ELTE | FACULTY OF INFORMATICS

Supervised Learning

Introduction to Machine Learning – Lecture 4
Computer Science BSc Course, ELTE Faculty of Informatics

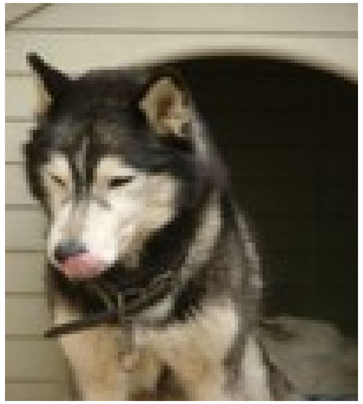
János Botzheim

Associate Professor
Department of Artificial Intelligence
botzheim@inf.elte.hu

ELTE | IK

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

Unsupervised learning

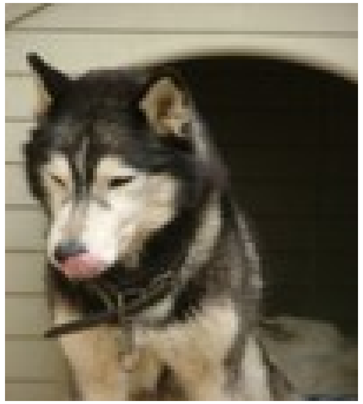


Supervised learning

label

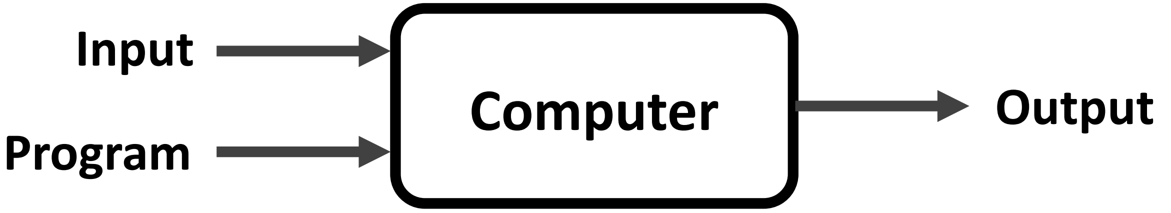
dog **cat** **bird**



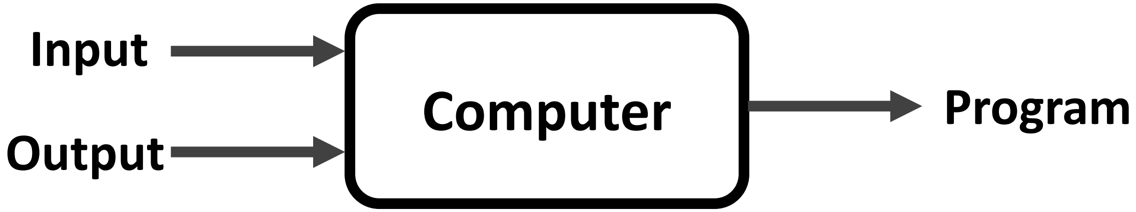


Learning - Programming

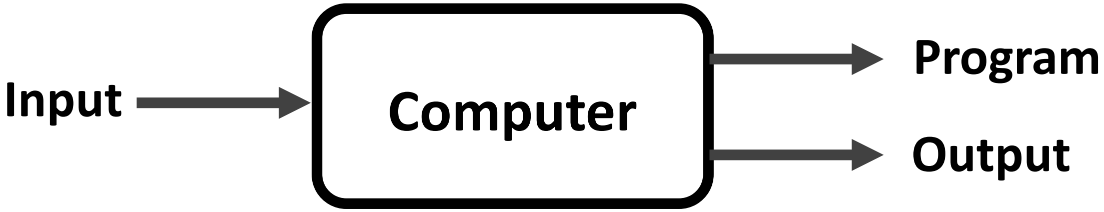
- Traditional computer programming:



- Supervised learning:



- Unsupervised learning:



Types of learning

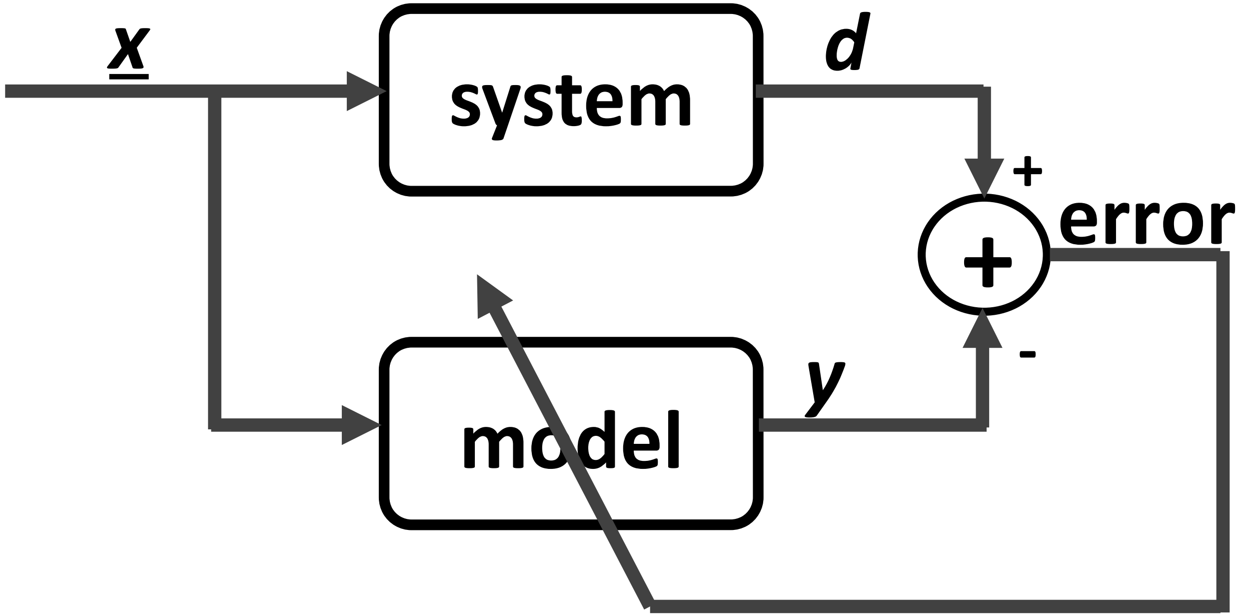
- **Learning**
  - Agent can improve its performance based on observations
- **Machine Learning**
  - Subset of artificial intelligence, computer system learns by patterns in observation data

- Learning tasks
  - **Classification:** output is one of a finite set of values
  - **Regression:** output is a numeric prediction

- Learning Feedback
  - **Supervised Learning** – learn a function that maps inputs to outputs by observing input-output pairs
  - **Unsupervised Learning** – learn patterns in data without explicit feedback (e.g. clustering)
  - **Reinforcement Learning** – learn beneficial actions based on rewards and punishments

Supervised learning

- given: input-output training patterns $(x_1^{(p)}, x_2^{(p)}, \ldots, x_n^{(p)}, d^{(p)})$
- p: number of patterns, n-dimensional input, scalar output



Supervised learning

- **Representation:**
  - rule based system
  - fuzzy system
  - decision tree
  - neural network
  - support vector machine
  - etc.
- **Evaluation:**
  - error
  - accuracy
  - cost
  - entropy
  - etc.
- **Optimization:**
  - gradient based algorithm
  - evolutionary algorithm
  - etc.

Supervised Learning

- More formally

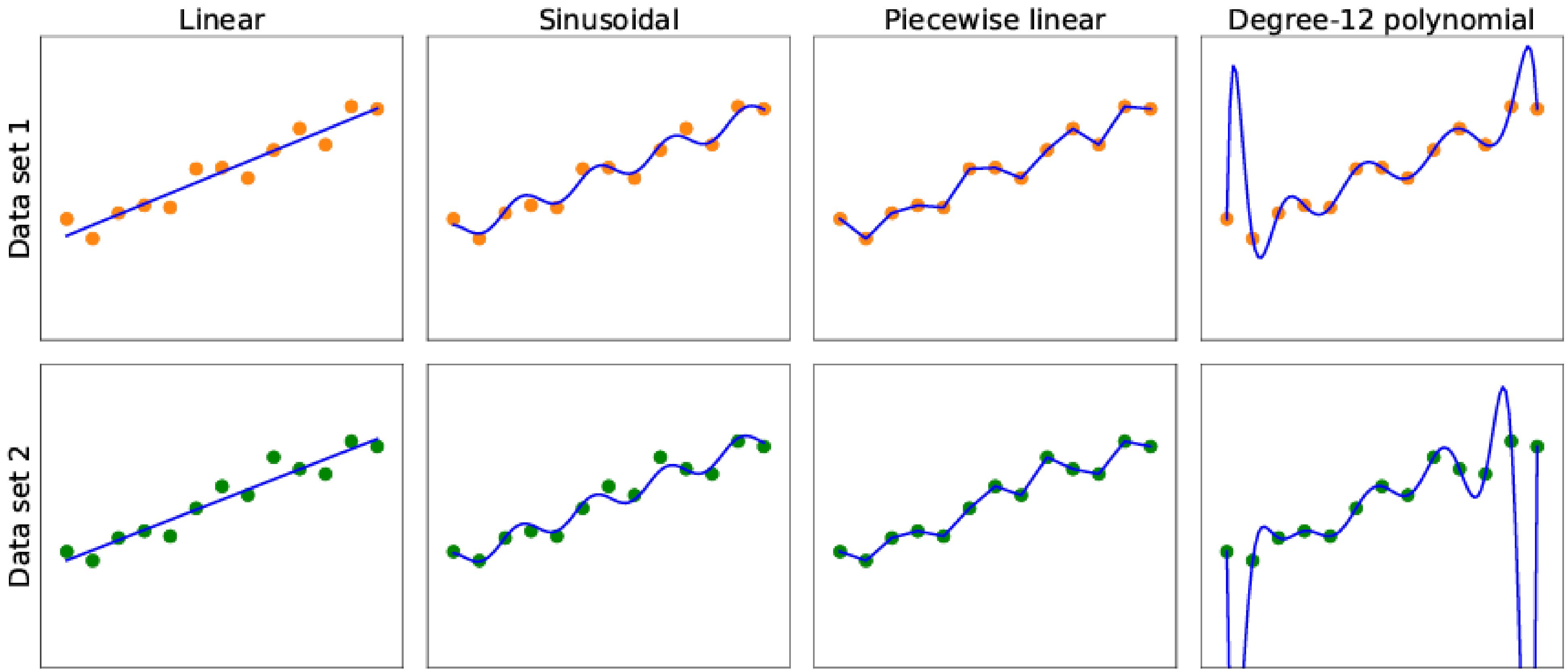
Given a **training set** of N example input–output pairs

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N),$$

where each pair was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.

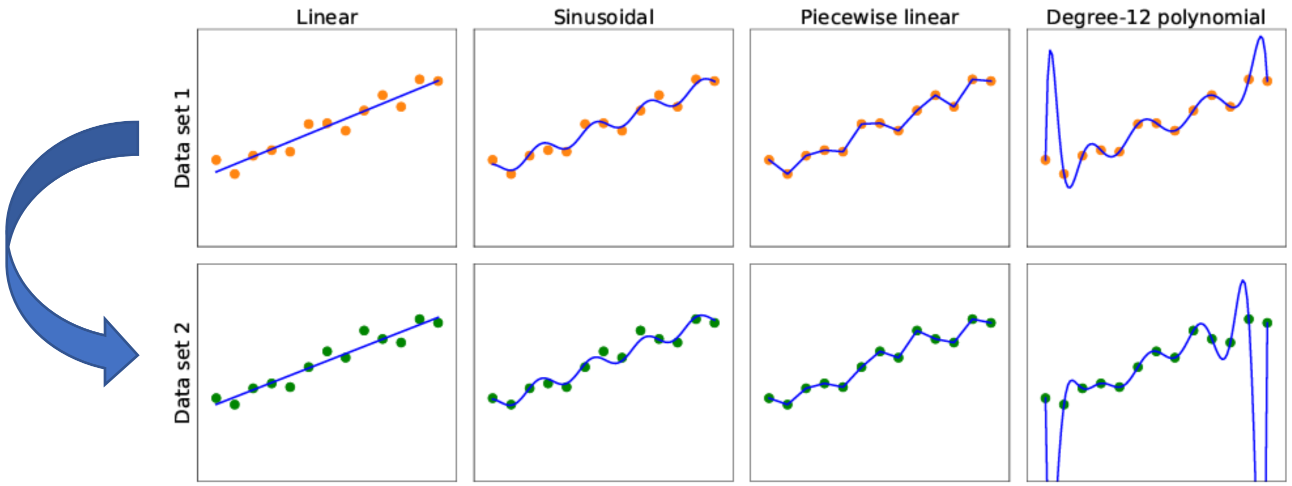
- The function ***h*** is called a **hypothesis** or a **model** of the data

Supervised Learning



Supervised Learning - Generalization

- Performance measures
  - Performance on the training set
  - Performance on test set
- **Generalization**
  - ***h*** generalizes well if it accurately predicts the outputs of the **test set**



Supervised Learning - Bias & Variance

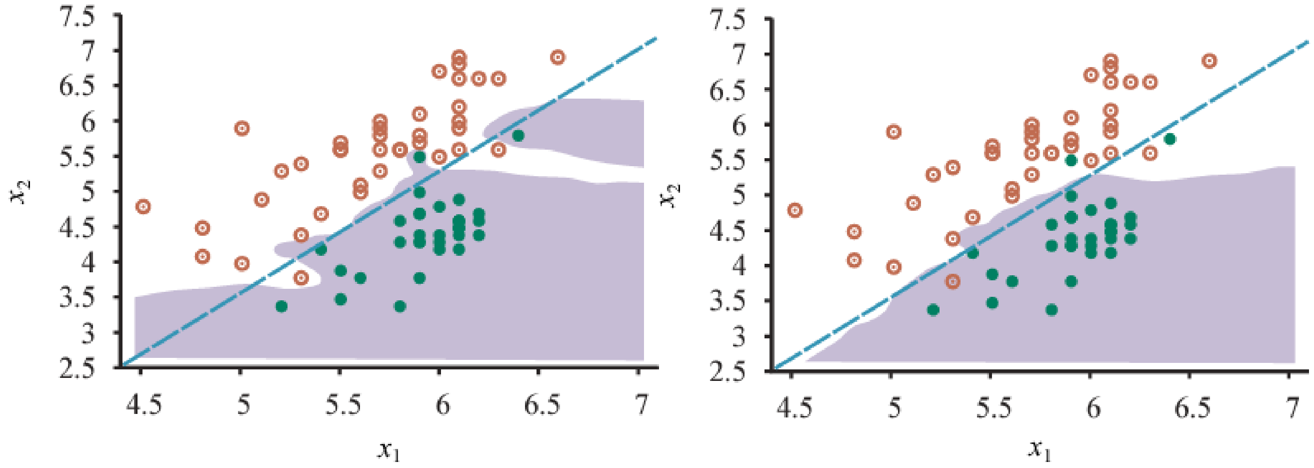
- One way to analyze **hypothesis spaces** is by the **bias** they impose (regardless of the training data set) and the **variance** they produce (from one training set to another).
- **Bias**
  - The tendency of a predictive hypothesis to deviate from the expected value when averaged over different training sets
- **Variance**
  - The amount of change in the hypothesis due to fluctuation in the training data

- **Bias–variance tradeoff**
  - more complex, low-bias hypotheses that fit the training data well
  - simpler, low-variance hypotheses that may generalize better

Supervised Learning - Underfitting & Overfitting

- **Underfitting**
  - The algorithm fails to find a pattern in the data
  - Model is too simple

- **Overfitting**
  - Fits too much to the **particular data set** it is trained on
  - Performs poorly on unseen data (train-test accuracy)
  - Model is too complex
  - Amount of data is not enough
  - Data is not diverse enough



Decision Trees

- Maps a vector of **attribute values** to a single output **“decision”**
- Sequence of tests
  - Start from root
  - Explore branches
  - Until a leaf is reached
- **Node = test of input attribute value**
- Input and output values can be **discrete** or **continuous**

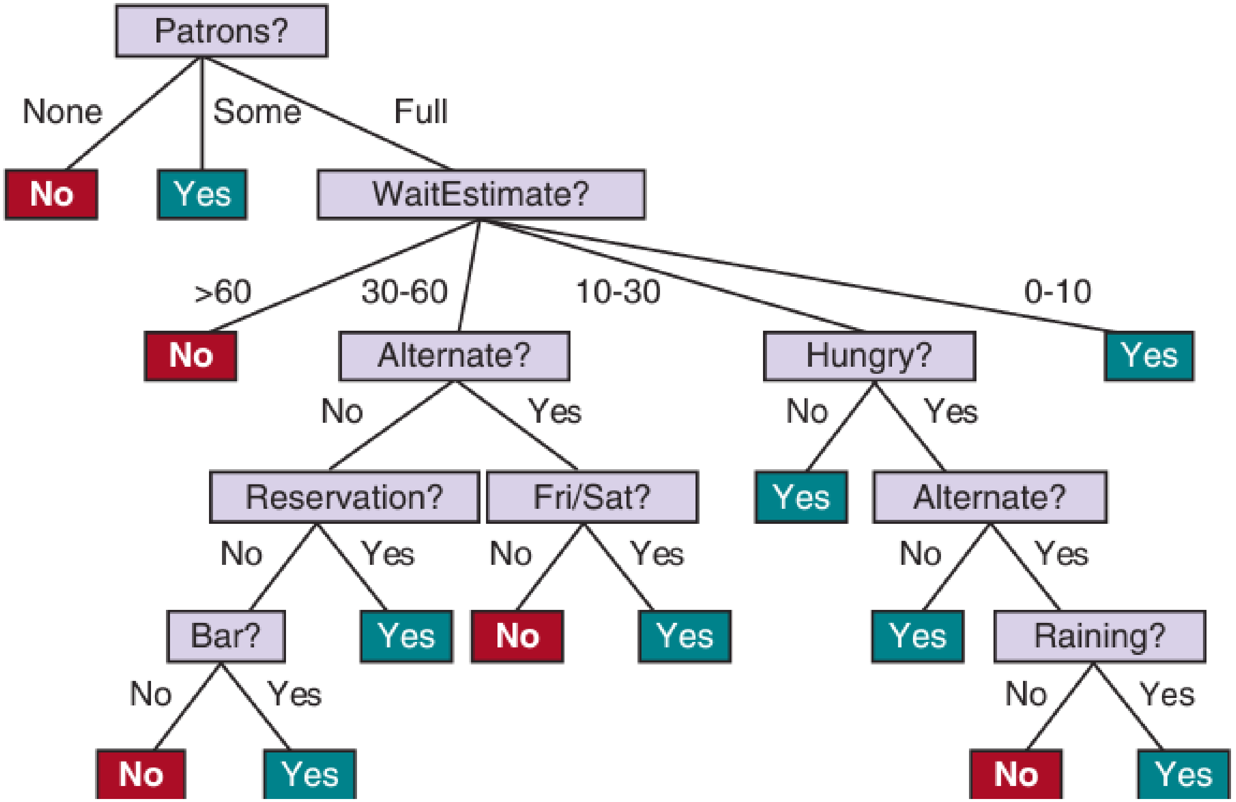
- Simple example: **Boolean decision tree**
- A Boolean decision tree is equivalent to a logical statement of the form:

$$Output \Leftrightarrow (Path_1 \vee Path_2 \vee \cdots),$$

- Where $(A_m = v_x \wedge A_n = v_y \wedge \cdots)$ are attribute-value tests corresponding to a path from the root to a true leaf

Decision Trees

- Example: Waiting in restaurant



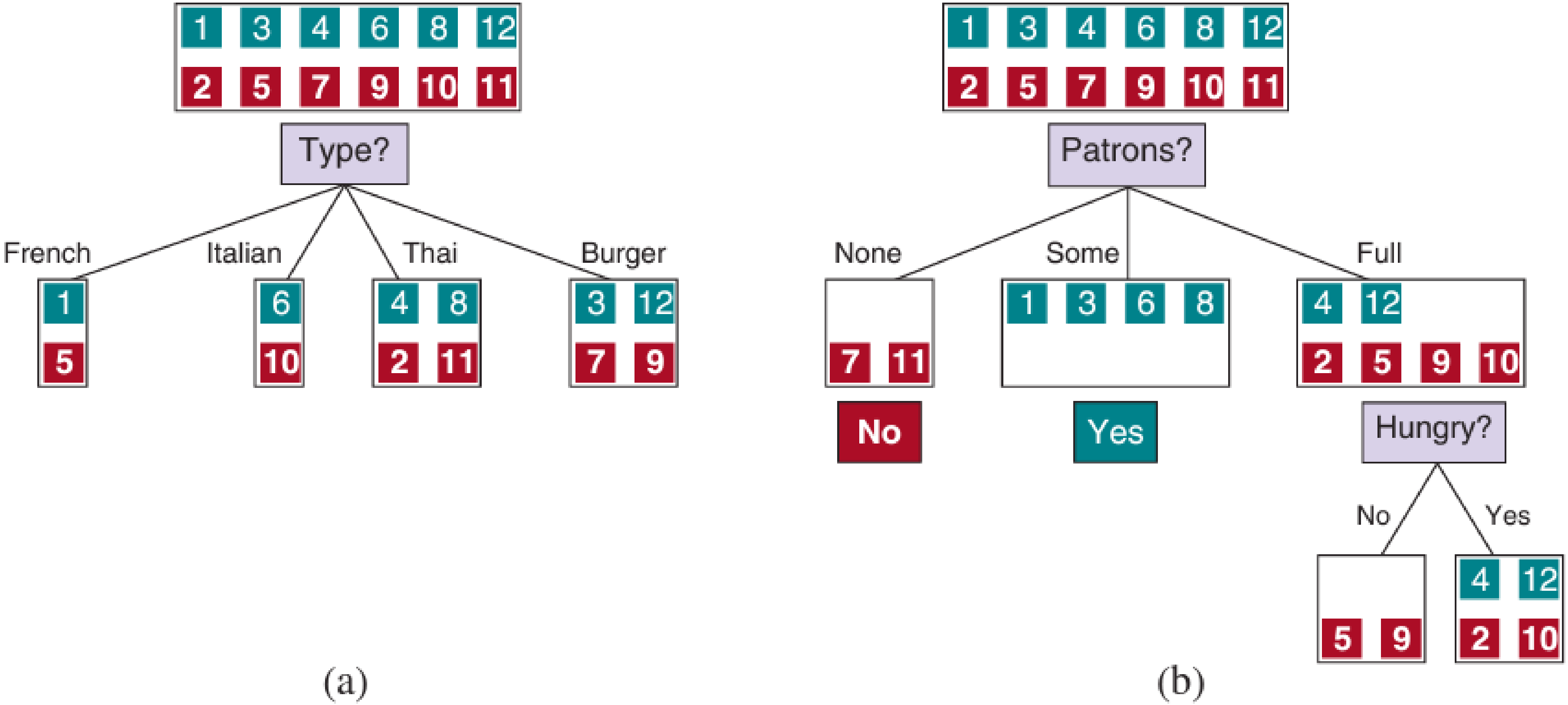
Decision Trees

- Training dataset

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*	$y_{12} = Yes$

Decision Trees

- Find most important features and solve sub-problems sequentially



Decision Trees

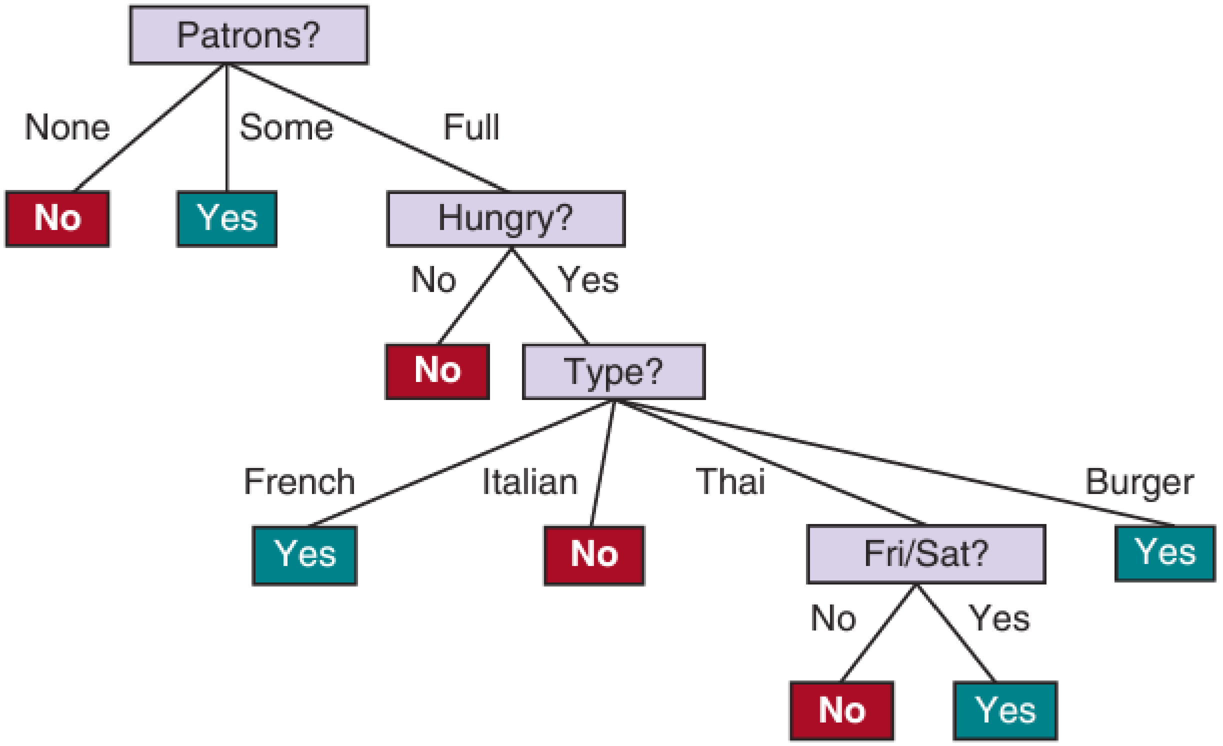
function LEARN-DECISION-TREE(*examples*, *attributes*, *parent_examples*) **returns** a tree

    if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
    else if all *examples* have the same classification **then return** the classification
    else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
    else
        $A \leftarrow \operatorname{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
        tree ← a new decision tree with root test A
        for each value v of A **do**
            $exs \leftarrow \{e : e \in examples$ **and** $e.A = v\}$
            subtree ← LEARN-DECISION-TREE(*exs*, *attributes* − A, *examples*)
            add a branch to *tree* with label $(A = v)$ and subtree *subtree*
        return *tree*

Decision Trees

- Final decision tree has short paths
- Simpler than the original tree
- Fitted to the training set
- **BUT** The learning algorithm looks at the examples, not at the correct function

- Selecting most important attributes based on **information gain** defined by **expected reduction in entropy** of the output variable



Decision Trees

- **Generalization**
  - If we **increase** the **number of attributes**, overfitting is **more likely**
  - If we **increase** the **number of training samples**, overfitting is **less likely**

- **Pruning**: improving generalization for decision trees
  - Eliminate nodes that are not clearly relevant
    - Look at a test node that has only leaf nodes as descendants
    - If the test appears to be irrelevant - eliminate and replace it with a leaf node
    - Use significance test to measure low information gain threshold

Decision Trees

- **Pros**
  - Ease of understanding
  - Scalability to large data sets
  - Versatility in handling discrete and continuous inputs
  - Performing classification and regression

- **Cons**
  - Suboptimal accuracy (largely due to the greedy search)
  - If trees are very deep, then getting a prediction for a new example can be expensive
  - Decision trees are unstable – adding just one new example can change the entire tree

Linear Regression and Classification

- **Univariate Linear Regression** = “fitting a straight line”

  *fit model on **(x, y) training examples** so that* $\boldsymbol{y = w_1 x + w_0}$
  where $\boldsymbol{w_0}$ *and* $\boldsymbol{w_1}$ *are **learned weights***

- The learned linear function

$$h_{\mathbf{w}}(x) = w_1 x + w_0$$

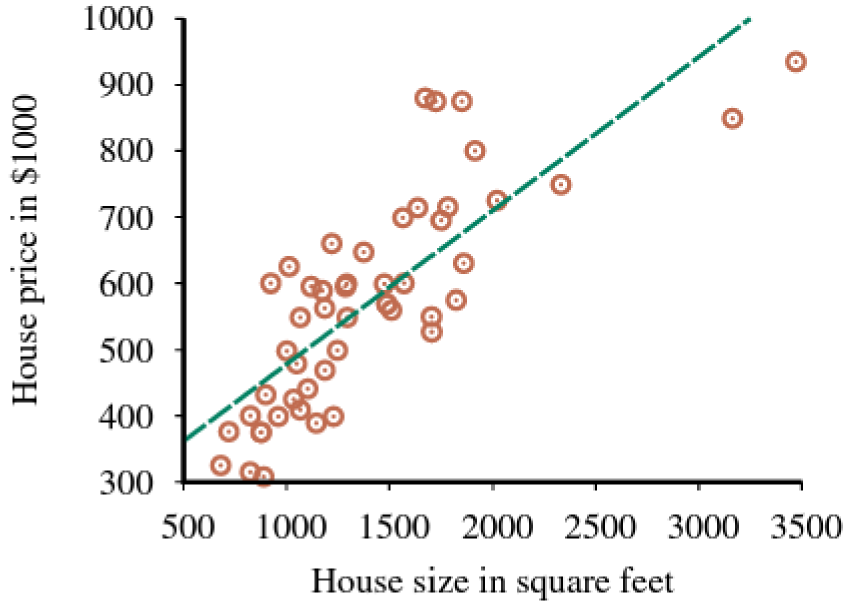
- The task of finding this linear function based on training data is called **linear regression**
  - Find values of w_0, w_1 that minimizes the empirical loss (e.g. L2 loss)

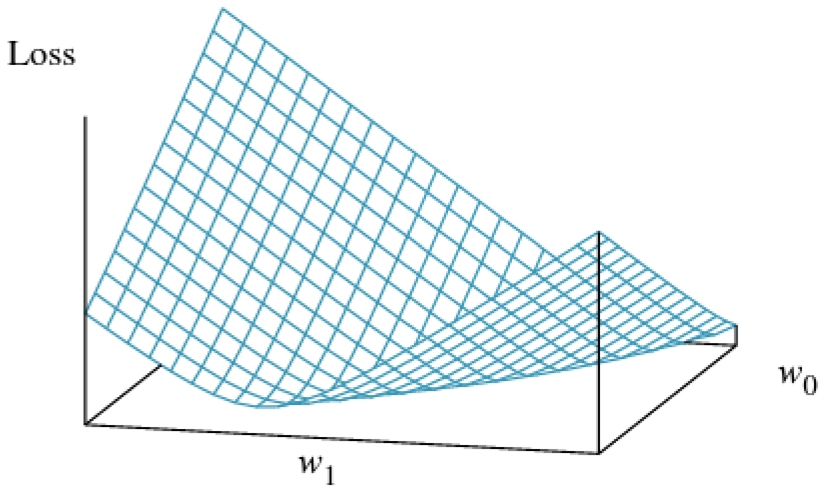
$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Linear Regression and Classification

- We can find the function that minimizes the loss using partial derivatives

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0$$





Linear Regression and Classification

- For more complex cases we need to introduce **gradient descent**
  - Algorithm for finding a point in the **weight space** that **minimizes the loss**
  - Similar to the hill climbing but here we are minimizing the loss, not maximizing the gain

- **Gradient Descent**
  - Select random starting point in weight space
  - Compute an estimate of the gradient
  - Move a small amount in the steepest downhill direction
  - Repeat until we converge on a point with minimum loss

  - Parameters
    - Learning rate (α) – determines the step size for the descent

$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged **do**
  for each w_i **in** $\mathbf{w}$ **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

Linear Regression and Classification

- For the univariate case the loss was quadratic - the partial derivative is linear
- We apply the chain rule

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))$$
$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)).$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

- Create the update function with the selected learning rate

$$w_0 \leftarrow w_0 + \alpha\,(y - h_{\mathbf{w}}(x)); \quad w_1 \leftarrow w_1 + \alpha\,(y - h_{\mathbf{w}}(x)) \times x$$

Single example

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

Batch