AI基础

Lecture 12: Learning from Examples

Bin Yang
School of Data Science and Engineering
byang@dase.ecnu.edu.cn

[Some slides adapted from Philipp Koehn, JHU]

Lecture 12 ILOs

- Linear regression and classification
- Regularization
- Learning decision trees
- Ensemble learning

Outline

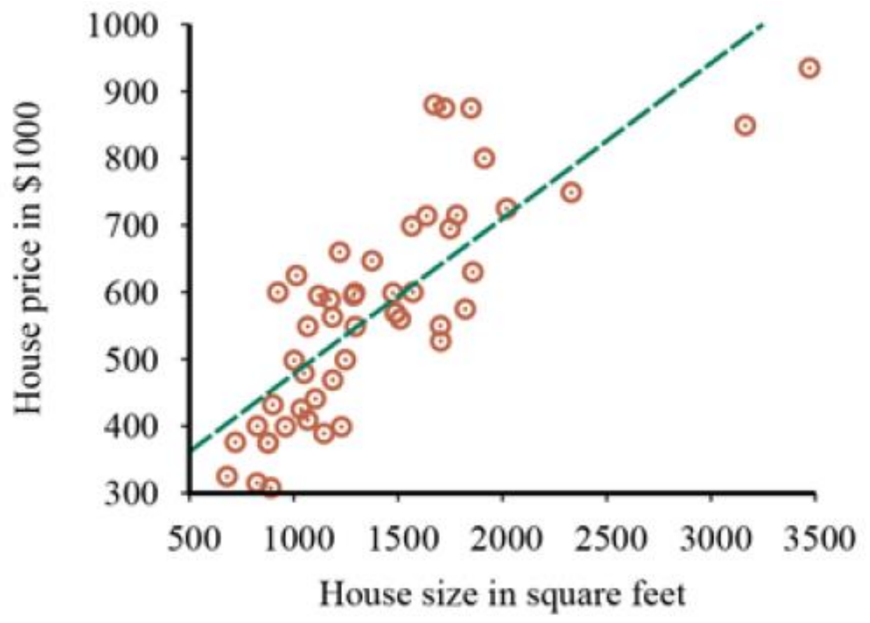
- Linear regression
  - Univaraite LR
  - Multivariate LR
- Regularization
  - L1, L2
- Linear classification
- Learning decision trees
- Ensemble learning

Univariate linear regression

- An example of a training set of n points in the x, y plane
  - each point representing the size in square feet x and the price of a house offered for sale y.
- A univariate linear function (a straight line) with input x and output y has the form

  $y=w_1x+w_0$ $h_{\mathbf{w}}(x) = w_1 x + w_0 .$

- The task of finding the h_w that best fits these data is called linear regression.

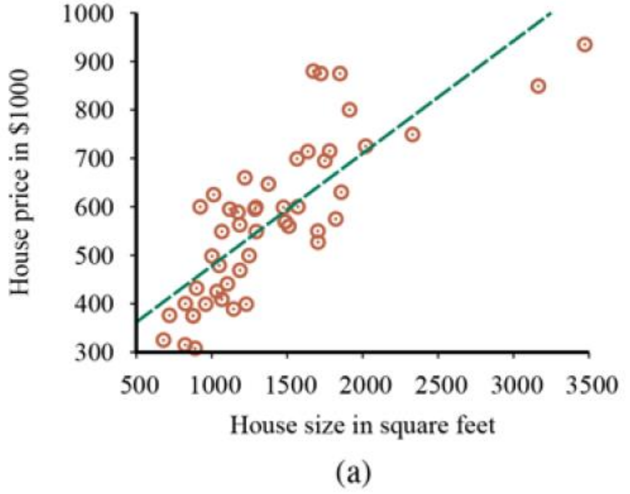


(a)

Univariate linear regression

- To fit a line to the data, all we have to do is find the values of the weights <w_0, w_1> that minimize the empirical loss.
    - Use the squared-error loss function, summed over all the training examples.

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



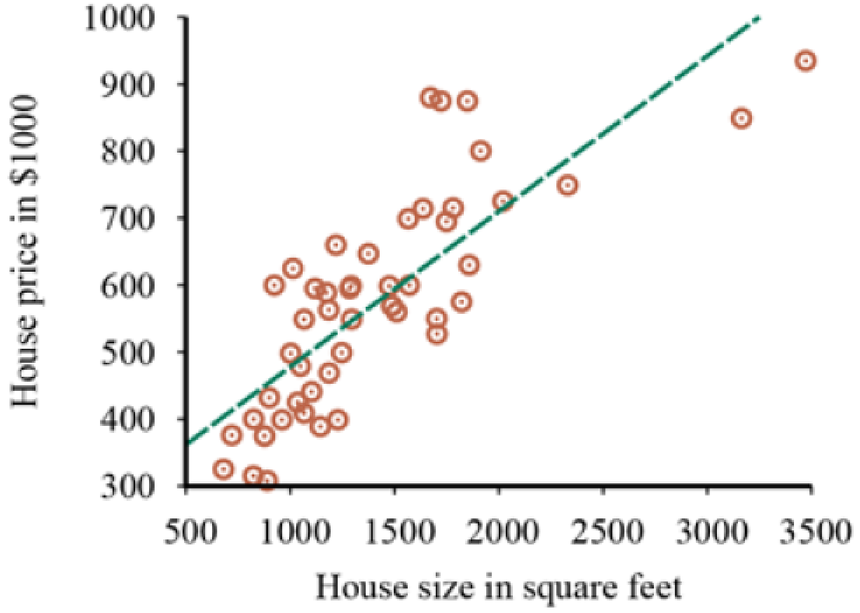
- To find the best **w** $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} Loss(h_{\mathbf{w}})$
- The loss is minimized when its partial derivatives with respect to w0 and w1 are zero:

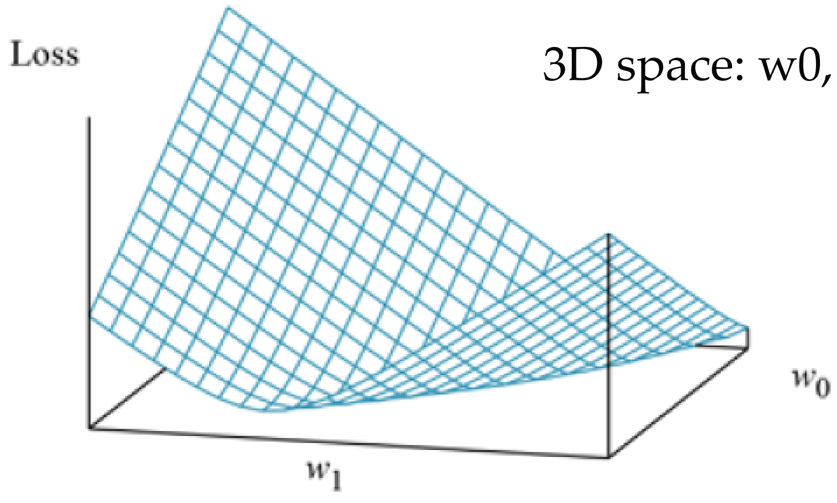
$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0.$$

Mini-quiz: can you get the optimal w0 and w1?

Univariate linear regression

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





Weight space—the space defined by all possible settings of the weights.

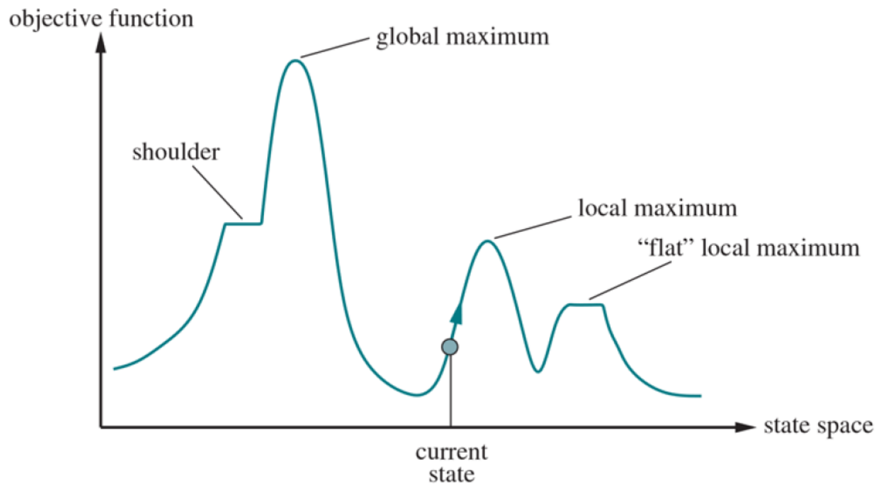
3D space: w0, w1, and loss.

Gradient descent

- The univariate linear model has the nice property that it is easy to find an optimal solution where the partial derivatives are zero.
- But that won’t always be the case, so we introduce here a method for minimizing loss that does not depend on solving to find zeroes of the derivatives, and can be applied to any loss function, no matter how complex.
- Hill climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
     neighbor ← a highest-valued successor state of current
     if VALUE(neighbor) $\le$ VALUE(current) then return current
     current ← neighbor
```

The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.



A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.

Gradient descent

- Hill climbing is for maximization. We go downhill to find minimization.
- We choose any starting point in weight space
  - here, a point in the (w0, w1) plane
- Compute an estimate of the gradient and move a small amount in the steepest downhill direction
- Repeating until we converge on a point in weight space with (local) minimum loss.
- α step size, learning rate

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

Chain rule

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

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

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

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

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

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

These updates make intuitive sense: if $h_{\mathbf{w}}(x) > y$ (i.e., the output is too large), reduce w_0 a bit, and reduce w_1 if x was a positive input but increase w_1 if x was a negative input.

Epoch, minibatch

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

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

- We have N training examples in total.
- Deterministic gradient descent, batch gradient descent
    - A step that cover all N training examples is called an epoch.
- Stochastic gradient descent
    - Selecting a minibatch of m out of the N examples
- How to choose m?
    - we can choose to take advantage of parallel vector operations, making a step with m examples almost as fast as a step with only a single example.
- SGD is widely applied to models other than linear regression, in particular neural networks.

Nonlinearity?

$\mathbf{z}' = (1, x_1, x_1^2, x_1^3)$

$\mathbf{z}' = (1, x_2, x_1 x_2, x_1^2)$

Multivariable linear regression

- $\mathbf{x}_j$ is an n-element vector
  - E.g., not only square feet, but also which floor, etc.

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

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

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

$$L(\mathbf{w}) = \| \hat{\mathbf{y}} - \mathbf{y} \|^2 = \| \mathbf{X}\mathbf{w} - \mathbf{y} \|^2.$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0.$$

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

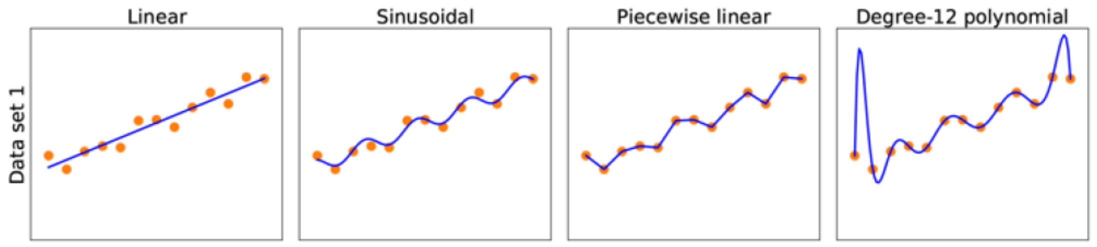
$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}.$$

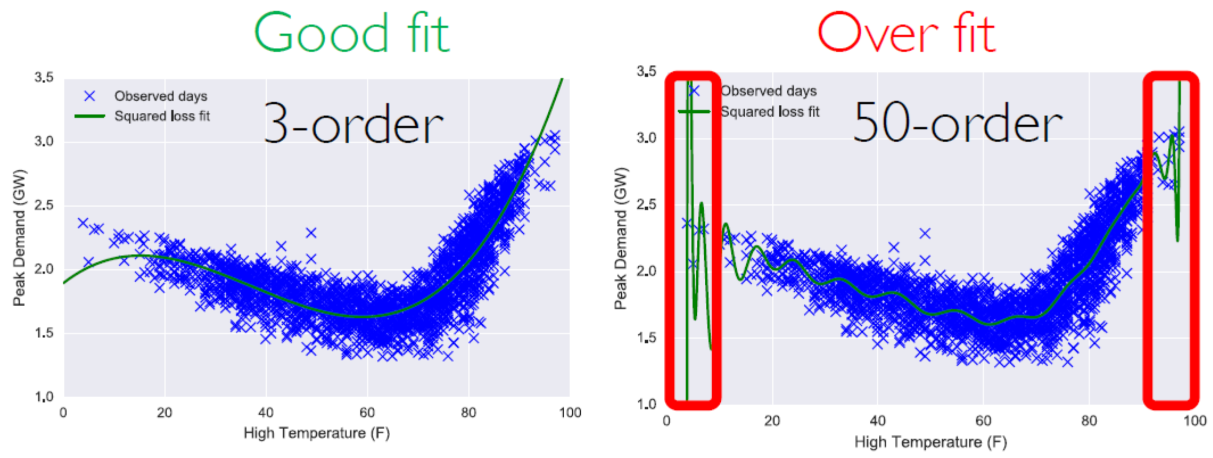
Outline

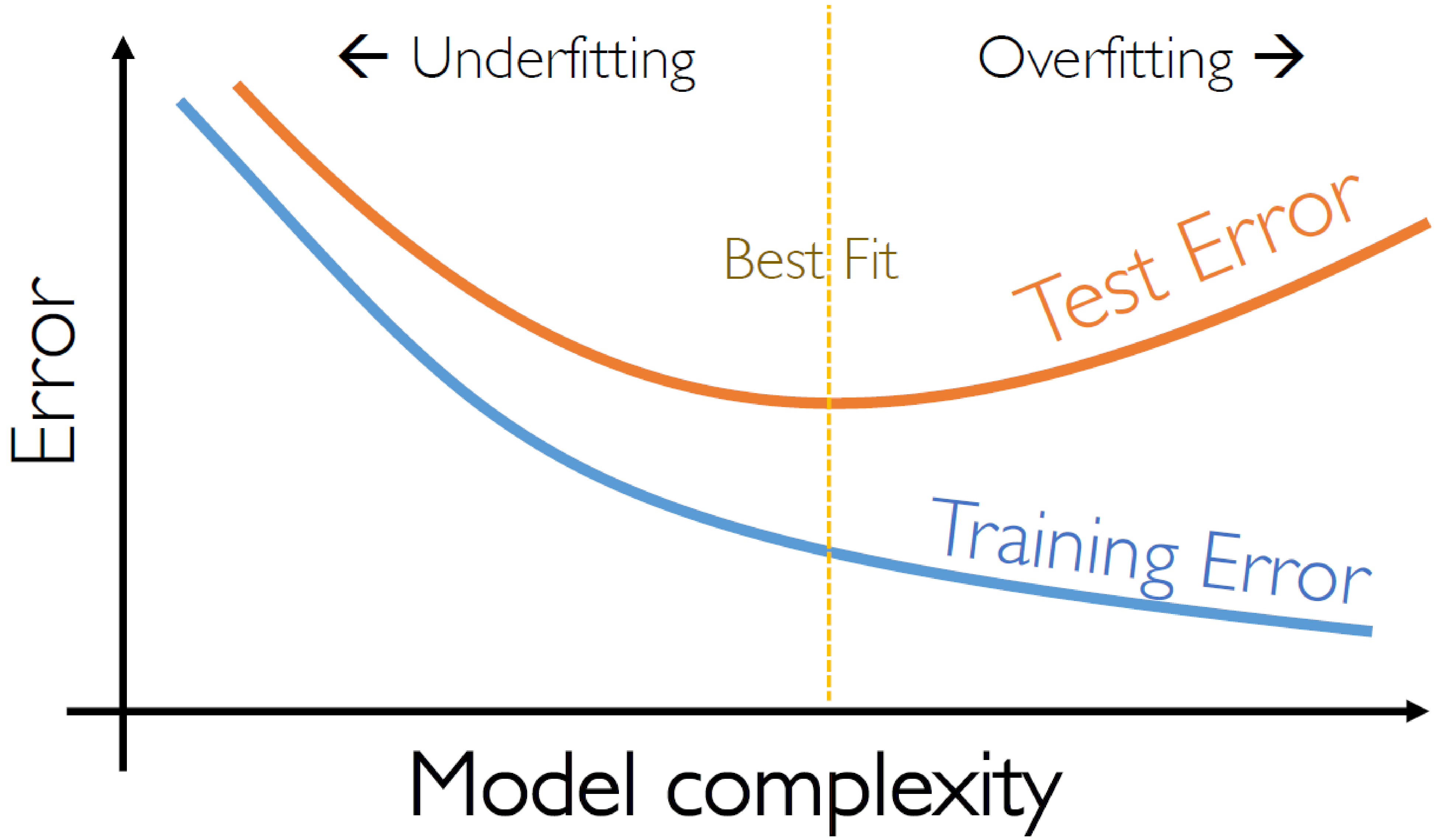
- Linear regression
  - Univaraite LR
  - Multivariate LR
- Regularization
  - L1, L2
- Linear classification
- Learning decision trees
- Ensemble learning

Regularization

- With univariate linear regression we didn't have to worry about overfitting. But with multivariable linear regression in high-dimensional spaces it is possible that some dimension that is actually irrelevant appears by chance to be useful, resulting in **overfitting**.






← Underfitting
Overfitting →
Best Fit
Test Error
Training Error
Error
Model complexity

Regularization

- It is common to use regularization on multivariable linear functions to avoid overfitting.
- With regularization, we minimize the total cost of a hypothesis, counting both the empirical loss and the complexity of the hypothesis.

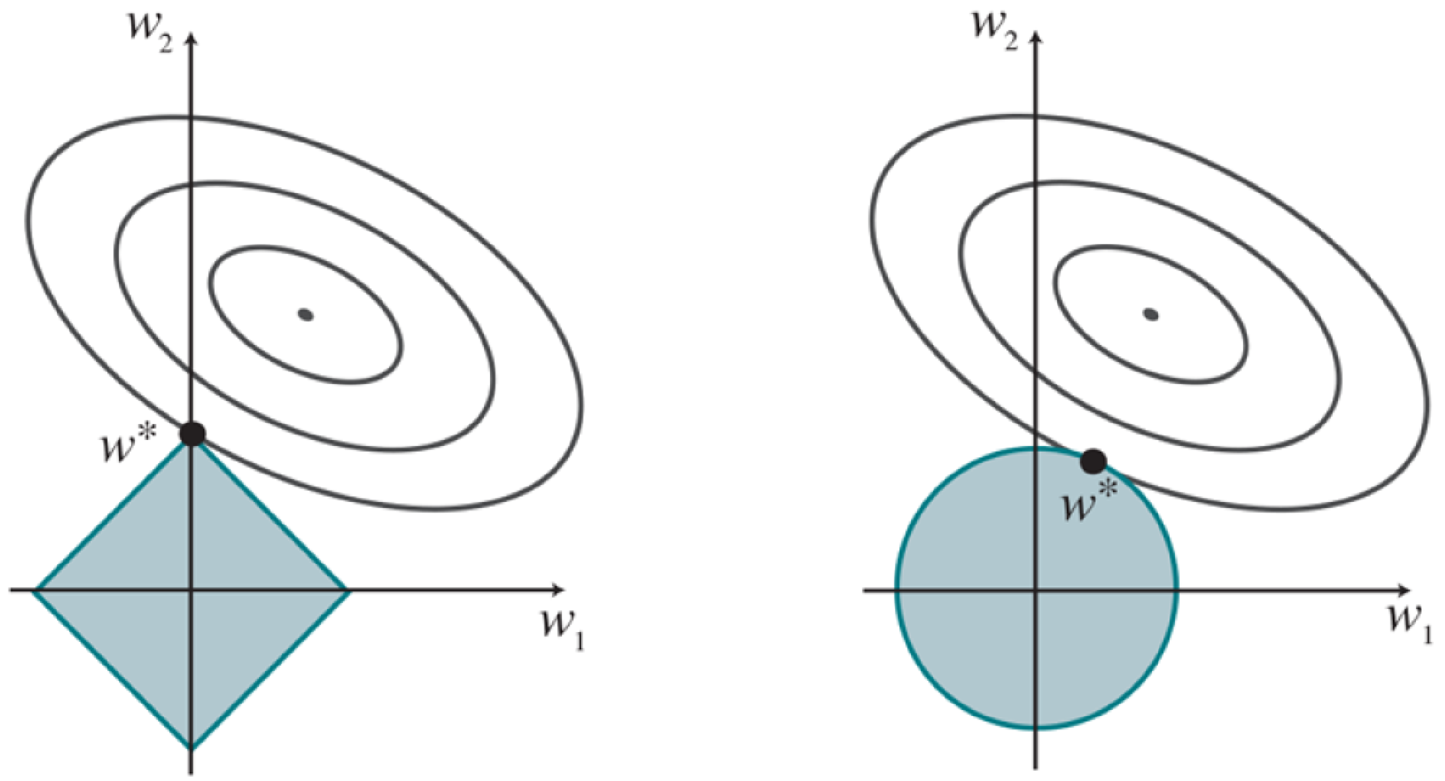
$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h).$$

$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q.$$

- L1: sum of the absolute values of the weights.
  - Tends to produce a sparse model, e.g., with many w_i being 0.
  - Declaring the corresponding attributes to be completely irrelevant
  - Can be easier for a human to understand, and may be less likely to overfit.
- L2: sum of the squares of the weights.

$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h).$$

- Minimizing the above cost function is equivalent to minimizing EmpLoss(w) while satisfying the constraint Complexity(h)<=c where c is a constant.
- L1: the constraint is a box.
- L2: the constraint is a circle.

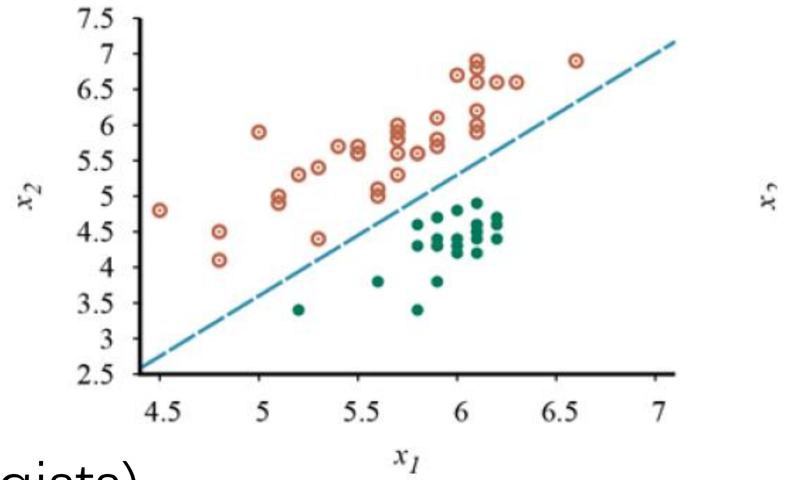


Why L_1 regularization tends to produce a sparse model. Left: With L_1 regularization (box), the minimal achievable loss (concentric contours) often occurs on an axis, meaning a weight of zero. Right: With L_2 regularization (circle), the minimal loss is likely to occur anywhere on the circle, giving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

Outline

- Linear regression
    - Univaraite LR
    - Multivariate LR
- Regularization
    - L1, L2
- Linear classification
- Learning decision trees
- Ensemble learning

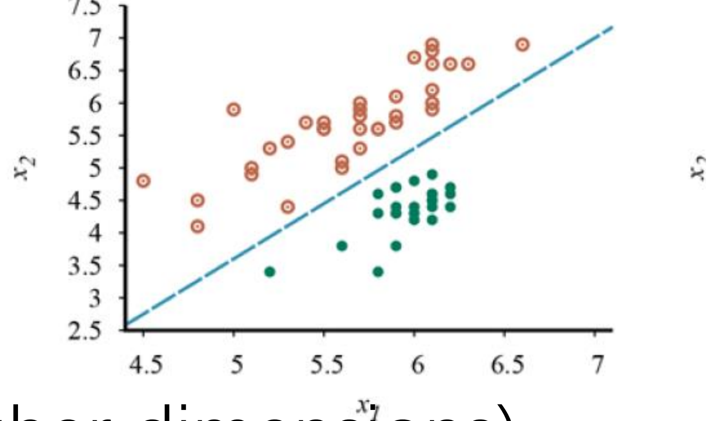
Linear classification



- Points of two classes:
  - earthquakes (which are of interest to seismologists)
  - underground explosions (which are of interest to arms control experts).
- Each point is defined by two input values, and that refer to body and surface wave magnitudes computed from the seismic signal.
- Given these training data, the task of classification is to learn a hypothesis that will take new points and return either 0 for earthquakes or 1 for explosions.

Decision boundary

The decision boundary is a straight line.



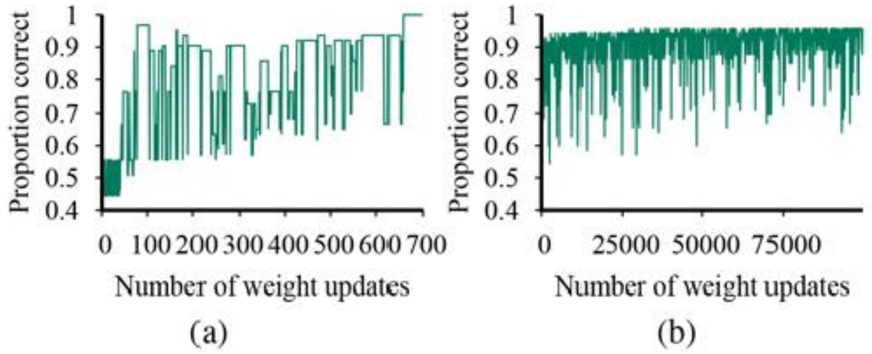
- A decision boundary is a line (or a surface, in higher dimensions) that separates the two classes.

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ if } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ and } 0 \text{ otherwise.}$$

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

- A linear decision boundary is called a **linear separator** and data that admit such a separator are called **linearly separable**.

A simple weight update rule



- A simple weight update rule that converges to a solution—that is, to a linear separator that classifies the data perfectly—provided the data are linearly separable

linearly separable
vs.
noisy, non linearly separable

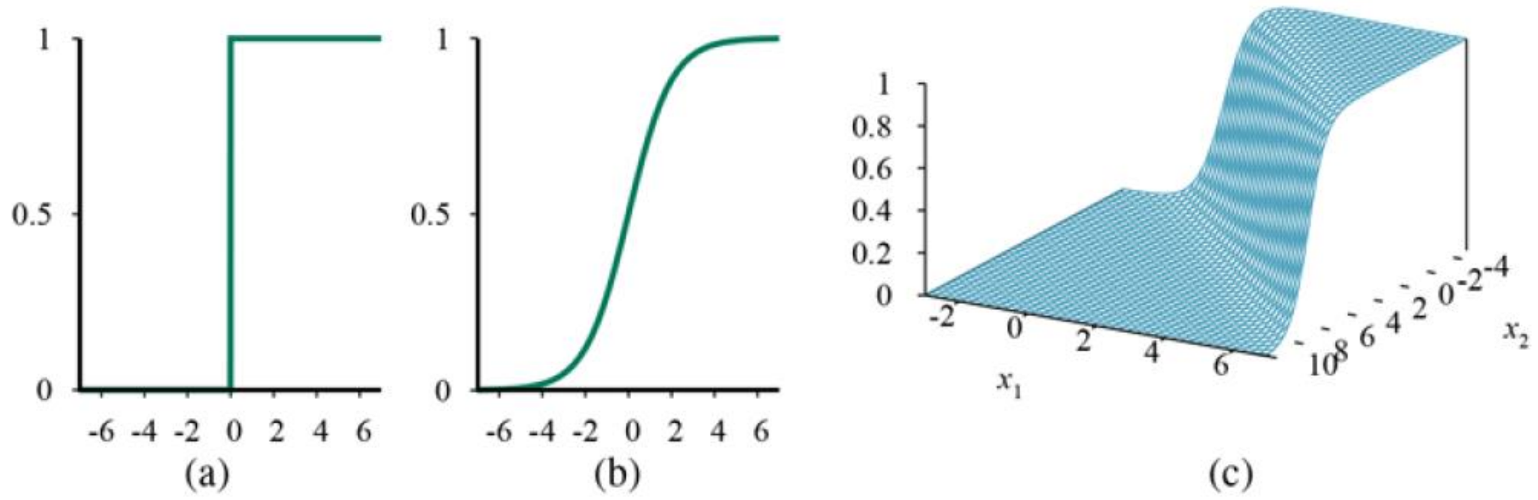
$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

- If the output is correct (i.e., $y = h_{\mathbf{w}}(\mathbf{x})$) then the weights are not changed.
- If y is 1 but $h_{\mathbf{w}}(\mathbf{x})$ is 0, then w_i is *increased* when the corresponding input x_i is positive and *decreased* when x_i is negative. This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ bigger so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 1.
- If y is 0 but $h_{\mathbf{w}}(\mathbf{x})$ is 1, then w_i is *decreased* when the corresponding input x_i is positive and *increased* when x_i is negative. This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ smaller so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 0.

Problems of linear classification

- The hypothesis $h_w(x)$ is not differentiable

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \text{ if } \mathbf{w} \cdot \mathbf{x} \geq 0 \text{ and } 0 \text{ otherwise.}$$



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

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

Logistic regression

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

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

- Because our hypotheses no longer output just 0 or 1, we will use the L2 loss function.

$$\begin{aligned}\frac{\partial}{\partial w_i}Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2\\ &= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))\\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w}\cdot\mathbf{x}\\ &= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w}\cdot\mathbf{x}) \times x_i.\end{aligned}$$

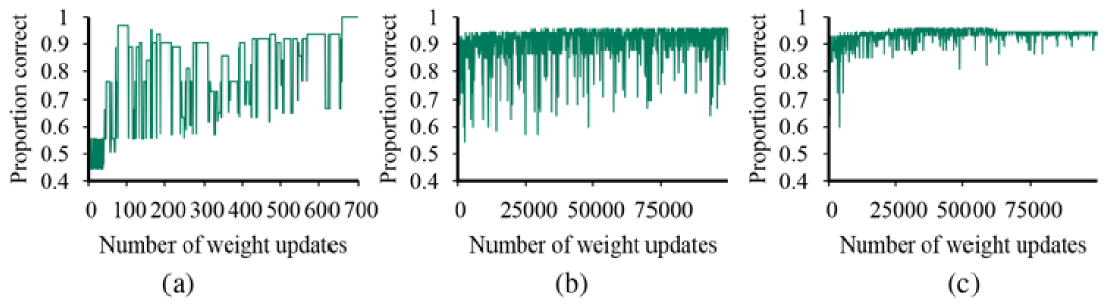
logistic function g has the following property:

$$g'(z) = g(z)(1-g(z))$$

$$g'(\mathbf{w}\cdot\mathbf{x}) = g(\mathbf{w}\cdot\mathbf{x})(1-g(\mathbf{w}\cdot\mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1-h_{\mathbf{w}}(\mathbf{x}))$$

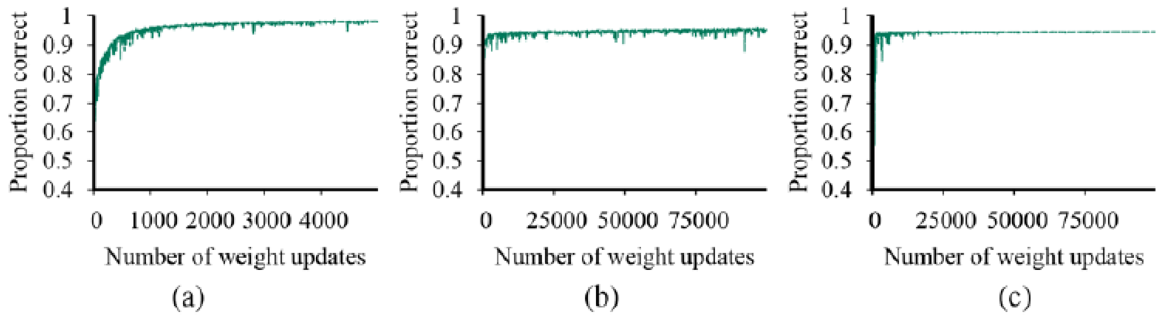
$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1-h_{\mathbf{w}}(\mathbf{x})) \times x_i.$$

Figure 19.16



(a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 19.15(a). (b) The same plot for the noisy, nonseparable data in Figure 19.15(b); note the change in scale of the x-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000 + t)$.

Figure 19.18



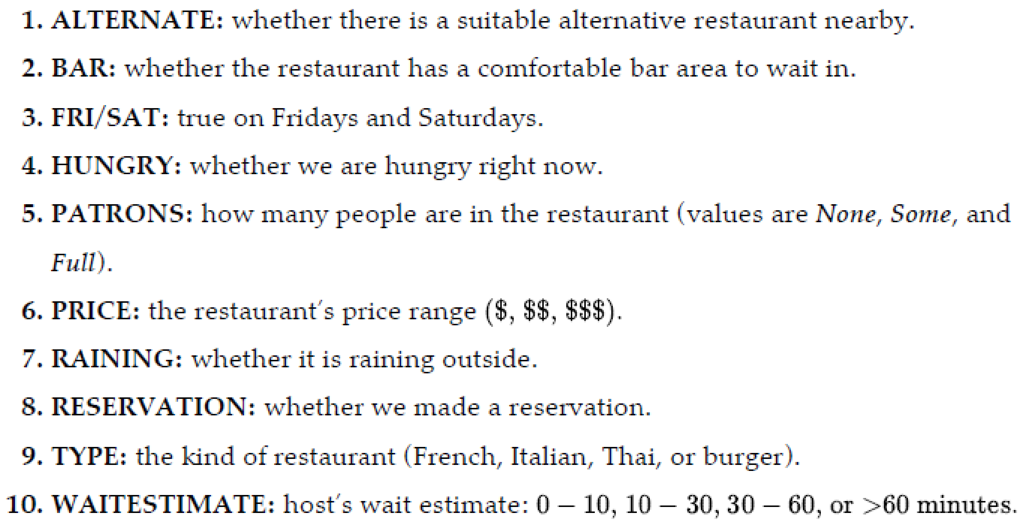
In (a), the linearly separable case, logistic regression is somewhat slower to converge, but behaves much more predictably. In (b) and (c), where the data are noisy and nonseparable, logistic regression converges far more quickly and reliably. These advantages tend to carry over into real-world applications, and logistic regression has become one of the most popular classification techniques for problems in medicine, marketing, survey analysis, credit scoring, public health, and other applications.

Outline

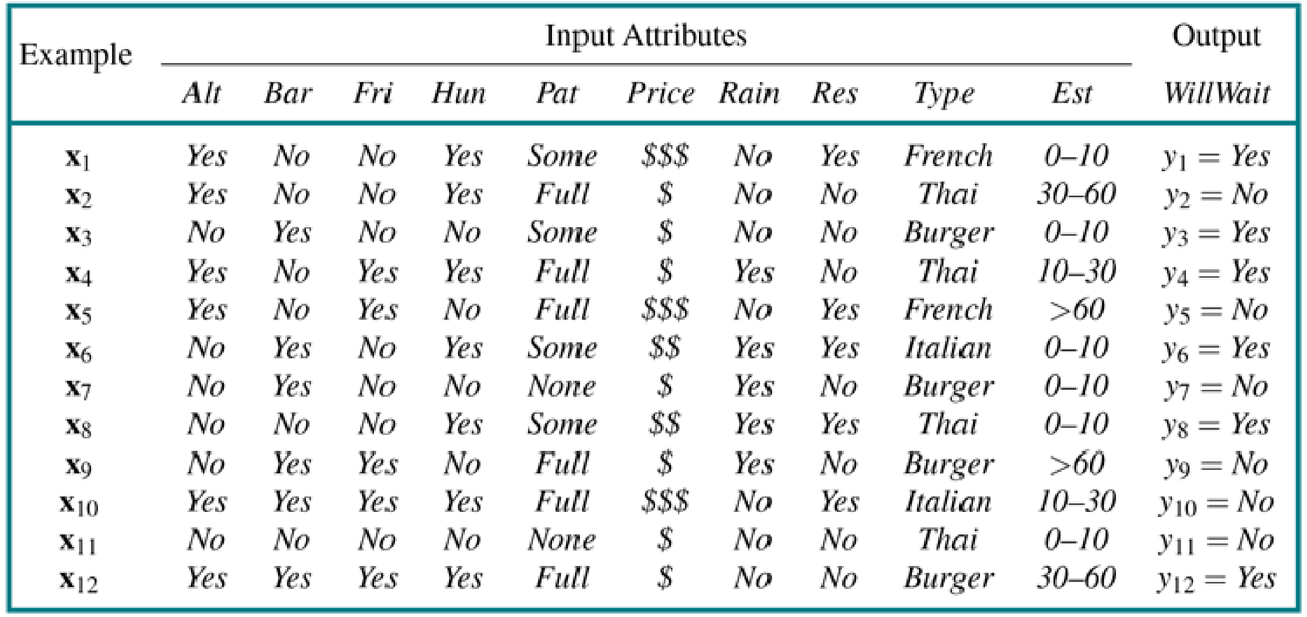
- Linear regression
    - Univaraite LR
    - Multivariate LR
- Regularization
    - L1, L2
- Linear classification
- Learning decision trees
- Ensemble learning

Learning decision trees

- Example problem: restaurant waiting
  - Whether to wait for a table at a restaurant
  - Output a Boolean variable WillWait, yes or no.



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 >60 minutes.



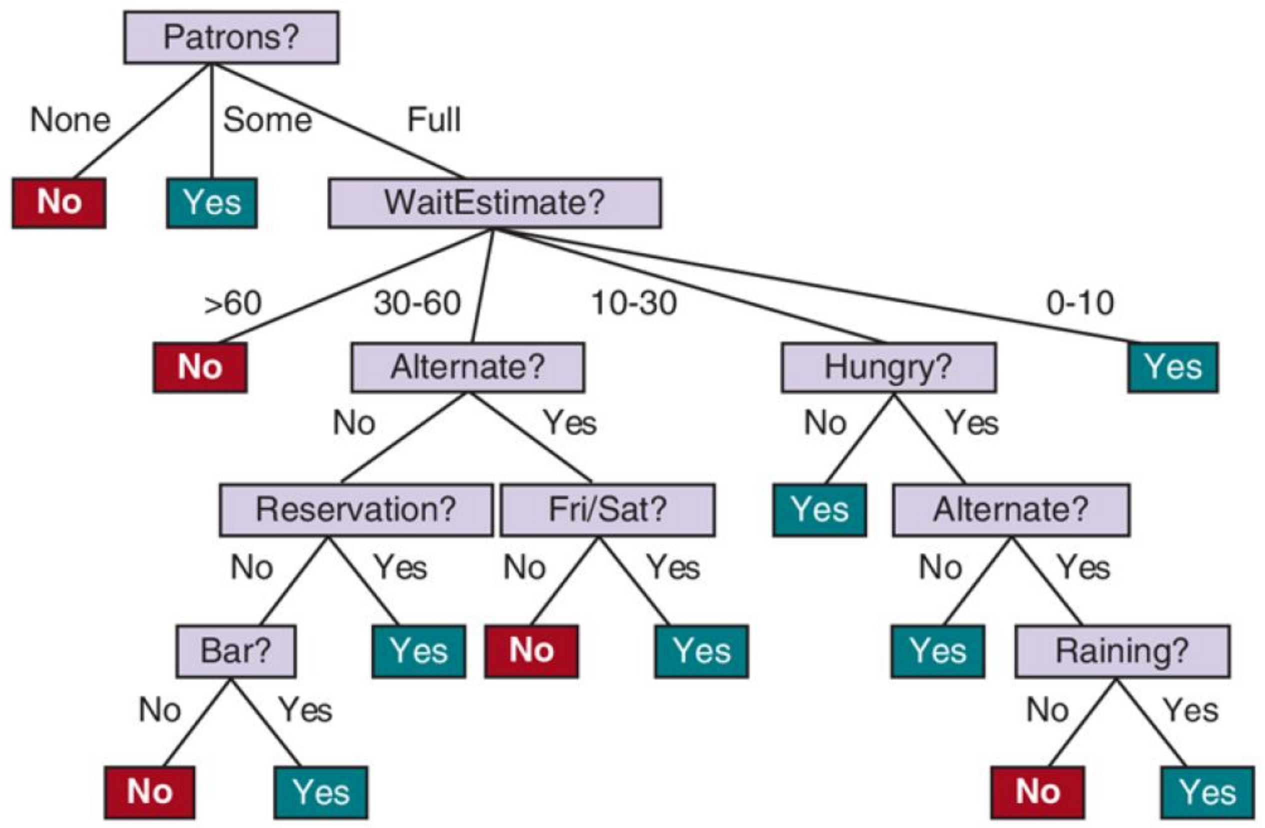
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*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*	$y_{12} = Yes$

$$2^6 \times 3^2 \times 4^2 = 9,216$$

12

Use the 12 known examples to estimate the WillWait for the remaining 9204 outputs.

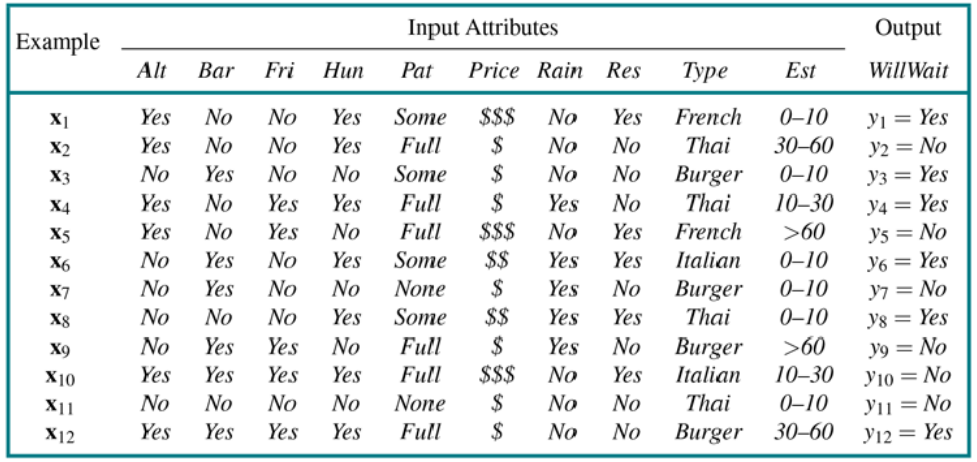
Figure 19.3



A decision tree for deciding whether to wait for a table.

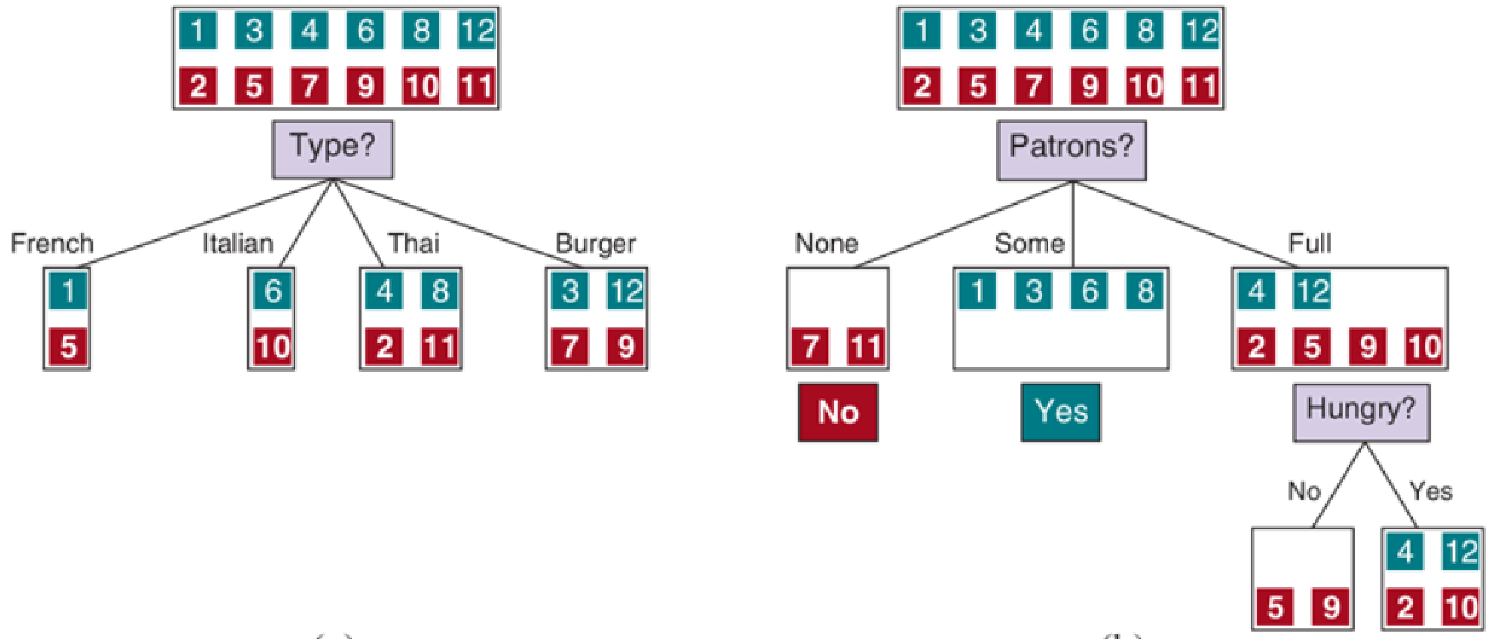
Mini-quiz:

- Is a tall tree or a short tree better?
- Explainability?
- Which attribute should be chosen first?



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*	$y_{12} = Yes$

Is Type a good attribute?
Is Patrons a good attribute?



(a) (b)

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.

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*	$y_{12} = Yes$

- 1. If the remaining examples are all positive (or all negative), then we are done: we can answer Yes or No.
  - This figure shows examples of this happening in the None and Some branches.
- 2. If there are some positive and some negative examples, then choose the **best** attribute to split them.
  - This figure shows Hungry being used to split the remaining examples.
- 3. If there are no examples left, it means that no example has been observed for this combination of attribute values, and we return the most common output value from the set of examples that were used in constructing the node’s parent.
- 4. If there are no attributes left, but both positive and negative examples, it means that these examples have exactly the same description, but different classifications. This can happen
  - because there is an error or noise in the data;
  - because the domain is nondeterministic;
  - because we can’t observe an attribute that would distinguish the examples.

  The best we can do is return the most common output value of the remaining examples.

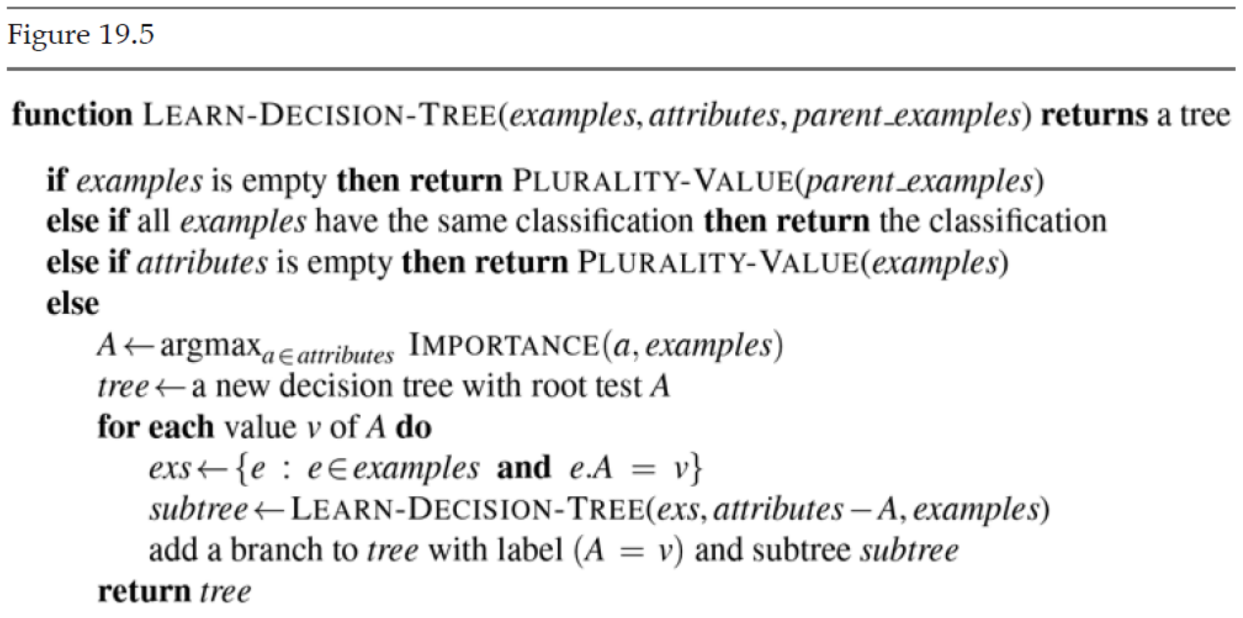
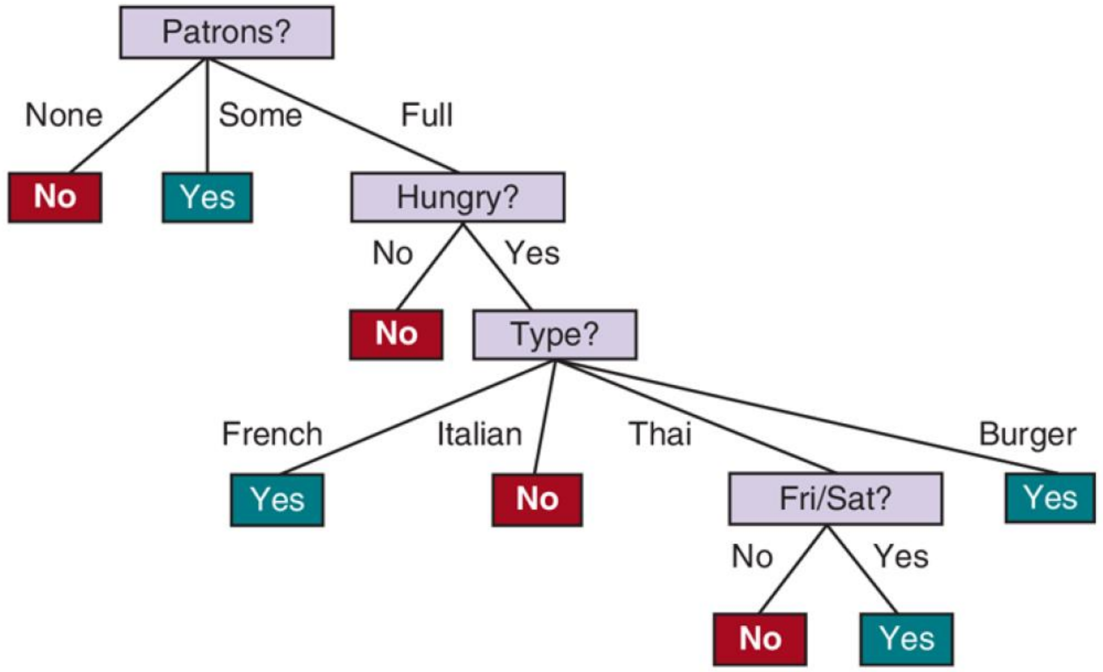


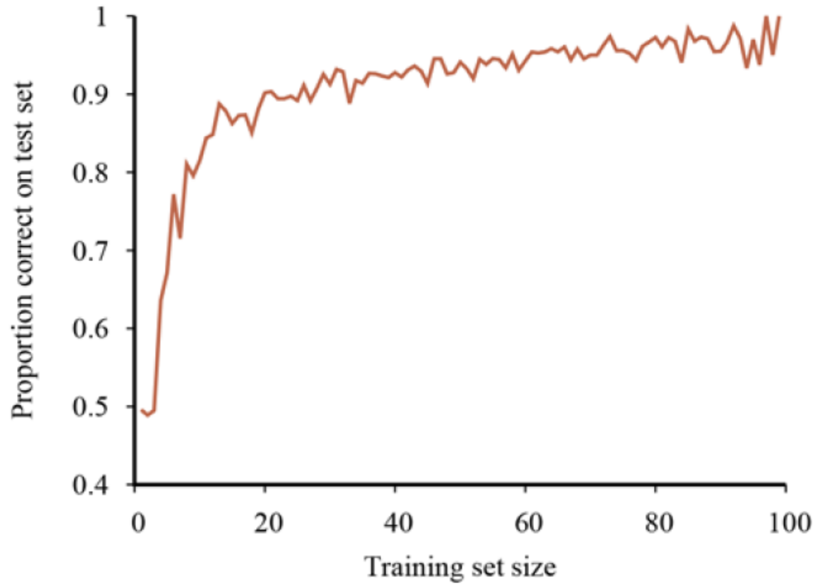
Figure 19.5

```
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
      $A \leftarrow \mathrm{argmax}_{a \in attributes}$ 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
```



Learning curve

- we have 100 examples at our disposal, which we split randomly into a training set and a test set.
- We learn a hypothesis with the training set and measure its accuracy with the test set.
- We can do this starting with a training set of size 1 and increasing one at a time up to size 99.



A 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.

Choosing attribute tests

- How to choose the best/most important attribute to split?
- How to measure importance, using the notion of **information gain**, which is defined in terms of **entropy**, which is the fundamental quantity in **information theory**.
- Entropy is a measure of the uncertainty of a random variable.
- The more information (more certainty), the less entropy.
  - A random variable with only one possible value—a coin that always comes up heads—has no uncertainty and thus its entropy is defined as zero.

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

Entropy examples

- A fair coin: tail and head each 50%

$$H(Fair) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1\ .$$

- A fair four-side die: each side has 25%

$$H(Die4) = -(0.25 \log_2 0.25 + 0.25 \log_2 0.25 + 0.25 \log_2 0.25 + 0.25 \log_2 0.25) = 2$$

- An unfair coin: 99% heads

$$H(Loaded) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08 \text{ bits.}$$

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

Entropy of a Boolean random variable

- Define B(q) as the entropy of a Boolean random variable that is true with probability q
    - WillWait in our example is a Boolean random varaible

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q)).$$

- If a training set contains p positive examples and n negative examples, then the entropy of the output variable on the whole set is

$$H(Output) = B\left(\frac{p}{p+n}\right).$$

- B(WillWait)=B(6/6+6)=1

An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$. Each subset E_k has p_k positive examples and n_k negative examples, so if we go along that branch, we will need an additional $B(p_k/(p_k+n_k))$ bits of information to answer the question. A randomly chosen example from the training set has the kth value for the attribute (i.e., is in E_k with probability $(p_k+n_k)/(p+n)$), so the expected entropy remaining after testing attribute A is

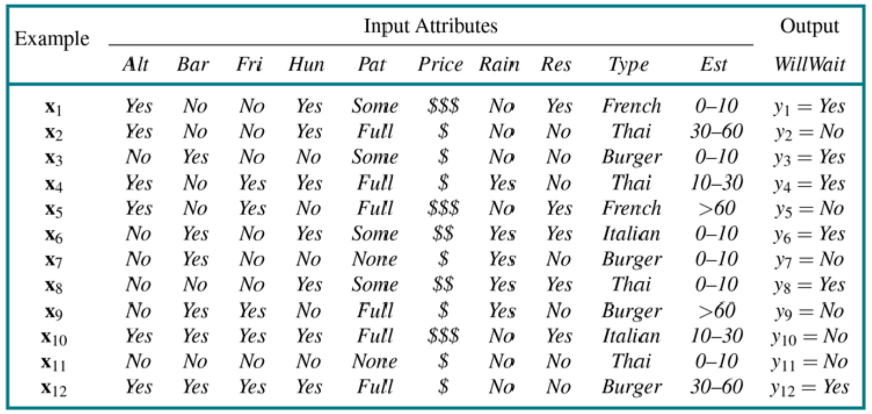
$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right).$$

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

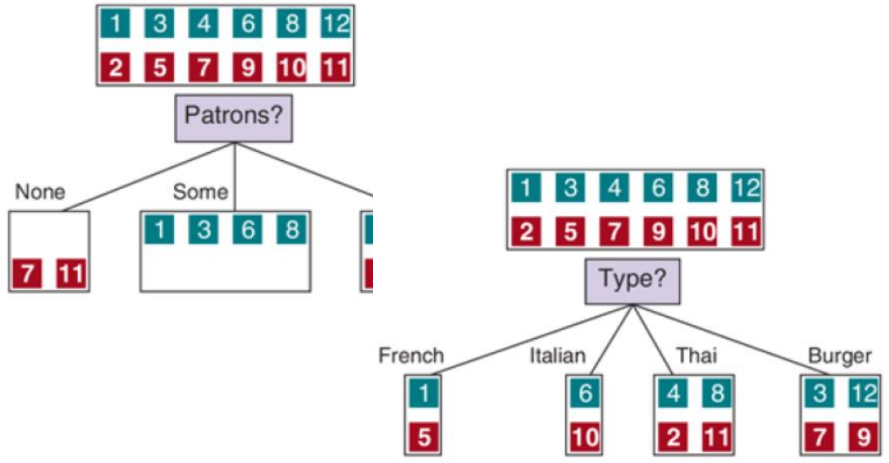
$$Gain(A) = B\left(\frac{p}{p+n}\right) - Remainder(A).$$

In fact $Gain(A)$ is just what we need to implement the IMPORTANCE function. Returning to the attributes considered in Figure 19.4, we have

$$Gain(Patrons) = 1 - \left[\tfrac{2}{12}B\left(\tfrac{0}{2}\right) + \tfrac{4}{12}B\left(\tfrac{4}{4}\right) + \tfrac{6}{12}B\left(\tfrac{2}{6}\right)\right] \approx 0.541 \text{ bits},$$
$$Gain(Type) = 1 - \left[\tfrac{2}{12}B\left(\tfrac{1}{2}\right) + \tfrac{2}{12}B\left(\tfrac{1}{2}\right) + \tfrac{4}{12}B\left(\tfrac{2}{4}\right) + \tfrac{4}{12}B\left(\tfrac{2}{4}\right)\right] = 0 \text{ bits},$$



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	$y_{12} = Yes$



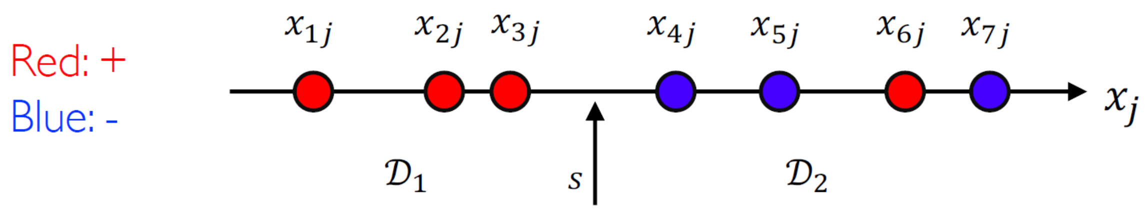
How to choose the **best/most important** attribute to split?

The attribute with the largest Gain(A).

Overfitting

- Overfitting becomes more likely as the number of attributes grows, and less likely as we increase the number of training examples.
- Larger hypothesis spaces (e.g., decision trees with more nodes or polynomials with high degree) have more capacity both to fit and to overfit.
- For decision trees, a technique called decision **tree pruning** combats overfitting.
  - Pruning works by eliminating nodes that are not clearly relevant.
  - We start with a full tree, as generated by LEARN-DECISION-TREE.
  - We then look at a test node, if a test node is testing an irrelevant attribute, we eliminate the test.
    - Suppose we are at a node consisting of p positive and n negative examples. If the attribute is irrelevant, we would expect that it would split the examples into subsets such that each subset has **roughly the same proportion** of positive examples as the whole set, and so the information gain will be **close to zero**.
- **Early stopping**, do not generate the nodes that will be pruned later.
  - stop generating nodes when there is no good attribute to split on.

Broadening the applicability of decision trees



- Continuous input attributes
  - Examples: Height, Weight, or Time. It may be that every example has a different attribute value.
  - The information gain measure would give its highest score to such an attribute, giving us a shallow tree with this attribute at the root, and single-example subtrees for each possible value below it.
  - But that doesn't help when we get a new example to classify with an attribute value that we haven't seen before.
- A better way is to use a **split point test**
  - Weight>160 vs Weight<=160
- A large number of possible values without order: **equality test**
  - Zipcode=10002 vs. Zipcode is other

Broadening the applicability of decision trees

- Continuous-valued output attribute
  - We need a regression tree rather than a classification tree
- A regression tree has at each leaf a linear function of some subset of numerical attributes.
- For example, the branch for two-bedroom apartments might end with a linear function of square footage and number of bathrooms.
- The learning algorithm must decide when to stop splitting and begin applying linear regression over the attributes.
- **CART**, standing for Classification And Regression Trees, is used to cover both classes.

Outline

- Linear regression
    - Univaraite LR
    - Multivariate LR
- Regularization
    - L1, L2
- Linear classification
- Learning decision trees
- Ensemble learning

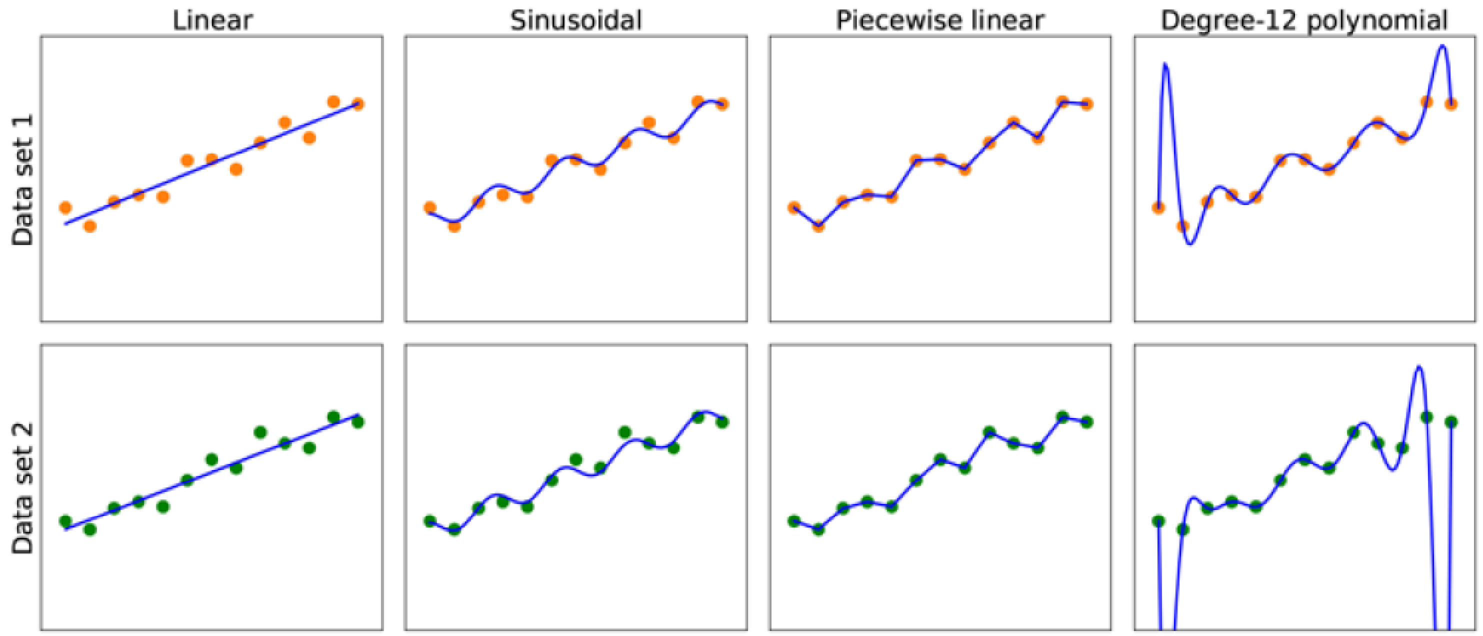
Ensemble learning

- So far we have looked at learning methods in which a single hypothesis is used to make predictions.
- The idea of ensemble learning is to select a collection, or ensemble, of hypotheses, and combine their predictions by averaging, voting, or by another level of machine learning.
- We call the individual hypotheses **base models** and their combination an **ensemble model**.

Benefits of using ensemble learning

- Bias vs. variance

Figure 19.1



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).

Benefits of using ensemble learning

- The first is to reduce bias.
  - The hypothesis space of a base model may be too restrictive, imposing a strong bias (such as the bias of a linear decision boundary in logistic regression).
  - An ensemble can be more expressive, and thus have less bias, than the base models.
  - An ensemble of n linear classifiers allows more functions to be realizable, at a cost of only n times more computation; this is often better than allowing a completely general hypothesis space that might require exponentially more computation.

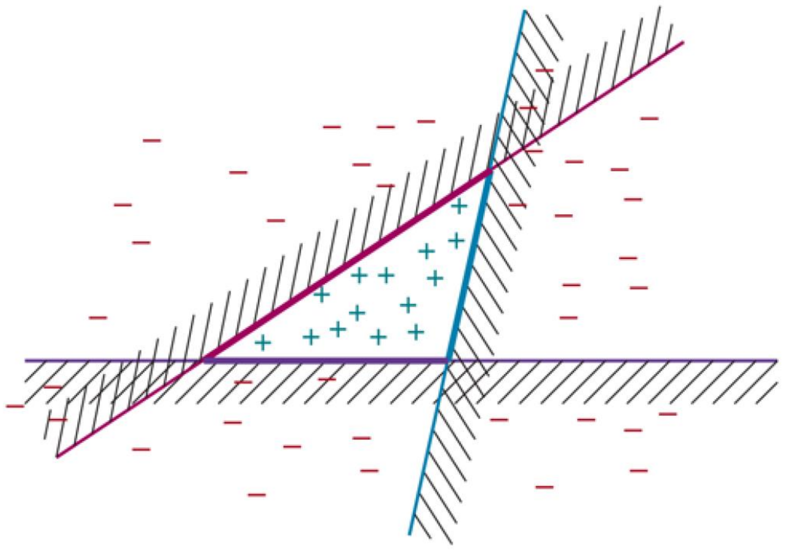


Illustration of the increased expressive power obtained by ensemble learning. We take three linear threshold hypotheses, each of which classifies positively on the unshaded side, and classify as positive any example classified positively by all three. The resulting triangular region is a hypothesis not expressible in the original hypothesis space.

Benefits of using ensemble learning

- The second reason is to reduce variance.
  - Consider an ensemble of K=5 binary classifiers that we combine using majority voting. For the ensemble to misclassify a new example, at least three of the five classifiers have to misclassify it. The hope is that this is less likely than a single misclassification by a single classifier.

- Four ways of creating ensembles:
  - bagging
  - random forests
  - stacking
  - boosting

Bagging

- We generate K distinct training sets by sampling with replacement from the original training set and then to get K base models.
    - We randomly pick N examples from the training set, but each of those picks might be an example we picked before. (This is to introduce some **diversity**.)
    - We then run our machine learning algorithm on the N examples to get a hypothesis.
- We repeat the above process K times, getting K different hypotheses (K base models).
- Then, when asked to predict the value of a new input, we aggregate the predictions from all 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: $h(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} h_i(\mathbf{x})$

When to use bagging

- Bagging tends to reduce variance and is a standard approach when there is limited data or when the base model is seen to be overfitting.
- Bagging can be applied to any class of model, but is most commonly used with decision trees. It is appropriate because decision trees are unstable: a slightly different set of examples can lead to a wildly different tree. Bagging smoothes out this variance.
- If you have access to multiple computers then bagging is efficient, because the hypotheses can be computed in parallel.

Random forests

- Unfortunately, bagging decision trees often ends up giving us trees that are highly correlated.
    - If there is one attribute with a very high information gain, it is likely to be the root of most of the trees.
- The random forest model is a form of decision tree bagging in which we take extra steps to make the ensemble of trees **more diverse**, to reduce variance.
- Random forests can be used for classification or regression.

Random forests

- The key idea is to randomly vary the attribute choices (rather than sampling the training examples as bagging does).
- 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.
- A further improvement is to use randomness in selecting the split point value:
    - for each selected attribute, we randomly sample several candidate values from a uniform distribution over the attribute’s range. Then we select the value that has the highest information gain.
    - That makes it more likely that every tree in the forest will be different. Trees constructed in this fashion are called extremely randomized trees (ExtraTrees).

Runtime

- Random forests are efficient to create.
- You might think that it would take times longer to create an ensemble of trees, but it is not that bad, for three reasons:
  - (a) each split point runs faster because we are considering fewer attributes,
  - (b) we can skip the pruning step for each individual tree, because the ensemble as a whole decreases overfitting,
  - (c) if we happen to have computers available, we can build all the trees in parallel.

Applications

- Random forests have been very successful across a wide variety of application problems.
- In Kaggle data science competitions they were the most popular approach of winning teams from 2011 through 2014, and remain a common approach to this day (although deep learning and gradient boosting have become even more common among recent winners).
- The randomForest package in R has been a particular favorite.
- In finance, random forests have been used for credit card default prediction, household income prediction, and option pricing.
- Mechanical applications include machine fault diagnosis and remote sensing.
- Bioinformatic and medical applications include diabetic retinopathy, microarray gene expression, mass spectrum protein expression analysis, biomarker discovery, and protein-protein interaction prediction.

Stacking

- Bagging combines multiple base models of the same model class trained on different data
  - Diversity comes from data.
- Stacked generalization (or stacking for short) combines multiple base models from different model classes trained on the same data.
  - Diversity comes from model classes.

$\mathbf{x}_1$=Yes, No, No, Yes, Some, \$\$\$, No, Yes, French, $0-10$; $y_1 =$ Yes

- Train, validation, test
- Training data: to create n different base models.
  - An SVM model, a logistic regression, and a decision tree model
- Validation data: use n different based models to make predictions.

$\mathbf{x}_2 =$ Yes, No, No, Yes, Full, \$, No, No, Thai, $30-60$, **Yes, No, No**; $y_2 =$ No

- Validation data + predictions: train a new model.

- The method is called "stacking" because it 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.
- In fact, it is possible to stack multiple layers, each one operating on the output of the previous layer.
- Stacking reduces bias, and usually leads to performance that is better than any of the individual base models.

Boosting

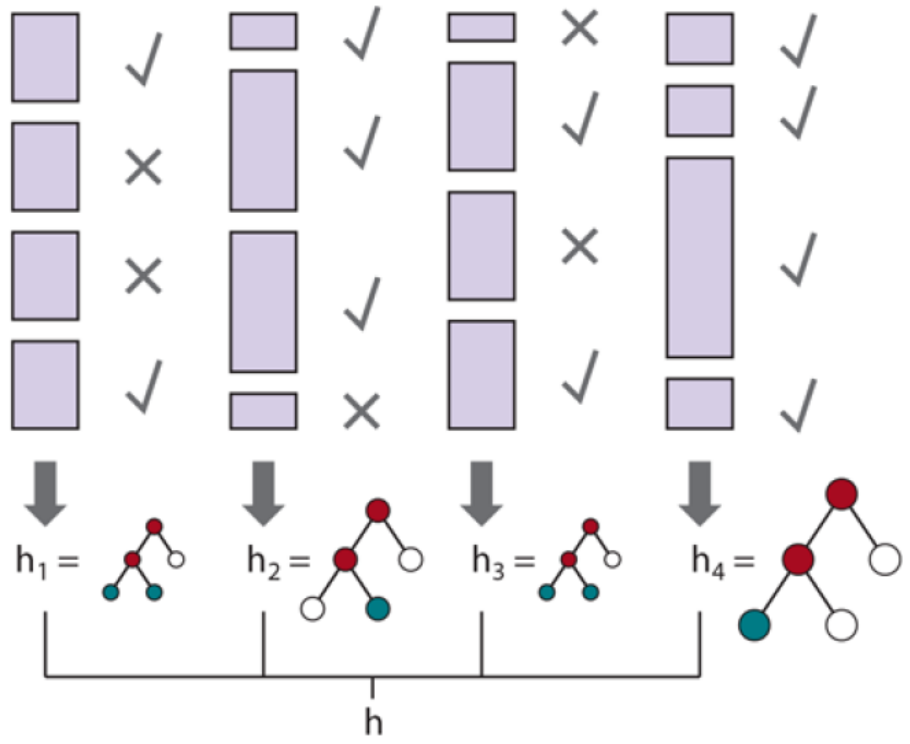
- Weighted training set
  - Each example has an associated weight w_j that describes how much the example should count during training.
  - For example, if one example had a weight of 3 and the other examples all had a weight of 1, that would be equivalent to having 3 copies of the one example in the training set.
- Boosting starts with equal weights $w_j = 1$ for all the examples.
- From this training set, it generates the first hypothesis h_1, In general, h_1 will classify some of the training examples correctly and some incorrectly.
- We would like the next hypothesis to do better on the misclassified examples, so we increase their weights while decreasing the weights of the correctly classified examples.

Boosting

- From this new weighted training set, we generate hypothesis h_2.
- The process continues in this way until we have generated K hypotheses, where K is an input to the boosting algorithm.
- Examples that are difficult to classify will get increasingly larger weights until the algorithm is forced to create a hypothesis that classifies them correctly.
  - Note that this is a greedy algorithm in the sense that it does not backtrack; once it has chosen a hypothesis it will never undo that choice; rather it will add new hypotheses.
  - It is also a sequential algorithm, so we can't compute all the hypotheses in parallel as we could with bagging.
- The final ensemble lets each hypothesis vote, as in bagging, except that each hypothesis gets a weighted number of votes

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

Figure 19.24



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.

AdaBoost

- It is usually applied with decision trees as the component hypotheses; often the trees are limited in size.
- AdaBoost has a very important property:
  - if the input learning algorithm is a weak learning algorithm—which means that always returns a hypothesis with accuracy on the training set that is slightly better than random guessing (that is, 50%+ϵ for Boolean classification)
  - then AdaBoost will return a hypothesis that classifies the training data perfectly for large enough K.
  - Thus, the algorithm **boosts** the accuracy of the original learning algorithm on the training data.

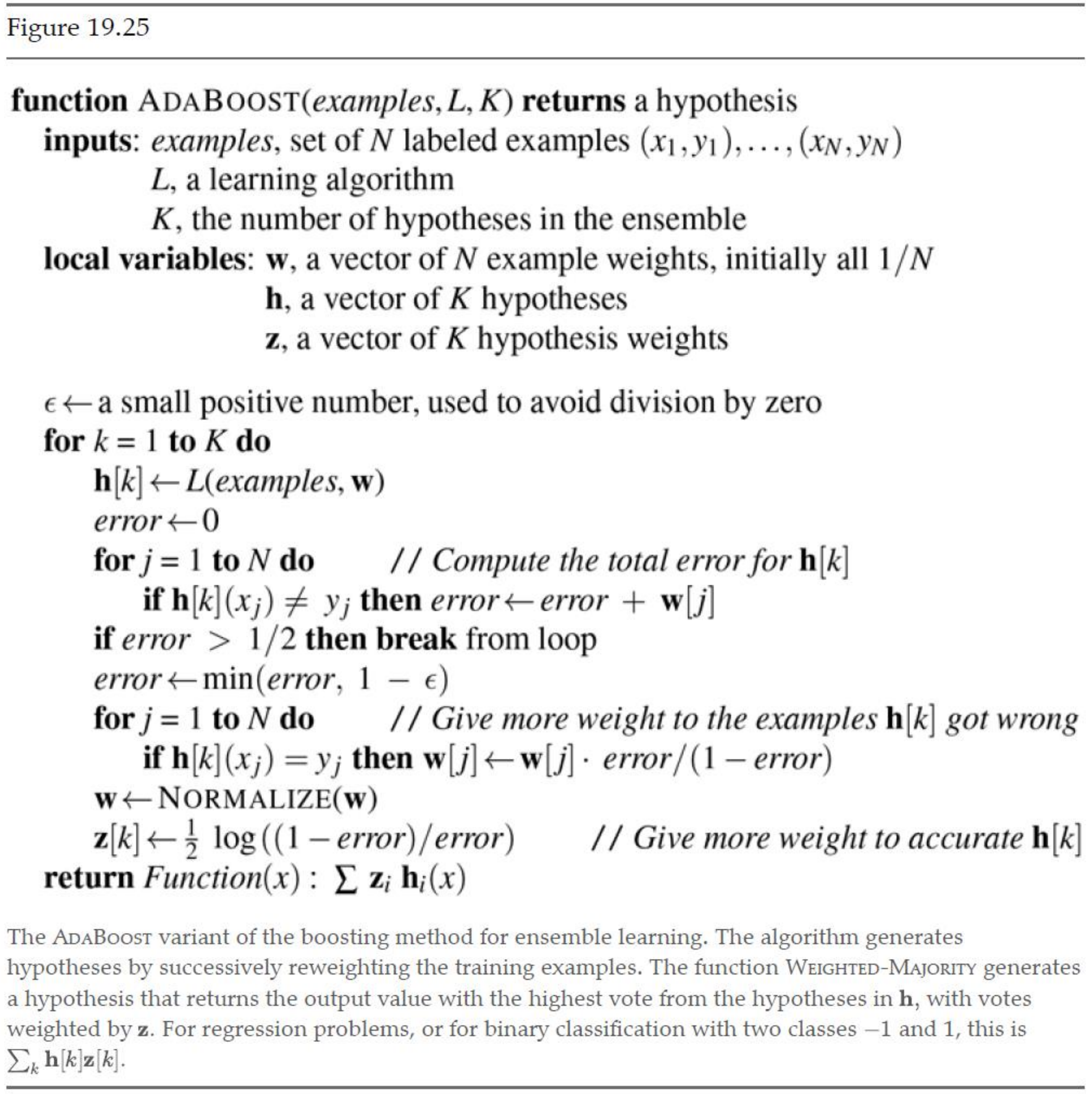
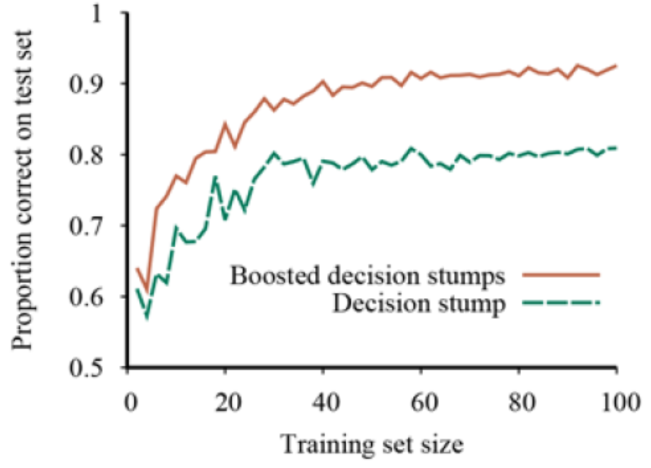


Figure 19.25

```
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: $\mathbf{w}$, a vector of $N$ example weights, initially all $1/N$
                   $\mathbf{h}$, a vector of $K$ hypotheses
                   $\mathbf{z}$, 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 $\mathbf{h}$, with votes weighted by $\mathbf{z}$. For regression problems, or for binary classification with two classes -1 and 1, this is $\sum_k \mathbf{h}[k]\mathbf{z}[k]$.

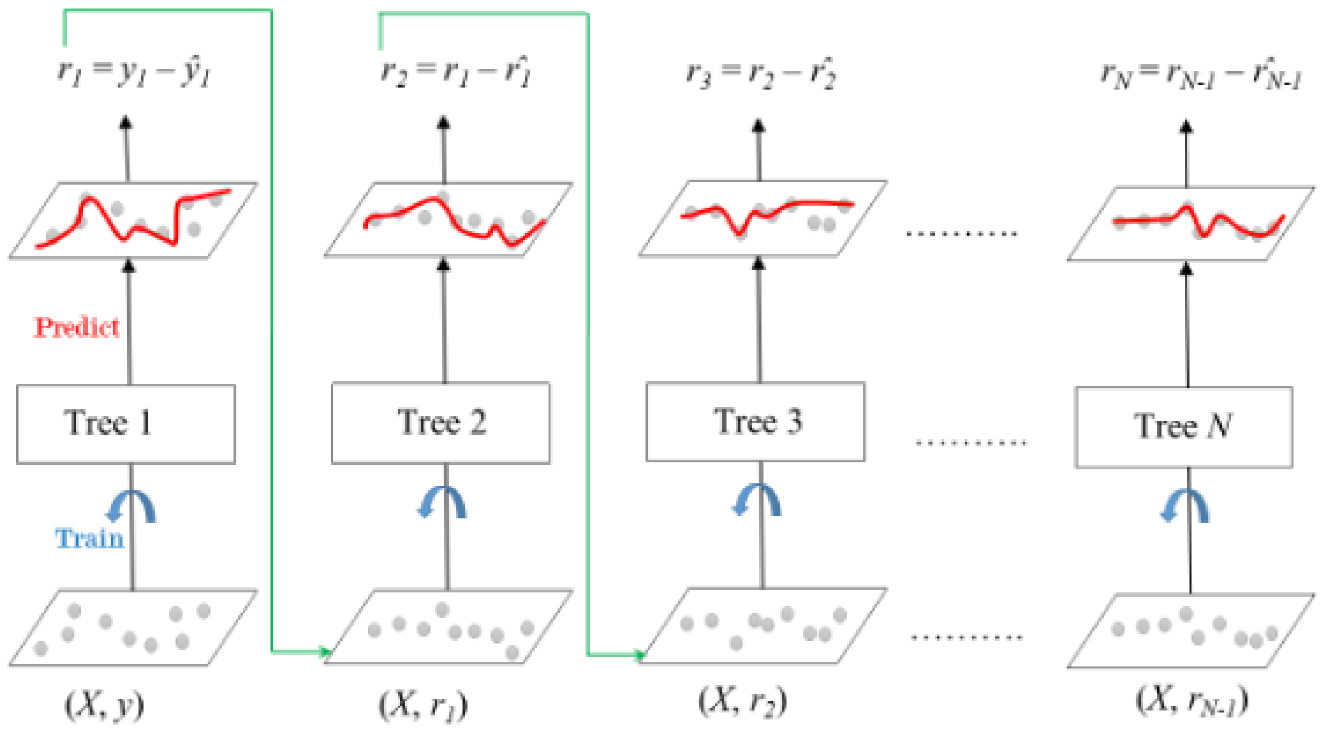


The original hypothesis space: **decision stumps**, which are decision trees with just one test, at the root.

Boosted decision stumps with K=5.

Gradient boosting

- As the name implies, gradient boosting is a form of boosting using gradient descent.
- Recall that in AdaBoost, we start with one hypothesis and boost it with a sequence of hypotheses that pay special attention to the examples that the previous ones got wrong.
- In gradient boosting we also add new boosting hypotheses, which pay attention not to specific examples, but to the gradient between the right answers and the answers given by the previous hypotheses.



Comparisons

	Bagging	Random Forest	Stacking	Boosting
How to introduce diversity	Different subsets of the training data Base models are of the same class.	Different trees. Base models are in the same category, but with different structures.	Different base models.	Sequential, different learning targets (wrongly classified examples, or the residual of the previous predictions)
Parallelism	Easy. Independent base models.	Easy. Independent base models.	Have multiple stacks. At the same stack, yes. But stack by stack is sequential.	Sequential. Next base model is based on the previous base models.

Lecture 12 ILOs

- Linear regression and classification
  - Two different ways to optimize the parameters
  - Classification, logistic regression
- Regularization
  - L1, L2
  - Pruning and early stopping in decision trees
- Learning decision trees
  - How to select the best attribute to split, information gain
- Ensemble learning
  - Bagging, random forest, stacking, boosting