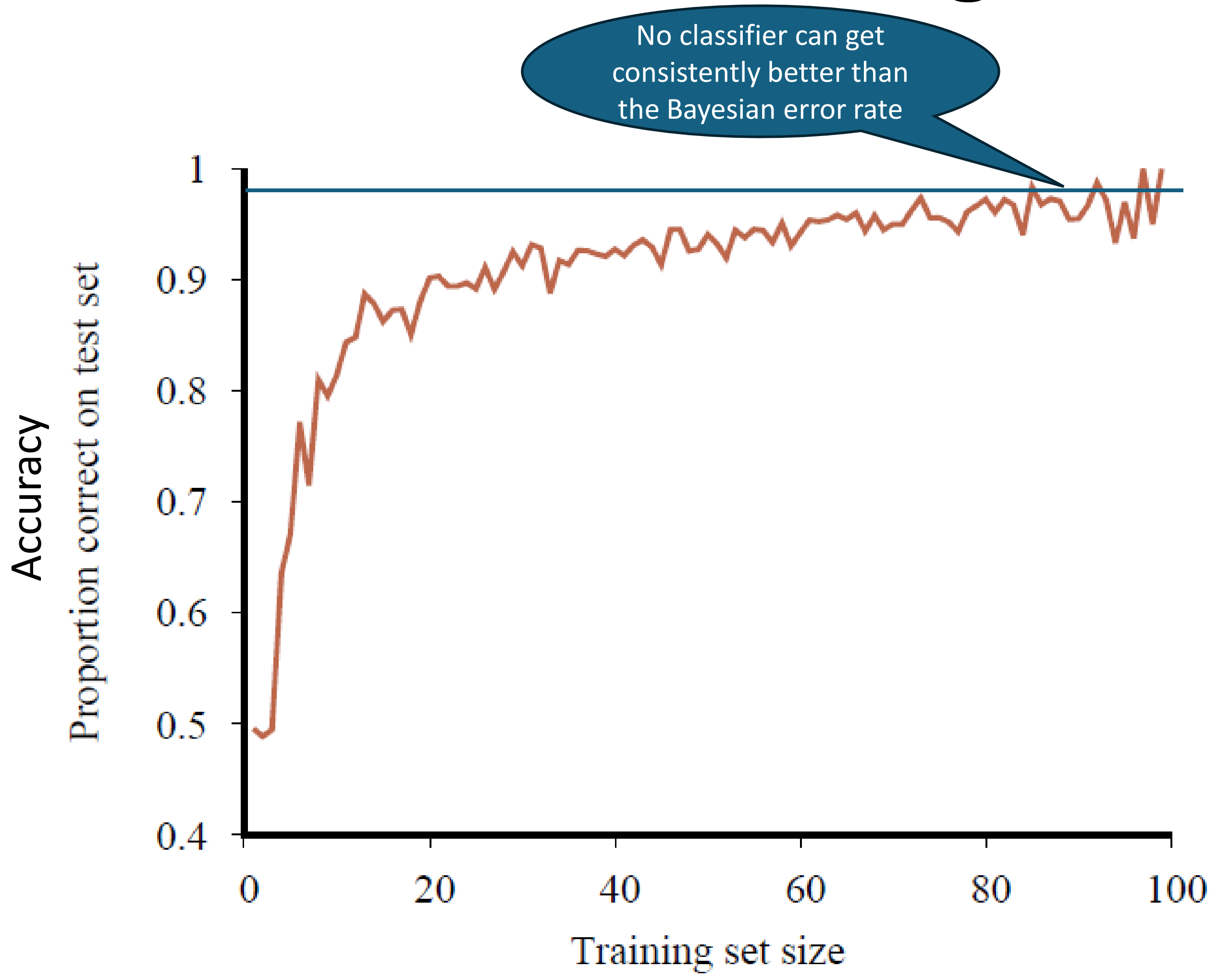


How to Split the Dataset

- **Random splits:** Split the data randomly in, e.g., 60% training, 20% validation, and 20% testing.
- **Stratified splits:** Like random splits, but balance classes or other properties of the examples.
- **k-fold cross validation:** Use training & validation data better
  - Split the training & validation data randomly into k folds.
  - For each of k rounds: Hold one fold back for testing/validation and use the remaining $k-1$ folds for training.
  - Use the average error/accuracy of the k rounds as a better estimate.
  - Some algorithms/tools do this internally for hyperparameter tuning.



Learning Curve: The Effect the Training Data Size



Accuracy increases when the amount of available training data increases.

More data is better!

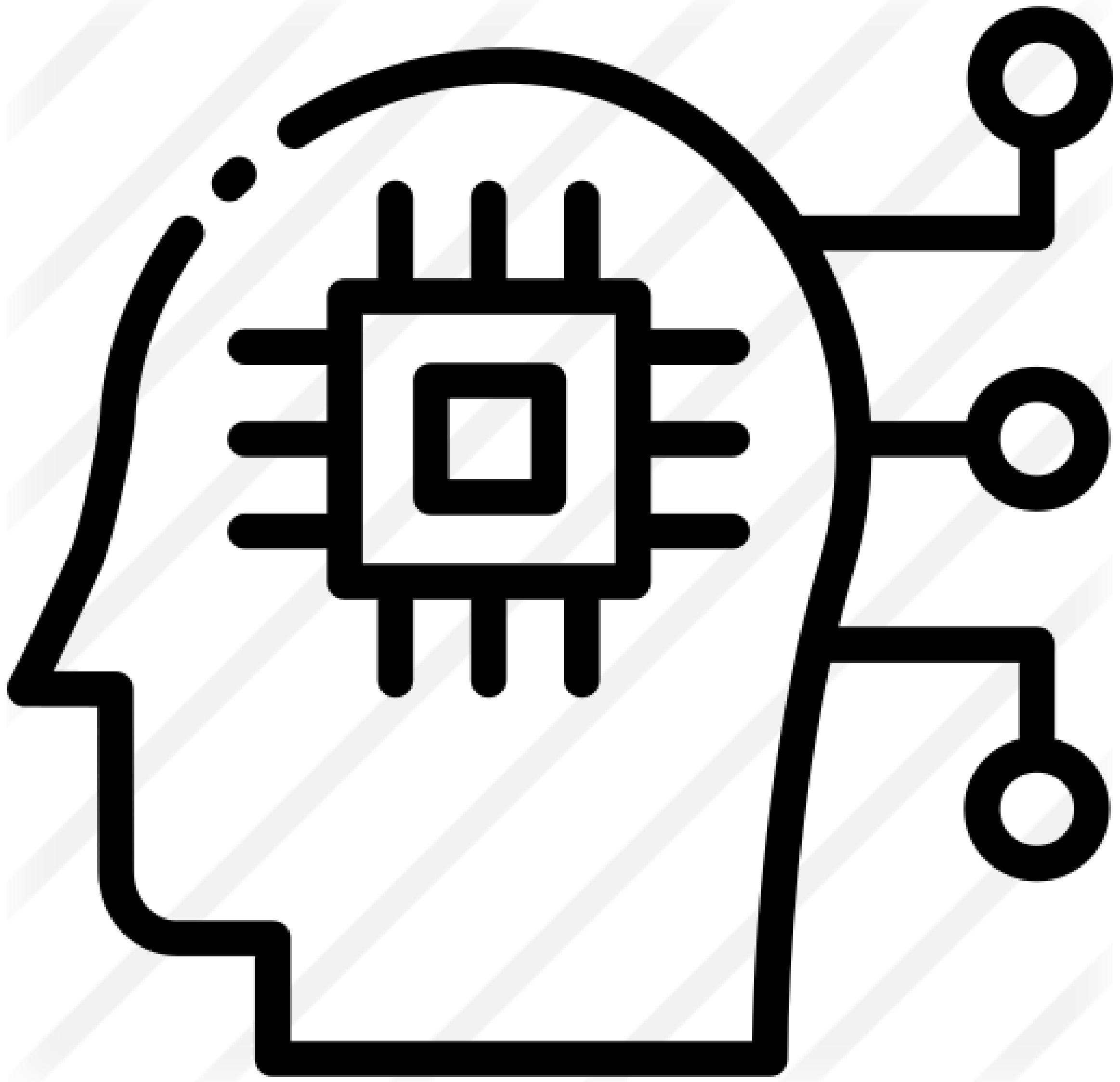
At some point, the learning curve flattens out, and more data does not contribute much!

Comparing to a Baselines

- First step: get a **baseline**
  - 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 published state-of-the-art as a baseline.



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

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

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

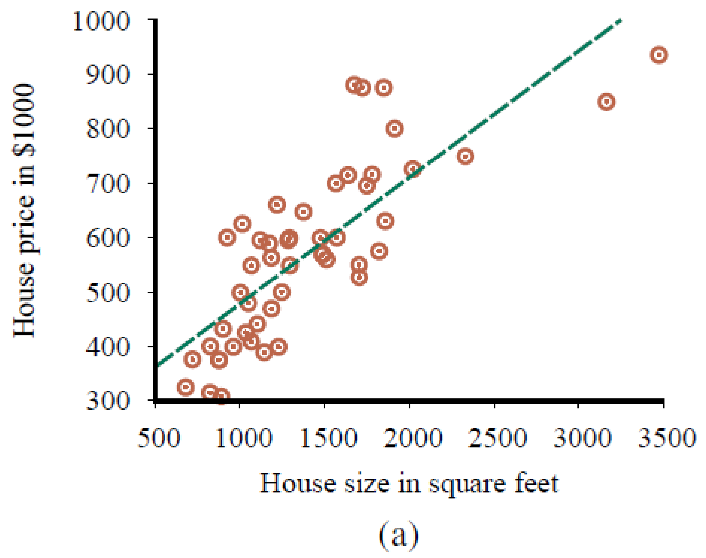
Find: $\nabla L(\boldsymbol{w}) = 0$

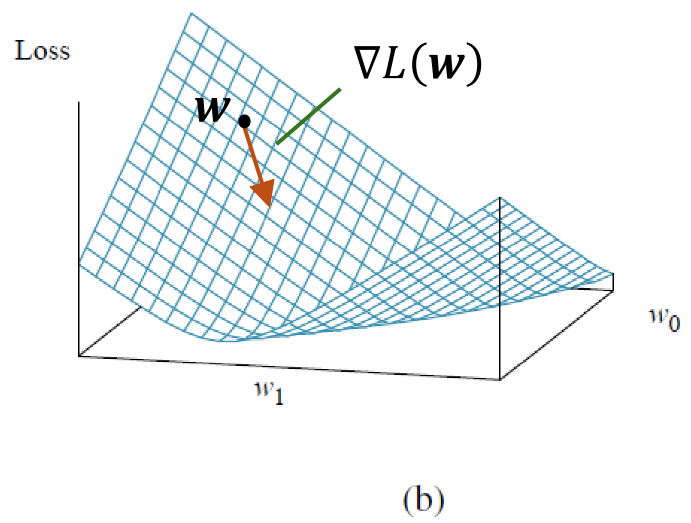
Gradient descent:

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

Analytical solution:

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





Bayes Classifier

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

Naïve Bayes Classifier

- 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 $P(y)$s and the $P(\boldsymbol{x}_i \mid y)$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:

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

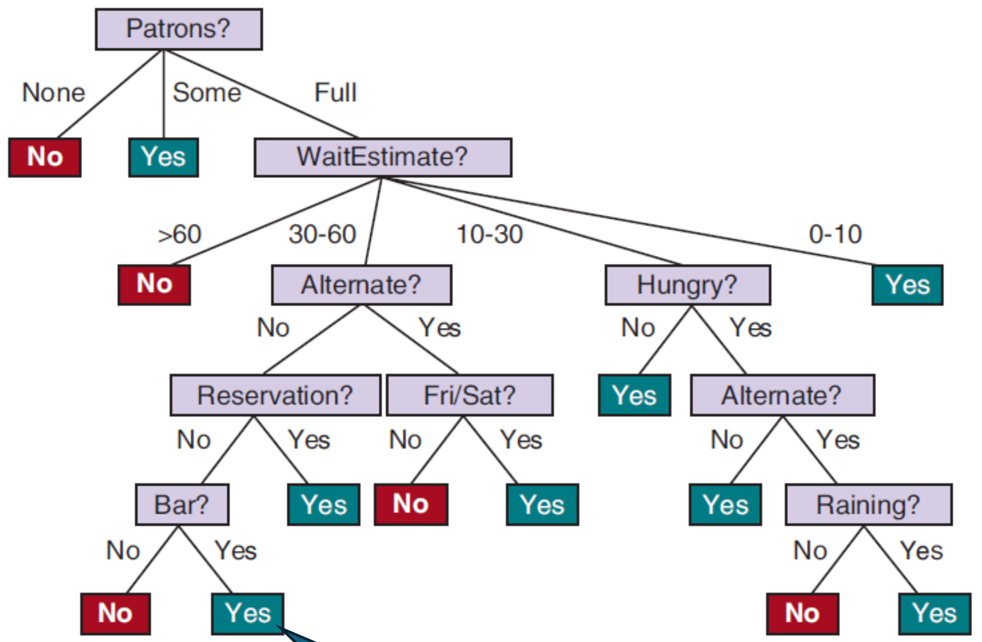
The parameters for the normal distribution $N\left(\mu_y, \sigma_y\right)$ are estimated from data.

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:

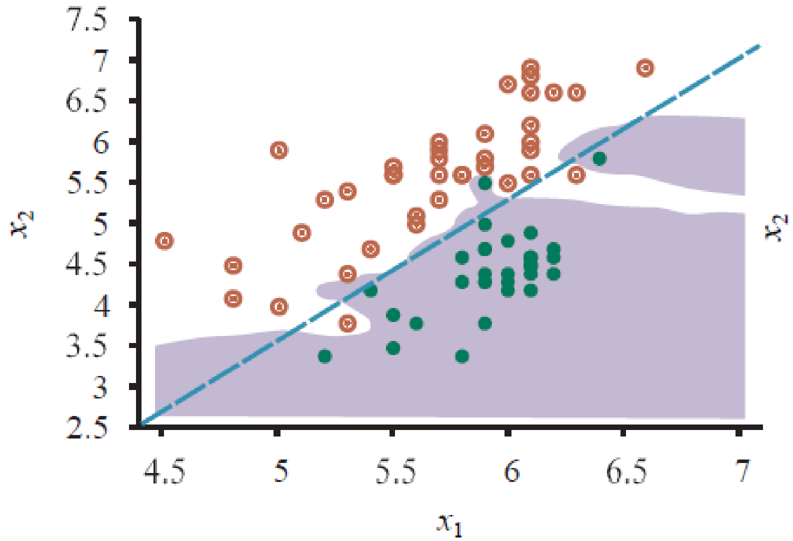
$$\hat{P}(\text{Yes}|\text{node N}) = \frac{N_{Yes}}{N_{Yes}+N_{No}}$$

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

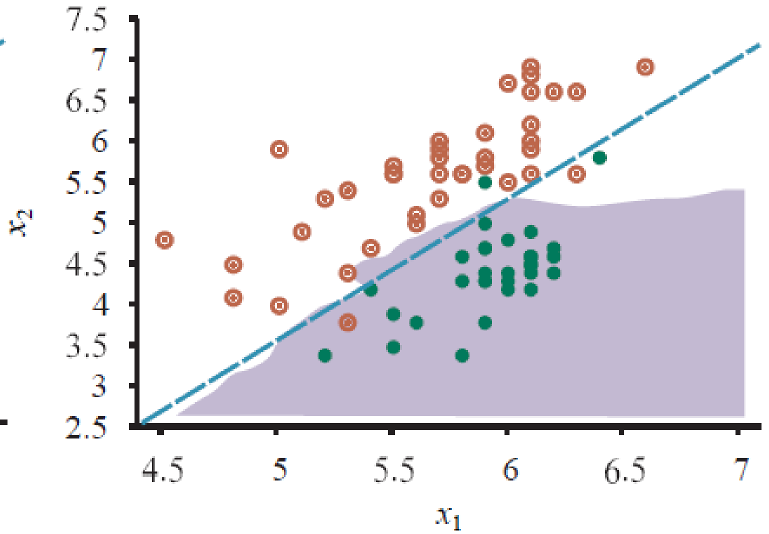
$$h(\boldsymbol{x}) = \underset{y}{\operatorname{argmax}}\ P(Y = y \mid \text{leafNodeMatching}(\boldsymbol{x}))$$

K-Nearest Neighbors Classifier

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



$(k = 1)$



$(k = 5)$

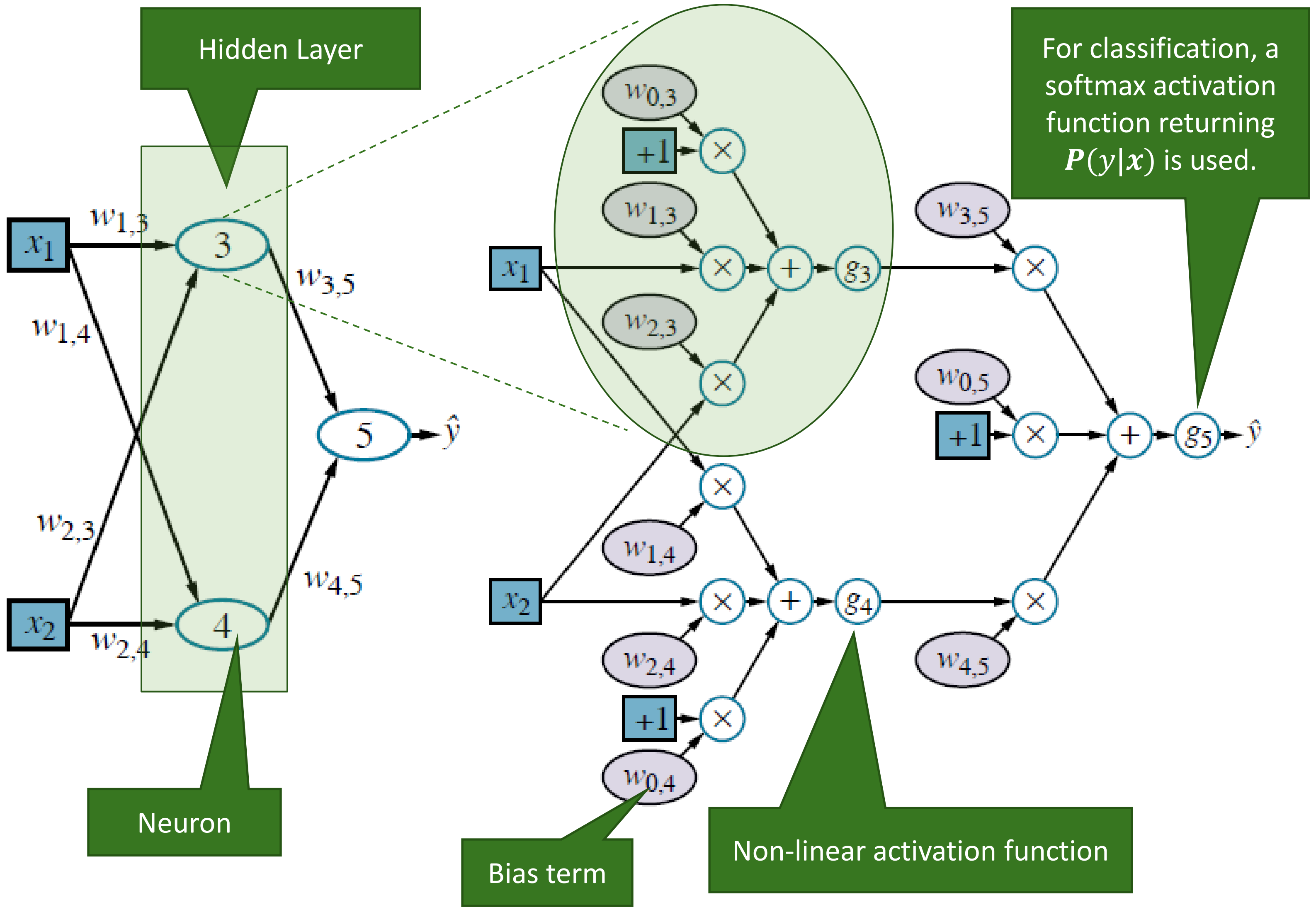
- Class is predicted by looking at the majority in the set of the k nearest **neighbors**. k is a hyperparameter. Larger k smooths the decision boundary.
- Neighbors are found using a distance measure (e.g., Euclidean distance between points).
- Approximates a Bayesian classifier by

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

Artificial Neural Networks

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

Network Topology **Computational graph**



- Represent
  $$\hat{y} = h(\mathbf{x}) = g^{[2]}\left(\mathbf{W}^{[2]}\, g^{[1]}\left(\mathbf{W}^{[1]}\mathbf{x}\right)\right)$$
  as a network of weighted sums with non-linear **activation functions** $g(\cdot)$ (e.g., sigmoid, ReLU).
- Learn weight matrices $\mathbf{W}$ 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.
- 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.

Other Popular Models and Methods

Many other models exist

- **Generalized linear model (GLM):** This important model family includes **linear regression, Poisson regression** and the classification method **logistic regression.**

Often used methods

- **Regularization:** Enforce simplicity and reduce overfitting by using a penalty for complexity.
- **Kernel trick:** Lets a linear classifier learn non-linear decision boundaries.
- **Ensemble Learning:** Use many models and combine the results (e.g., random forest, boosting).
- **Embedding and Dimensionality Reduction:** Learn how to represent data in a simpler way. E.g., principal components analysis (PCA), variational autoencoders, text embeddings.

Some Use Cases of ML for Intelligent Agents

Learn a Policy

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

  $action = h(state)$

- 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 for states.

  $eval = h(state)$

- Can learn a **heuristic** for minimax search from examples.
- In reinforcement learning we learn action values $q(state, action)$.

Learn Perception for Sensors

- **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.

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!