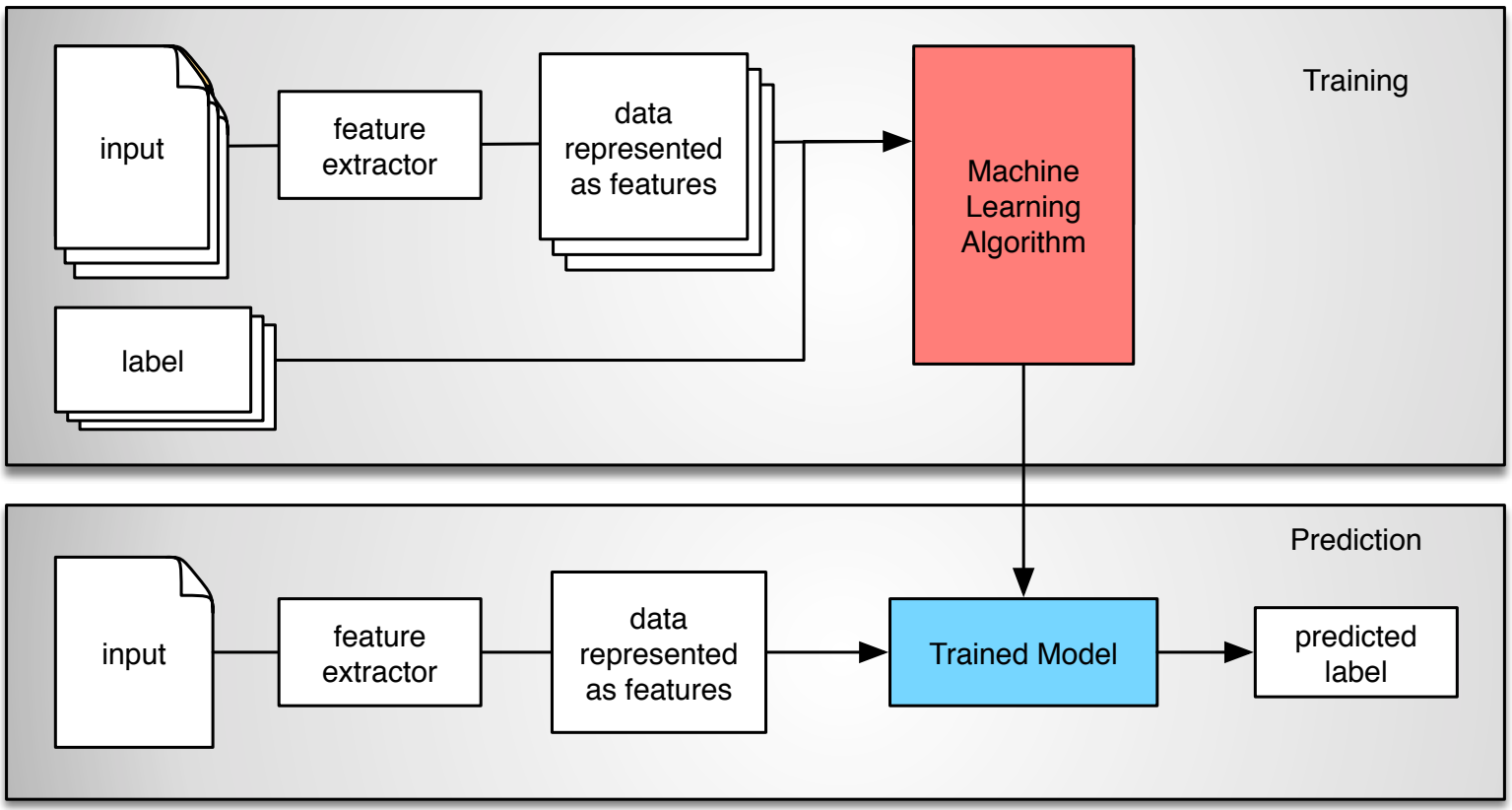
Lecture 2: Inductive learning

Helen Yannakoudakis and Oana Cocarascu

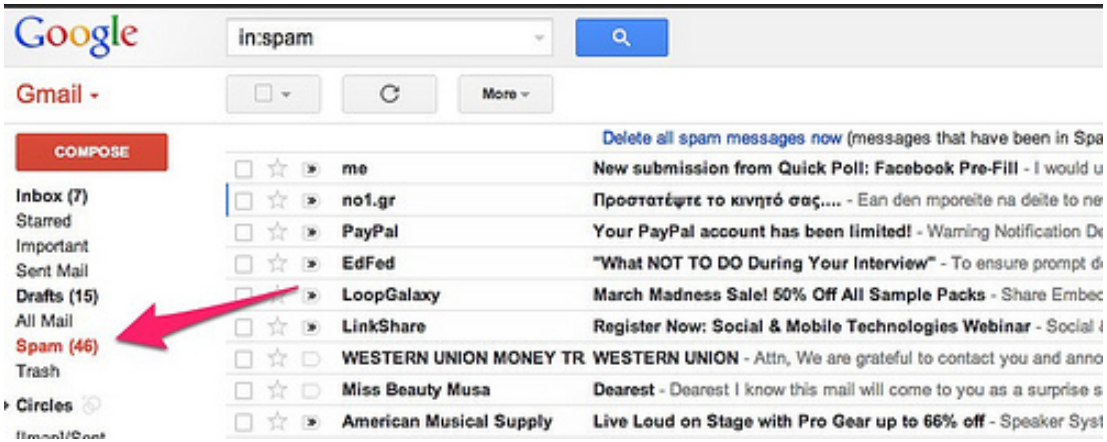
Department of Informatics
King's College London

(Version 1.7)

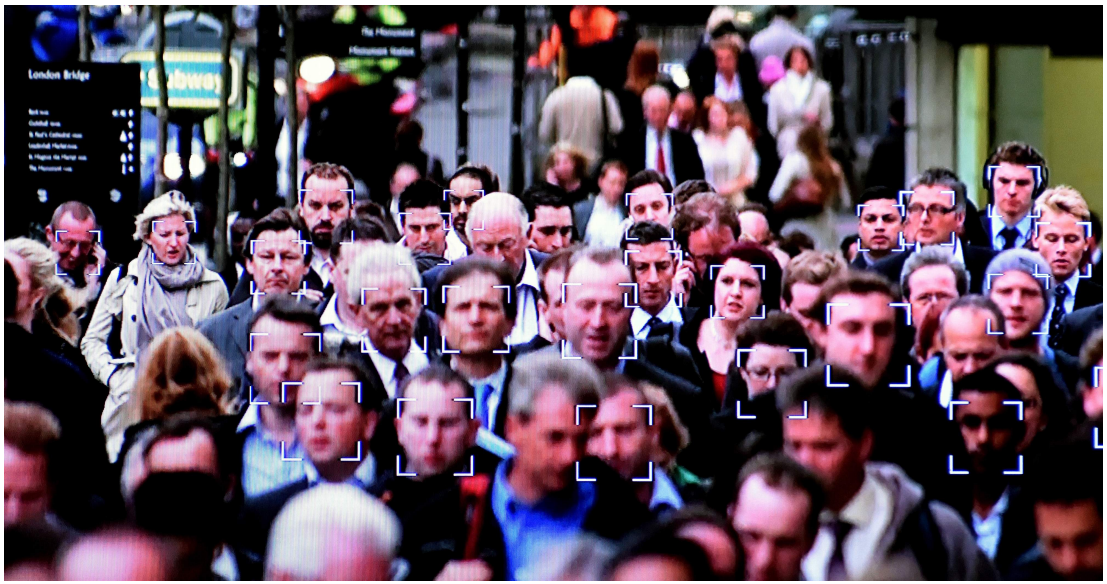
Supervised learning



What is this good for?



(www.whatcounts.com)



(The Guardian/ZUMA Press Inc)

Today

- Introduction
- Inductive Learning
- Probabilistic models 1
- Probabilistic models 2
- Kernel Methods
- Neural Networks
- Evolutionary Algorithms
- Reinforcement Learning 1
- Reinforcement Learning 2
- Learning from Demonstration

KING'S College LONDON

Today

- Inductive learning = learning from examples.
- Inductive learning hypothesis: any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

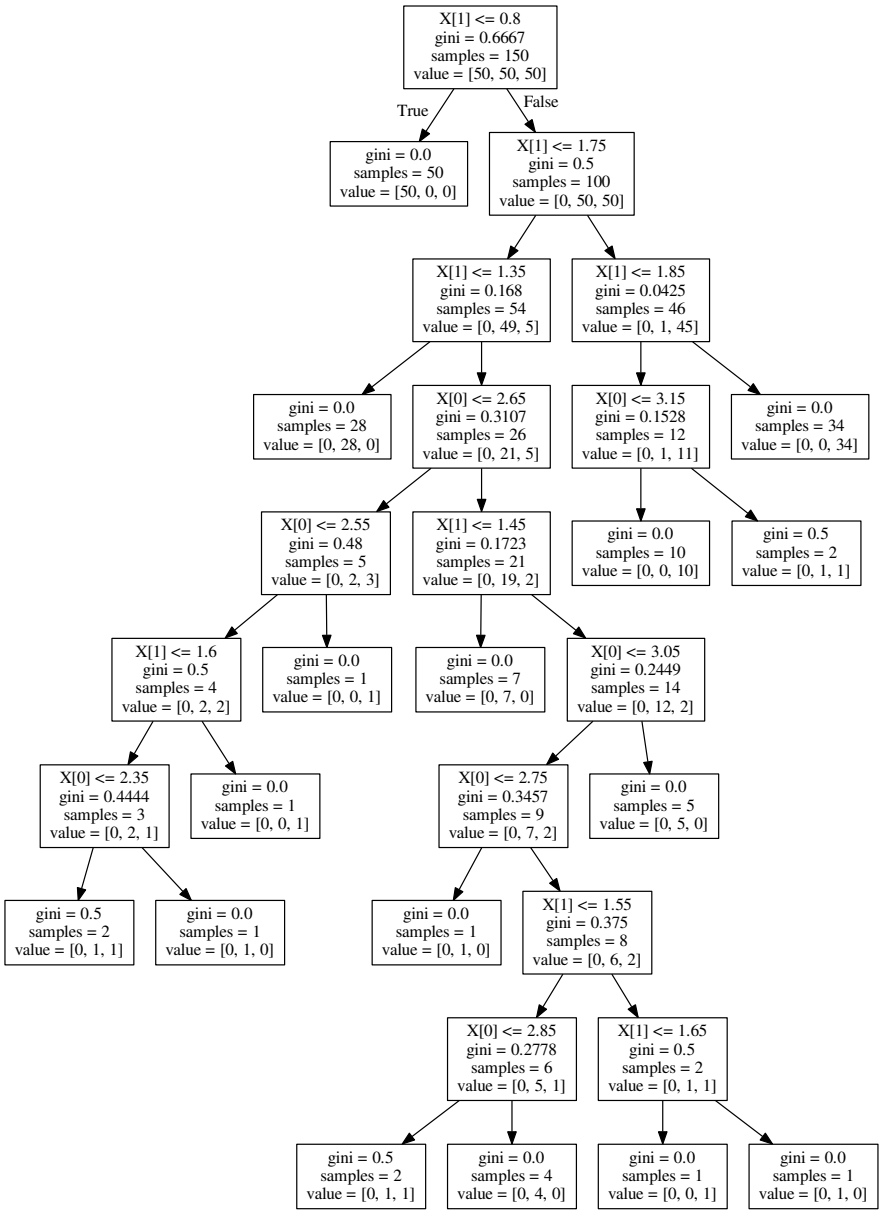
- Simple classifiers and regression models.
- Classification: classify examples into one of a discrete set of possible categories (eg, setosa, virginica).
- Regression: predict real-valued scalar (eg, income level).

- Optimisation: identify the “best” model parameters.

Today

- Decision trees
- Linear regression
- Linear classifiers
- Logistic regression
- Ensemble methods

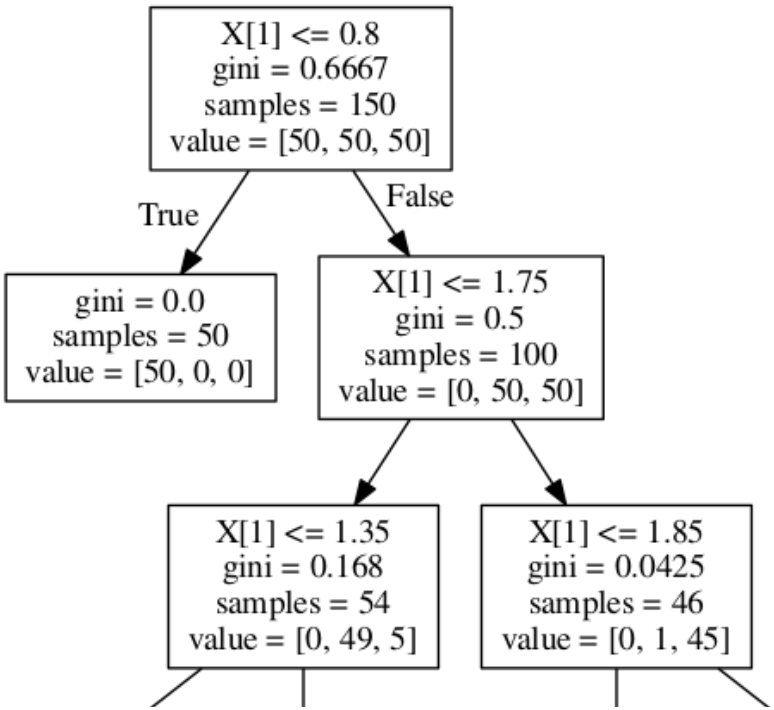
Decision tree for Iris example



- Approximating discrete-valued target functions.
- Learned function represented by a decision tree.
- Node: a test of the value of a feature for a data point.
- Leaf node: specifies the value to be returned.
- Disjunction of conjunctions of constraints on the feature values of instances / data points.
- Learned trees can be re-represented as sets of if–then rules.

KING'S College LONDON

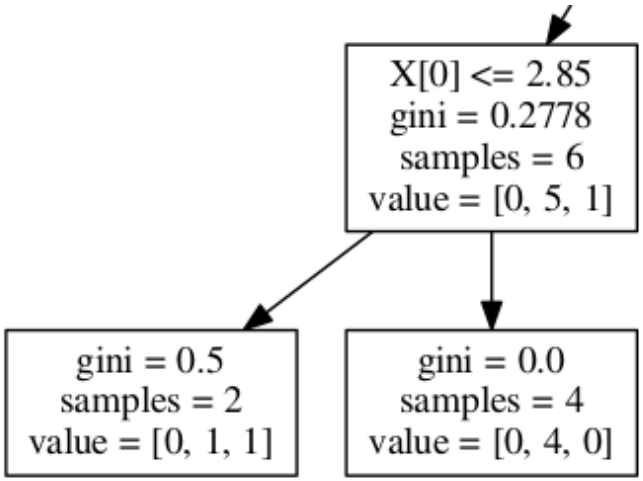
Decision tree for Iris example



- Approximating discrete-valued target functions.
- Learned function represented by a decision tree.
- Node: a test of the value of a feature for a data point.
- Leaf node: specifies the value to be returned.
- Disjunction of conjunctions of constraints on the feature values of instances / data points.
- Learned trees can be re-represented as sets of if–then rules.

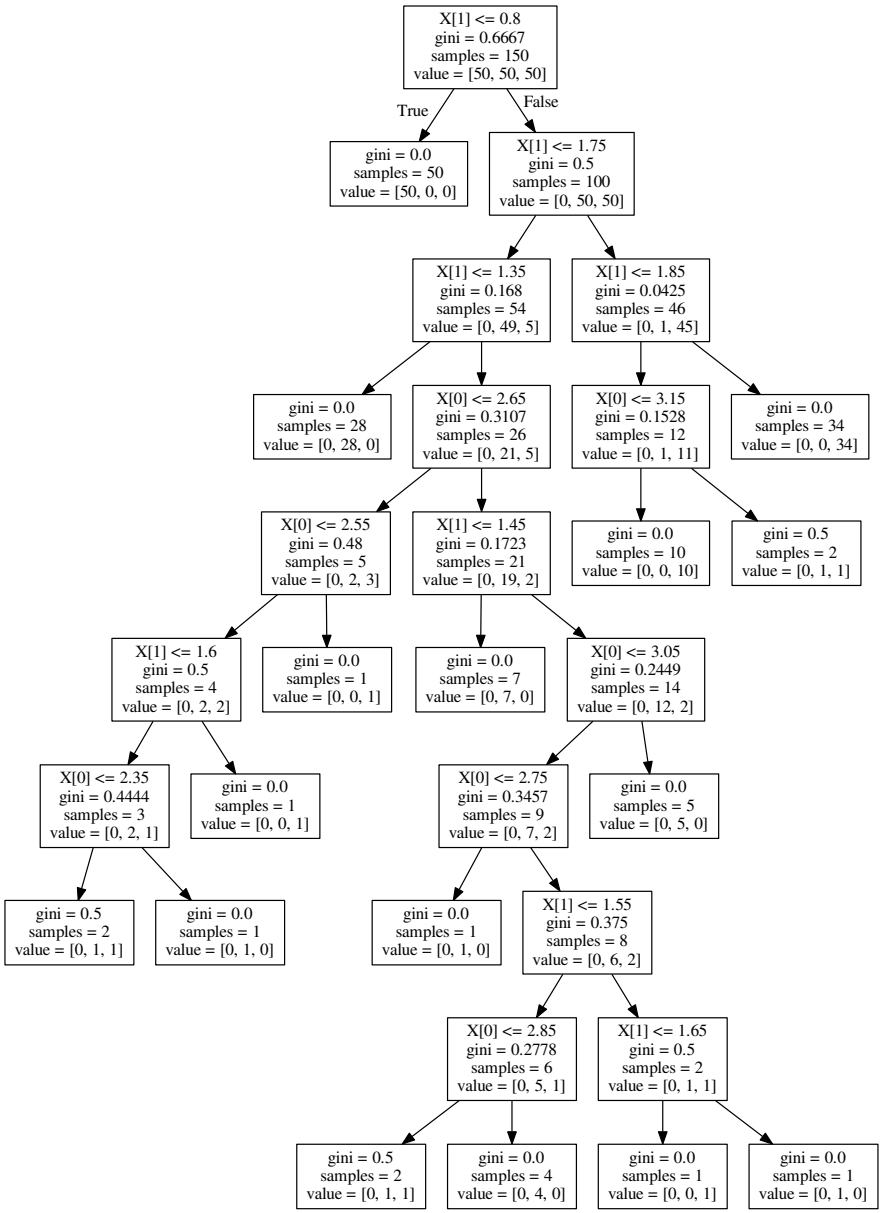
KING'S College LONDON

Decision tree for Iris example



- Approximating discrete-valued target functions.
- Learned function represented by a decision tree.
- Node: a test of the value of a feature for a data point.
- Leaf node: specifies the value to be returned.
- Disjunction of conjunctions of constraints on the feature values of instances / data points.
- Learned trees can be re-represented as sets of if–then rules.

Decision tree for Iris example



- Approximating discrete-valued target functions.
- Learned function represented by a decision tree.
- Node: a test of the value of a feature for a data point.
- Leaf node: specifies the value to be returned.
- Disjunction of conjunctions of constraints on the feature values of instances / data points.
- Learned trees can be re-represented as sets of if–then rules.

KING'S College LONDON

Concrete example

Should we wait for a table at a restaurant?

What is our learning target?

Dataset:

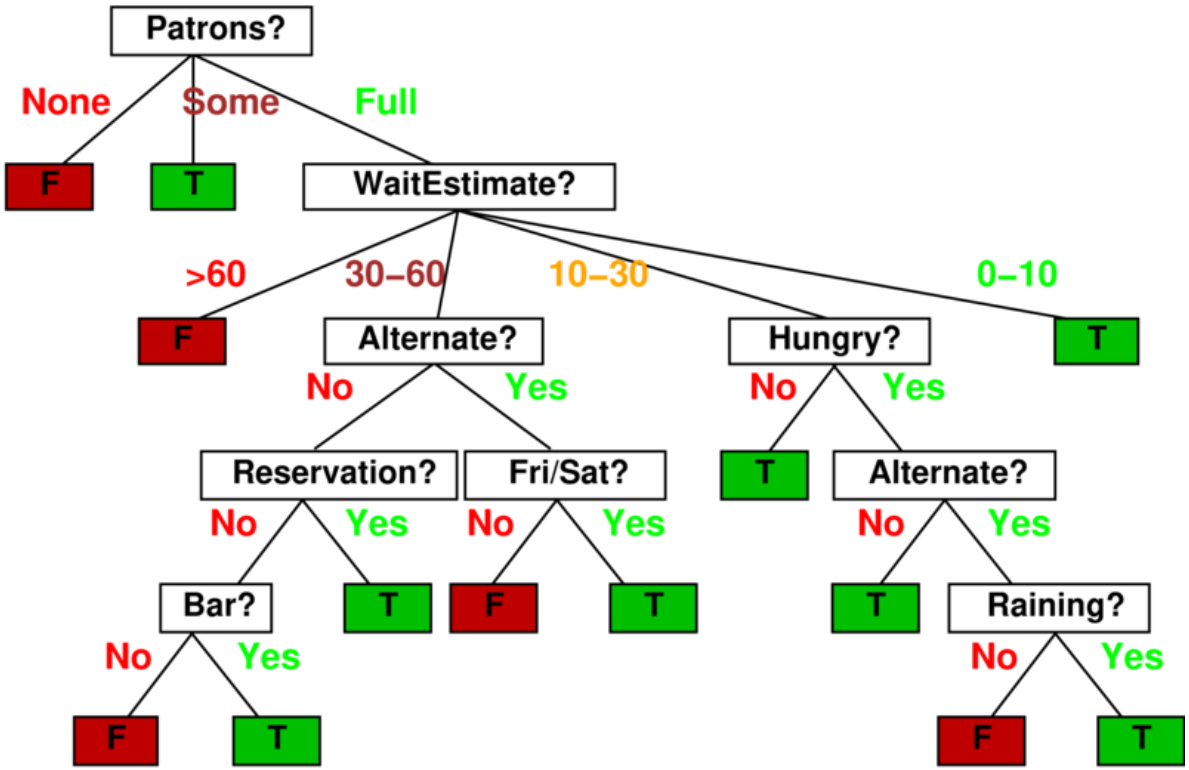
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*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*

Suitable alternative nearby; has a bar; Friday; hungry; patrons: how many people in the restaurant; price range; raining outside; have reservation; type; wait estimate by host.

Different X different datapoint / restaurant.

Decision trees

- Here is the “true” tree for deciding whether to wait:



(Russell & Norvig)

Price and type features?

Decision trees: expressiveness

- Trivially, $\exists$ a consistent decision tree for any training set with one path to leaf for each example.
  - Path tests each feature in turn and follows the value for the example.
  - Leaf has the classification of the example.
  - (Unless f non-deterministic in x).

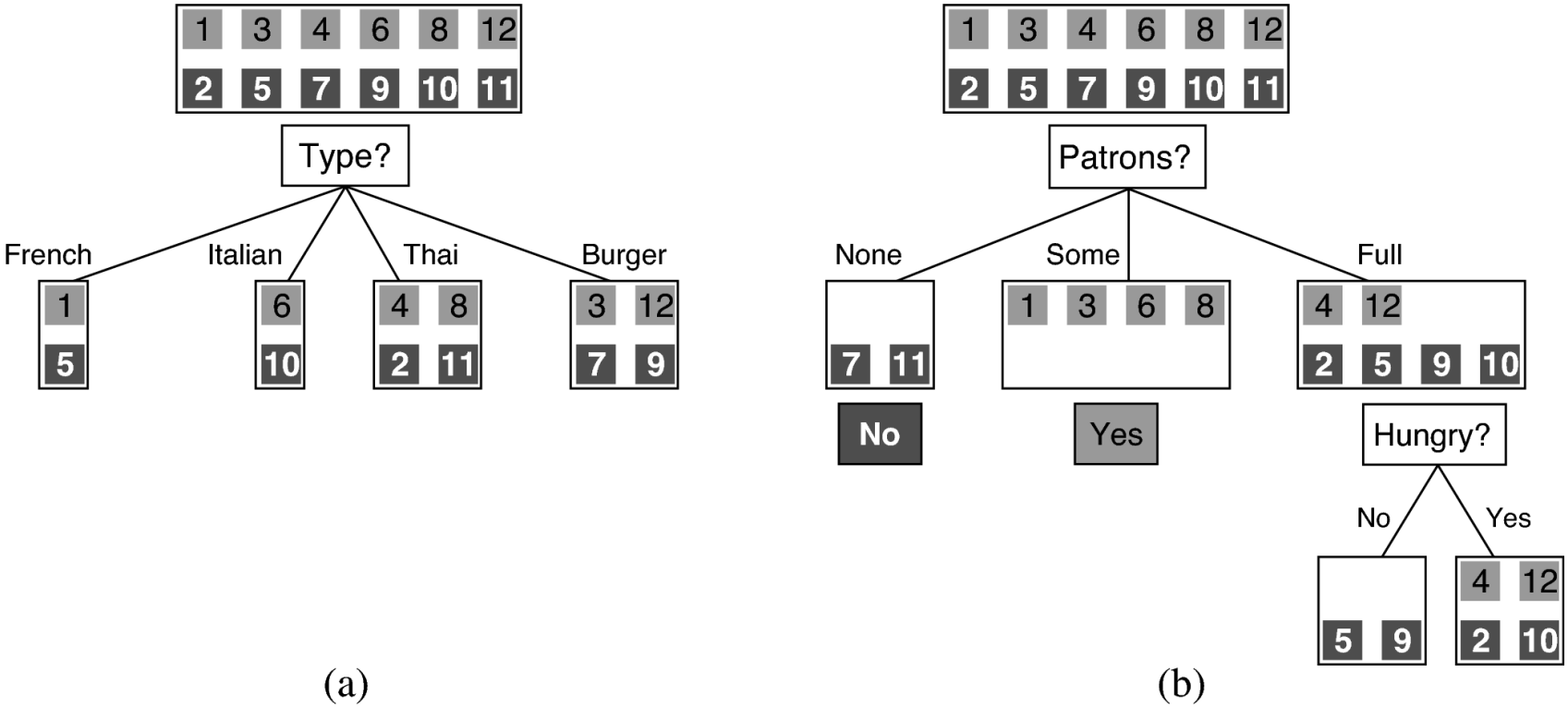
Decision trees: expressiveness

- Trivially, $\exists$ a consistent decision tree for any training set with one path to leaf for each example.
  - Path tests each feature in turn and follows the value for the example.
  - Leaf has the classification of the example.
  - (Unless f non-deterministic in x).
- This trivial tree probably won't generalize to new examples.
- Prefer to find more compact decision trees.

Decision tree learning

- Aim: find a small tree consistent with the training examples.
- Basic idea: test the most important feature first – why?
- (Recursively) choose “most significant” feature as root of (sub)tree.

Some features are better than others



(Russell & Norvig)

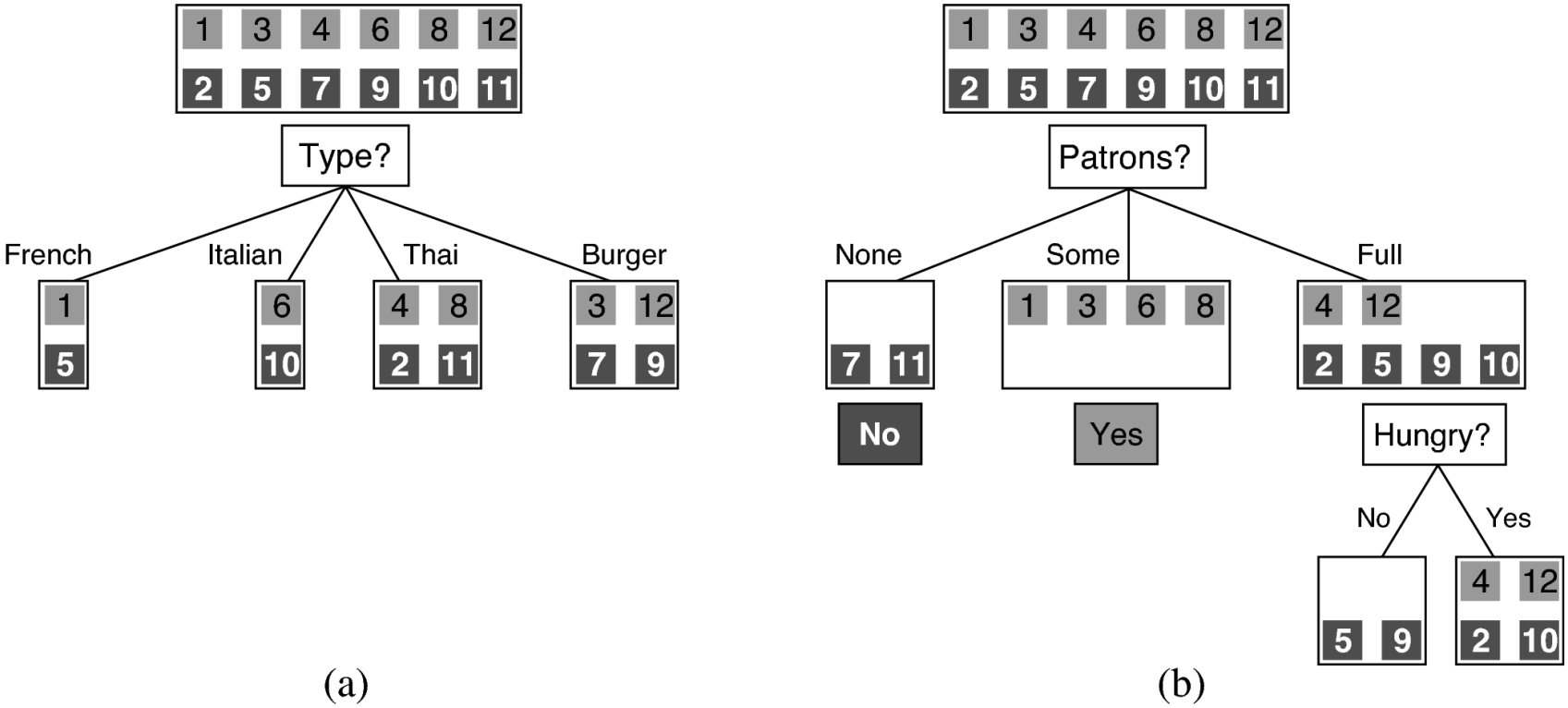
Decision tree learning

- After we pick a feature, one of five things can be the case.

Decision tree learning

- **1.** If all remaining examples are positive, then we are done. We can answer “yes”.
- See *Some* in our example (next slide).

Decision tree learning

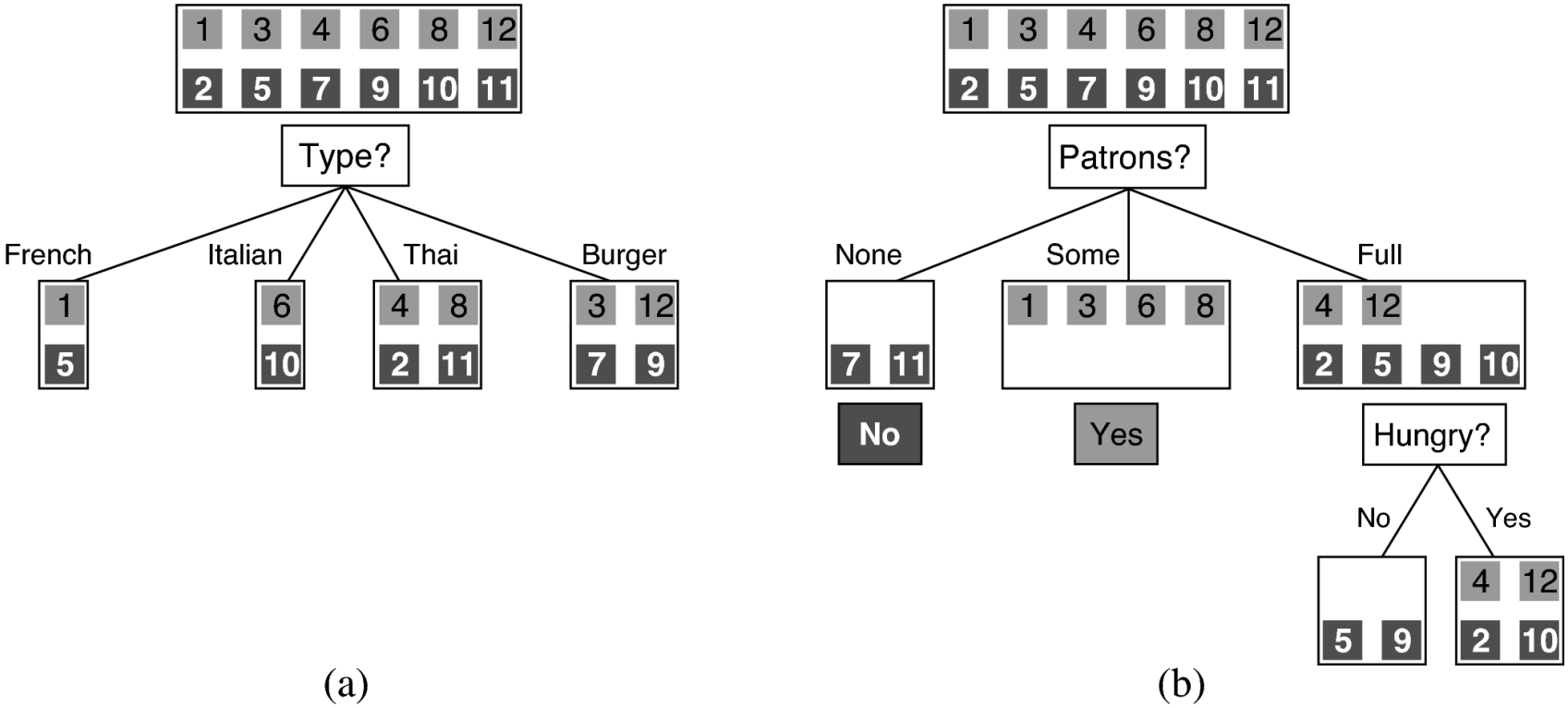


(Russell & Norvig)

Decision tree learning

- **2.** If all remaining examples are negative, then we are done. We can answer “no”.
- See *None* in the example.

Decision tree learning

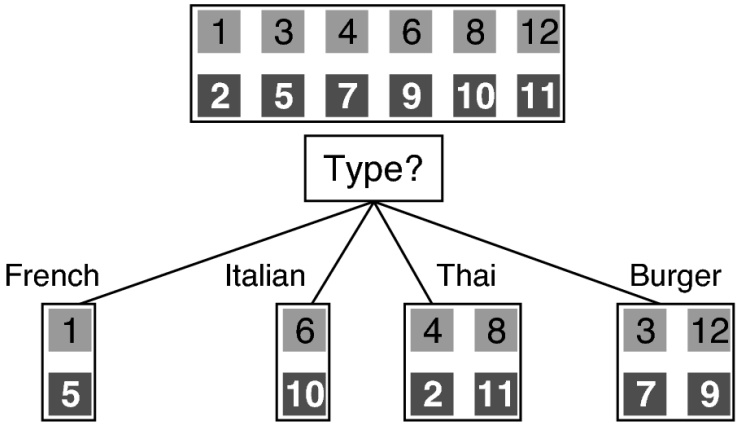


(Russell & Norvig)

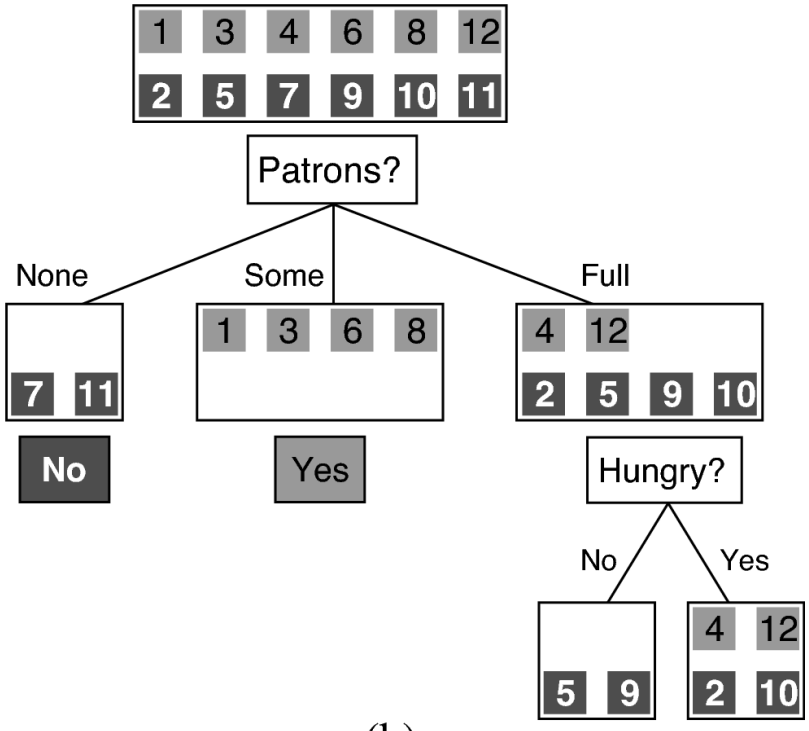
Decision tree learning

- **3.** If there are some positive and some negative examples, then choose the best feature to split them.
- See *Full*.

Decision tree learning



(a)



(b)

(Russell & Norvig)

Decision tree learning

- **4.** If there are no examples left, then there are no examples that fit this case.
- Ending up here means we have no data on the specific case we are asking about / with this combination of feature values.
- Best we can do is return a default value.
- Plurality classification.
- Best guess based on the parent node.
- Could be majority, ie., Yes if most examples at the parent node are classified “Yes”.
- Could be random pick, weighted by ratio of examples at parent node.
- Could be a probability, based on the ratio of examples at parent node.

Decision tree learning

- **5.** If there are no features left, but there are still positive and negative examples, these examples have the same description but different classifications.
- Can be due to noise.
- Can be due to unobservability of features.
- Plurality classification as a simple solution.

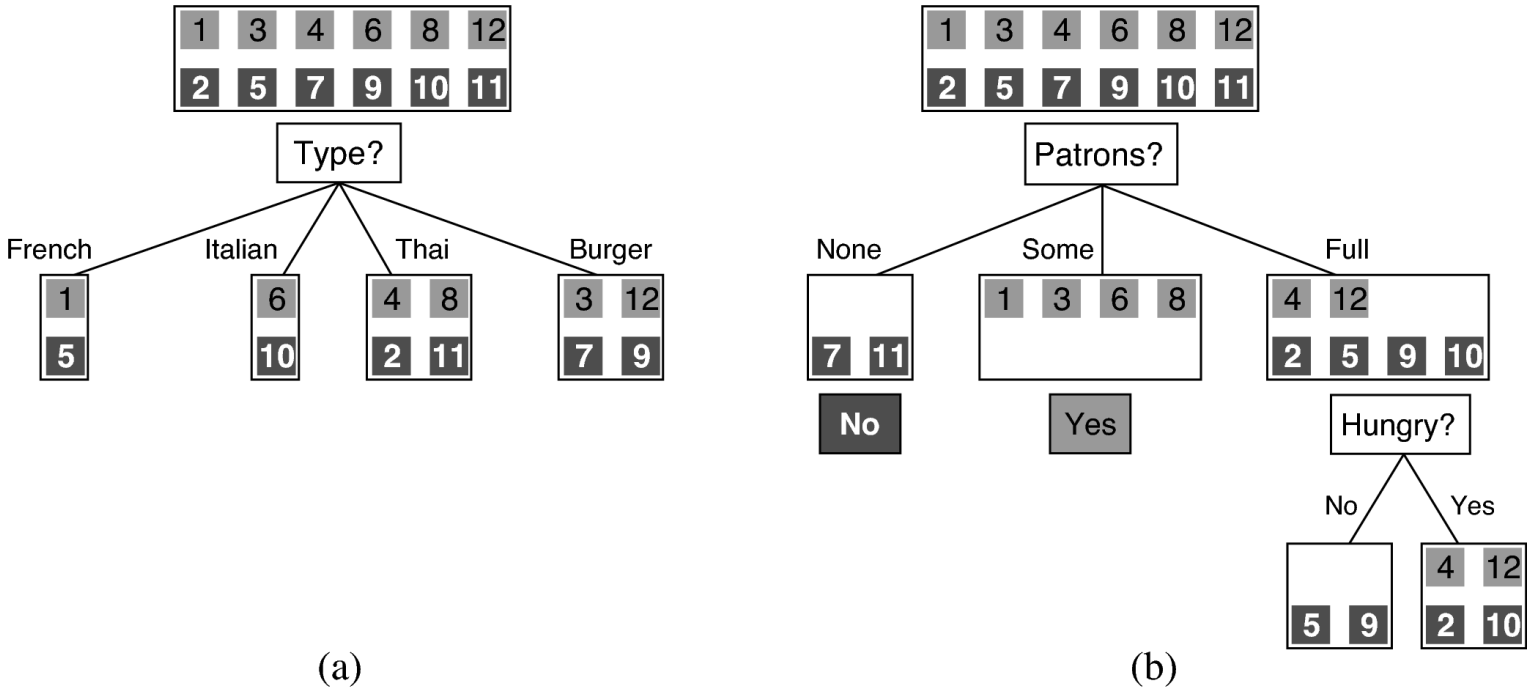
Decision tree learning

function DTL(*examples, attributes, parent_examples*) **returns** a decision tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_eamples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
best ← MOST-IMPORTANT-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* $= v_i$}
remain ← *attributes* − *best*
subtree ← DTL($examples_i$, *remain*, *examples*)
add a branch to *tree* with label v_i and subtree *subtree*
return *tree*

Choosing the “best” feature

- Idea: a good feature splits the examples into subsets that are (ideally) “all positive” or “all negative”.



(Russell & Norvig)

- *Patrons*? is a better choice—gives information about the classification.

Information and Entropy

- We will measure the information in a feature by looking at how it reduces entropy.
- Measure of uncertainty.
- Entropy in a probability distribution $\langle P_1, \ldots, P_n \rangle$ is

$$H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$$

- Maximum for a uniform distribution, minimum for a single point.

Calculating $\log_2$

- Your calculator probably doesn't have a $\log_2$ function.
- Instead it has log and ln.
- ln is no use here.
- log is really $\log_{10}$.
- Can compute $\log_2$ using $\log_{10}$ by:

$$log_2(x) = \frac{\log_{10}(x)}{\log_{10}(2)}$$

Entropy

- Suppose we have a set S of p positive and n negative examples at the root.
- If we had to pick the class at this point, we could compute the probability based on the positive and negative examples.
- Chance of $Class = 1$ would be

$$p(Class = 1) = \frac{p}{p+n}$$

- So we have a probability distribution over classes, and we can compute the entropy of S.

$$H(S) = H\left(\left\langle \frac{p}{p+n}, \frac{n}{p+n} \right\rangle\right)$$

- For 12 restaurant examples, $p = n = 6$ so the entropy is 1.

Entropy

- For this example, we only need to handle the entropy of a distribution over two values, positive and negative.
- Generalizes to more classes:
- For c classes:

$$H(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

where p_i is the proportion of the examples in c_i.

Information Gain

- A feature A splits the examples S into subsets S_i.
- Each of these subsets is a new branch in the decision tree.
- Each of these subsets will have its own entropy.
- We can measure the "goodness" of A by the reduction (if any) in the entropy of S and the entropy of all the S_i.
- We call this difference information gain.

$$Gain(S, A) = H(S) - \sum_i \frac{|S_i|}{|S|} H(S_i)$$

- So the entropy of the S_i collectively is the weighted sum of their entropies.
- The weight is the proportion of examples in that set.

KING'S College LONDON

Information Gain

- So, if S_i has p_i positive and n_i negative examples.

$$H(S_i) = H\left(\left\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\rangle\right)$$

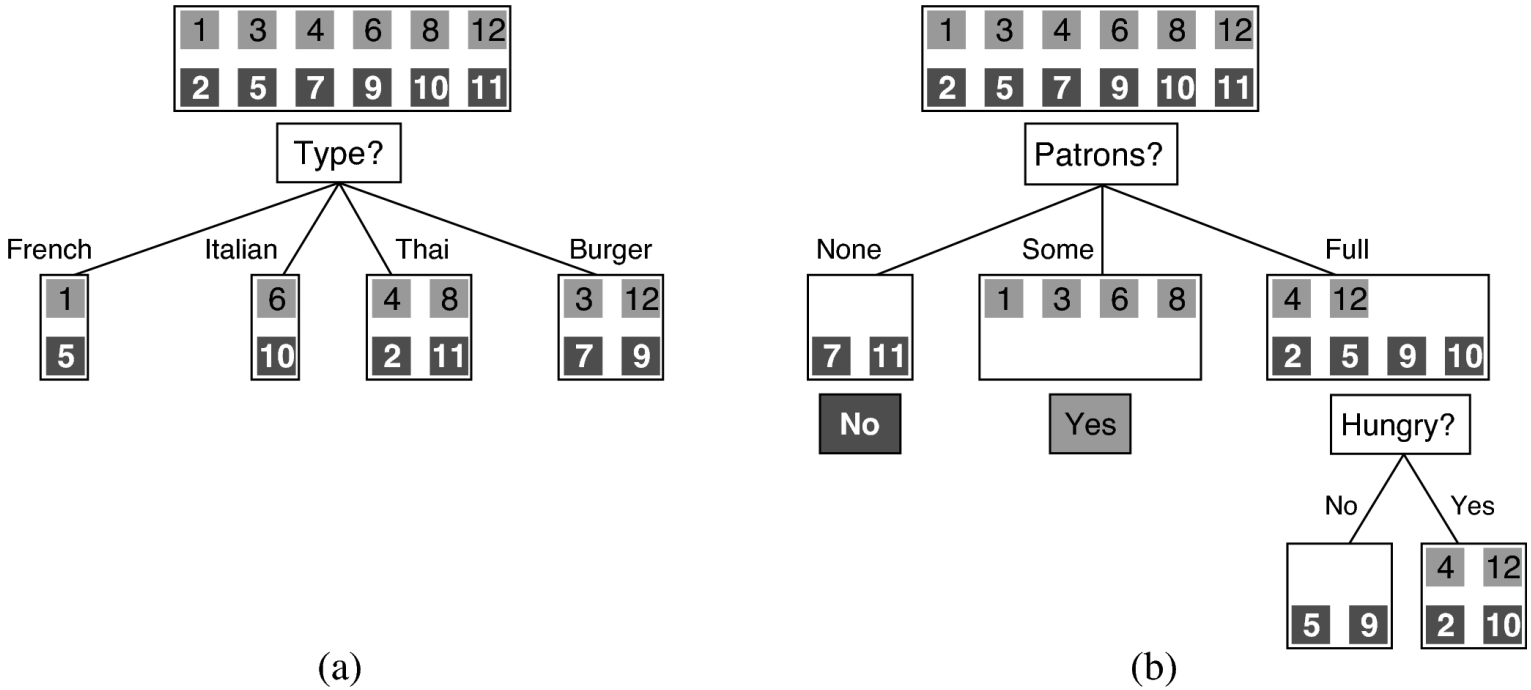
  is the entropy of S_i
- And so:

$$Gain(S, A) = H(S) - \sum_i \frac{p_i + n_i}{p + n} H\left(\left\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\rangle\right)$$

- The feature A with the biggest $Gain(S, A)$ is the one to pick.

KING'S College LONDON

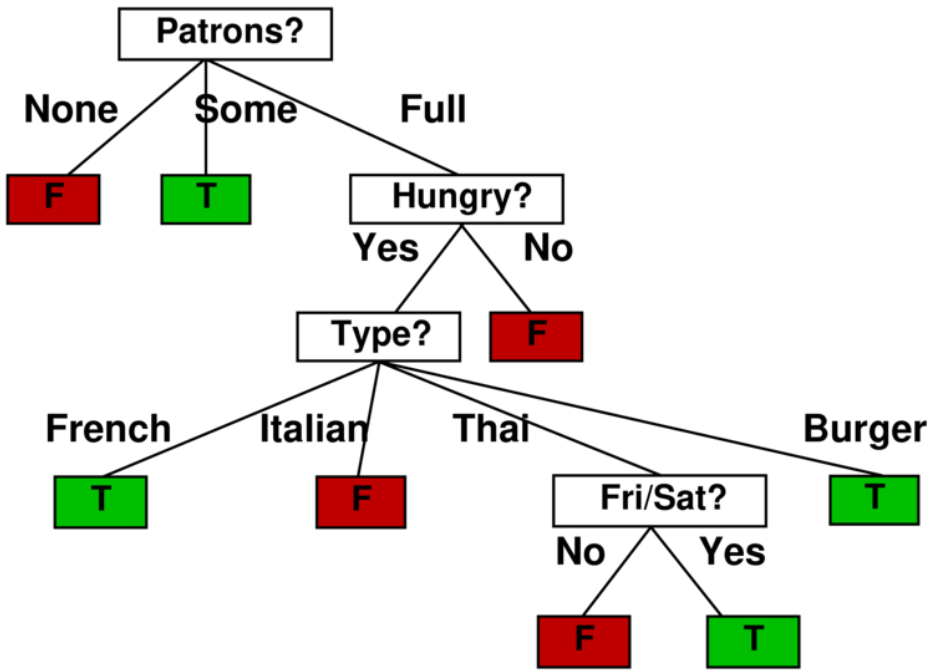
Information



- For *Patrons*?, gain is 0.541.
- For *Type*? gain is 0.
- Thus *Patrons*? is better than *Type*?.

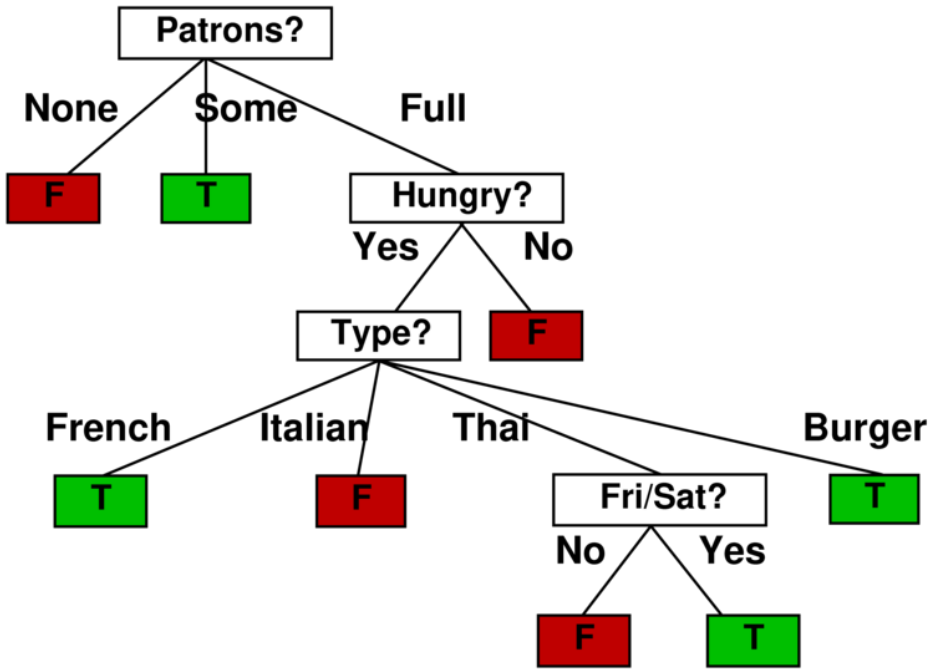
Back to the example

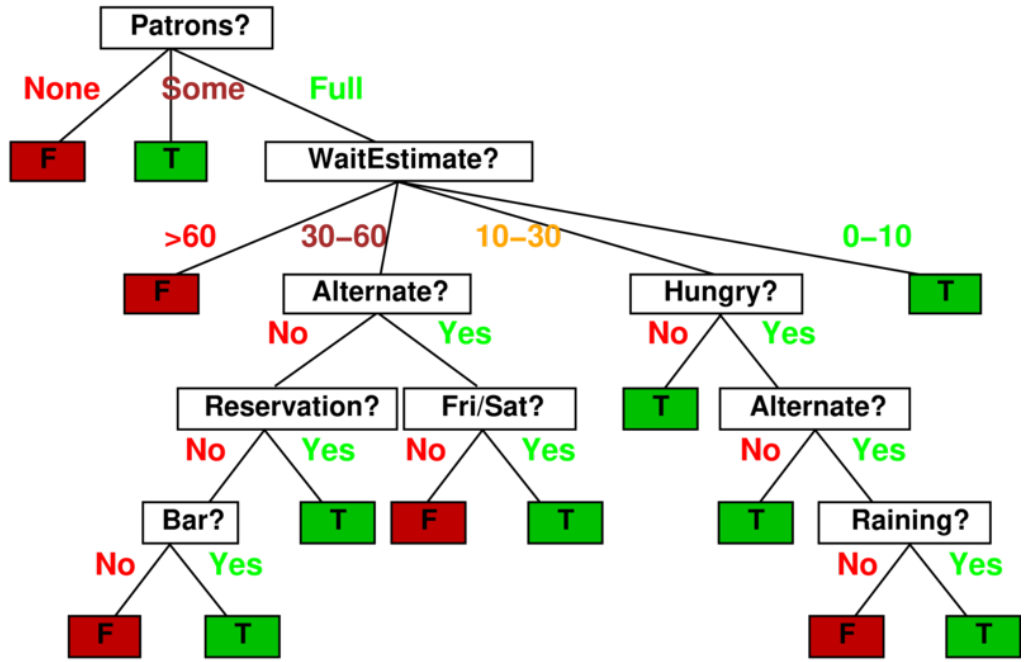
Decision tree learned from the 12 restaurant examples (left):



Back to the example

Decision tree learned from the 12 restaurant examples (left):





Substantially simpler than original tree we saw (right)—a more complex hypothesis isn't justified by small amount of data.

Other metrics for picking attributes

- Looking at information gain has a bias.
- Favours features that have many values.
- Imagine adding *RestaurantName* or *Time* to the restaurant example.
- *RestaurantName* enables a unique classification (information gain would have its highest value for this feature).
- With enough precision, *Time* would uniquely identify each case.
  This would look very good, but would be useless.

Gain ratio

- One solution is to add in a metric that penalizes features with lots of values:

$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

  intrinsic information content: the amount of information contained in the answer to the question, "What is the value of this feature?"

- We then combine this with *Gain* to get a new metric:

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$

- Now use *GainRatio* as the way of picking the best feature.

KING'S College LONDON

Gini Impurity

- Alternative to entropy to choose features.
- Suppose we have a set S of p positive and n negative examples at the root.
- We assign a probability based on the number of examples.
- E.g., chance of $Class = yes$ would be

$$p(Class = yes) = \frac{p}{p+n}$$

- Gini impurity is the probability of this classification mislabelling a randomly selected example.

Gini Impurity

- The general formula for Gini impurity when there are c classes is:

$$G(S) = \sum_{i=1}^{c} p_i \sum_{k \neq i} p_k$$

- Often easier to apply as:

$$G(S) = 1 - \sum_{i=1}^{c} p_i^2$$

- Now, if we just have two classes:

$$G(S) = 1 - \left(\left(\frac{p}{p+n} \right)^2 + \left(\frac{n}{p+n} \right)^2 \right)$$

KING'S College LONDON

Gini Impurity

- If we then have a feature A that splits S into subsets S_i, the Gini impurity of these sets is:

$$G(S, A) = \sum_i \frac{|S_i|}{|S|} G(S_i)$$

  (here $|S_i|$ indicates the size of S_i, the number of things it contains).
- To pick a feature we calculate $G(S, A)$ for each feature.
- The feature with the lowest Gini impurity is the best choice.

KING'S College LONDON

Overfitting

- DTL grows the tree just enough to perfectly classify all the examples.
- If the set is too small — or the set contains noise — then this can easily lead to overfitting.
- Given two decision trees d and d', d is said to overfit the training data if:
  1. d has a smaller error than d' on the training data.
  2. d' has a smaller error that d on all the other instances.
- Of course, this is not a very practical definition since you need somehow to know what all the possible examples are.

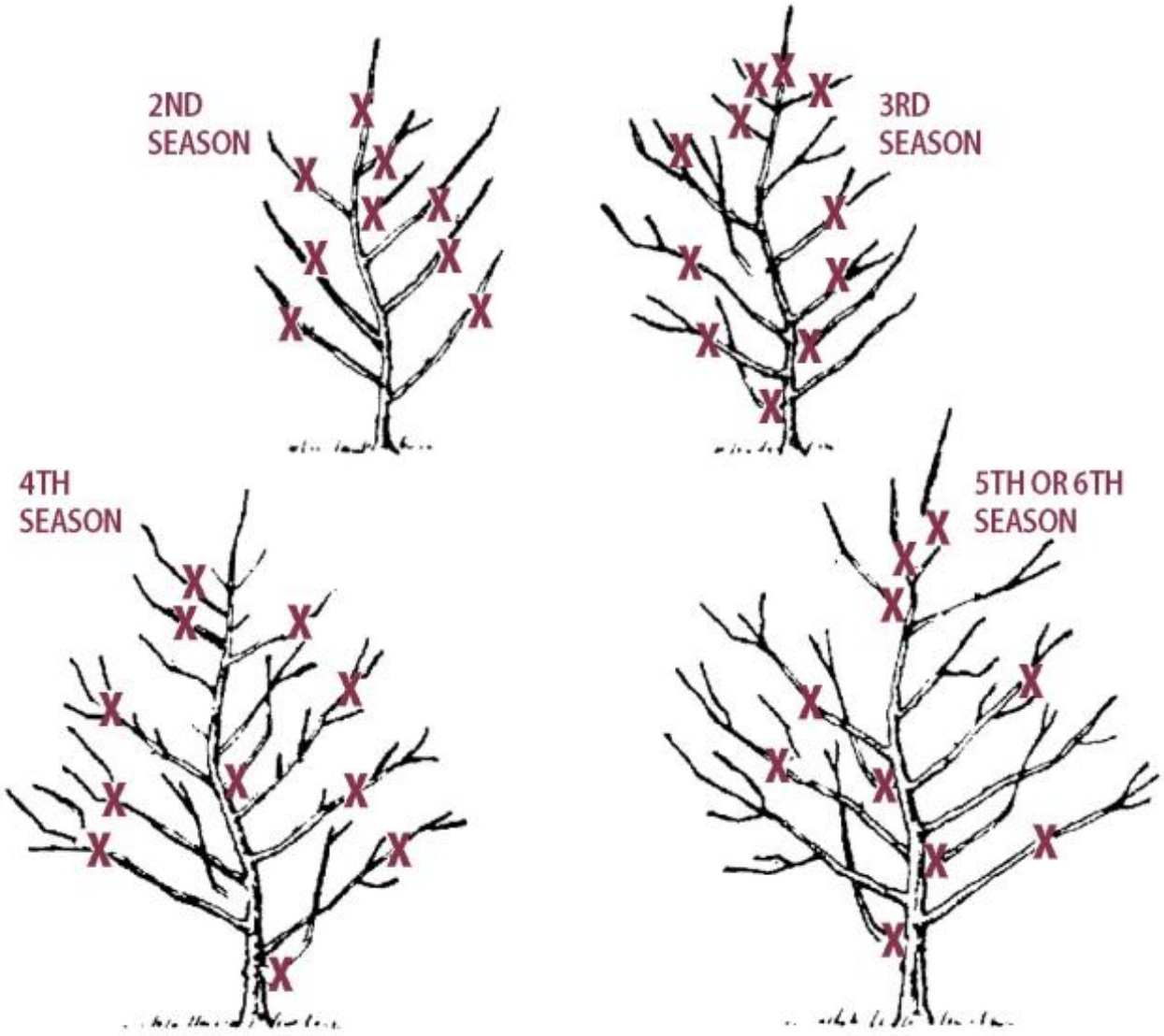
Overfitting

- Two main approaches to preventing overfitting.
  1. Stop growing the tree earlier.
     - Don't get to the point of overfitting.
  2. Allow to (possibly) overfit, and then prune the tree.
- The second seems more successful.

Overfitting

- And, all of this assumes that we do the usual data splits / validation that we covered last time.

Reduced error pruning



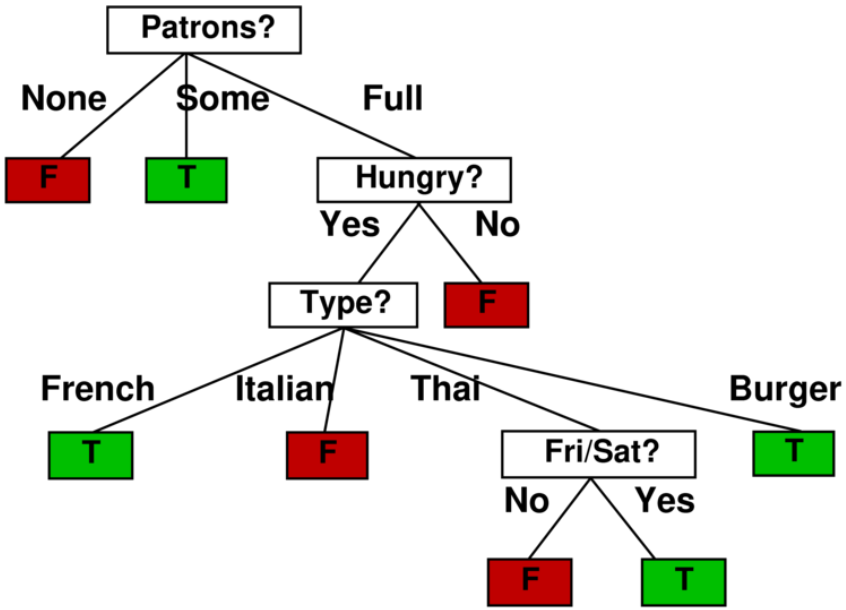
(starkbros.com)

Reduced error pruning

- Consider every branch in the tree as a candidate for pruning.
- Remove the branch, and make the root of the branch into a leaf.
- Use the plurality classification for examples at that leaf.
- Test the new tree.
- Keep a branch pruned if the pruned tree performs better on some validation set than the original tree.
- Repeat while it is possible to prune a branch and improve performance.

Reduced error pruning

- What does it mean on this tree?



(Russell & Norvig)

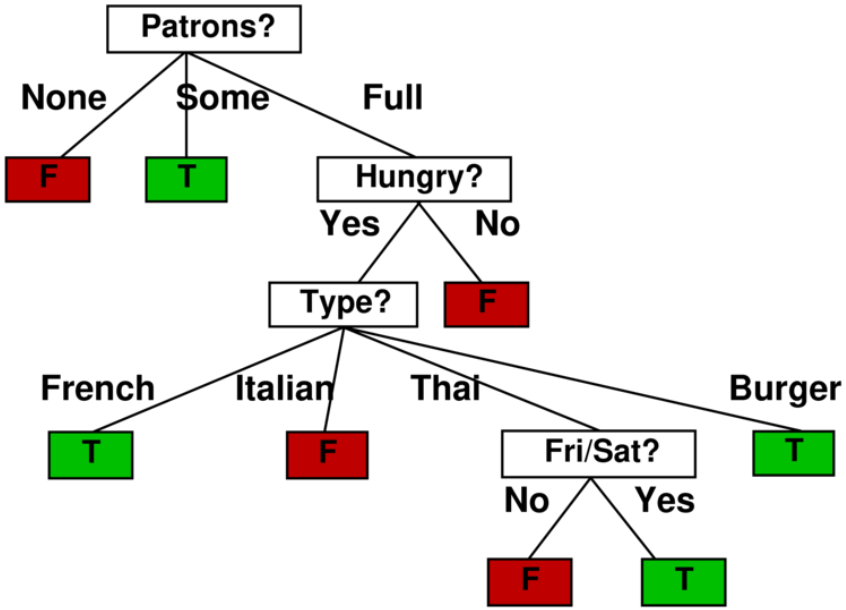
Reduced error pruning

- Requires another partition of the data:
- Now need:
  1. Training data
  2. Pruning data (development set / validation set)
  3. Test data
- When data is limited, this is a problem.

Rule post-pruning

- Solution to the “pruning with limited data” problem.
  1. Build the decision tree as before.
  2. Create a rule set
     One rule for each path from root to leaf.
  3. Remove preconditions (feature tests) from rules if that improves their accuracy.
  4. Sort rules by accuracy and apply them in order/sequence when classifying examples.
- Let’s see how this works on the restaurant example.

Rule post-pruning



(Russell & Norvig)

- If ($Patrons? = None$)
  Then $\neg Wait$
- If ($Patrons? = Full$) and ($Hungry? = No$)
  Then $\neg Wait$

Rule post-pruning

- Removing preconditions means, for example, trying the second rule
  'If ($Patrons? = Full$) and ($Hungry? = No$)'
  without:
  $$Patrons = Full$$
  $$Hungry = No$$
  and seeing if this improves performance.
- Advantage of rules is that pruning is more flexible than pruning branches.
- Can remove part of a branch.
- Can remove root while keeping lower branches.

Broadening decision tree approach

- Multivalued features
  - When features have many values, information gain gives an inappropriate estimation of the usefulness of the feature. Tend to split examples into small classes (ie. ExactTime)
  - Convert to Boolean tests.
- Continuous/integer input features
  - Infinite sets of possible values.
  - Modify approach to identify split points which give highest information gain.
    $Weight > 160$
- Continuous output values
  - When trying to predict continuous output values need to create a regression tree, which ends with a linear function.

KING'S College LONDON

DTL variants

- The basic algorithm given above is the core of the ID3 algorithm.
- ID3 includes using entropy to make the choice about which feature to split on.
- C4.5 extended ID3, adding post-processing to simplify the tree.

DTL variants

- CART is a similar method, with a complex relationship with ID3/C4.5:
  The historical lineage of C4.5 offers an interesting study into how different sub-communities converged on more or less like-minded solutions to classification. ID3 was developed independently of the original tree induction algorithm developed by Friedman [13], which later evolved into CART [4] with the participation of Breiman, Olshen, and Stone. But, from the numerous references to CART in [30], the design decisions underlying C4.5 appear to have been influenced by (to improve upon) how CART resolved similar issues, such as procedures for handling special types of attributes.

  (The Top 10 Algorithms in Data Mining)

Linear regression

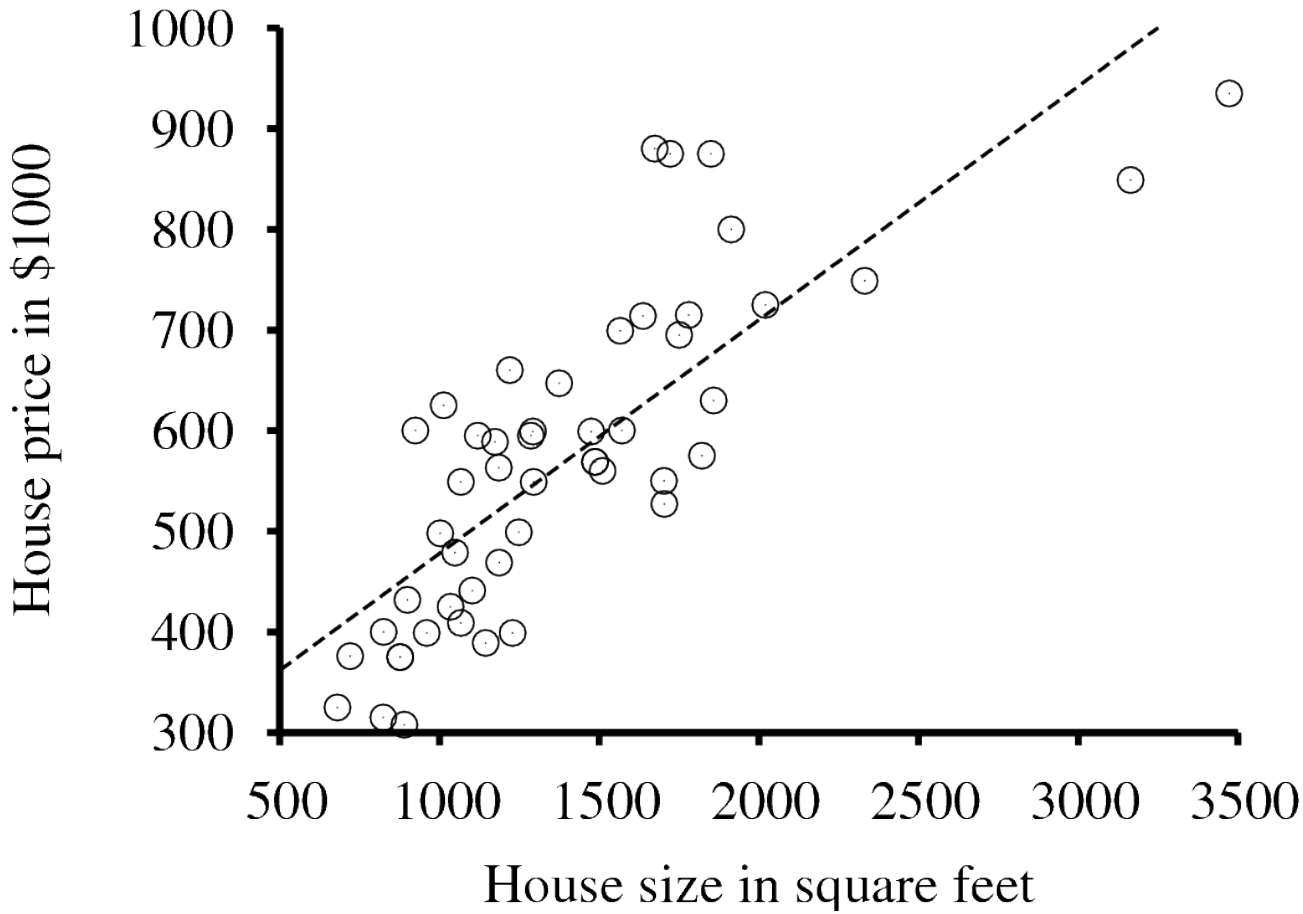
- Learning a linear function of continuous inputs.
- Equation is of the form (univariate):

$$h_w(x) = w_1 x + w_0$$

  where the w subscript indicates the vector $[w_0, w_1]$.
- Idea is that we want to estimate the values of w_0 and w_1 from data.

Linear regression

Example: predicting house prices by floor area.



(Russell & Norvig)

Linear regression

- Finding the h_w that best fits the data is linear regression (choose w so that h_w is close to y for our training examples).
- To fit the line, we find the $[w_0, w_1]$ that minimize the loss/error.
- Traditionally we use the squared loss function L_2:

$$\begin{aligned} J(h_w) = Loss(h_w) &= \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) \\ &= \sum_{j=1}^{N} (y_j - h_w(x_j))^2 \\ &= \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 \end{aligned}$$

summed over all the training examples, and where the data we have are pairs (x_i, y_i). (aka cost function)

- So we would like to find (optimisation objective):

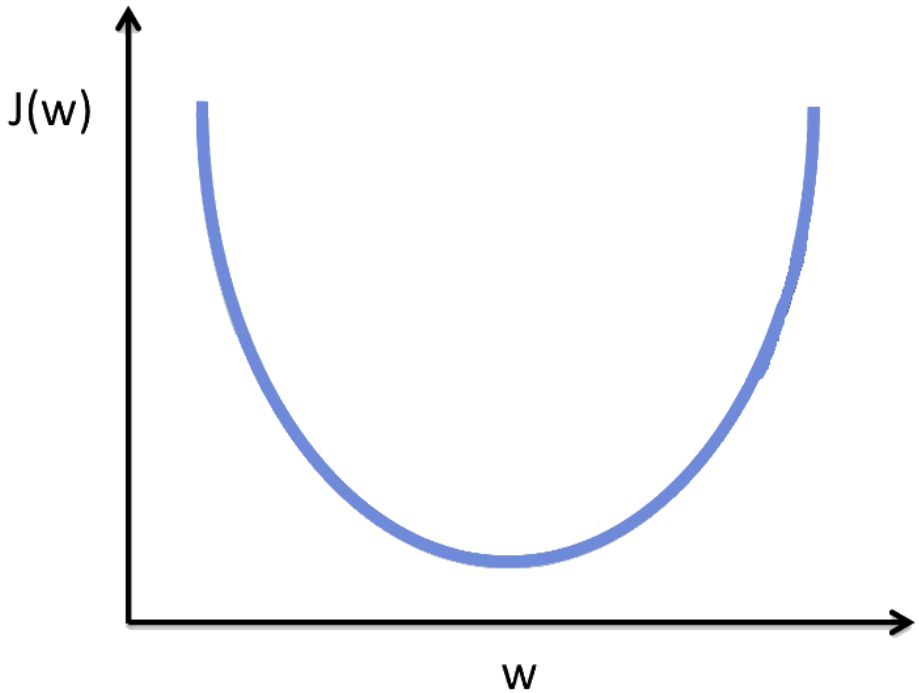
$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} Loss(h_{\mathbf{w}})$$

Linear regression

- We use the squared loss function because Gauss showed that for normally distributed noise, this gives us the most likely values of the weights.

Linear regression: cost function

- Plotting the lost / cost function (for simplicity, $h_w(x) = w_1 x$).
- For the house price case the loss function looks like (assume we remove w_0 for now and $w = w_1$):

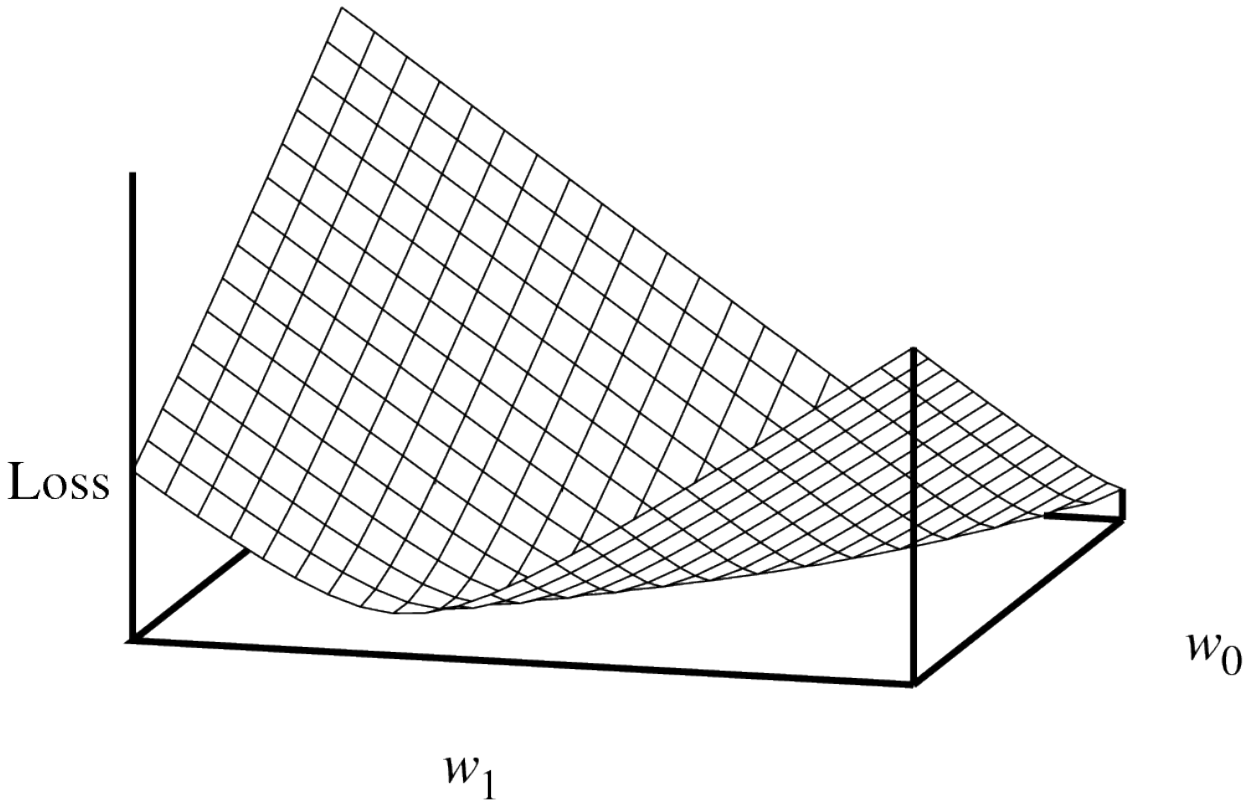


(https://rasbt.github.io)

- Each value of w corresponds to a different function h.
- Which w shall we choose?

Linear regression: cost function

- For the house price case the loss function looks like:



(Russell & Norvig)

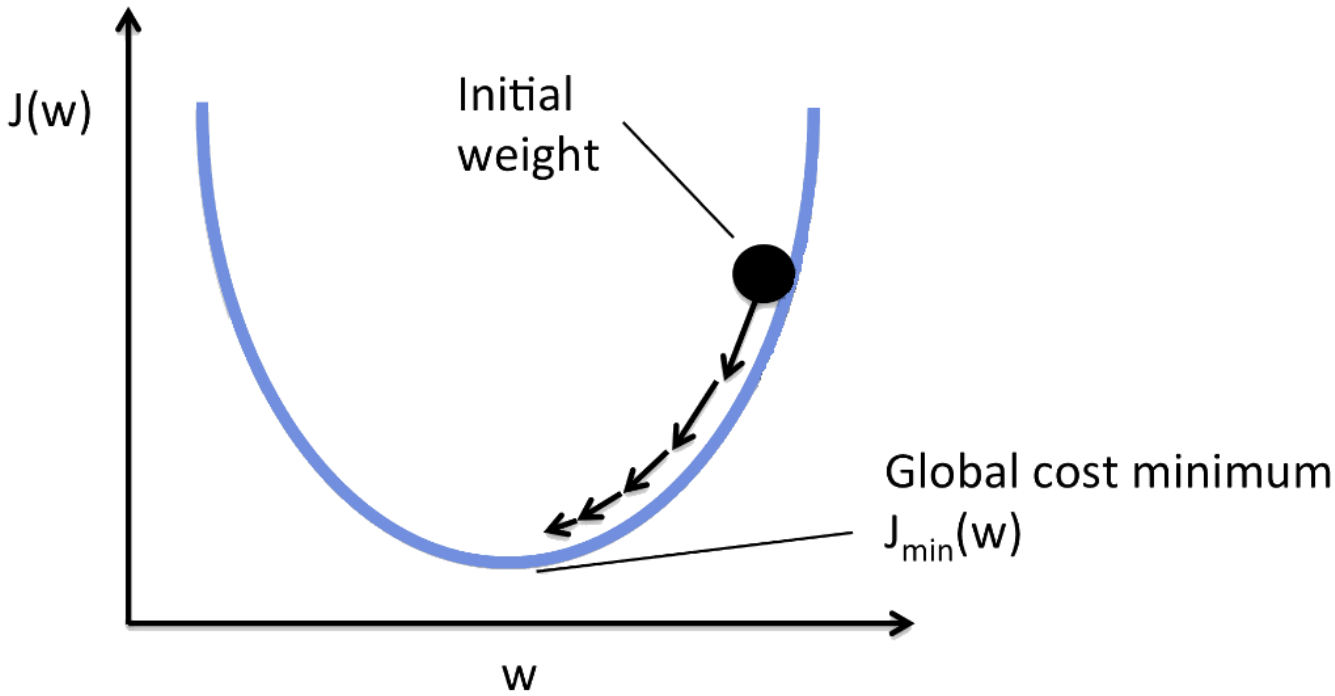
Gradient descent algorithm

Outline:

- Start with some initial values for w_0, w_1.
- Keep changing these values so that the cost / loss function $J(h_w)$ is reduced.
- Continue this until you reach a minimum.

Gradient descent

- For the house price case the loss function looks like (assume we remove w_0 for now and $w = w_1$):



(https://rasbt.github.io)

Gradient descent

- Start at any point in the (w_0, w_1) plane and move to a neighbouring point that is downhill.
- For each w_i we update with:

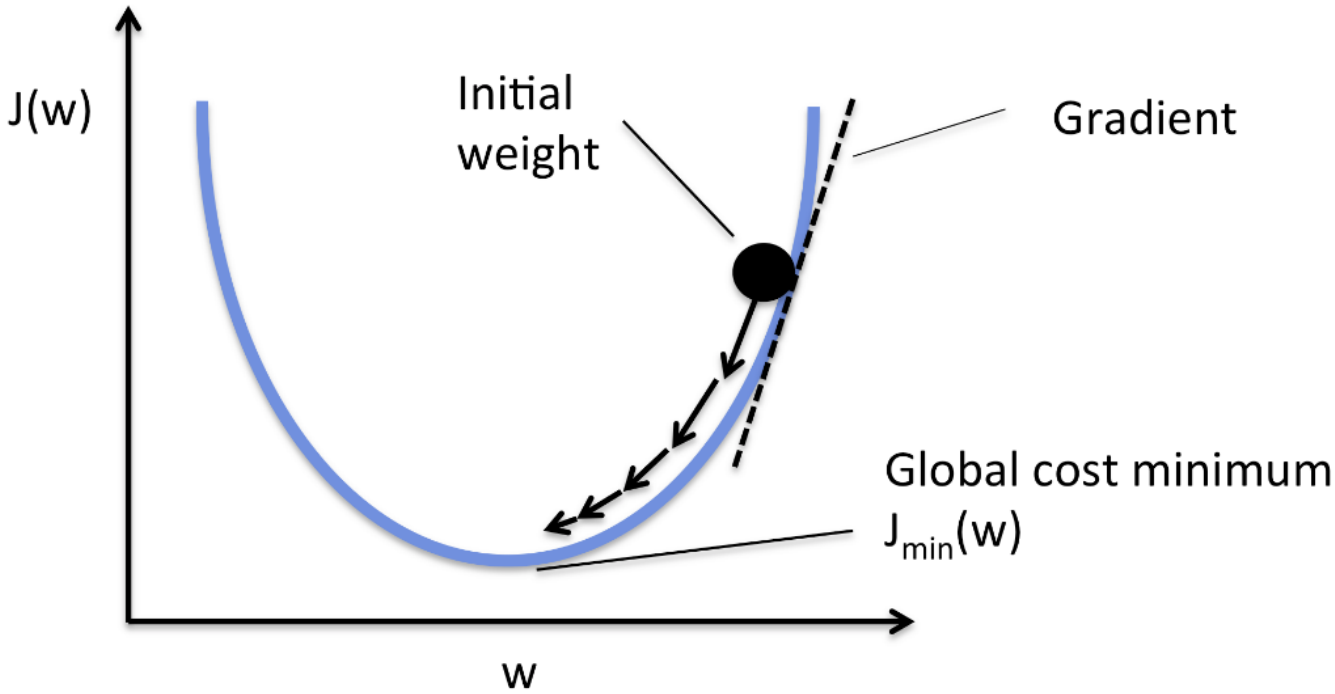
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$$

  where α is the learning rate and controls how big a step we take downhill.
- Just repeat until convergence.
- Note: must simultaneously update the w.
- (Gradient descent algorithm can be used for other functions/problems too).

KING'S College LONDON

Gradient descent

- For the house price case the loss function looks like (assume we remove w_0 for now and $w = w_1$):



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

(https://rasbt.github.io)

Linear regression

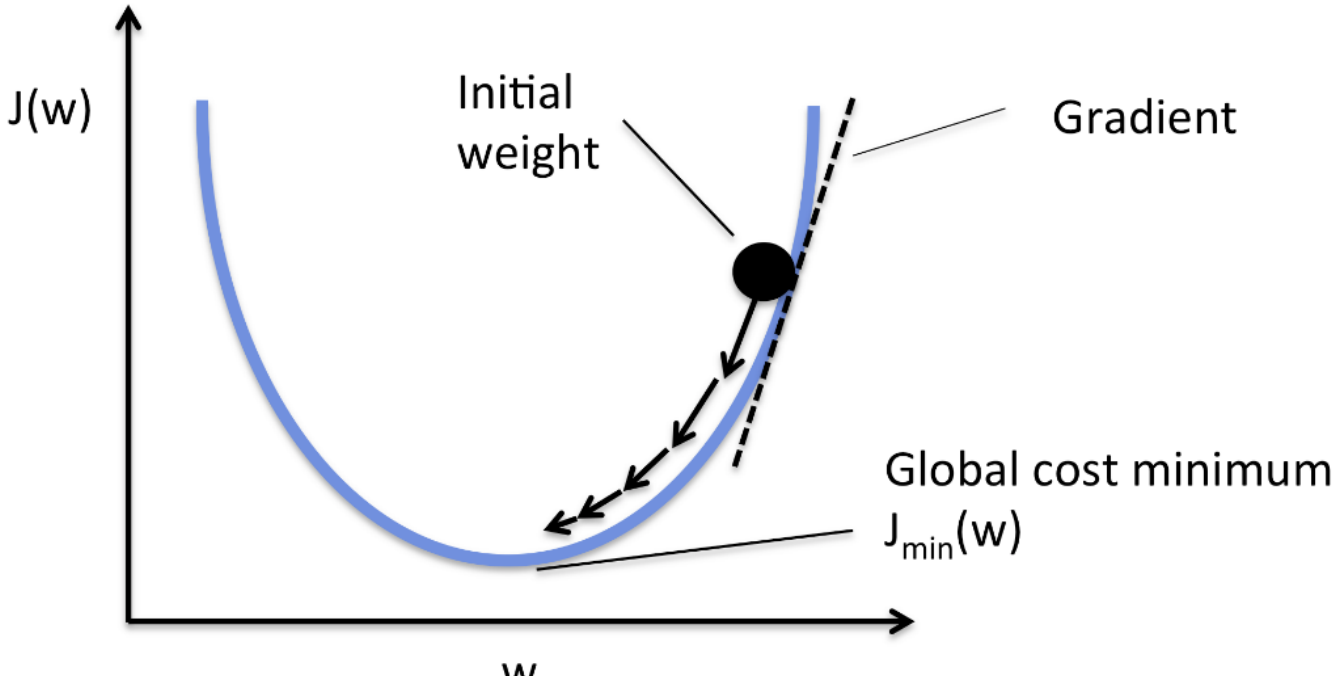
- The slope at the minimum is 0 which means $\frac{\partial}{\partial w_i} Loss(w) = 0$.
- Essentially the sum below is minimized when its partial derivatives with respect to w_0 and w_1 are zero:

$$
\begin{aligned}
J(h_w) = Loss(h_w) &= \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) \\
&= \sum_{j=1}^{N} (y_j - h_w(x_j))^2 \\
&= \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2
\end{aligned}
$$

Learning rate

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

- α is the learning rate and controls how big a step we take downhill.
- α too small, then tiny steps downhill (slow).
- α too large, we might miss the minimum and may not converge.



Gradient descent

- Batch gradient descent vs. stochastic gradient descent.

Batch gradient descent

- Consider all the training examples simultaneously.
- At each step we consider all the training examples, and update the weights using:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) x_j$$

- This is guaranteed to converge.
- But it can be slow since we need to compute for all N examples at each step.

Stochastic gradient descent

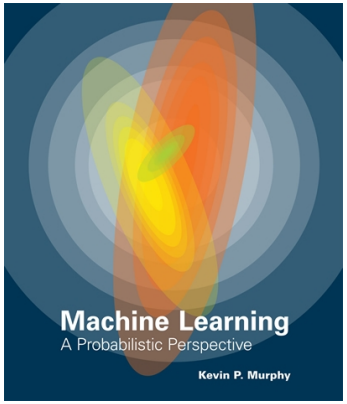
- We can just do:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(y - h_w(x)) \\ w_1 &\leftarrow w_1 + \alpha(y - h_w(x))x \end{aligned}$$

  For each of the N examples in turn.
- Often quicker.
- If learning rate is constant, may not converge.
- Can often be made to converge by decreasing the learning rate over time.

More sophisticated gradient descent

- Techniques exist to estimate the best step size/learning rate.
- See Murphy, pages 245–252.



Multivariate linear regression

- Now we have more variables:

$$x_{j,1}, \ldots, x_{j,i}, \ldots x_{j,n}$$

  and are interested in a vector of weights w_i.
- Simplify the handling of the weights by creating a dummy attribute to pair with w_0.

$$x_{j,0} = 1$$

- Then $h_w(x_j)$ is just the weighted sum of the variable values:

$$h_w(x_j) = \sum_{i=0}^{i=n} w_i x_{j,i}$$

KING'S College LONDON

Multivariate linear regression

- We learn by doing gradient descent, as before.
- Batch gradient descent:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_w(x_j)) x_{j,i}$$

- Stochastic gradient descent:

$$w_i \leftarrow w_i + \alpha (y_j - h_w(x_j)) x_{j,i}$$

- Not really much harder than the univariate case.
- We just adjust more weights each time.

KING'S College LONDON

Multivariate linear regression

- BUT, have to worry about overfitting.
- Take the complexity of the model into account.

Multivariate linear regression

- BUT, have to worry about overfitting.
- Take the complexity of the model into account.
- We add a regularisation term to the loss function:

$$Loss'(h) = Loss(h) + \lambda Complexity(h)$$

where

$$Complexity(h_{\mathbf{w}}) = \sum_i w_i^2$$

Intuition: smaller w will give simpler functions $h_{\mathbf{w}}$.
(e.g., consider $w_0 + w_1 x + w_2 x^2 + w_3 x^3$)

KING'S College LONDON

Multivariate linear regression

- BUT, have to worry about overfitting.
- Take the complexity of the model into account.
- We add a regularisation term to the loss function:

$$Loss'(h) = Loss(h) + \lambda Complexity(h)$$

where

$$Complexity(h_{\mathbf{w}}) = \sum_i w_i^2$$

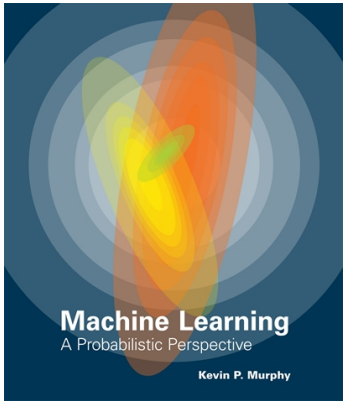
Intuition: smaller w will give simpler functions $h_{\mathbf{w}}$.
(e.g., consider $w_0 + w_1 x + w_2 x^2 + w_3 x^3$)

- λ is the regularisation parameter: trade-off between fitting the data and having a simple function. What if too high?
- Then use $Loss'(h)$ when computing the weight updates.

KING'S College LONDON

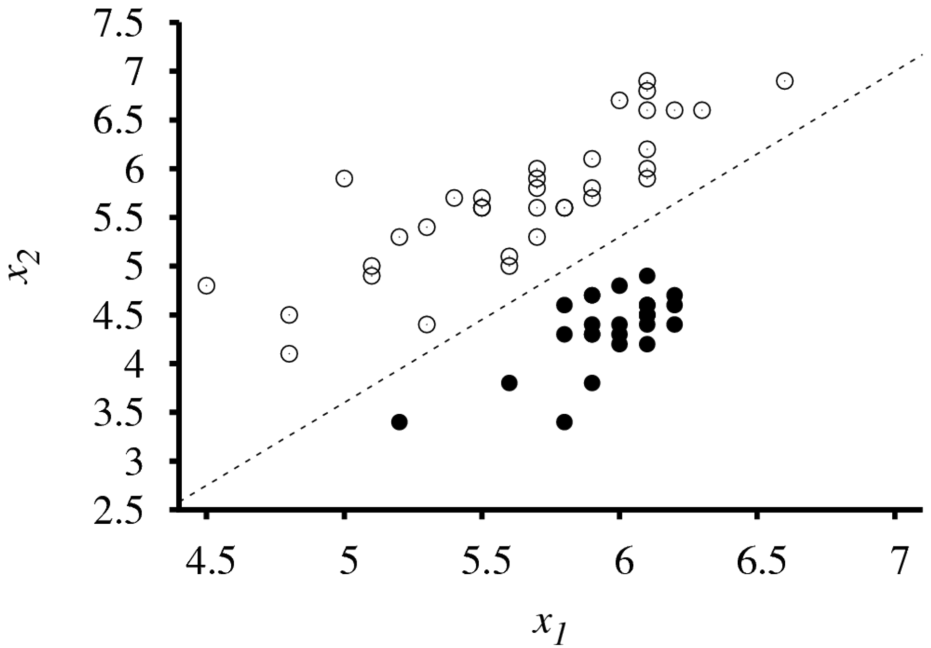
More on regularization

- Again, there is a battery of more sophisticated methods.
- See Murphy, pages 225–228, 252.



Linear classifiers

- Can turn a linear function into a classifier.
- Function defines the decision boundary that seperates the two classes.



(Russell & Norvig)

- A linear decision boundary will separate two linearly separable classes.
- Classify based on where a point lies in relation to the line.

KING'S College LONDON

Linear classifiers

- In the above example (seismic data due to earthquakes and nuclear explosions) the linear separator is:

$$x_2 = 1.7x_1 - 4.9 \;\; OR \;\; -4.9 + 1.7x_1 - x_2 = 0$$

- Explosions are to the right of the line (points for which the function is > 0):

$$-4.9 + 1.7x_1 - x_2 > 0$$

- Thus we do classification as follows:

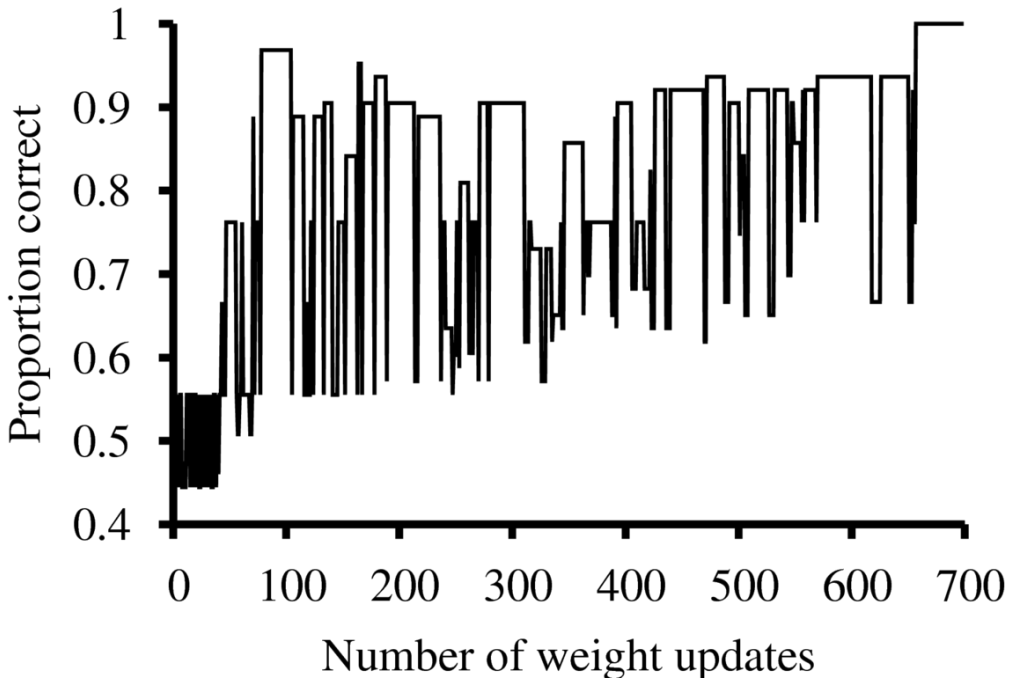
$$h_w(x_j) = 1 \; if \sum_{i=0}^{i=n} w_i x_{j,i} > 0$$

otherwise the classifier returns 0: $h_w(x_j) = 0$

Linear classifiers

- Learn the decision boundary just as we learnt the linear function.
- Starting with arbitrary weights
  1. Use the decision rule (as in previous slide) on an example.
  2. If it classifies correctly, do not update weights.
  3. Otherwise update weights as before.
- It is common to use stochastic gradient descent (one example per update) here.

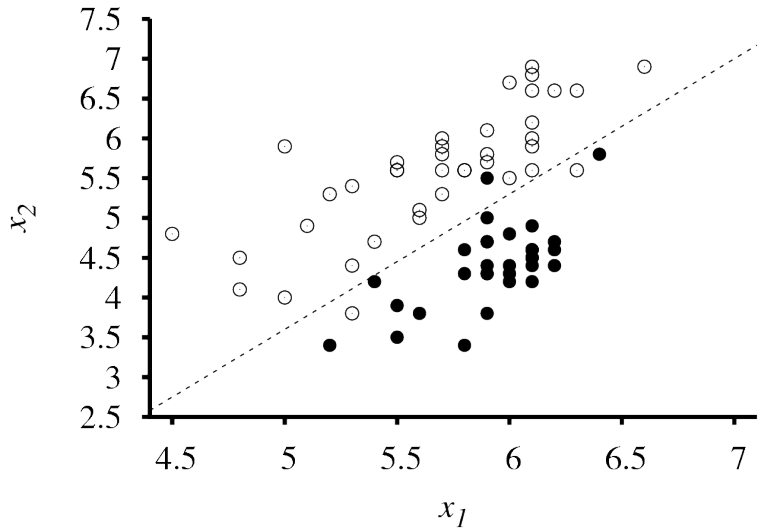
Linear classifiers: learning curve

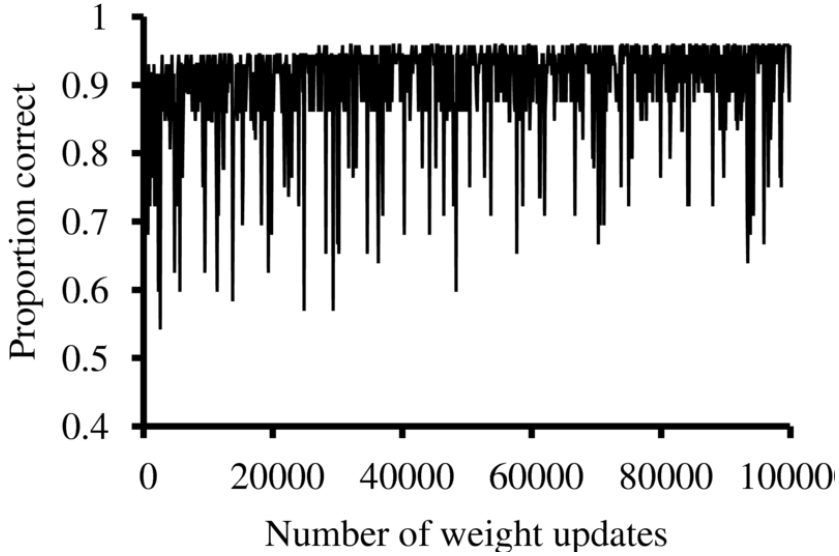


(Russell & Norvig)

- Typically, the variation across runs is very large.
- The curve is not smooth because the boundary is hard, so can misclassify a lot of examples even a long way into learning.

Linear classifiers

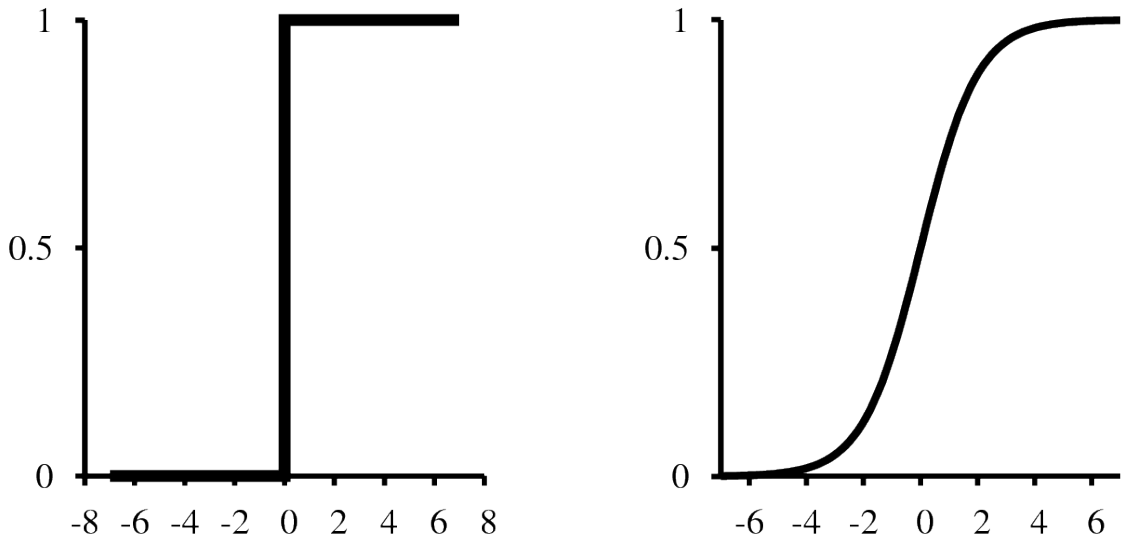




(Russell & Norvig)

- Worse if the data is noisy.

Linear classifiers: logistic regression



(Russell & Norvig)

- The linear classifier always predicts 0 or 1, even for examples that are very close to the boundary.
- In many tasks, we need more gradated predictions.
- Soften the threshold function: approximate the hard threshold with a continuous function.
- Use the logistic function as a threshold:
  $h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$

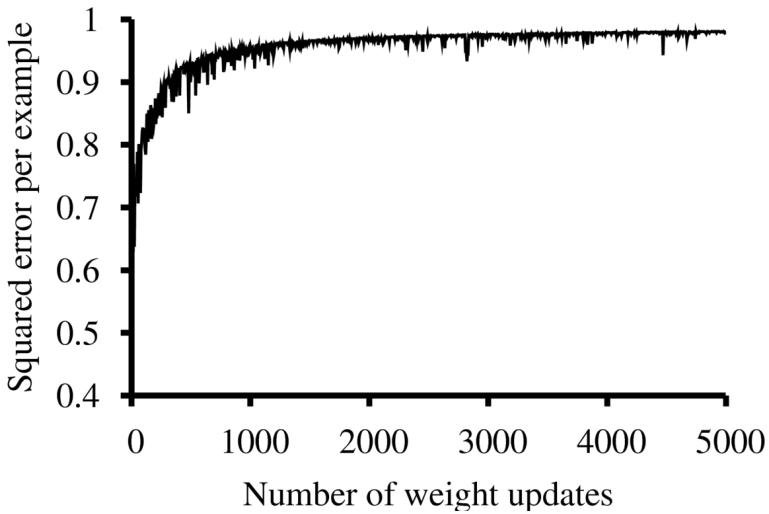
Linear classifiers: logistic regression

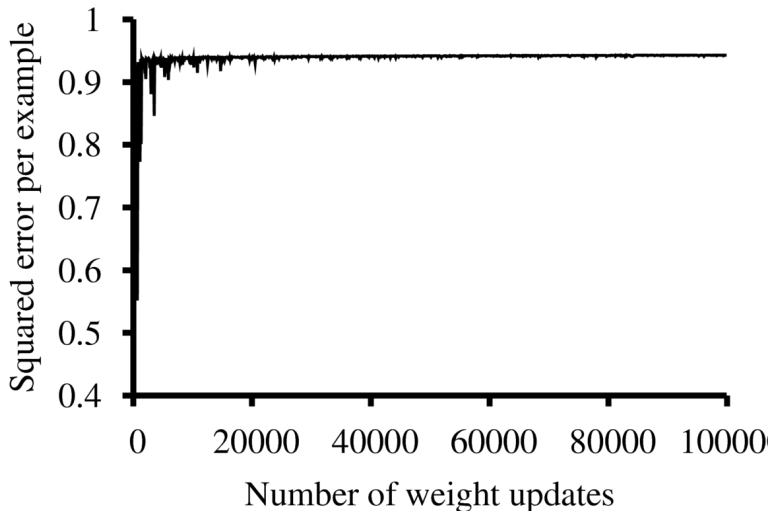
- The logistic function taxes one's calculus a little, but the update rule is pretty simple (commonly uses **cross-entropy loss** function).

$$w_i \leftarrow w_i + \alpha(y - h_w(x)).h_w(x)(1 - h_w(x)).x_i$$

- Logistic regression

Linear classifiers

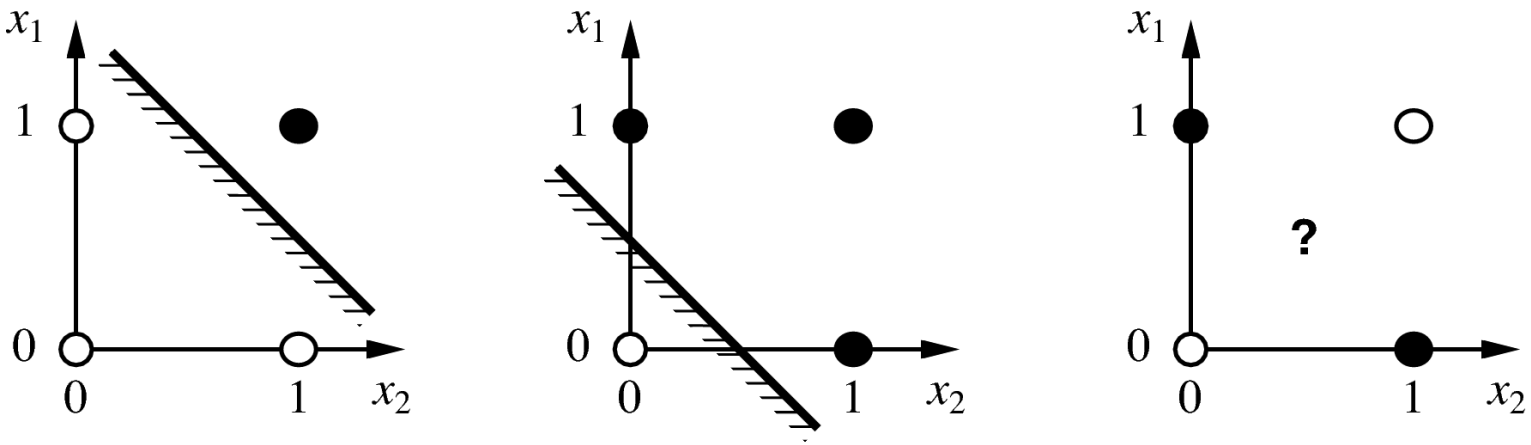




(Russell & Norvig)

- Convergence is slower, but behaves much more predictably.

Issues with linear separability



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

(Russell & Norvig)

- As a result, there is lots of work on methods for non-linear boundaries.
  - Neural networks/deep learning.
  - Support vector machines
- Note that decision trees can handle XOR functions.

Ensemble learning

- Every classifier has an error rate
  - Will always misclassify some examples.
- Using an ensemble is an easy way to improve on this.
- Take *N* classifiers, use them all on the same example.
- Have them vote on the classification.
- For a binary classification and 5 classifiers, error rate drops from 10% (say) to less than 1%.
  Assuming that the classifiers are independent
  (i.e. different enough).

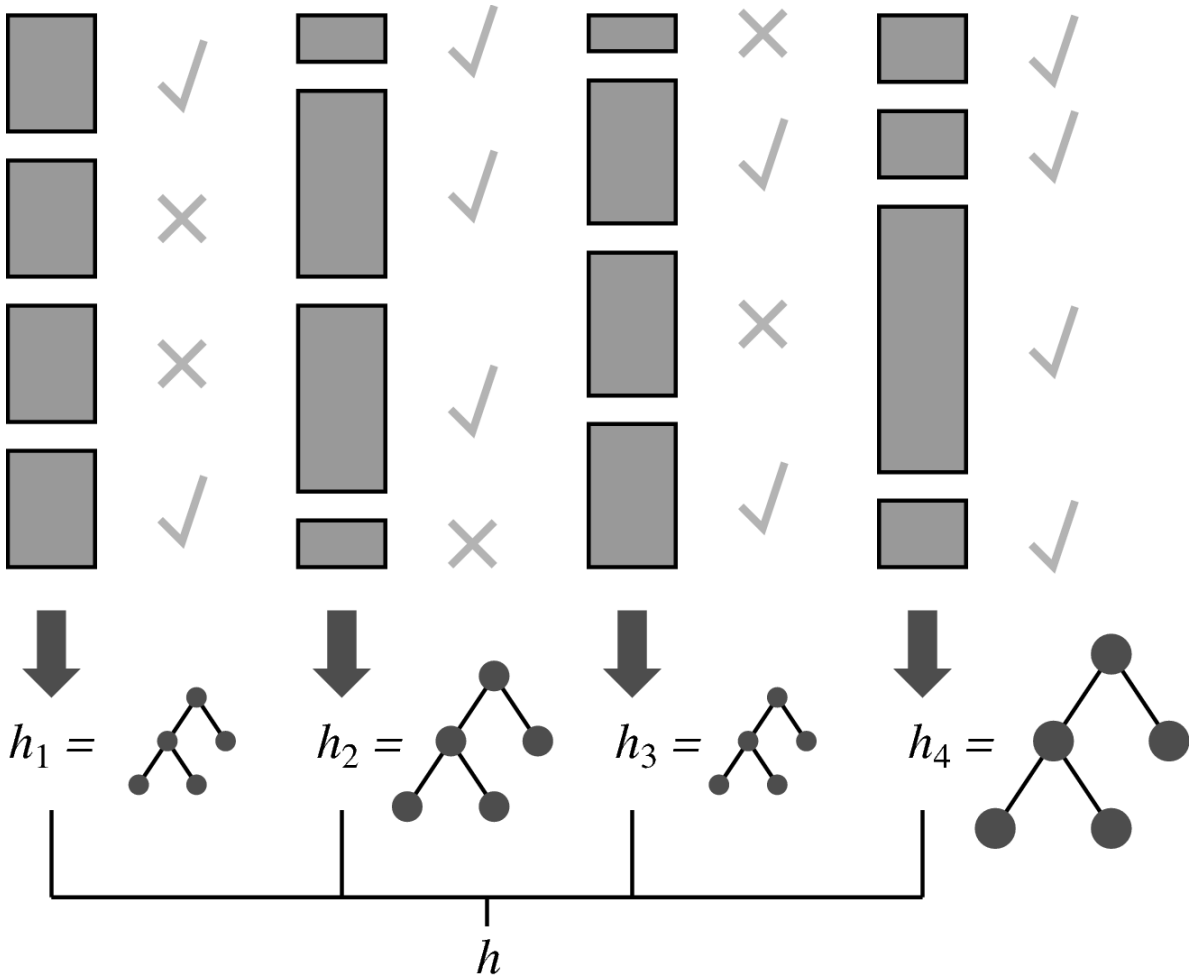
Boosting

- Boosting extends the idea of a simple ensemble.
- Builds on the idea of a weighted training set
- Higher weighted examples are counted as more important during training.
- (For example we put more copies into the training set)

Boosting

- Starts with all examples of equal weight, and learns a classifier h_1.
- Test it.
- Increase the weights of the misclassified examples and learn a new classifier h_2.
- Repeat.
- Final ensemble is the combination of all the classifers, weighted by how well they perform on the training set.

Boosting



(Russell & Norvig)

AdaBoost

- Many variants of the basic boosting idea.
- The AdaBoost algorithm is a commonly used approach to boosting.
- Given an initial classifier that is slightly better than random (weak model), AdaBoost can generate an ensemble that will perfectly classify the training set.

Bagging

- "Bootstrap aggregation"
- From a training set $\mathcal{D}$, we create multiple training sets:

$$\mathcal{D}_1, \ldots \mathcal{D}_n$$

- From each of these we learn a classifier.
- To classify an example, combine the results of the ensemble.
- The $\mathcal{D}_i$ are sampled from $\mathcal{D}$
  Uniform distribution, with replacement.
- Results in multiple sets with the same properties.

Combining results

- Typical combination for classifiers is by voting.
- Can also combine regression ensembles, by averaging.
- Both can be weighted (for boosting).

Random forests

- Bagging, applied to decision trees.
- So, a set of decision trees learnt from bagged training sets.
- PLUS, random selection of features for each tree.
  Features are selected randomly, with replacement (between trees).
  So, bagging at the feature level.

Random subspace method

- This last bit:
  - . . . random selection of features for each tree.
  - Features are selected randomly, with replacement (between trees).
  - So, bagging at the feature level.

  is called the *random subspace method*
- Can apply it to other kinds of ensemble.

Summary

- Decision trees
  - Learning
  - Entropy
  - Other metrics
  - Pruning
- Linear regression
  - Learning
- Linear classifiers
  - Logistic function
- Ensemble methods
  - Boosting
  - Bagging
  - Random forests