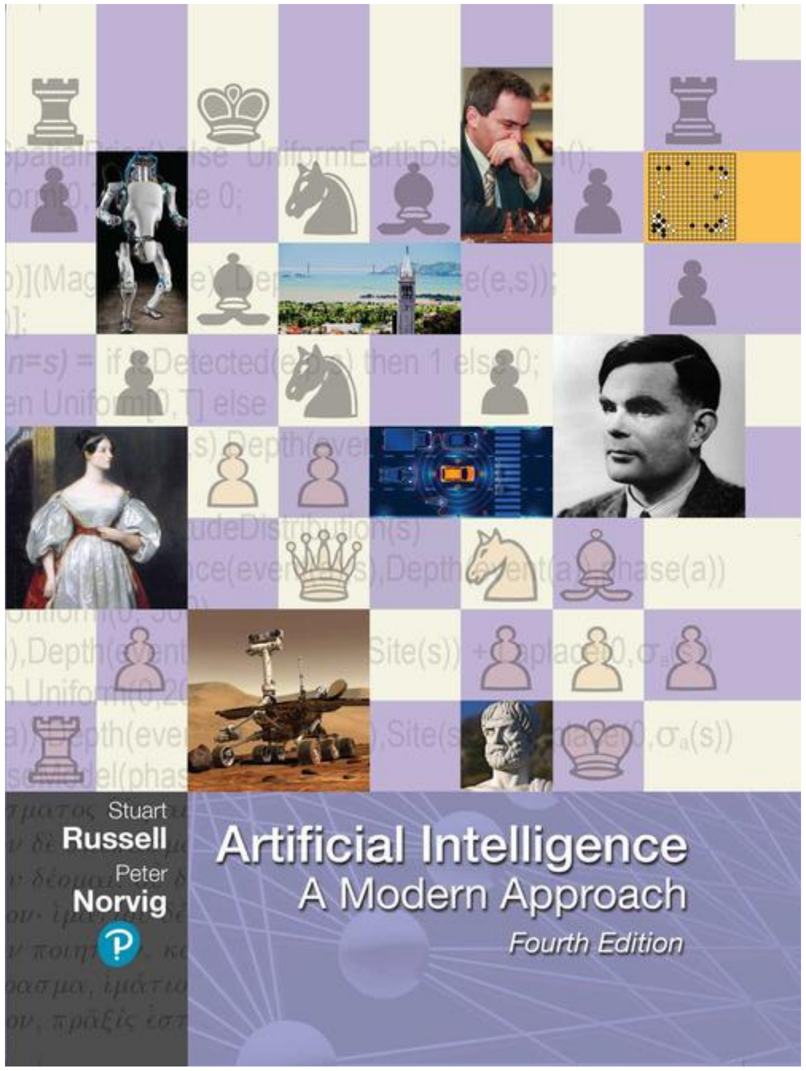
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 19

Learning From Examples

Forms of Learning

- There are three types of feedback that can accompany the inputs, and that determine the three main types of learning:
  - Supervised Learning
    - agent observes input-output pairs
    - learns a function that maps from input to output
  - Unsupervised Learning
    - agent learns patterns in the input without any explicit feedback
    - clustering
  - Reinforcement Learning
    - agent learns from a series of reinforcements: rewards & punishments

Supervised Learning

- There are three types of feedback that can accompany the inputs, and that determine the three main types of learning:
  - Supervised Learning
    - agent observes input-output pairs
    - learns a function that maps from input to output
  - Unsupervised Learning
    - agent learns patterns in the input without any explicit feedback
    - clustering
  - Reinforcement Learning
    - agent learns from a series of reinforcements: rewards & punishments

Supervised Learning

- Training set of examples of input output (N)
  - $(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$,
  - $y = f(x)$,

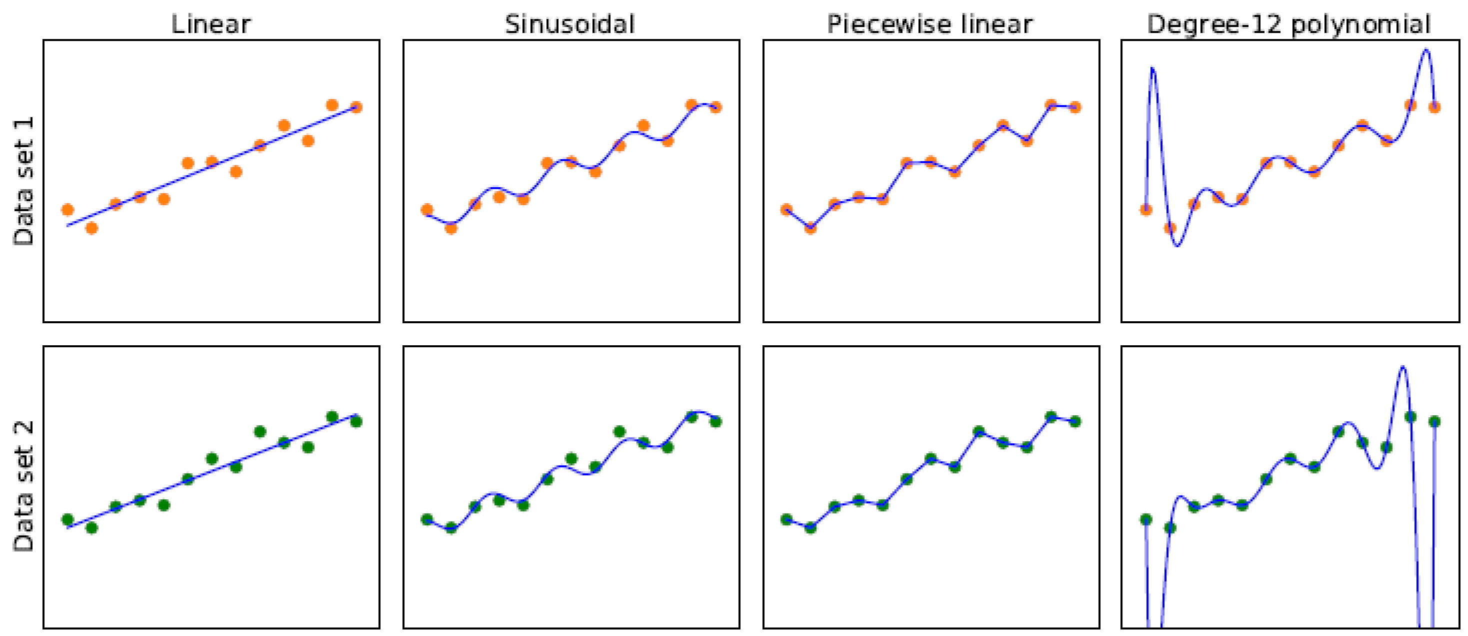
  function h is hypothesis about the world, approximates the true function f
  - drawn from a hypothesis space H of possible functions
  - h Model of the data, drawn from a model class H

***Consistent* hypothesis**: an h such that each x_i in the training set has $h(x_i) = y_i$.
- *look* for a **best-fit function** for which each $h(x_i)$ is close to y_i

The true measure of a hypothesis, depends on how well it handles inputs it has not yet seen. Eg: a second sample of (x_i, y_i)

h generalizes well if it accurately predicts the outputs of the test set

Supervised Learning



Finding hypotheses to fit data.
Top row: four plots of best-fit functions from four different hypothesis spaces trained on data set 1.
Bottom row: the same four functions, but trained on a slightly different data set (sampled from the same $f(x)$ function).

Supervised Learning

- Use **bias** to analyze hypothesis space
  - the tendency of a predictive hypothesis to deviate from the expected value when averaged over different training set
- **Underfitting**: fails to find a pattern in the data
- **Variance:** the amount of change in the hypothesis due to fluctuation in the training data.
- **Overfitting:** when it pays too much attention to the particular data set it is trained on, causing it to perform poorly on unseen data.
- **Bias–variance tradeoff**: a choice between more complex, low-bias hypotheses that fit the training data well and simpler, low-variance hypotheses that may generalize better.

Supervised Learning

- Determine how **probable a hypothesis is** not just if possible
- hypothesis h^* that is most probable given the data:

$$h^* = \underset{h \in H}{\operatorname{argmax}} \; P(h|data)$$

- By Bayes' rule this is equivalent to

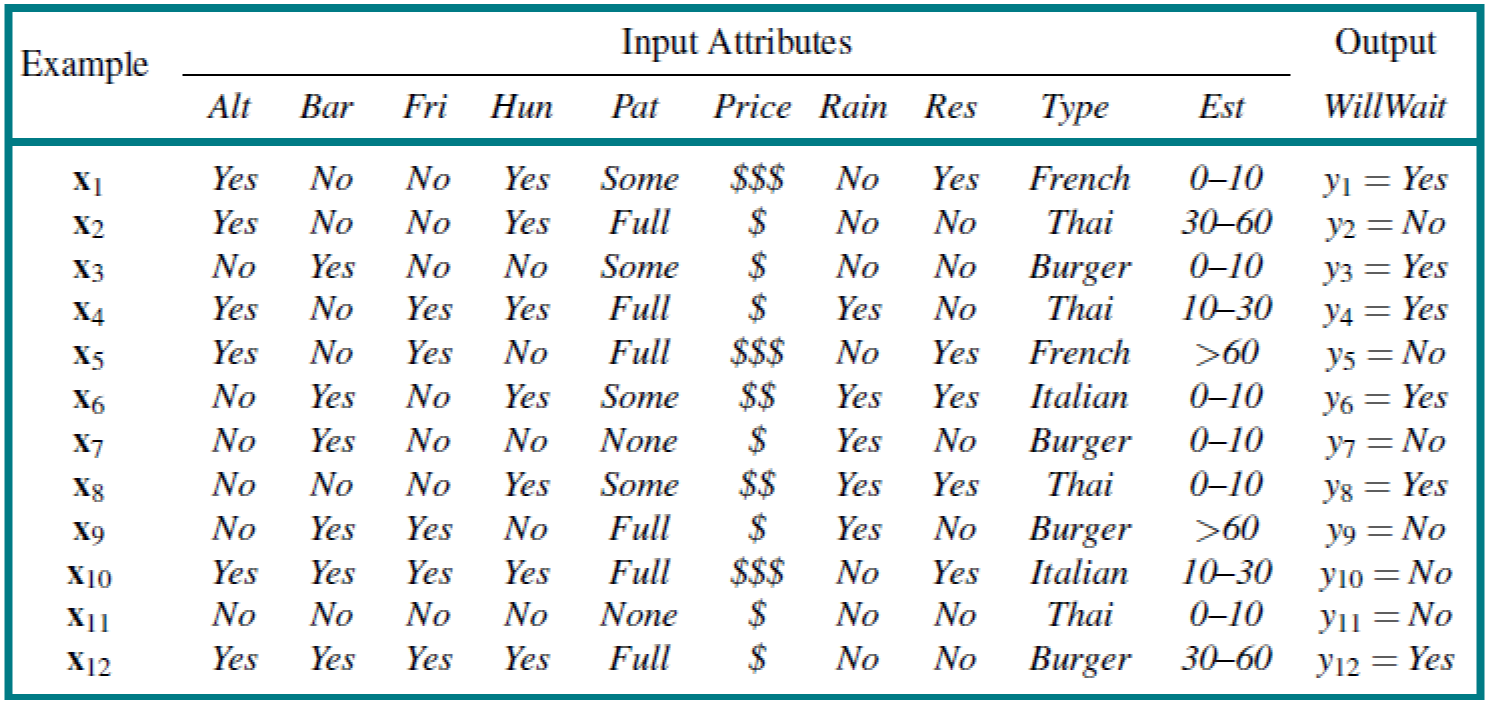
$$h^* = \operatorname{argmax} \; P(h|data) \, P(h)$$

Supervised Learning

Example problem: Restaurant waiting

- the problem of deciding whether to wait for a table at a restaurant.
- For this problem the **output, *y*,** is a Boolean variable that we will call ***WillWait***.
- **The input, *x*,** is a vector of ten attribute values, each of which has discrete values:
  1. *Alternate*: whether there is a suitable alternative restaurant nearby.
  2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
  3. *Fri/Sat*: true on Fridays and Saturdays.
  4. *Hungry*: whether we are hungry right now.
  5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
  6. *Price*: the restaurant's price range ($, $$, $$$).
  7. *Raining*: whether it is raining outside.
  8. *Reservation*: whether we made a reservation.
  9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
  10. *WaitEstimate*: host's wait estimate: 0–10, 10–30, 30–60, or >60minutes

Supervised Learning



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*	$y_{12} = Yes$

Examples for the restaurant domain.

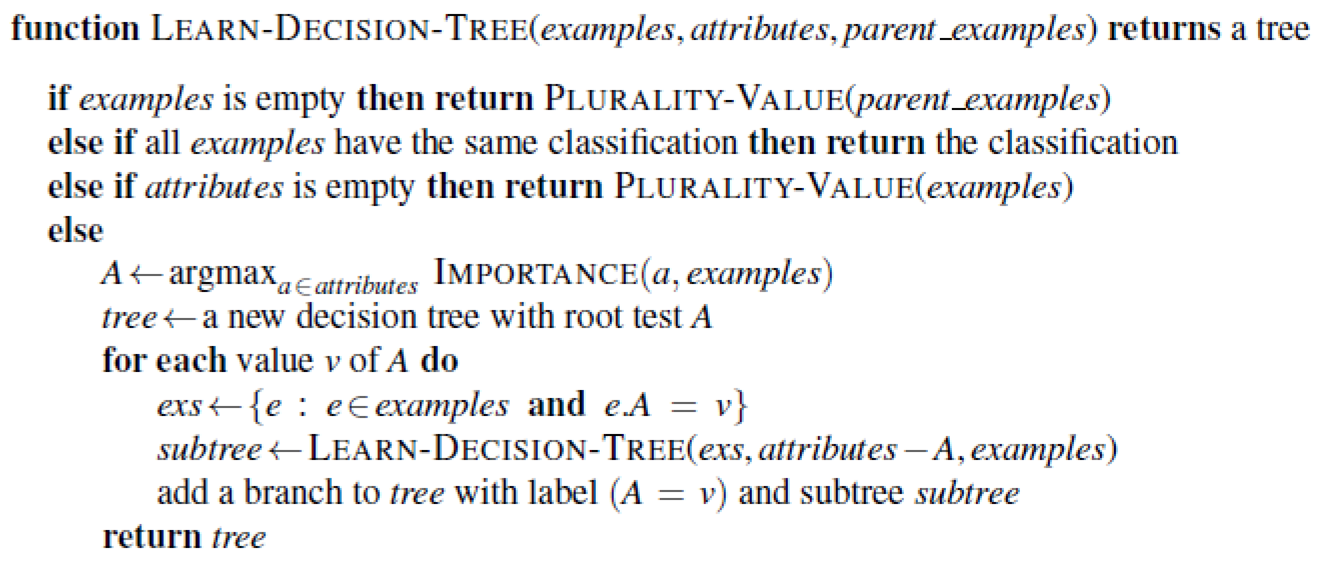
Decision Trees

A decision tree is a representation of a function that maps a vector of attribute values to a single output value—a “decision.”

- reaches its decision by performing a sequence of tests, starting at the **root** and following the appropriate branch until a leaf is reached.
- each **internal node** in the tree corresponds to a test of the value of one of the input attributes
- the **branches** from the node are labeled with the possible values of the attribute,
- the **leaf** nodes specify what value is to be returned by the function.

Boolean decision tree is equivalent to a logical statement of the form:
$Output \Leftrightarrow (Path_1 \vee Path_2 \vee \cdots)$

Decision Trees



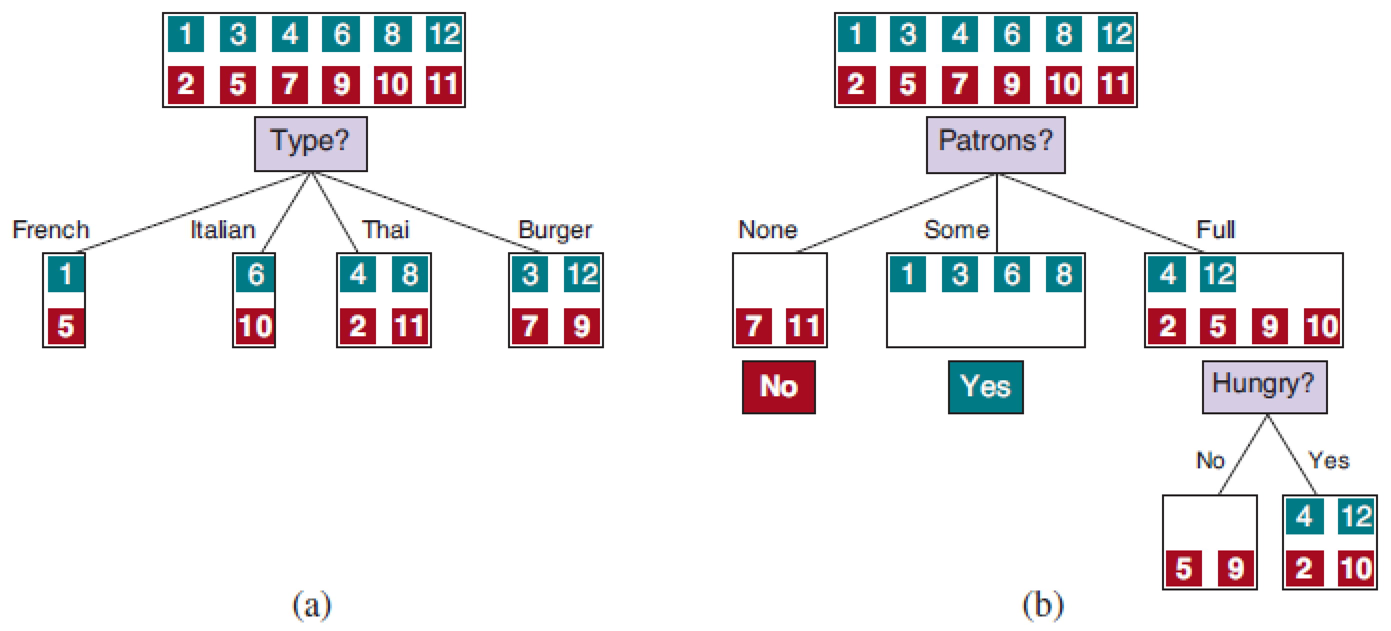
```
function LEARN-DECISION-TREE(examples, attributes, parent_examples) returns a tree
  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  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
```

The decision tree learning algorithm. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

Aim: find a small tree consistent with the training examples

Idea: (recursively) choose “most significant” attribute as root of (sub)tree

Decision Trees

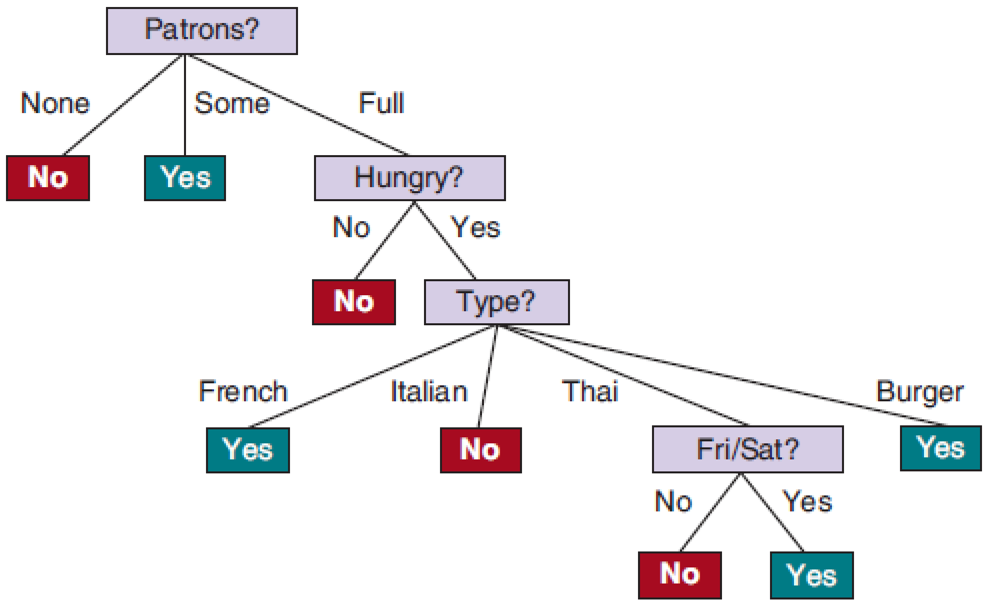


Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining.

(a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples.

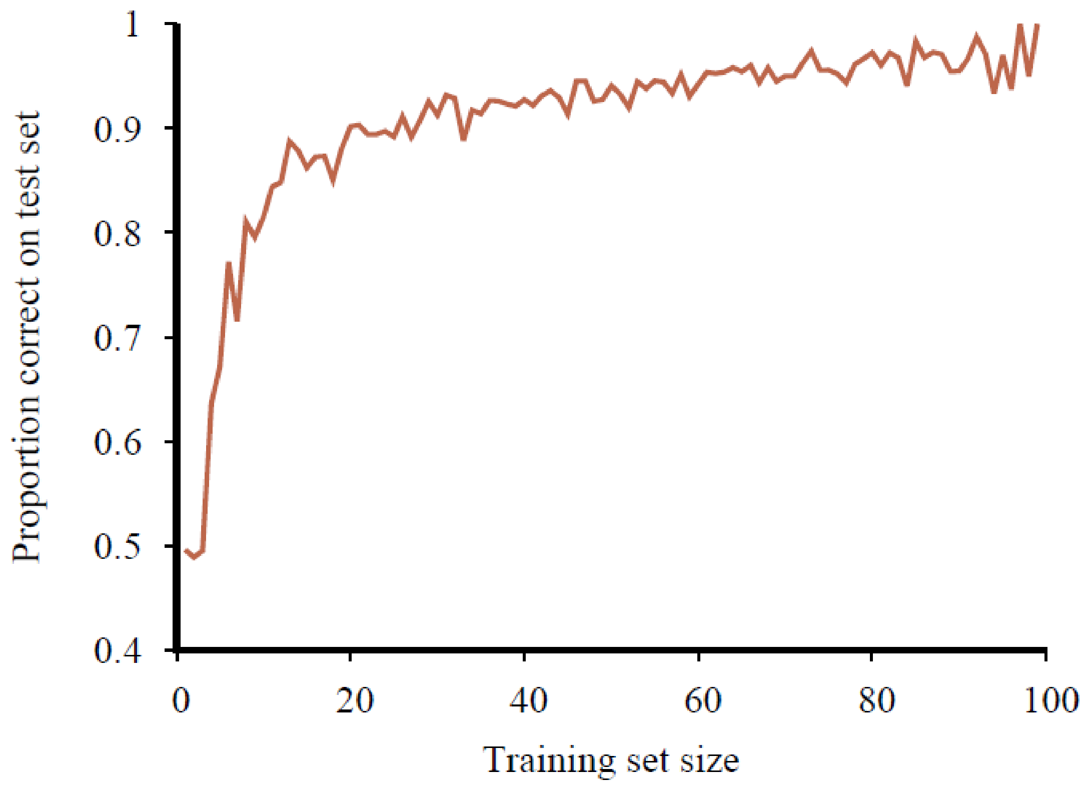
(b) Splitting on *Patrons* does a good job of separating positive and negative examples. After splitting on *Patrons*, *Hungry* is a fairly good second test

Decision Trees



The decision tree induced from the 12-example training set.

Decision Trees



The learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.

Decision Trees

Choosing attribute tests

Entropy: measure of the uncertainty of a random variable;

- the more information, the less entropy
- fundamental quantity in information theory

In general, the entropy of a random variable V with values v_k having probability

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k) \,.$$

The **information gain** from the attribute test on *A* is the expected reduction in entropy:

$$Gain(A) = B\left(\tfrac{p}{p+n}\right) - Remainder(A) \,.$$

Decision Trees

- Decision tree pruning helps combat overfitting
  - Eliminating nodes that are not clearly relevant

- How large a gain should we require in order to split on a particular attribute?
- Significance test
  - Start with null hypothesis
  - Calculate extent data deviates from perfect absence of pattern
  - degree of deviation is statistically unlikely (<=5% probability)

- node consisting of p positive and n negative examples. expected numbers, $\hat{p}_k$ and $\hat{n}_k$,
- Measure deviation & total deviation

$$\hat{p}_k = p \times \frac{p_k + n_k}{p+n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p+n}. \qquad \Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}.$$

Decision Trees

Broadening the applicability of decision trees

Decision trees can be made more widely useful by handling the following complications:

- Missing data
- Continuous and multivalued input attributes
- Continuous-valued output attribute

Decision trees are also **unstable** in that adding just one new example can change the test at the root, which changes the entire tree

Model Selection and Optimization

- Task of finding a good hypothesis as two subtasks:
  - **Model selection**: model selection chooses a good hypothesis space
  - **Optimization** (training) finds the best hypothesis within that space.

A **training set** to create the hypothesis, and a **test set** to evaluate it.

Error rate: the proportion of times that $h(x) \neq y$ for an (x, y)

Three data sets are needed:

1. A **training** set to train candidate models.
2. A **validation** set, also known as a development set or dev set, to evaluate the candidate models and choose the best one.
3. A **test** set to do a final unbiased evaluation of the best model.

When insufficient amount of data to create three sets: *k*-fold cross-validation

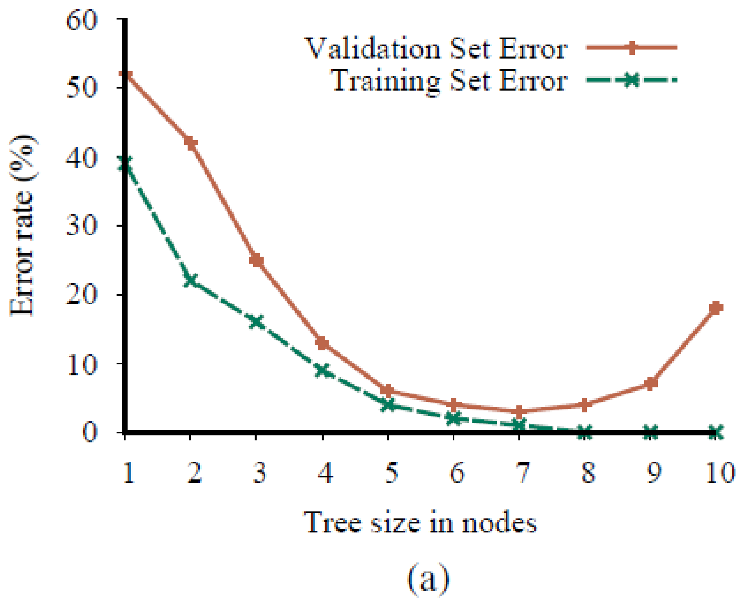
- split the data into *k* equal subsets
- perform *k* rounds of learning
- on each round 1/*k* of the data are held out as a validation set and the remaining examples are used as the training set.
- Popular values for *k* are 5 & 10
- **leave-one-out cross-validation** or **LOOCV,** *k*=*n*

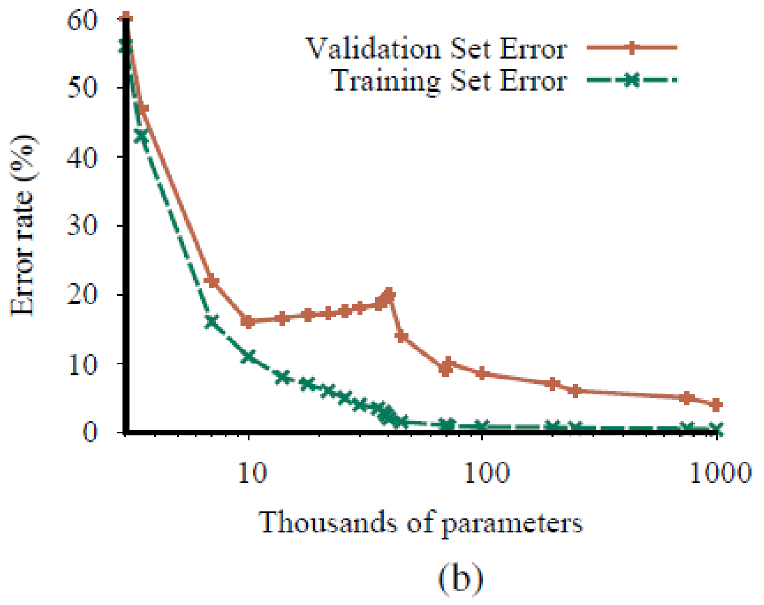
Model Selection and Optimization

```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do
     err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
     if err is starting to increase significantly then
        best_size ← the value of size with minimum err[size]
        h ← Learner(best_size, training_set)
        return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
    validation_set ← examples[(i − 1) × N/k:i × N/k]
    training_set ← examples − validation_set
    h ← Learner(size, training_set)
    errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k        // average error rate on validation sets, across k-fold cross-validation
```

An algorithm to select the model that has the lowest validation error. It builds models of increasing complexity, and choosing the one with best empirical error rate, *err*, on the validation data set. *Learner*(*size, examples*) returns a hypothesis whose complexity is set by the parameter *size*, and which is trained on *examples*. In CROSS-VALIDATION, each iteration of the **for** loop selects a different slice of the *examples* as the validation set, and keeps the other examples as the training set. It then returns the average validation set error over all the folds. Once we have determined which value of the *size* parameter is best, MODEL-SELECTION returns the model (i.e., learner/hypothesis) of that size, trained on all the training examples, along with its error rate on the held-out test examples.

Model Selection and Optimization





Error rates on training data (lower, green line) and validation data (upper, orange line) for models of different complexity on two different problems. MODEL-SELECTION picks the hyperparameter value with the lowest validation-set error. In (a) the model class is decision trees and the hyperparameter is the number of nodes. The data is from a version of the restaurant problem. The optimal size is 7. In (b) the model class is convolutional neural networks (see Section 22.3) and the hyperparameter is the number of regular parameters in the network. The data is the MNIST data set of images of digits; the task is to identify eachdigit. The optimal number of parameters is 1,000,000 (note the log scale).

Model Selection and Optimization

From error rates to loss

- Minimize a **loss function** rather than maximize a utility function. The loss function $L(x, y, \hat{y})$ is defined as the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$:

$$\begin{aligned} L(x,y,\hat{y}) &= \mathit{Utility}(\text{result of using } y \text{ given an input } x) \\ &- \mathit{Utility}(\text{result of using } \hat{y} \text{ given an input } x) \end{aligned}$$

- Simplified version independent of x: $L(y, \hat{y})$

- The learning agent maximizes its expected utility by choosing the hypothesis that minimizes expected loss over all input–output pairs it will see.

- prior probability distribution $\mathbf{P}(X,Y)$ over examples.

- the set of all possible input–output examples (ε)

Model Selection and Optimization

Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$

- **Generalization loss** for a hypothesis h (with respect to loss function L):

$$GenLoss_L(h) = \sum_{(x,y)\in\mathcal{E}} L(y, h(x))\, P(x, y)\,,$$

- **Empirical loss**

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

- The estimated **best hypothesis** $\hat{h}^*$, minimum empirical loss

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

Model Selection and Optimization

Regularization

- process of explicitly penalizing complex

- minimizes the weighted sum of empirical loss and the complexity of the hypothesis
- Complexity of the hypothesis

$$\begin{aligned} Cost(h) &= EmpLoss(h) + \lambda\, Complexity(h) \\ \hat{h}^* &= \operatorname*{argmin}_{h \in \mathcal{H}} Cost(h)\,. \end{aligned}$$

- The choice of regularization function depends on the hypothesis space.

- **Feature selection:**
  - reduce the dimensions that the models work with
  - discard attributes that appear to be irrelevant

Model Selection and Optimization

Hyperparameter tuning

- **Hand-tuning:** guess parameter values based on history, repeat until satisfactory performance

- **Grid search:** try all combinations of values and see which performs best on the validation data (small number of possible values)

- Random search

- Bayesian optimization: treats the task of choosing good hyperparameter values AS A machine learning problem

- Population-based training (PBT)