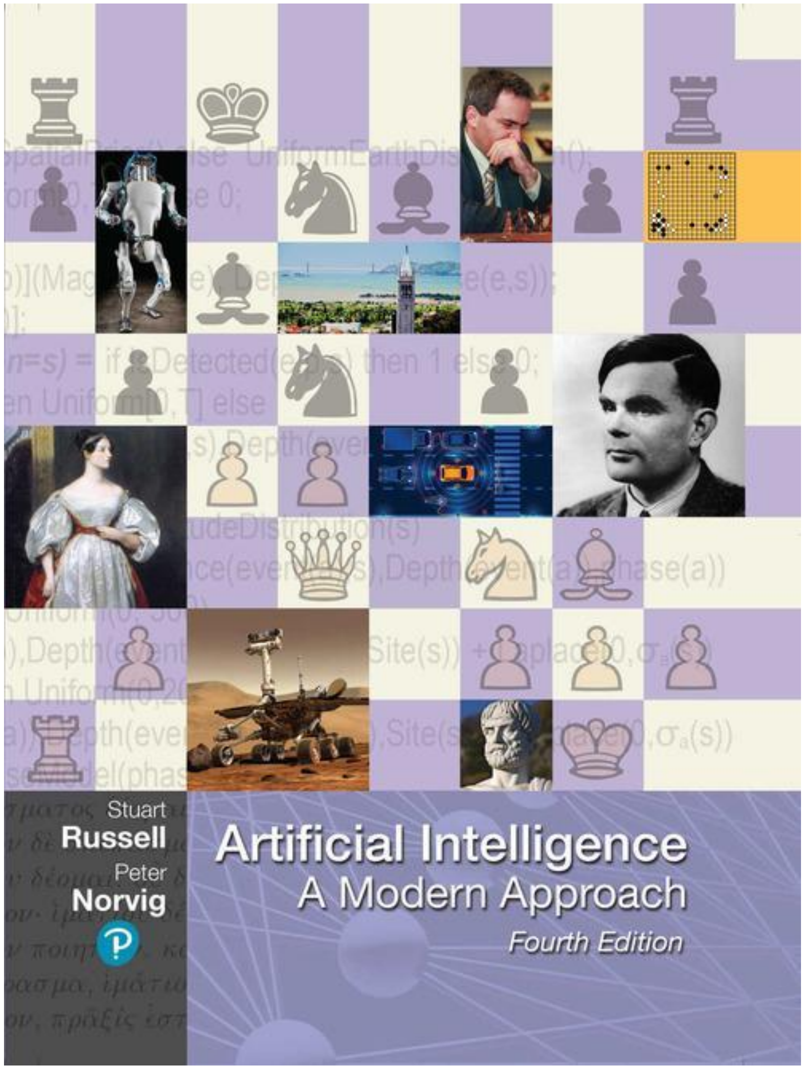
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 19

Learning From Examples

Pearson

Outline

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Model Selection and Optimization
- The Theory of Learning
- Linear Regression and Classification
- Nonparametric Models
- Ensemble Learning
- Developing Machine Learning Systems

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*

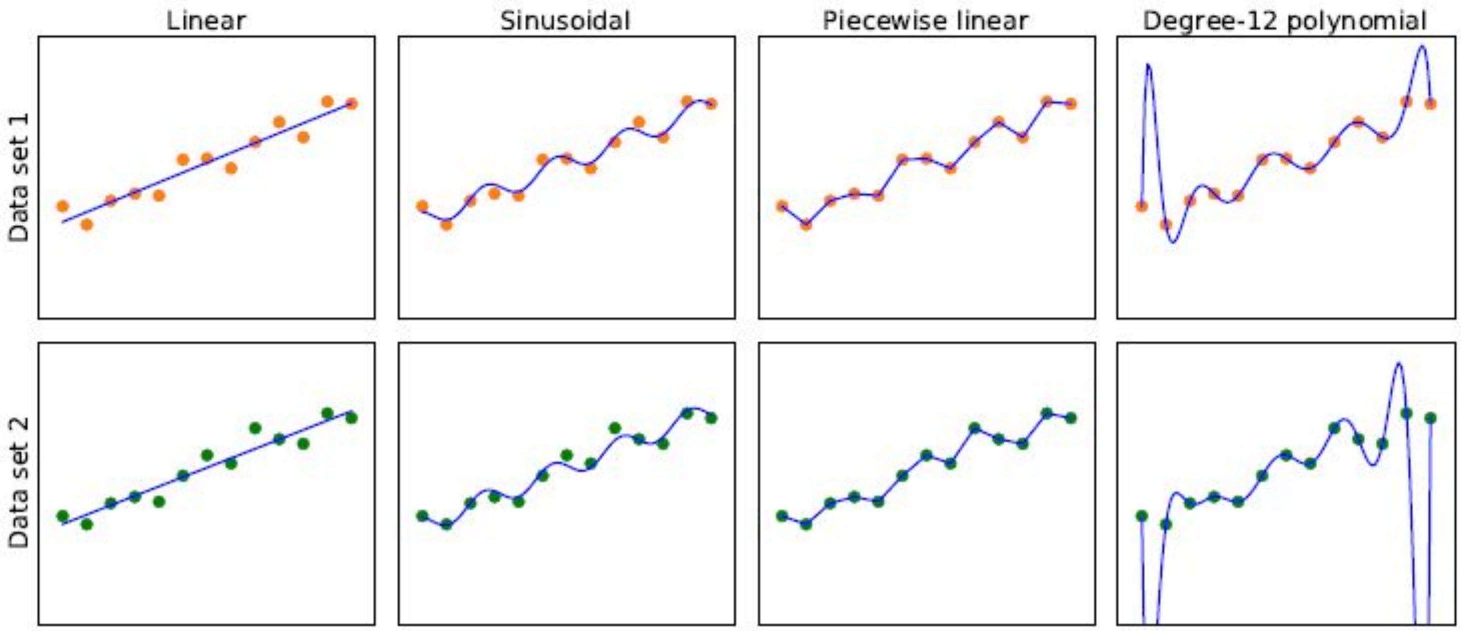
***Consistent* hypothesis**: an *h* such that each x_i in the training set has $h(x_i) = y_i$.

- *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

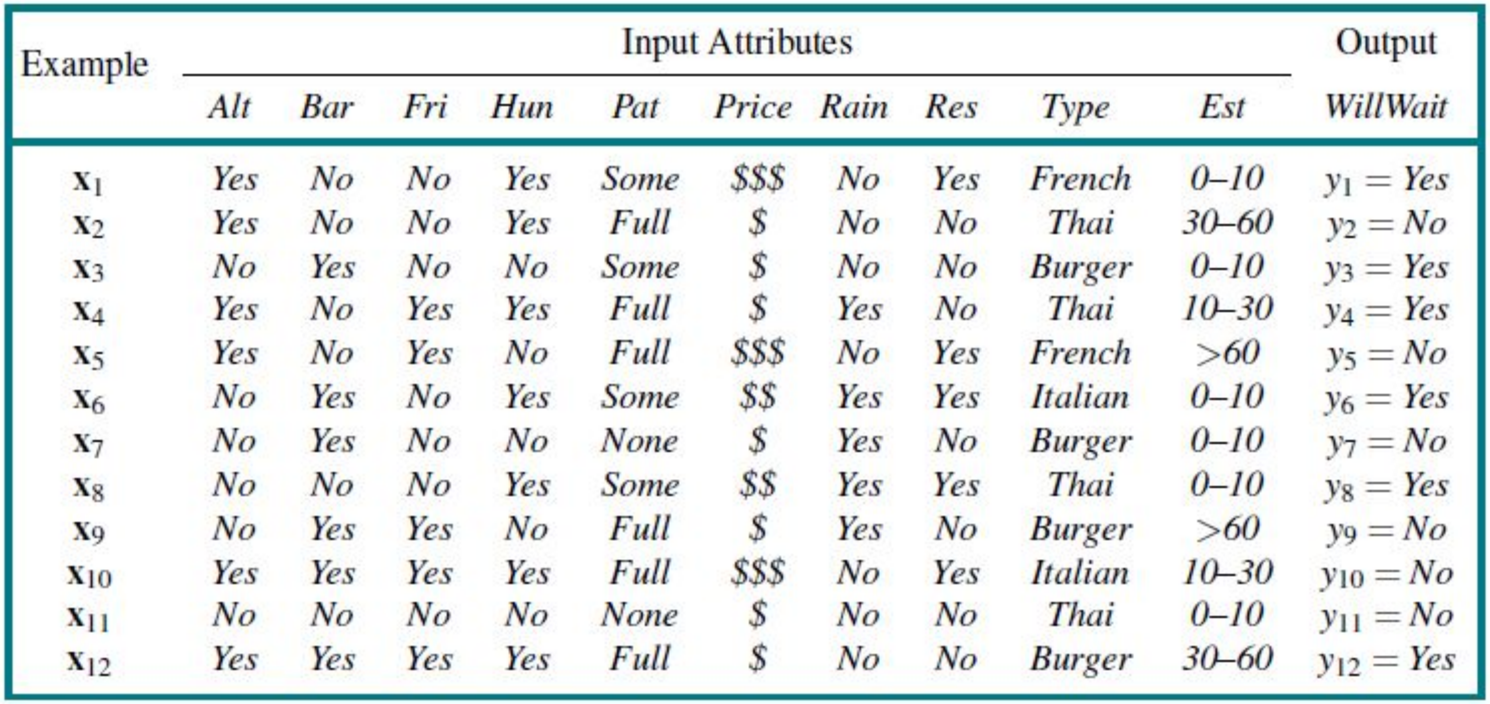
$$h^* = \operatorname{argmax} \; P(h|data)\,P(h)$$

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

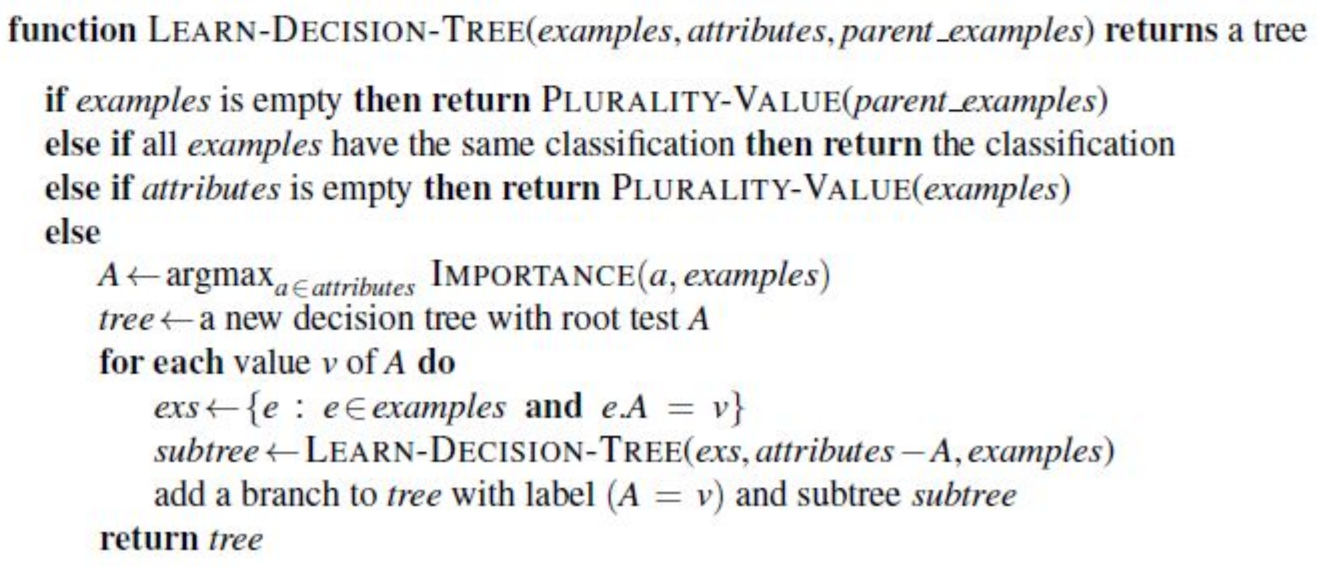
A decision tree is a representation of a function that maps a vector of attribute values to a single output value—a “decision.”

- 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



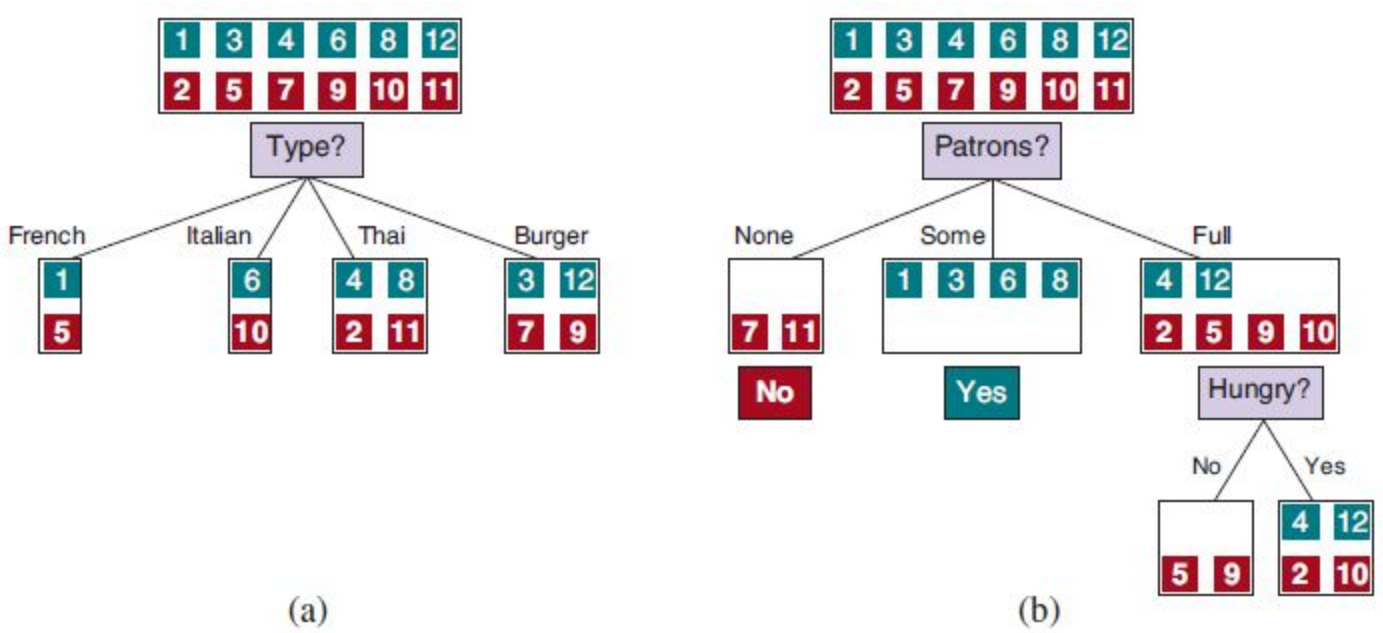
```
function LEARN-DECISION-TREE(examples, attributes, parent_examples) returns a tree
  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
     $A \leftarrow \text{argmax}_{a \in attributes}$ 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

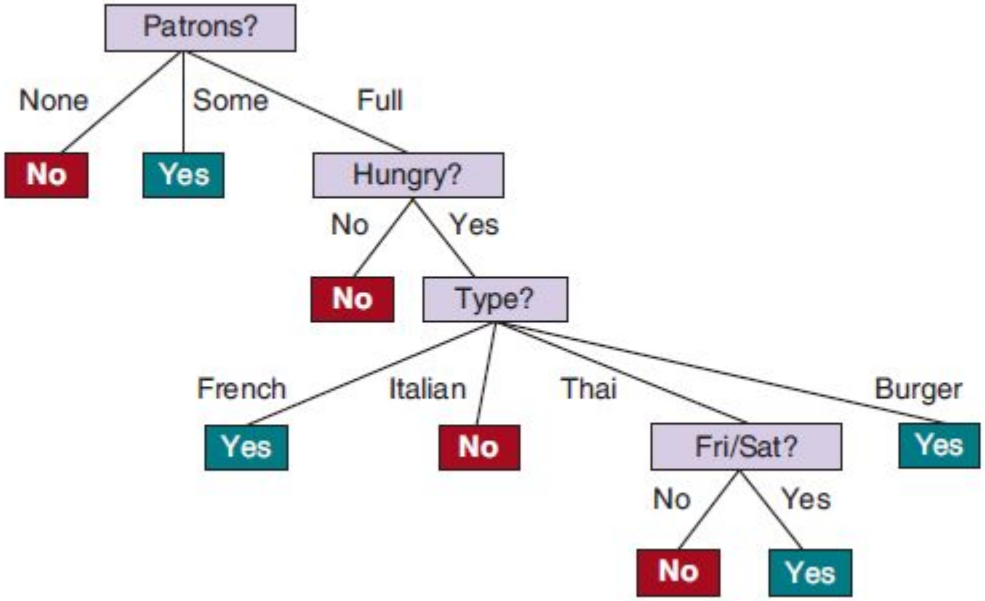


Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining.

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

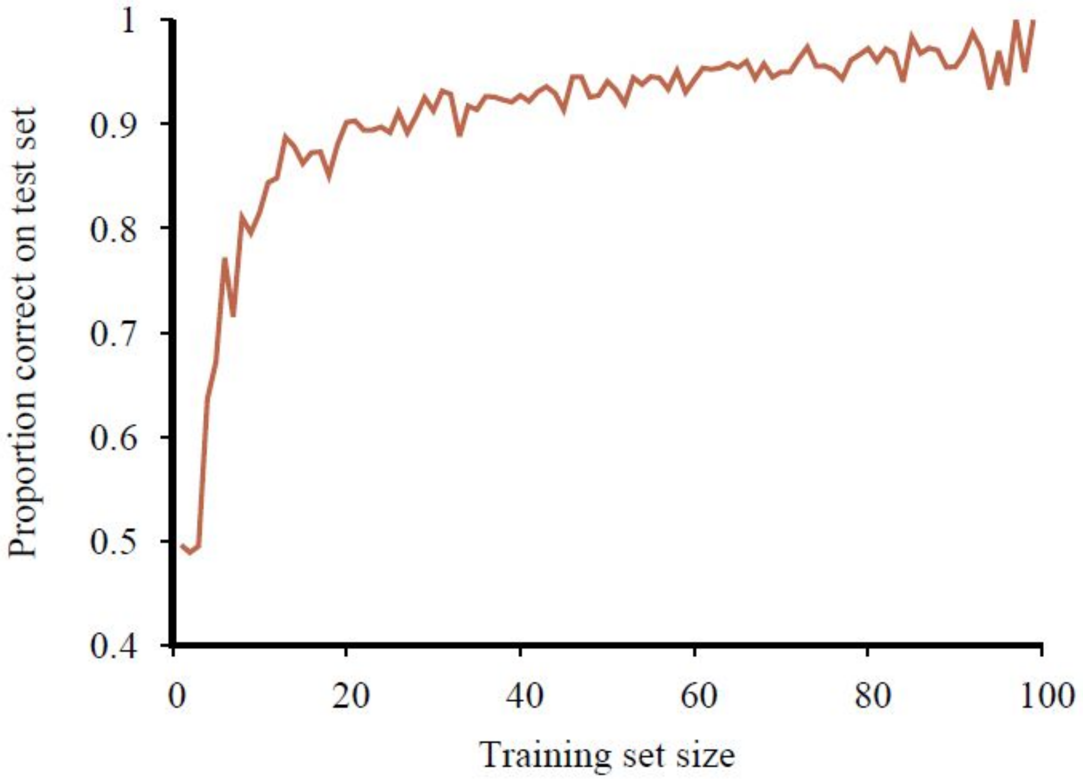
(b) Splitting on *Patrons* does a good job of separating positive and negative examples. After splitting on *Patrons*, *Hungry* is a fairly good second test

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:

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

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 $h(x)$ /= y for an (x, y)

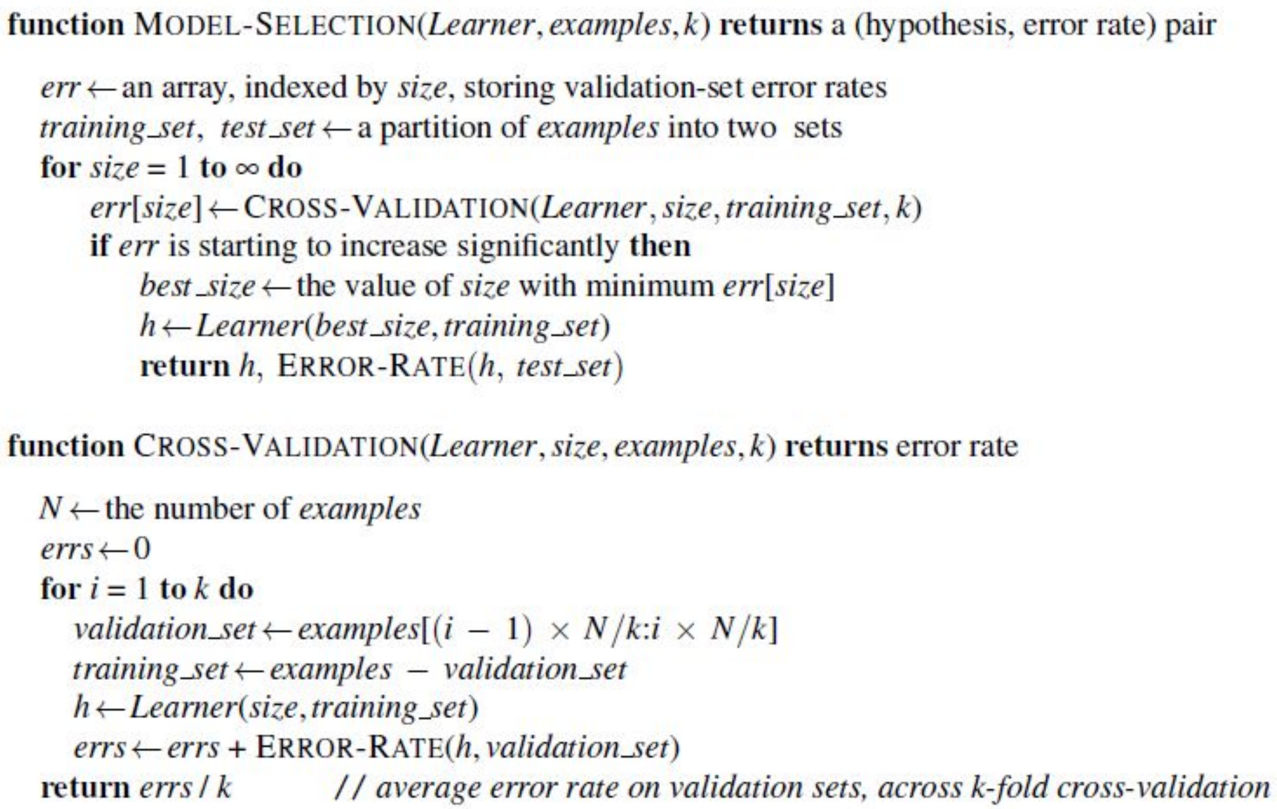
Three data sets are needed:

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*

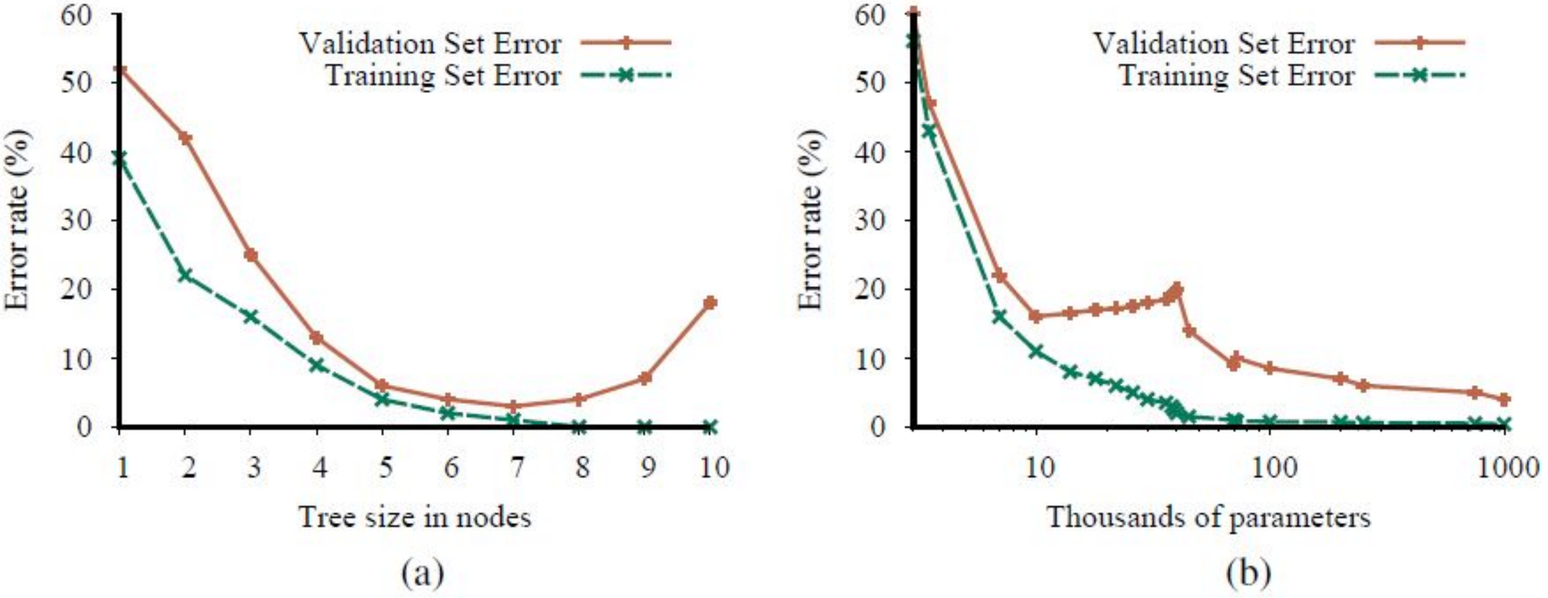
Model Selection and Optimization



```
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 $L(x, y, y\text{ˆ})$ is defined as the amount of utility lost by predicting $h(x) = y\text{ˆ}$ 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: $L(y, y\text{ˆ})$

- 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)

The Theory of Learning

Probably approximately correct (PAC): any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong

PAC learning algorithm: returns PAC hypotheses

A hypothesis h is called **approximately correct** if error(h) ε, where ε is a small constant.

$$P(h_b \text{ agrees with } N \text{ examples}) \leq (1-\epsilon)^N .$$

$$P(\mathcal{H}_{\text{bad}} \text{ contains a consistent hypothesis}) \leq |\mathcal{H}_{\text{bad}}|(1-\epsilon)^N \leq |\mathcal{H}|(1-\epsilon)^N$$

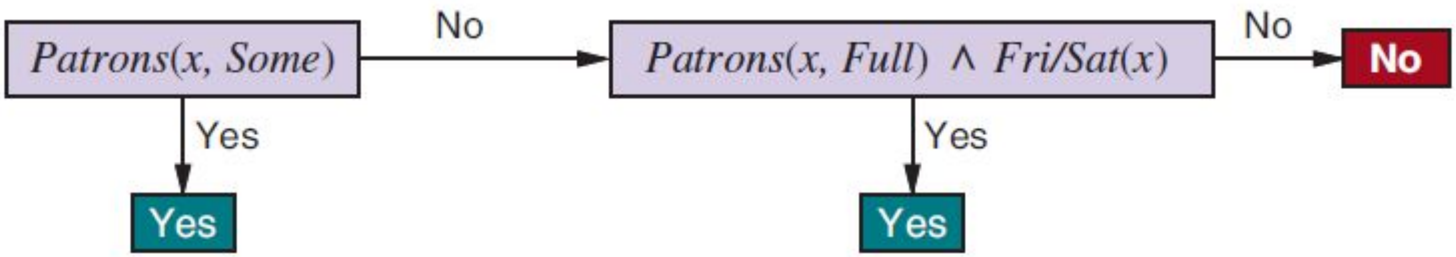
$$P(\mathcal{H}_{\text{bad}} \text{ contains a consistent hypothesis}) \leq |\mathcal{H}|(1-\epsilon)^N \leq \delta .$$

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|\mathcal{H}|\right)$$

The Theory of Learning

Example of decision lists

$$WillWait \Leftrightarrow (Patrons = Some) \lor (Patrons = Full \land Fri/Sat)\ .$$



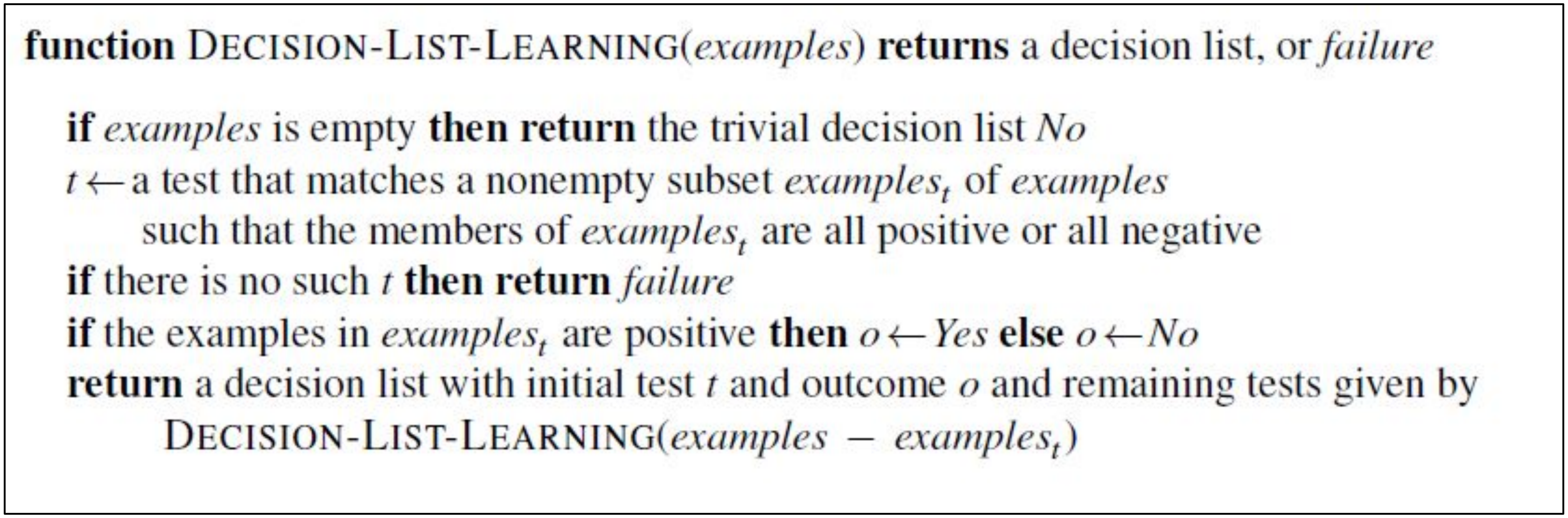
$$|k\text{-DL}(n)| \leq 3^c c! \text{ where } c = |Conj(n,k)|\ .$$

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)\ .$$

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}\ .$$

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

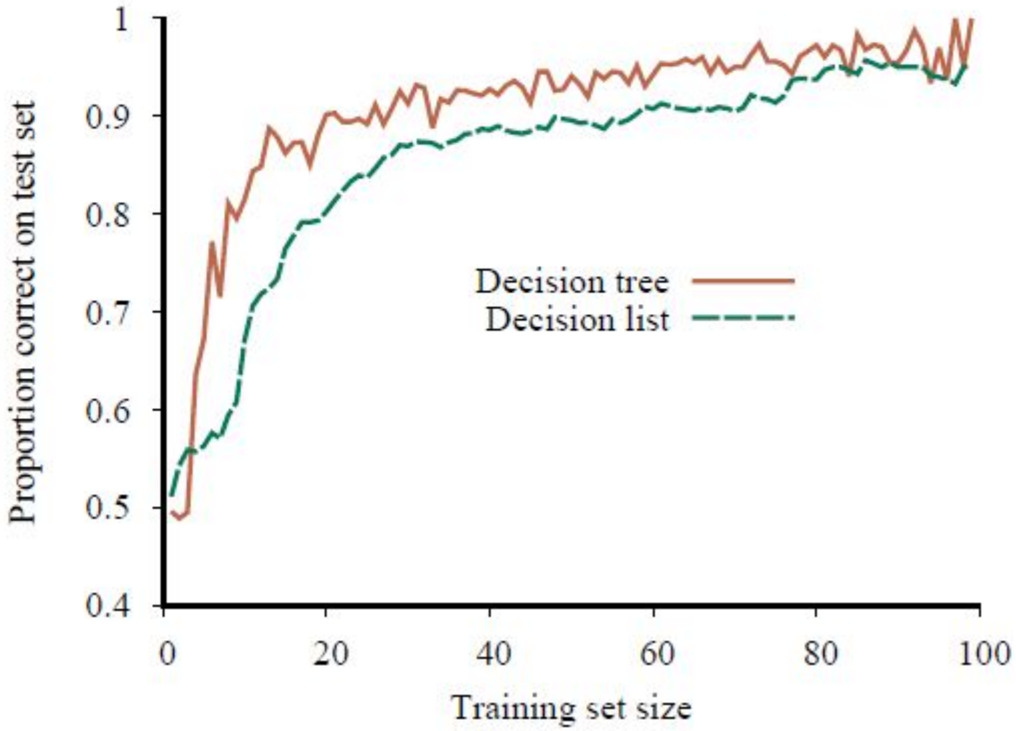
The Theory of Learning



```
function DECISION-LIST-LEARNING(examples) returns a decision list, or failure

  if examples is empty then return the trivial decision list No
  t ← a test that matches a nonempty subset $examples_t$ of examples
      such that the members of $examples_t$ are all positive or all negative
  if there is no such t then return failure
  if the examples in $examples_t$ are positive then o ← Yes else o ← No
  return a decision list with initial test t and outcome o and remaining tests given by
        DECISION-LIST-LEARNING($examples - examples_t$)
```

An algorithm for learning decision lists

The Theory of Learning



Learning curve for DECISION-LIST-LEARNING algorithm on the restaurant data. The curve for LEARN-DECISION-TREE is shown for comparison; decision trees do slightly better on this particular problem

Linear Regression and Classification

Univariate linear regression

Input x and output y

$$y = w_1 x + w_0$$

Linear function

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

Linear regression: finding the $h_{\mathbf{w}}$ that best fits these data is
Find the values of the weights (w_0, w_1) that minimize the empirical loss.

Squared-error loss function, L_2, summed over all the training examples:

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

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

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N .$$

Linear Regression and Classification

Gradient descent

- search through a continuous weight space by incrementally modifying the parameters (minimizing loss)

```
$\mathbf{w} \leftarrow$ any point in the parameter space
while not converged do
    for each $w_i$ in $\mathbf{w}$ do
        $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$
```

- α: step size/learning rate that can be a fixed constant or decay over time

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

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

Linear Regression and Classification

Batch gradient descent

- minimize the sum of the individual losses

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

- The loss surface is convex,
- no local minima to get stuck in, and convergence to the global minimum is guaranteed
- As long *α* is not too large, overshoots.
- **Epoch**: step that covers all the training examples

Stochastic gradient descent or SGD

- it **randomly** selects a small number of training examples at each step
- select a minibatch of *m* out of the *N* examples.
- choose *m* to take advantage of parallel vector operations,

Linear Regression and Classification

Multivariable linear regression

- $\mathbf{x}_j$ is an n-element vector
- h : dot product of the weights and the input vector

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^{\top}\mathbf{x}_j = \sum_i w_i x_{j,i}$$

- Best vector of weights, $\mathbf{w}^*$, minimizes squared-error loss

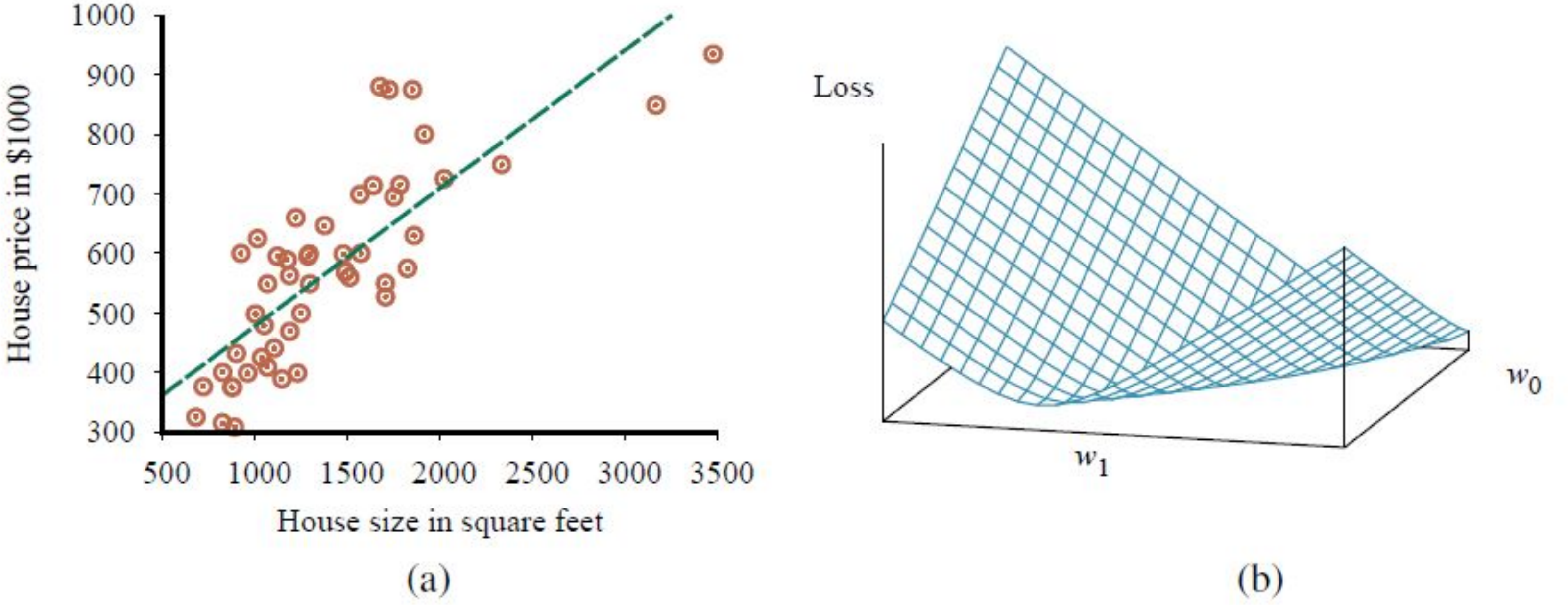
$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)\,.$$

- Gradient descent will reach the (unique) minimum of the loss function; the update equation for e

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}\,.$$

$$\mathbf{w}^* = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\mathbf{y}\,.$$

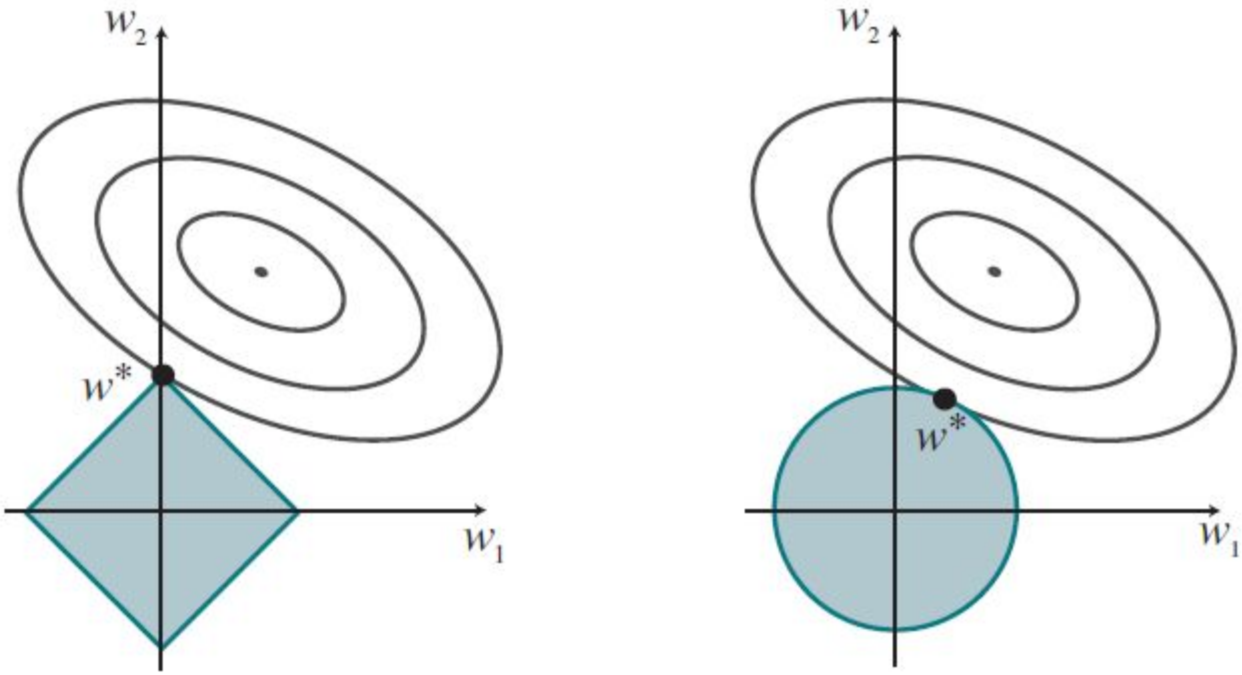
Linear Regression and Classification



(a) Data points of price versus floor space of houses for sale in Berkeley, CA, in July 2009, along with the linear function hypothesis that minimizes squared-error loss: $y = 0.232x + 246$.

(b) Plot of the loss function $\sum_j (y_j - w_1 x_j + w_0)^2$ for various values of w_0, w_1. Note that the loss function is convex, with a single global minimum.

Linear Regression and Classification



Why L_1 regularization tends to produce a sparse model. Left: With L_1 regularization (box), the minimal achievable loss (concentric contours) often occurs on an axis, meaning a weight of zero. Right: With L_2 regularization (circle), the minimal loss is likely to occur anywhere on the circle, giving no preference to zero weights.

Linear Regression and Classification

Linear classifiers with a hard threshold

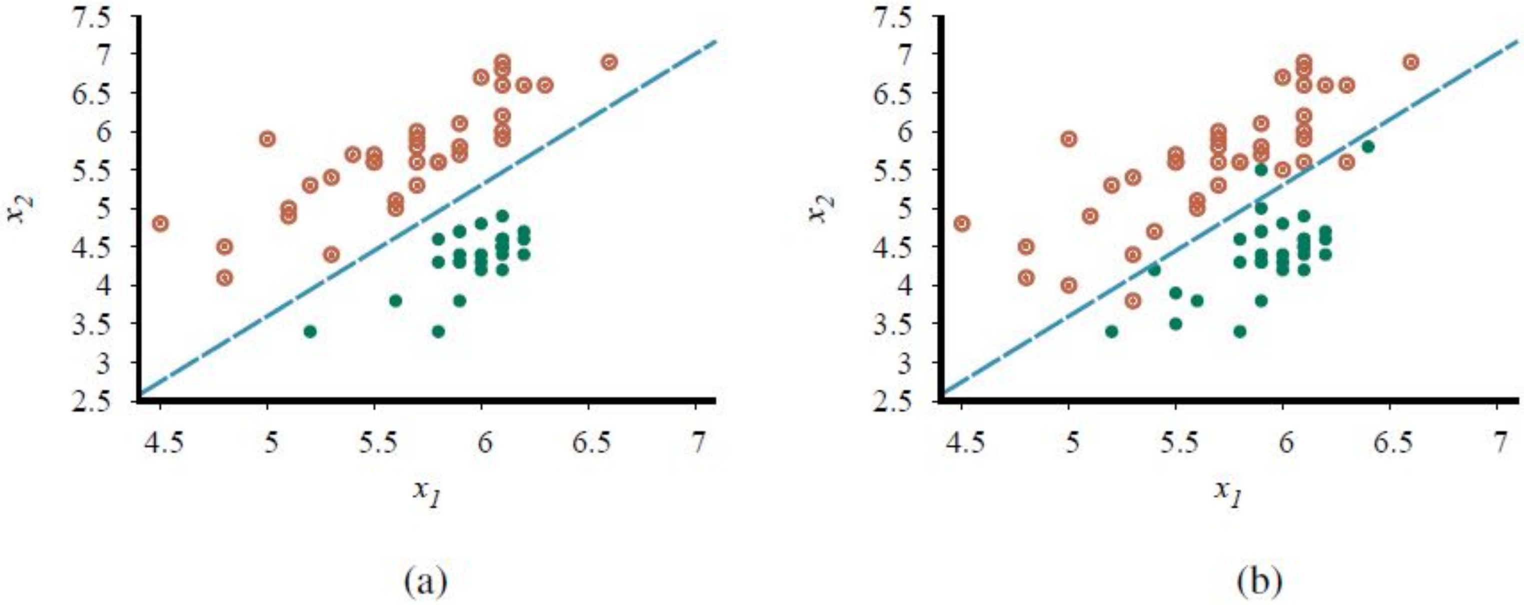
- Linear functions can be used to do classification as well as regression.
- **Decision boundary:** a line (or a surface, in higher dimensions) that separates the two classes.
- **Linear separator**: linear decision boundary for linearly separable data

- h: result of passing the linear function $\mathbf{w} \cdot \mathbf{x}$ through a **threshold function**:

$h_{\mathbf{w}}(\mathbf{x}) = \mathit{Threshold}(\mathbf{w} \cdot \mathbf{x})$ where $\mathit{Threshold}(z) = 1$ if $z \geq 0$ and 0 otherwise.

- Possibilities of outputs for weight update during training:
  - Correct output: weight unchanged
  - y is 1 but $h_{\mathbf{w}}(\mathbf{x})$ is 0: w_i is *increased* when the corresponding input x_i is positive *and decreased* when x_i is negative.
  - y *is* 0 but $h_{\mathbf{w}}(\mathbf{x})$ is 1: w_i is *decreased* when the corresponding input x_i is positive *and increased* when x_i is negative.

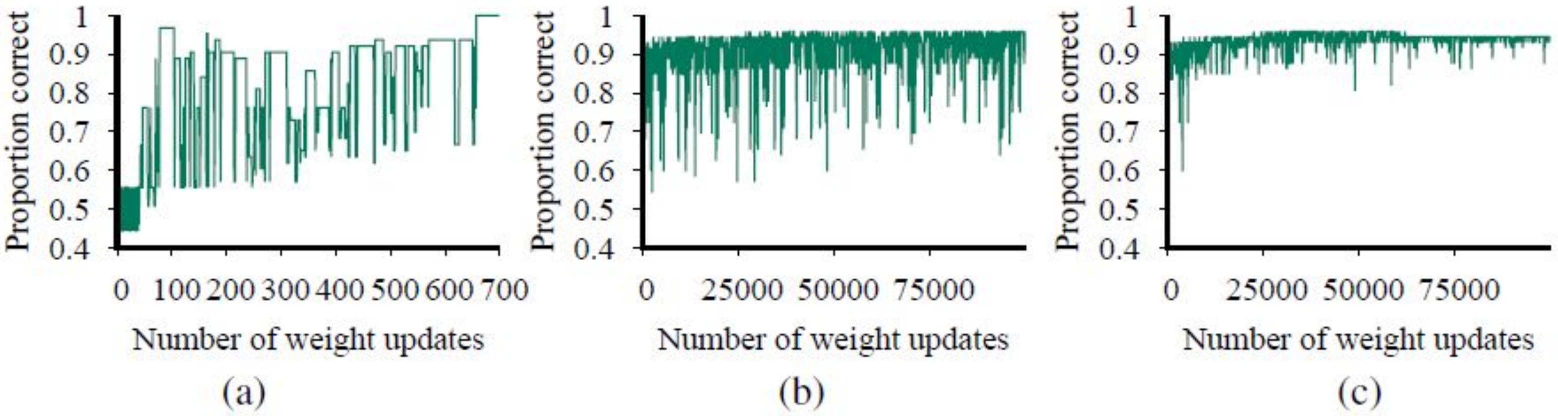
Linear Regression and Classification



(a) Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East. Also shown is a decision boundary between the classes.

(b) The same domain with more data es and explosions are no longer linearly separable.

Linear Regression and Classification



(a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule.
(b) The same plot for the noisy, nonseparable data; note the change in scale of the *x*-axis.
(c) The same plot as in (b), with a learning rate schedule $a(t) = 1000/(1000 + t)$.

Linear Regression and Classification

Linear classification with logistic regression

- softening the threshold function— approximating the hard threshold with a continuous, differentiable function
- Logistic function:

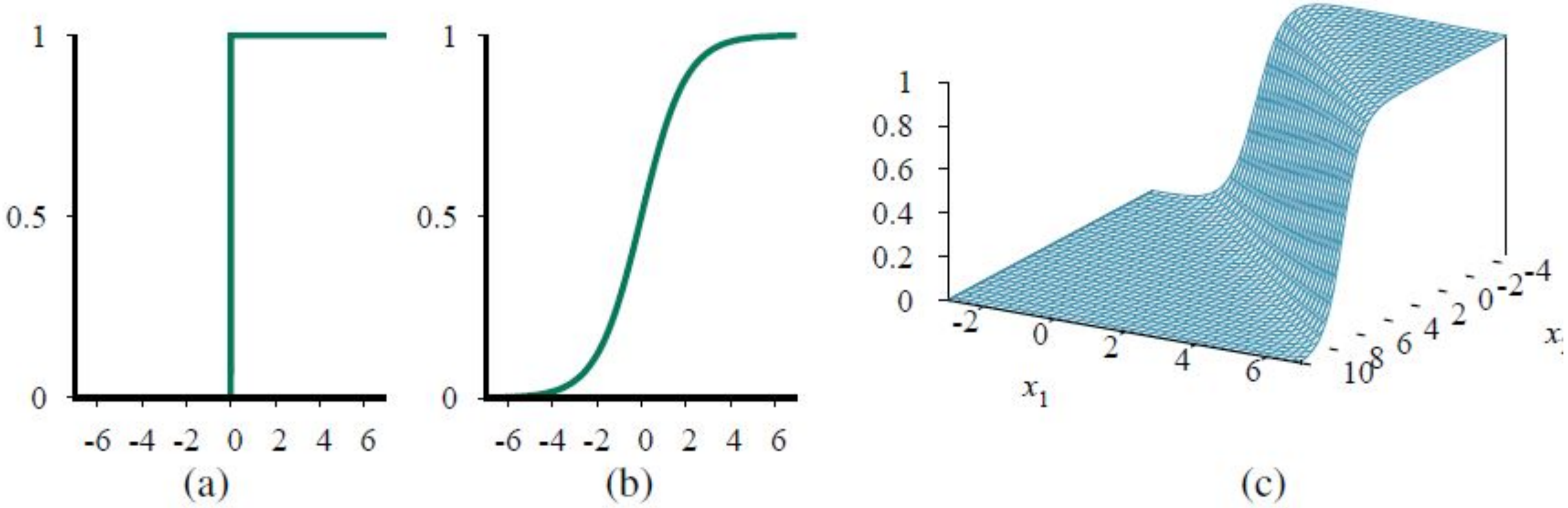
$$Logistic(z) = \frac{1}{1+e^{-z}}$$

- Used to replace the threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}\,.$$

- **Logistic regression**: process of fitting the weights of this model to minimize loss on a data set

Linear Regression and Classification



(a)

(b)

(c)

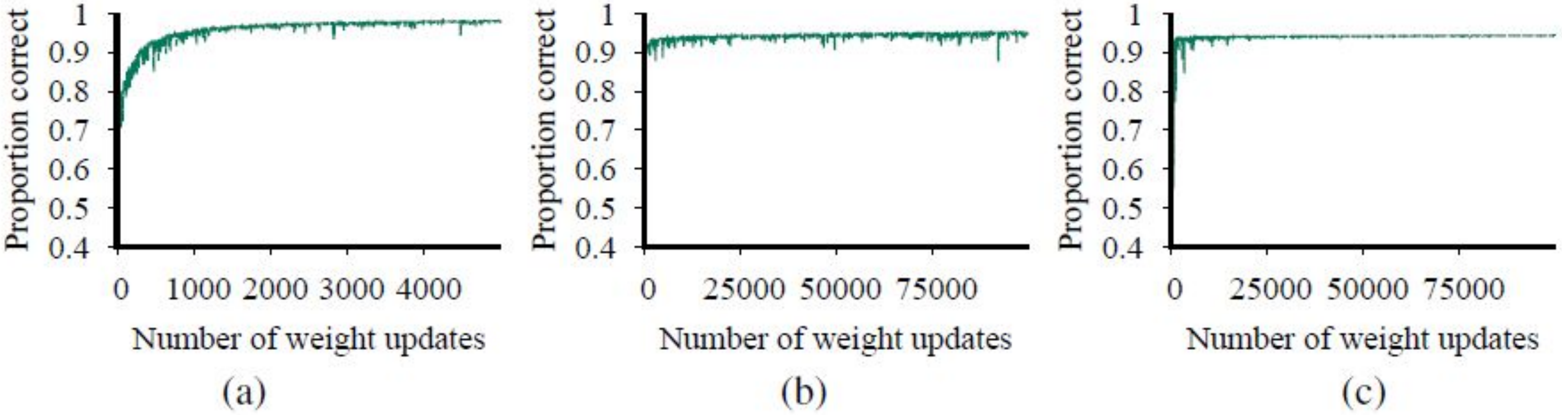
(a) The hard threshold function *Threshold*(z) with 0/1 output. Note that the function is nondifferentiable at $z = 0$.

(b) The logistic function (sigmoid function), *Logistic*$(z) = \frac{1}{1+e^{-z}}$

(c) Plot of a logistic regression hypothesis $h_{\mathbf{w}}(\mathbf{x}) =$ *Logistic*$(\mathbf{w} \cdot \mathbf{x})$

1

Linear Regression and Classification



Logistic regression on previous data. The plot in (a) covers 5000 iterations rather than 700, while the plots in (b) and (c) use the same scale as before.)

1

Nonparametric Models

Parametric model: learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples)

Nonparametric model: model that cannot be characterized by a bounded set of parameters

One example **piecewise linear function** that retains all the data points as
part of the model. (instance-based learning or memory-based learning)

Simplest instance-based learning method: **table lookup**

- take all the training examples, put them in a lookup table, and then when asked for $h(\mathbf{x})$, see if $\mathbf{x}$ is in the table; if it is, return the corresponding y.

Nonparametric Models

Nearest-neighbor models

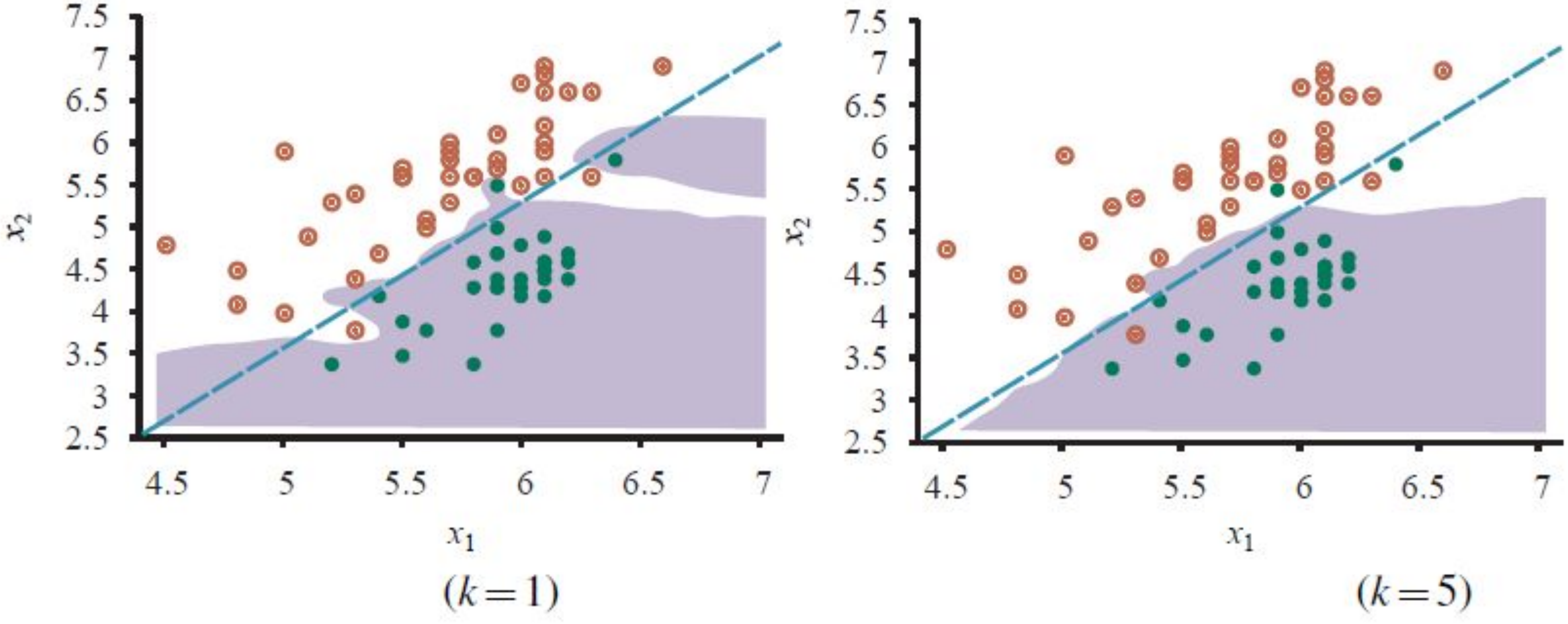
k-nearest-neighbors

- given a query $\mathbf{x}_q$, instead of finding an example that is equal to $\mathbf{x}_q$, find the *k* examples that are *nearest* to $\mathbf{x}_q$.
- set of *k* neighbors nearest to $\mathbf{x}_q$. = $NN(k, \mathbf{x}_q)$
- for example, if $k = 3$ and the output values are *Yes, No, Yes* , then the classification will be *Yes*.
- To avoid ties on binary classification, k is usually chosen as an odd number.
- Measure the distance from a query point $\mathbf{x}_q$ to an example point $\mathbf{x}_j$ using **Minkowski distance** or L^p

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p} .$$

- p =2, **Euclidean distance**
- p =1, **Manhattan distance**
- Boolean attribute values, **Hamming distance**
- **Mahalanobis distance:** takes into account the covariance between dimensions.

Nonparametric Models



($k = 1$)

($k = 5$)

(a) A k-nearest-neighbors model showing the extent of the explosion class for previously used data, with k = 1. Overfitting is apparent.

(b) k = 5, the overfitting problem goes away for this data set.

Nonparametric Models

Locality-sensitive hashing

- Hash tables have the potential to provide even faster lookup than binary trees
- near points grouped together in the same bin, **locality-sensitive hash** (LSH)
- create multiple random projections and combine them
  - Random subset of the bit-string representation
  - Projection is just a random subset of the bit-string representation. We choose ,e different random projections and create ,e hash tables, $g_1(\mathbf{x}),\ldots, g_\ell(\mathbf{x})$.
  - Enter all the examples into each hash table
  - Fetch the set of points in bin $g_i(\mathbf{x}_q)$ of each hash table,
  - Union these ℓ sets together into a set of candidate points, C.
  - Compute the actual distance to $\mathbf{x}_q$ for each of the points in C and return the k closest points.

Nonparametric Models

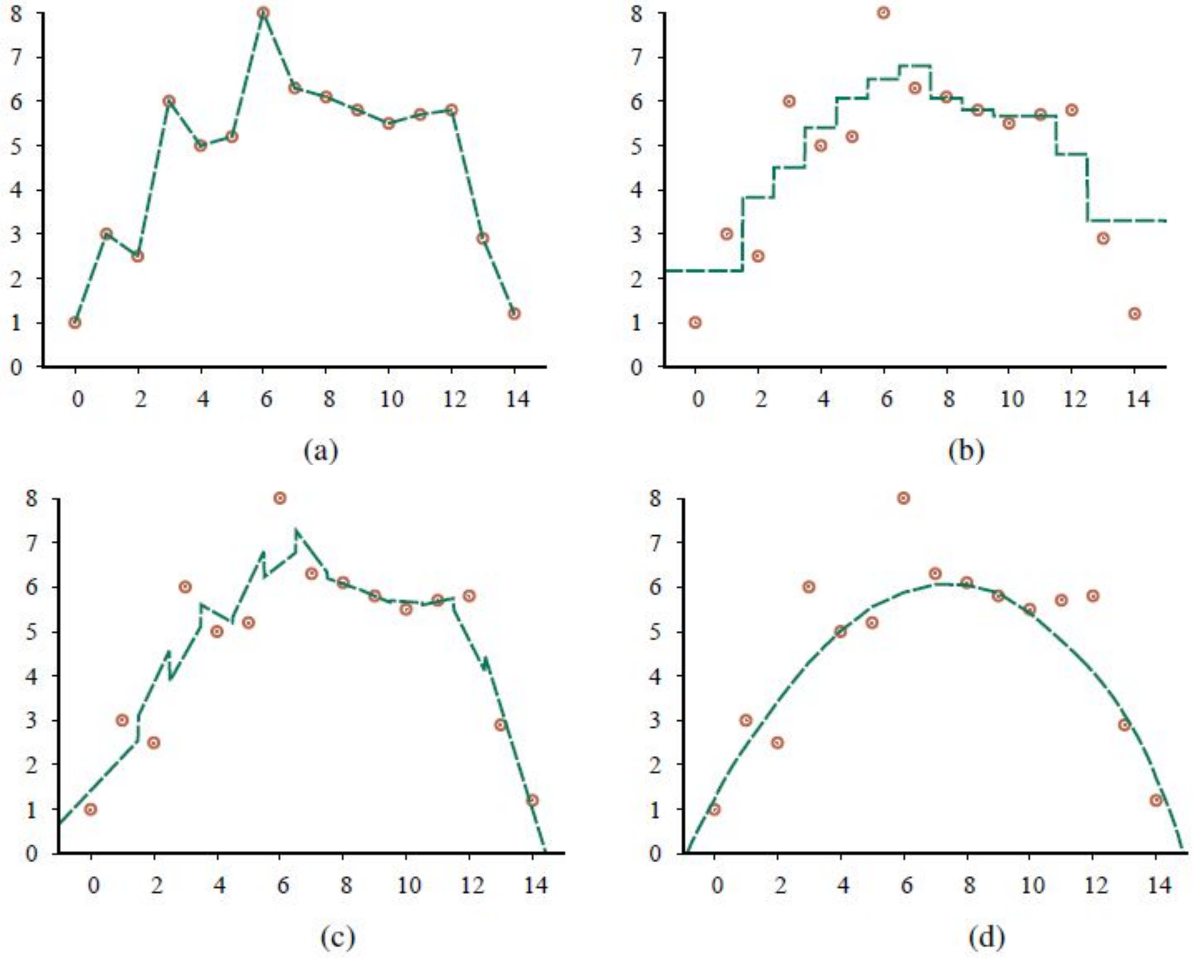
Nonparametric regression

- *k*-**nearest-neighbors regression** improves on connect-the-dots.

- **Locally weighted regression**: gives us the advantages of nearest neighbors without the discontinuities
  - at each query point x_q, the examples that are close to x_q are weighted heavily, and the examples that are farther away are weighted less heavily, and the farthest not at all.
  - decide how much to weight each example with a function known as a kernel, whose input is a distance between the query point and the example

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathbf{x}_j))\,(y_j - \mathbf{w}\cdot\mathbf{x}_j)^2 ,$$

Nonparametric Models



Nonparametric regression models: (a) connect the dots, (b) 3-nearest neighbors average, (c) 3-nearest-nei hbors linear regression, (d) locally weighted regression with a quadratic kernel of width 10.

Nonparametric Models

Support vector machines (SVM)

SVMs retain three attractive properties over deep learning networks and random forests:

- SVMs construct a maximum margin separator
  - a decision boundary with the largest possible distance to example points
- SVMs create a linear separating hyperplane,
- SVMs are nonparametric
  - the separating hyperplane is defined by a set of example

Instead of minimizing expected empirical loss on the training data, SVMs attempt to minimize expected generalization loss.

Nonparametric Models

- The separator between points is defined as the set of points $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$.
- Optimal solution is found by solving

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

- Expression is convex; it has a single global maximum that can be found efficiently.
- The data enter the expression only in the form of dot products of pairs of points
- A final important property is that the weights α_j associated with each data point are *zero* except for the **support vectors**—the points closest to the separator.

Nonparametric Models

The kernel trick

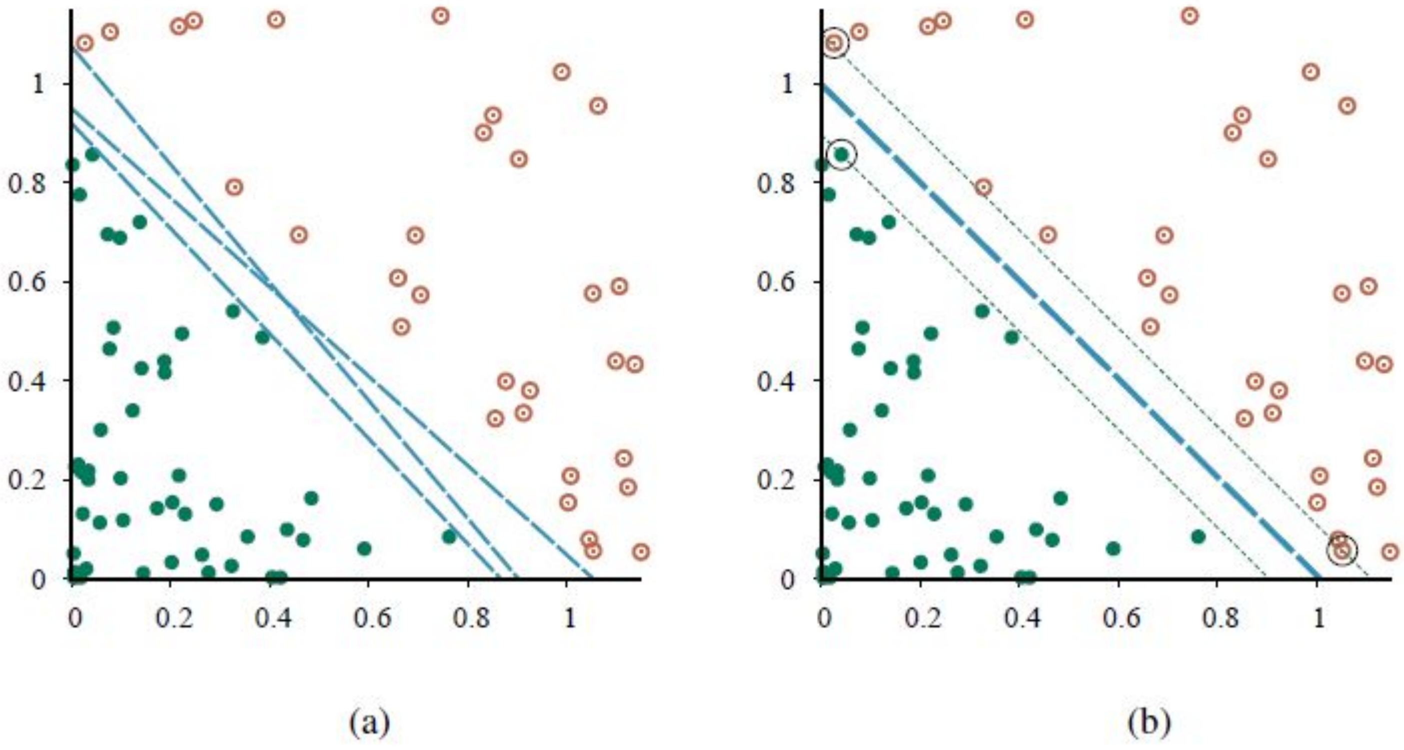
- Plugging kernel into equation below, optimal linear separators can be found

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k)$$

- The resulting linear separators, when mapped back to the original input space, can correspond to arbitrarily wiggly, nonlinear decision boundaries between the positive and negative examples.

- The kernel method can be applied with any other algorithm that can be reformulated to work only with dot products of pairs of data points (kernelized version of the algorithm)

Nonparametric Models



Support vector machine classification:

(a) Two classes of points (orange open and green filled circles) and three candidate linear separators.

(b) The maximum margin separator (heavy line), is at the midpoint of the margin (area between dashed lines). The support vectors (points with large black circles) are the examples closest to the separator;here there are three.

Ensemble Learning

- The idea of ensemble learning is to select a collection, or ensemble, of hypotheses, $h_1, h_2, \ldots, h_n$, and combine their predictions by averaging, voting, or by another level of machine learning

- individual hypotheses: **base models**

- Combination of base models**: Ensemble models**

- **Reasons to do ensemble learning**
  - **Reduce bias,** ensemble can be more expressive thus less bias than base models
  - **Reduce variance**, it is hoped it is less likely multiple classifiers will misclassify

Ensemble Learning

Bagging

- generate K distinct training sets by sampling with replacement from the original training set.
- randomly pick N examples from the training set, but each of those picks might be an example picked before.
- run our machine learning algorithm on the N examples to get a hypothesis
- repeat this process K times, getting K different hypotheses
- aggregate the predictions from all K hypotheses.
- for classification problems, that means taking the plurality vote (the majority vote for binary classification).
- for regression problems, the final output is the average of hypotheses:

$$h(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} h_i(\mathbf{x})$$

Ensemble Learning

Random forests

- a form of decision tree bagging

- randomly vary the attribute choices

- At each split point in constructing the tree, we select a random sampling of attributes, and then compute which of those gives the highest information gain

- Given n attributes, $\sqrt{n}$ common number of attributes randomly picked at each split for classification *n*/3 for regression problems.

- **Extremely randomized trees (ExtraTrees):**
  - for each selected attribute, randomly sample several candidate values from a uniform distribution over the attribute's range.
  - select the value that has the highest information gain.

- Pruning prevents overfitting

Ensemble Learning

Stacking

- combines multiple base models from different model classes trained on the same data

- approach:
  - use the same training data to train each of the base models,
  - use the held-out validation data (plus predictions) to train the ensemble model.
  - Also possible to use cross-validation if desired.

- can be thought of as a layer of base models with an ensemble model stacked above it, operating on the output of the base models

Ensemble Learning

Boosting

- weighted training set: each example has an associated weight $w_j \geq 0$ that describes how much the example should count during training.

- Start with first hypothesis h_1.

- increase their weights while decreasing the weights of the correctly classified examples.

- process continues in this way until we have generated K hypotheses, where K is an input to the boosting algorithm.

- Similar to a Greedy algorithm in the sense that it does not backtrack; once it has chosen a hypothesis h_i it will never undo that choice; rather i[illegible] hypotheses

$$h(\mathbf{x}) = \sum_{i=1}^{K} z_i h_i(\mathbf{x})$$

Ensemble Learning

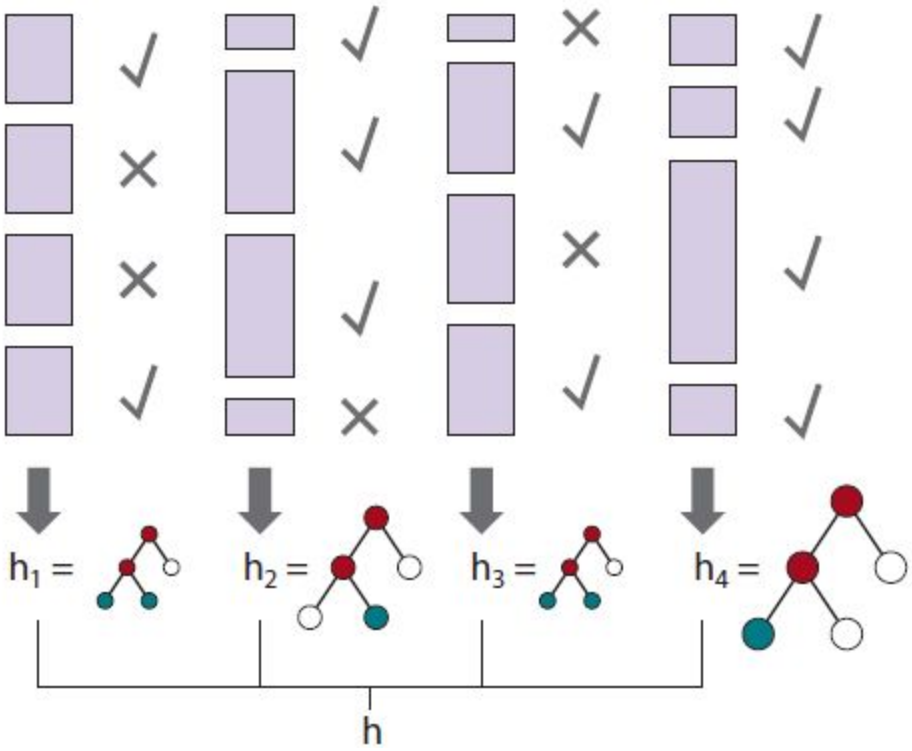
Input learning algorithm (L)

Weak learning algorithm: L always returns a hypothesis with accuracy on the training set that is slightly better than random guessing

ADABOOST

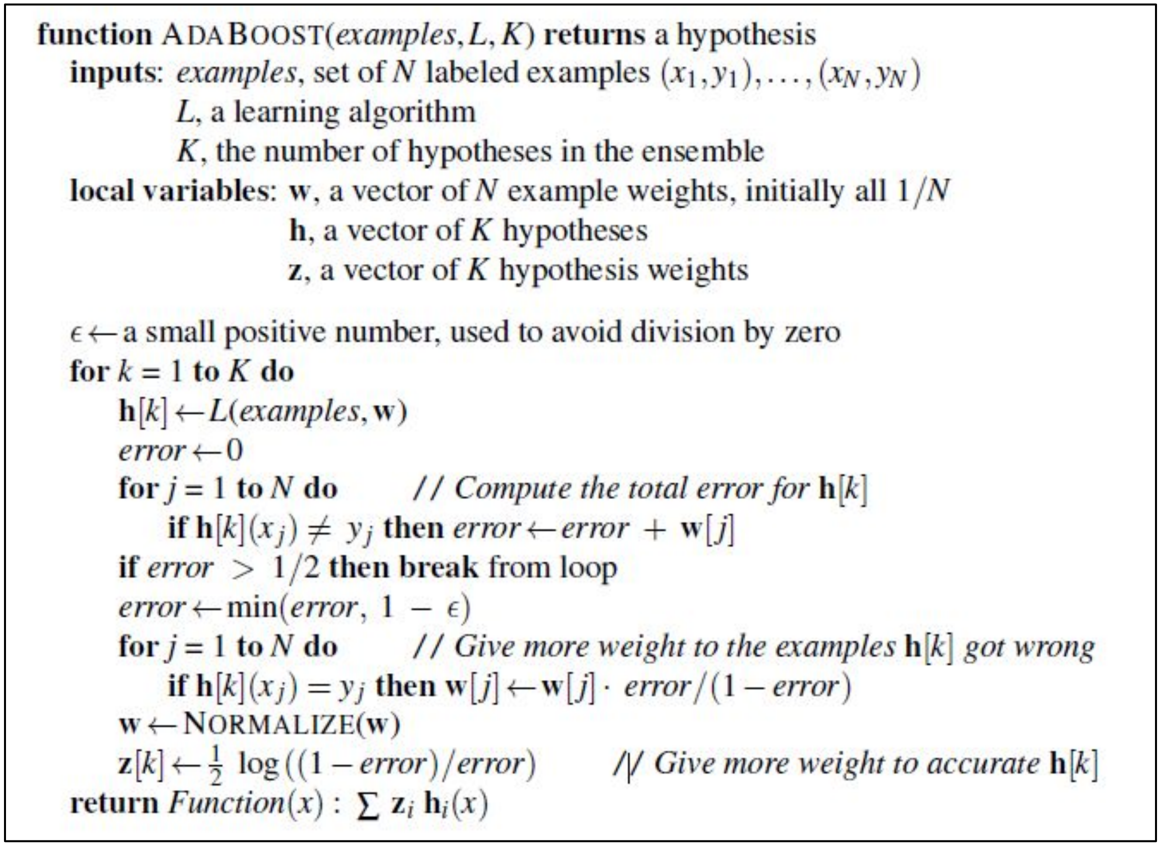
- will return a hypothesis that *classifies the training data perfectly* for large enough K
- boosts the accuracy of the original learning algorithm
- overcome any amount of bias in the base model,
- As long the base model is ϵ is better than random guessing

Ensemble Learning



How the boosting algorithm works. Each shaded rectangle corresponds to an example; the height of the rectangle corresponds to the weight. The checks and crosses indicate whether the example was classified correctly by the current hypothesis. The size of the decision tree indicates the weight of that hypothesis in the final ensemble

Ensemble Learning



```
function ADABOOST(examples, L, K) returns a hypothesis
  inputs: examples, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
          L, a learning algorithm
          K, the number of hypotheses in the ensemble
  local variables: w, a vector of N example weights, initially all $1/N$
                   h, a vector of K hypotheses
                   z, a vector of K hypothesis weights

  $\epsilon \leftarrow$ a small positive number, used to avoid division by zero
  for $k = 1$ to $K$ do
      $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
      $error \leftarrow 0$
      for $j = 1$ to $N$ do        // Compute the total error for $\mathbf{h}[k]$
          if $\mathbf{h}[k](x_j) \neq y_j$ then $error \leftarrow error + \mathbf{w}[j]$
      if $error > 1/2$ then break from loop
      $error \leftarrow \min(error,\ 1 - \epsilon)$
      for $j = 1$ to $N$ do        // Give more weight to the examples $\mathbf{h}[k]$ got wrong
          if $\mathbf{h}[k](x_j) = y_j$ then $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
      $\mathbf{w} \leftarrow$ NORMALIZE($\mathbf{w}$)
      $\mathbf{z}[k] \leftarrow \frac{1}{2} \log((1 - error)/error)$        /|/ Give more weight to accurate $\mathbf{h}[k]$
  return Function$(x)$ : $\sum \mathbf{z}_i\ \mathbf{h}_i(x)$
```

The ADABOOST variant of the boosting method for ensemble learning. The algorithm generates hypotheses by successively reweighting the training examples. The function WEIGHTED-MAJORITY generates a hypothesis that returns the output value with the highest vote from the hypotheses in **h**, with votes weighted by **z**. For regression problems, or for binary classification with two classes -1 and 1, this is $\sum_k \mathbf{h}[k]\mathbf{z}[k]$.

Online Learning

- an agent receives an input x_j from nature, predicts the corresponding y_j, and then is told the correct answer.
- the process repeats with x_{j+1}, and so on.
- **Randomized weighted majority algorithm:** keep track of how well each expert performs, and choose to believe them in proportion to their past performance.
- *Initialize* a set of weights $\{w_1, \ldots, w_K\}$ all to 1.
- for each problem to be solved do

```
1. Receive the predictions $\{y\hat{}_1, \ldots, y\hat{}_K\}$ from the experts.
2. Randomly choose an expert $k^*$ in proportion to its weight: $P(k) = w_k$.
3. yield $\hat{\boldsymbol{y}}_k *$ as the answer to this problem.
4. Receive the correct answer $y$.
5. For each expert $k$ such that $y\hat{}_k /= y$, update $w_k \leftarrow \beta w_k$
6. Normalize the weights so that $\sum_k w_k = 1$.
```

Here β is a number, $0 < \beta < 1$, that tells how much to penalize an expert for each mistake.

- Measure of success using regret: number of additional mistakes we make compared to the expert.
- M^* : number of mistakes made by the best expert)
- M : number of mistakes made by the random weighted majority algorithm)

$$M < \frac{M^* \ln(1/\beta) + \ln K}{1-\beta}.$$

Developing Machine Learning Systems

- Problem formulation
  - Determine problem/solution, specify a loss function
  - metrics that should be tracked

- Data collection, assessment, and management
  - When data are limited, data augmentation can help
  - For unbalanced class, undersample the majority, over-sample the minority
  - Consider outliers
  - Feature engineering, Exploratory data analysis (EDA)

- Model selection and training
  - receiver operating characteristic (ROC) curve
  - AUC provides a single-number summary of the ROC curv
  - confusion matrIX

- Trust, interpretability, and explainability
  - Source control Testing Review, Monitoring, Accountability,
  - Inspect the actual model and understand why it got a particular answer for input

- Operation, monitoring, and maintenance
  - monitor your performance on live data
  - nonstationarity—the world changes over time

Developing Machine Learning Systems

Tests for Machine Learning Infrastructure

(1) Training is reproducible. (2) Model specification code is unit tested. (3) The full ML pipeline is integration tested. (4) Model quality is validated before attempting to serve it.

(5) The model allows debugging by observing the step-by-step computation of training or inference on a single example. (6) Models are tested via a canary process before they enter production serving environments. (7) Models can be quickly and safely rolled back to a previous serving version.

Monitoring Tests for Machine Learning

(1) Dependency changes result in notification. (2) Data invariants hold in training and serving inputs. (3) Training and serving features compute the same values. (4) Models are not too stale. (5) The model is numerically stable. (6) The model has not experienced regressions in training speed, serving latency, throughput, or RAM usage. (7) The model has not experienced a regression in prediction quality on served data.

Developing Machine Learning Systems

Tests for Features and Data

(1) Feature expectations are captured in a schema. (2) All features are beneficial. (3) No feature's cost is too much. (4) Features adhere to meta-level requirements. (5) The data pipeline has appropriate privacy controls. (6) New features can be added quickly. (7) All input feature code is tested.

Tests for Model Development

(1) Every model specification undergoes a code review. (2) Every model is checked in to a repository. (3) Offline proxy metrics correlate with actual metrics (4) All hyperparameters have been tuned. (5) The impact of model staleness is known. (6) A simpler model is not better. (7) Model quality is sufficient on all important data slices. The model has been tested for considerations of inclusion.

Summary

- If the available feedback provides the correct answer for example inputs, then the learning problem is called supervised learning.
- Learning a function whose output is a continuous or ordered value (like weight) is called regression;
- Learning a function with a small number of possible output categories is called classification;
- Decision trees can represent all Boolean functions. The information-gain heuristic
- provides an efficient method for finding a simple, consistent decision tree.
- A linear classifier with a hard threshold—also known as a perceptron—can be trained by a simple weight update rule to fit data that are linearly separable.
- Logistic regression replaces the perceptron’s hard threshold with a soft threshold defined by a logistic function
- Nonparametric models use all the data to make each prediction, rather than trying to summarize the data with a few parameters
- Support vector machines find linear separators with maximum margin to improve the generalization performance of the classifier
- Ensemble methods such as bagging and boosting often perform better than individual methods.