Lecture 14: Machine Learning (Decision Trees and Linear Classifiers)

CS 486/686: Introduction to Artificial Intelligence

Plan for Today

- Supervised Learning
  - Decision Trees
  - Linear Regression and classification

Example

- Question:
  - Should you wait for a table at a restaurant?
- Input:
  - a set of input vectors, x_i, of 10 discrete attribute values
- Output:
  - a Boolean variable, Wait, indicating whether you should wait or not

Data

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Decision Trees

- Recall that for a learning problem, we need to decide on the representation for H
  - Decision Trees: representation of a function that maps vectors of attribute values to a single output value (the decision)
    - Each node is a test of value of one of the input variables
    - Branches from nodes are labelled with possible attribute values
    - Leaves specify the decision to return
  - Perform tests starting at the root until a leaf is reached
- This class: Boolean classification
  - Discrete values and positive/negative outputs

Decision Tree

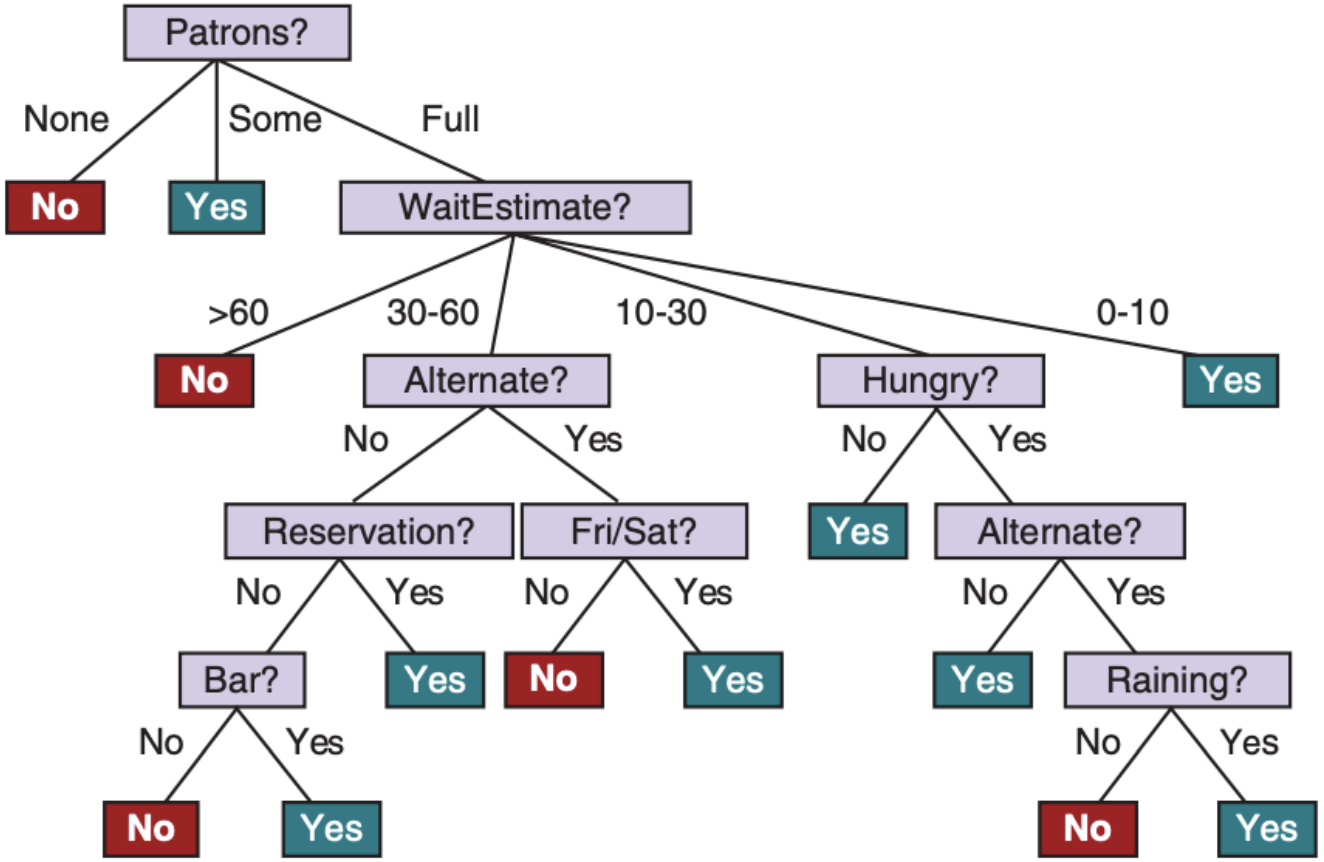


Figure 19.3 A decision tree for deciding whether to wait for a table.

Decision Trees

- When we talk about learning a decision tree we are “searching” through the space of all possible decision trees to find a simple one that is consistent with the input data
  - Why simple/small?

- Given attributes
  - Test most important attribute first
    - One that would make the most difference to the classification decision
  - Continue recursively

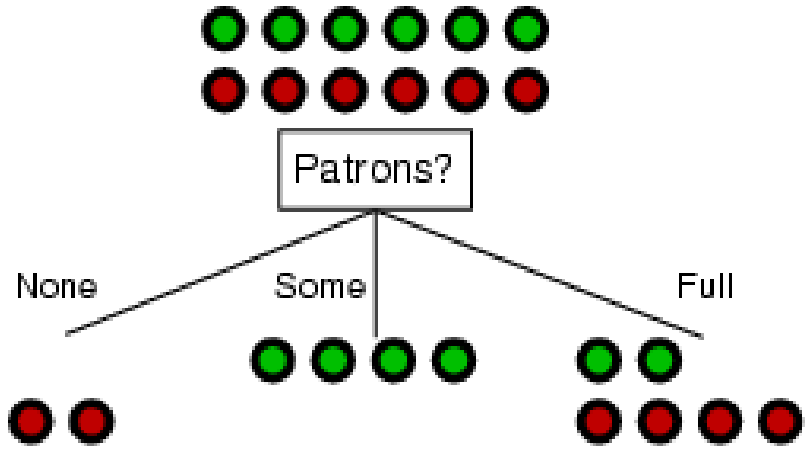
Learning a Decision Tree

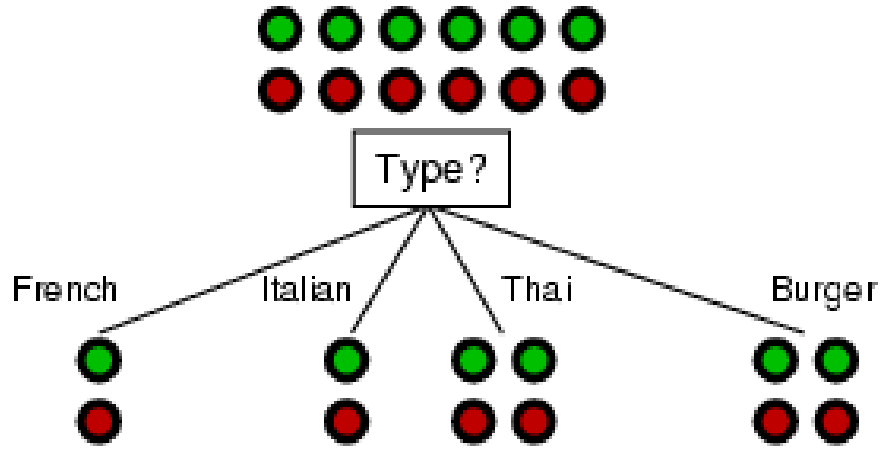
- Aim: Find a small tree consistent with the data

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an Attribute

There are different ways of selecting attributes, but a “good attribute” splits the training examples appropriately.





Information Theory

- How do we determine what a ”good attribute” is?
  - Entropy

- Entropy measures the uncertainty about a random variable
  - Consider flipping a coin:
    - If the coin is weighted so that it is always H, there is 0 uncertainty (0 entropy)
    - A fair coin (0.5 H, 0.5T) has maximum uncertainty (entropy is 1)
- In general

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

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

Restaurant Example

There are 12 examples, 6 are positive (p) and 6 are negative (n).

Therefore

$$H(output) = -\left(\frac{6}{12}\log_2\left(\frac{6}{12}\right) + \frac{6}{12}\log_2\left(\frac{6}{12}\right)\right)$$

$$H(output) = 1.0$$

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*	$y_{12} = Yes$

Figure 19.2 Examples for the restaurant domain.

Restaurant Example

But what if we test attribute A with d values?

- Gives us additional information
- Reduces overall entropy
- Splits the dataset into $E_1,\ldots E_d$ where E_k = subset where A_k is true (p_k and n_k pos and neg examples)
- After the test, remaining entropy is

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

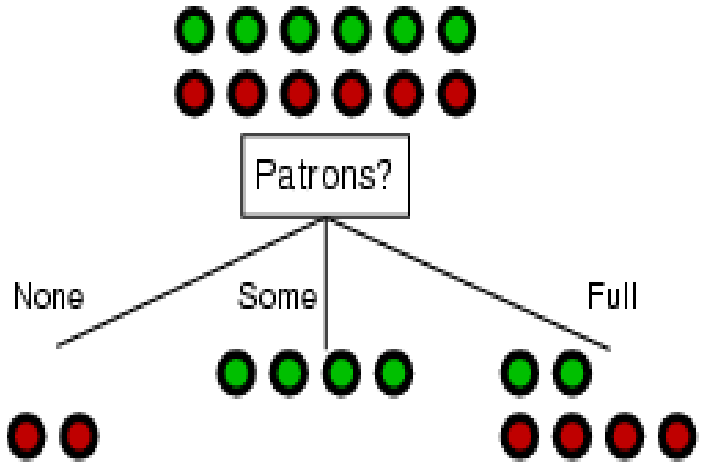
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$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

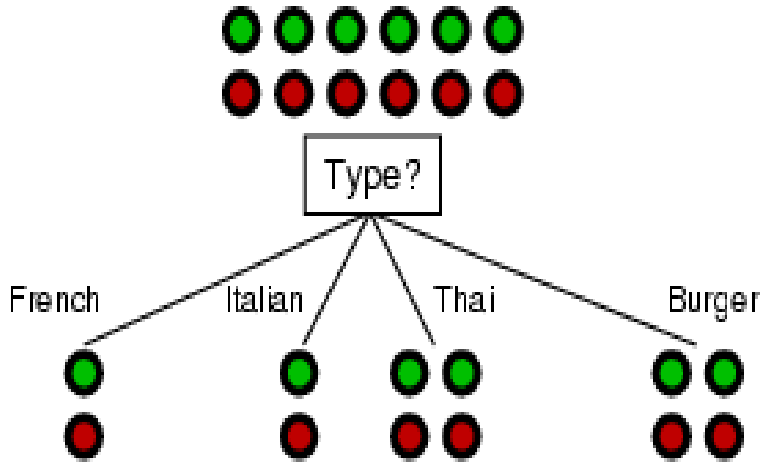
Figure 19.2 Examples for the restaurant domain.

The **Information Gain (Gain)** or reduction in entropy from using attribute A is

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

Choosing an Attribute





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

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

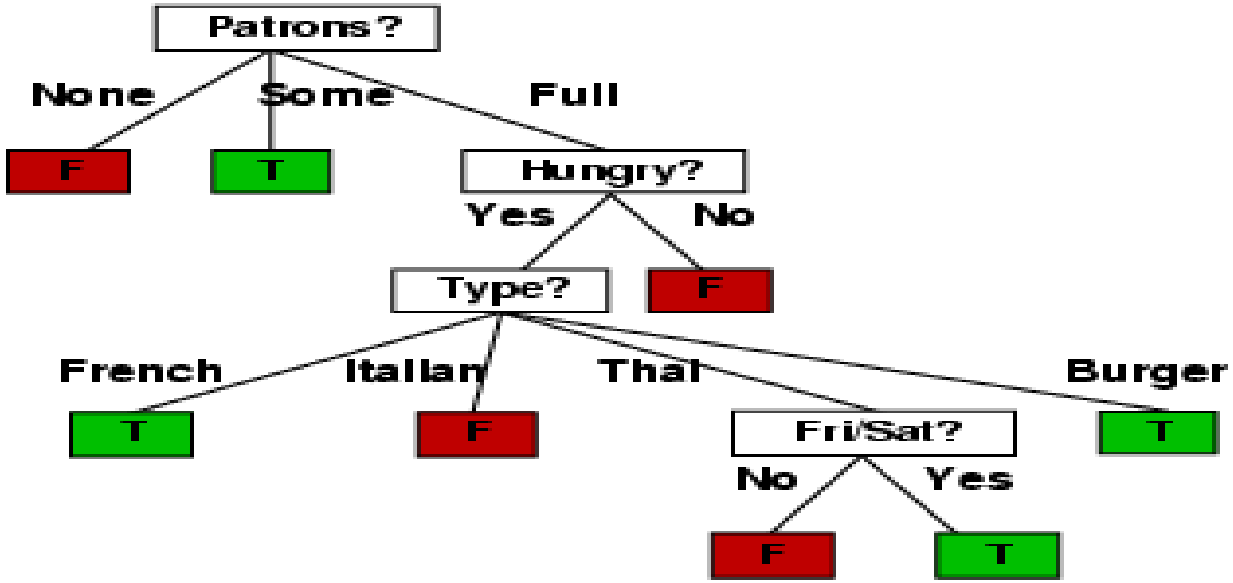
Learning a Decision Tree

- Aim: Find a small tree consistent with the data

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

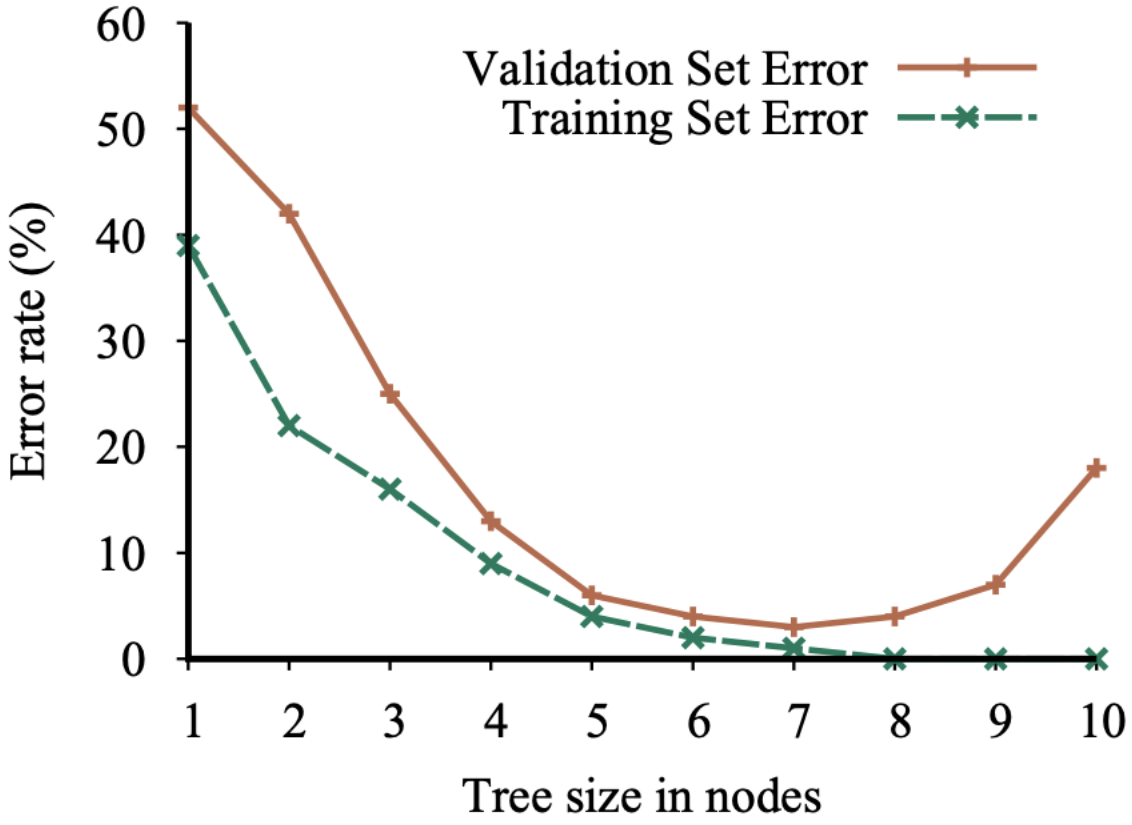
Decision Tree

- Significantly “simpler” than a tree using all attributes
  - A more complex hypothesis is not justified by the data we have



Decision Trees: Learning

- When do we stop splitting on attributes?
  - Only 1 feature remains
  - Data subset has the same label
  - Reached predefined depth/number of nodes
  - According to model selection criteria on **validation set**

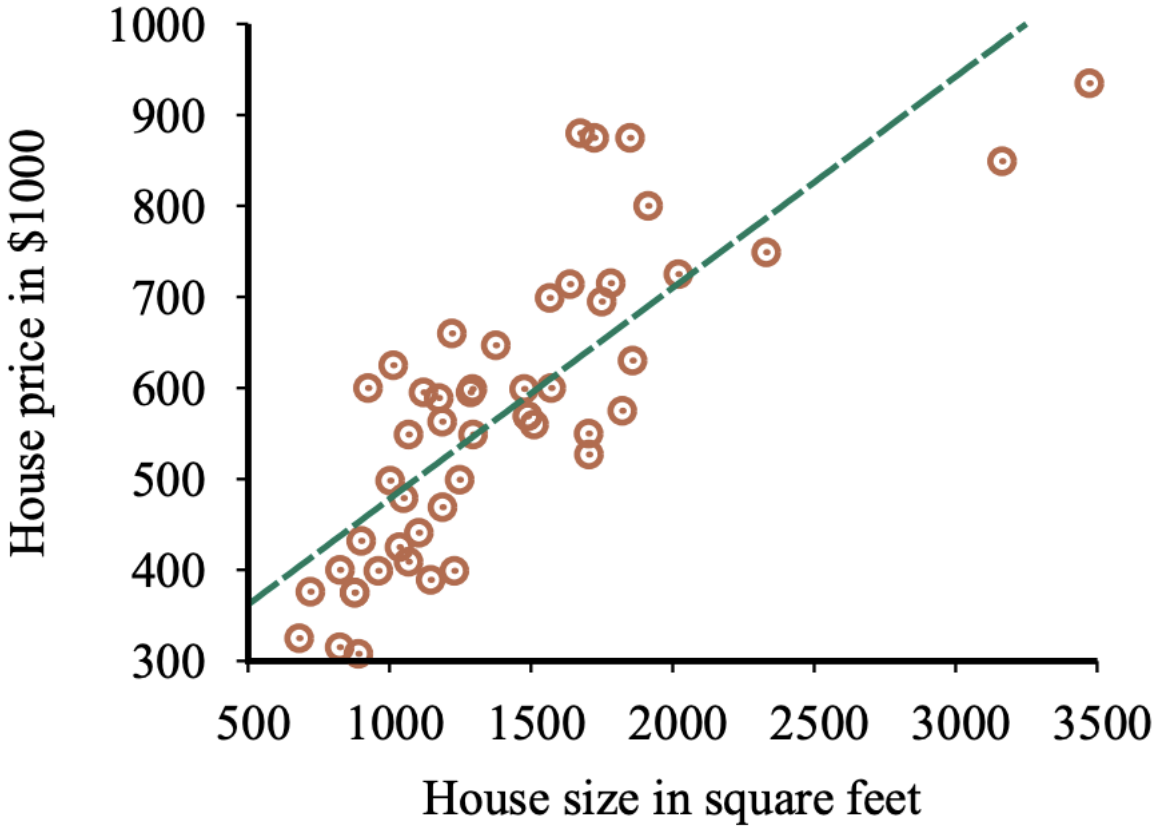


Linear functions

- Linear functions are another representation that are a completely different hypothesis space
  - Naturally for fitting a straight line for regression
  - Can also be used for classification

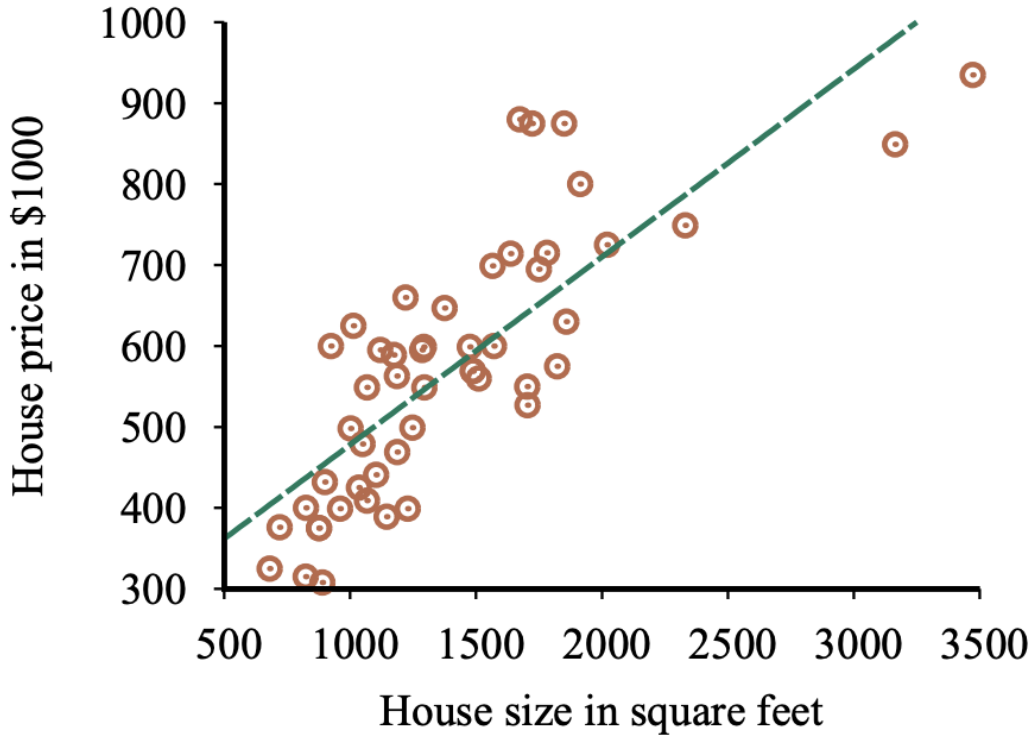
Linear Functions

- Consider data with one input feature, x, and one output feature, y.
- Univariate linear function hw predicts y using weighted sum over x
  - $h_w(x)=w_0+w_1x$
  - w: weights of the linear function



Linear Regression

- Suppose we have a dataset of N examples $\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$
- Linear regression finds the h_w that best fits the data
  - Finds w* that minimizes cumulative loss over the dataset



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

Linear Regression

- How do we do this?

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

- Observation: At w*, gradient with respect to w is 0

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

Solution 1: Solve Directly

- How do we do this?

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

- Observation: At w*, gradient with respect to w is 0

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

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N. \qquad (19.3)$$

Solution 2: Gradient Descent

- How do we do this?

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

- Always move in the direction that minimizes the loss

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

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w}) \qquad (19.4)$$

Gradient Descent

- Chain Rule: $\partial g(f(x))/\partial x = g'(f(x))\partial f(x)/\partial x,$
- For simplified case with one example

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

$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) . \qquad (19.5)$$

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

- Update rules (alpha is the learning rate or step size)

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

Gradient Descent

- For one example

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

- For a dataset of examples

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

- For multivariate hypotheses:

$$h_{\mathbf{w}}(\mathbf{x}_j) \;=\; w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum\nolimits_j w_i x_{j,i} \,. \qquad h_{\mathbf{w}}(\mathbf{x}_j) \;=\; w \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i} \,.$$

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

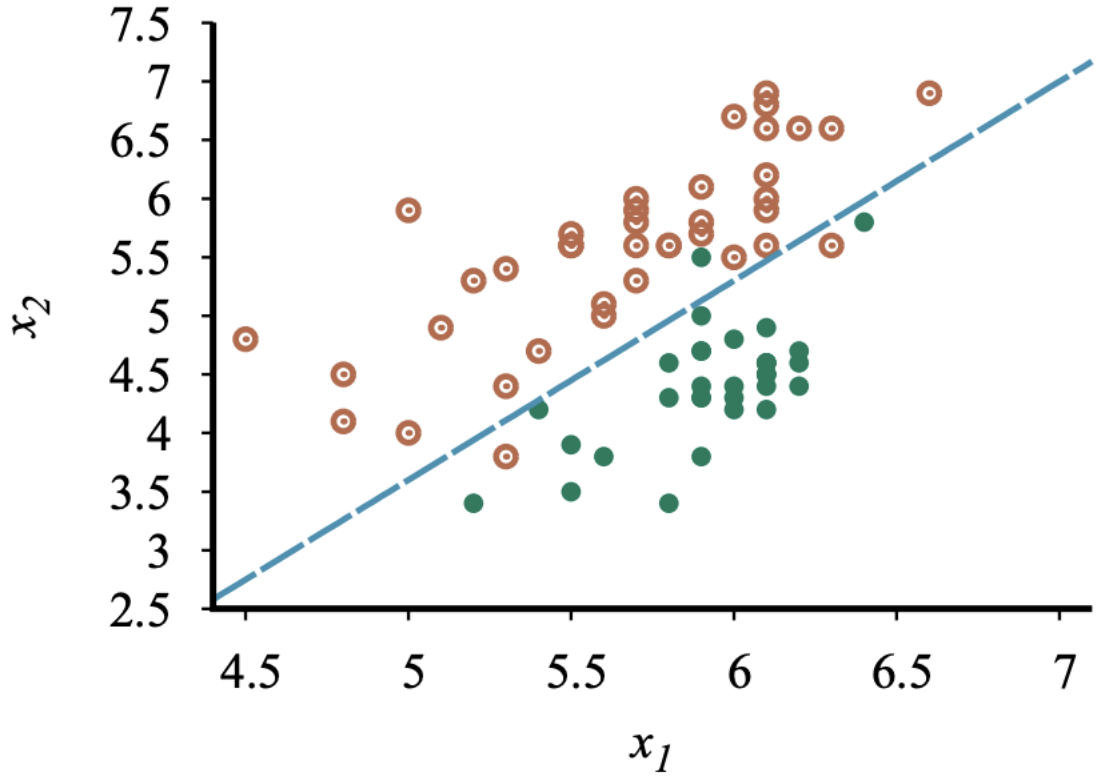
$$w_i \leftarrow w_i + \alpha \sum\nolimits_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i} \,. \qquad (19.6)$$

Flavours of Gradient Descent

- Optimize total loss over gradient descent: *batch* or *deterministic*
  - Slow because we must sum over entire dataset for each update
  - One update per epoch (scan over entire dataset)
- Randomly choose one example for update: *stochastic (classic)*
  - Fast: *N* updates per epoch
  - But gradient estimate has high variance
- Randomly choose batch of b examples for update: *stochastic*
  - Fast: *N/b* updates per epoch
  - Gradient estimate is more stable

Linear Classification

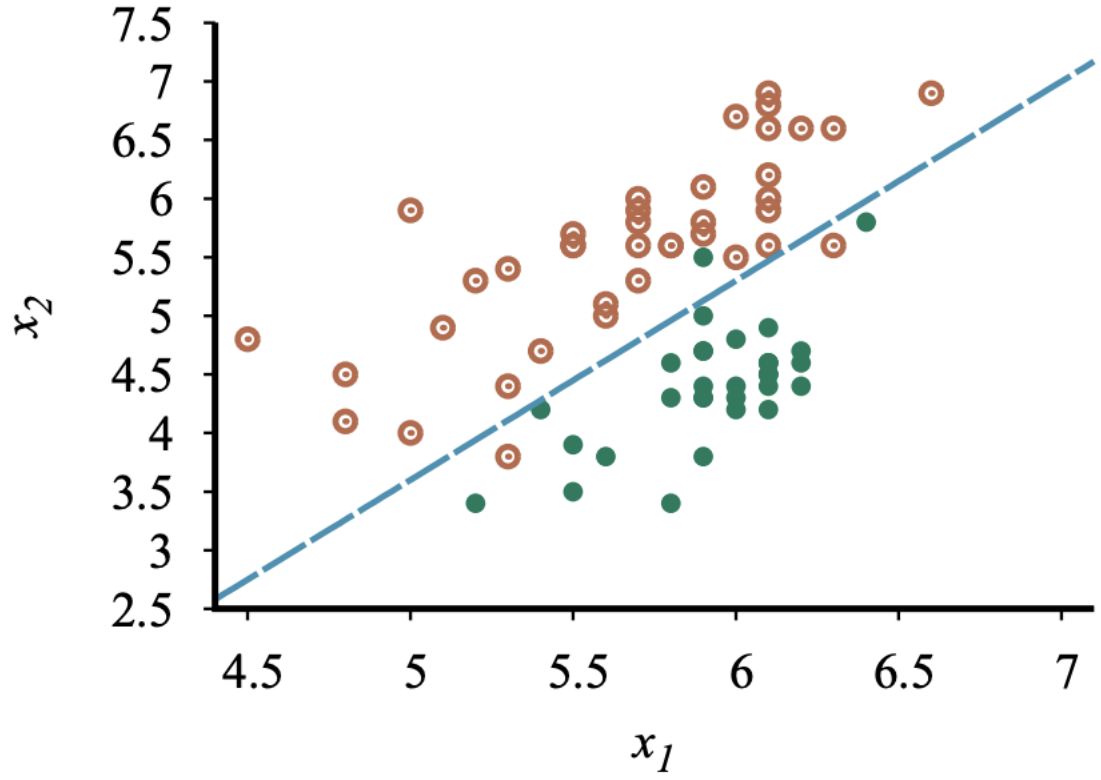
- Linear classification is actually quite similar to regression
  - Regression: line predicts y given x
  - Classification: line is a decision boundary that separates classes



Linear Classification

- Consider classifying
  - Earthquakes
  - Underground explosions
- Input and output
  - X1: body wave magnitude
  - X2: surface wave magnitude
  - Y: 0 for earthquake, 1 for explosion
- Hypothesis:

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



Linear Classifiers

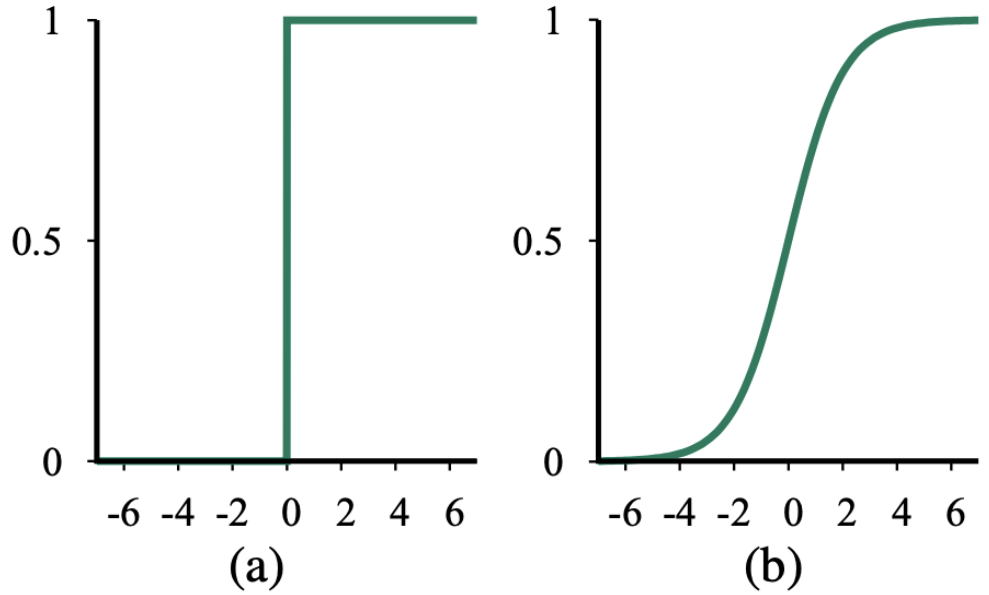
- Hypotheses using threshold functions

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

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

- Some different thresholds
  - Hard thresholds
  - Soft thresholds (Sigmoid or Logistic)

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



Linear Classification (Hard Thresholds

- Perceptron Update Rule

$$w_i \leftarrow w_i + \alpha(y - h_w(x))x_i$$

- Difference from regression is that y and $h_w(x)$ are in {0,1}
  - If $y = h_w(x)$: weights unchanged
  - Else
    - If $y = 1$ but $h_w(x) = 0$: w_i increased when x_i positive and decreased when x_i negative
      - Great because we want to make *h* bigger to 1
    - If $y = 0$ but $h_w(x) = 1$: w_i decreased when x_i positive and increased when x_i negative
      - Great because we want to make *h* smaller to 0

Linear Classification (Sigmoid Threshold)

- We call this logistic regression $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$.
  - *h* outputs range [0, 1], so use L_2 loss
  - Let's call Logistic function *g*, its derivative *g'* happens to be

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

  - Apply Chain rule:

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

Linear Classification

- Hard Threshold

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

- Sigmoid Threshold

$$w_i \leftarrow w_i + \alpha\,(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i\,. \quad (19.9)$$