Forms of Learning: F(features) $\rightarrow$ output

There are three types of feedback that can accompany the inputs, and that determine the three main types of learning:

- Supervised Learning
  - agent observes input-output pairs
  - learns a function that maps from input to output
- Unsupervised Learning
  - agent learns patterns in the input without any explicit feedback
  - clustering
- Reinforcement Learning (Next major topic)
  - agent learns from a series of reinforcements: rewards & punishments

- Many variations and combinations

Example Algorithms

Supervised Learning

- K-Nearest Neighbors
- Linear and Logistic Regression
- Support Vector Machines
- Decision Trees and Random Forests

Unsupervised Learning

- Clustering
  - K-means
  - Hierachical Cluster Analysis (HCA)
- Visualization and Dimension Reduction
  - Principal Components Analysis

Supervised Learning: Classification

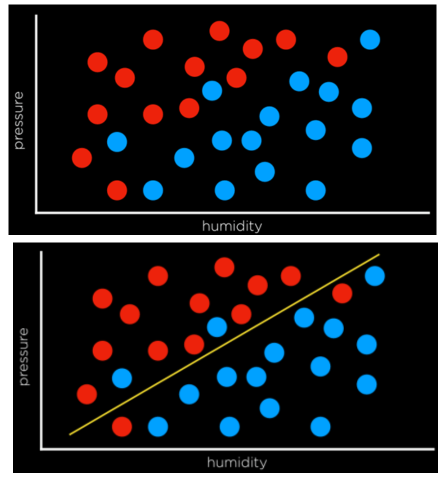
- Given some data with certain characteristics (feature vector) where each data point is associated with one of a set of discrete classes
- Classification: Develop a function (a classifier) that compute the class to which a data point belongs.
  Classifier f : {feature vectors} $\rightarrow$ {class values}
- Examples,
  - *Grading fruit* for sale at different prices
  - Determining the correct interpretation of a handwritten digit

First Step: Exploratory Data Analysis (EDA)

- Many approaches and algorithms, want to find the best (few) models and algorithms for determining what the data can tell you about your problem
- Plots, graphs, visualization, etc.
- Decide on most promising ways to model and analyze the data. May use one or more models and analysis approaches
- Need to evaluate and pick from many algorithms, may pick more than one

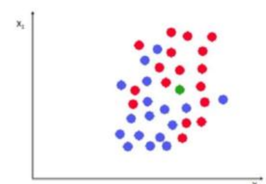
Nearest Neighbor Classification

Idea: Classify new data by checking the neighborhood



K Nearest Neighbor (KNN) Classification

- Idea: Classified by “MAJORITY VOTES” for its neighbor classes
  - Assigned to the most common class amongst its K-nearest neighbors (by measuring “distance” between data)



Ref: https://www.slideshare.net/tilanigunawardena/k-nearest-neighbors

Nearest Neighbor Classification Algorithm:

- Specify a positive integer k (**<u>odd</u>**) and given a new sample (feature vector)
- Select the k entries in our database which are closest to the new sample
- Find the most common classification of these entries
- This is the classification we give to the new sample

Perceptron (or McCulloch-Pitts neuron)

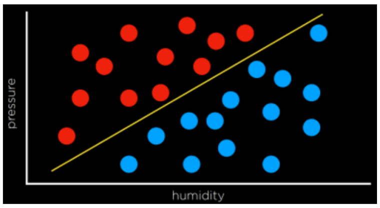
- In machine learning, the perceptron (or McCulloch-Pitts neuron) is an algorithm for supervised learning of binary classifiers.
- A Binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.[1]
- It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Perceptron Learning

- Looking the whole data set create a decision boundary.
- In two-dimensional data, draw a line between the two types of observations.
- New data points will be classified based on the side of the line on which it is plotted.

Example:

- Features Humidity and Pressure
- Classify as Rain, No Rain



- Almost always will misclassify some points since boundaries are seldom linear and data is noisy.

Sample Data

Date	Humidity	Pressure	Rain
January 1	93%	999.7	Rain
January 2	49%	1015.5	No Rain
January 3	79%	1031.1	No Rain
January 4	65%	984.9	Rain
January 5	90%	975.2	Rain

Model: x_1= Humidity x_2 = Pressure

Rain: Rrain or No Rain

$h_w(x) = w_0 + w_1 x_1 + w_2 x_2 \geq 0$ --- Rain

Perceptron Model: Linear separation (hypothesis function)

x_1 = Humidity x_2 = Pressure

want $h(x_1, x_2)$ = Rain if $w_0 + w_1 x_1 + w_2 x_2 \geq 0$

No Rain otherwise

Weight Vector w: (w_0, w_1, w_2) **Input Vector $\bar{\mathbf{x}}$** : $(1, x_1, x_2)$

Hypothesis Function:

$h(x_1, x_2) = 1$ if $w_0 + w_1 x_1 + w_2 x_2 \geq 0$ Rain

0 otherwise Not Rain

Perceptron Learning Rule: $w_{i,new} = w_{i,old} + \alpha(y - h(x)) * x_i$

Perceptron Model: Linear separation
n inputs --- separated by a n-1 dimensional plane

Weight Vector $\vec{\boldsymbol{w}}$: (w_0, w_1, w_2, …, w_n)
Input Vector $\vec{\boldsymbol{x}}$: (1, x_1, x_2 , …, x_n)

$\vec{w} \cdot \vec{x} \geq 0$ dot product: $\vec{w} \cdot \vec{x} = \sum_{i=0}^{n} w_i * x_i$

$h_w(\vec{x}) = 1$ if $\vec{w} \cdot \vec{x} \geq 0$ Rain
0 otherwise Not Rain

Perceptron Learning Rule: Learn from the data points one at a time to improve h

$w_{i\ new} = w_{i\ old} + \alpha$(actual value – estimated value) * x_i

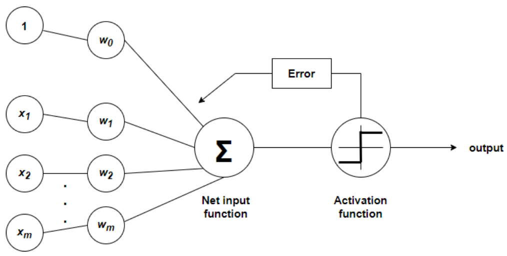
$w_{i\ new} = w_{i\ old} + \alpha(y - h(\vec{\boldsymbol{x}})) * x_i$ **α: learning rate**

Example: Perceptron Learning

- Initialization of Weights: The process begins with assigning random weights to the inputs.
- Output Calculation: Inputs are multiplied by their respective weights, summed, and passed through the activation function to produce an output.
- Output Comparison: The predicted output is compared with the actual output.
- Weight Adjustment: In the event of misclassification, weights are adjusted accordingly.
- Iteration: Steps 2–4 are repeated until the perceptron accurately classifies the inputs.

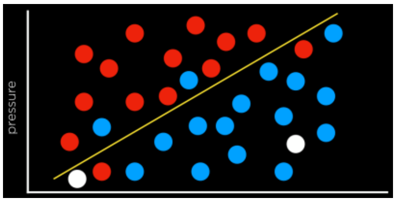
Perceptron Model: Limitations

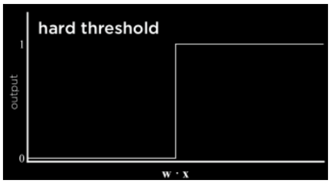
- Only allows for linear separation

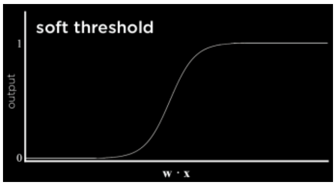


Activation or threshold function

Different functions possible: want to soften impact of data points that are close to boundary: strength of belief and easier for calculations later





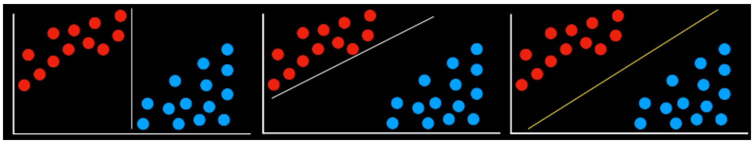


Perceptron is a type of artificial neural network component

- **Input Layer**: The input layer consists of one or more input neurons, which receive input signals.
- **Weights**: Each neuron is associated with weights, which represents the strength of the connection between the input and the output.
- **Bias**: A bias term (constant) is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.
- **Activation Function**: The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias term.
- **Output**: The output of the perceptron is a single binary value, either 0 or 1, which indicates the class or category to which the input data belongs.
- **Training Algorithm**: The perceptron is typically trained using a supervised learning algorithm such as the perceptron learning algorithm or backpropagation. During training, the weights and biases of the perceptron are adjusted to minimize the error between the predicted output and the true output for a given set of training examples.

Support Vector Machines, SVMs

- Many lines may classify correctly. But are they equally good?
- The third line below seems better, that is the idea behind SVMs



- Idea: Find a boundary, which is as far as possible from the two groups it separates, called the **Maximum Margin Separator**.
- Works by finding separating hyperplane in a higher dimensional space
- Allows for non-linear boundaries using kernel functions (e.g polynomials, logistic,..)

Linear Regression

- Shift in focus to a function of **continuous** inputs that results in a **continuous value** rather than a *small* set of discrete values.
- Examples:
  - Market share as function of marketing expenditures
  - Employment as function of federal funds rate
  - Ocean temperature as function of CO_2 concentration in the atmosphere
  - Home prices as function of lot size, neighborhood, age, house size,...

Loss Functions

- Loss Function: A function that quantifies the difference between predicted and actual values. It guides the optimization process by providing feedback on how well it fits the data.
- Examples:
  - Classification Loss Function: Cross-Entropy/Logistic Loss (CE) Measures the distance from the actual class to the predicted value, which is usually a real number between 0 and 1
  - Classic Regression Loss function: Mean Square Error has a closed form solution. (that is a simple matrix equation that computes h)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Linear Regression and Classification: Gradient descent

- search through a continuous weight space by incrementally modifying the parameters (minimizing loss)
- α: step size/learning rate that can be a fixed constant or decay over time

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

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

Decision Trees: Problem Setting

Idea:

1. Develop a set of questions based on a set of **features** that are believed to influence or cause a label or an action.
2. Organize the questions as a tree that will result in the label or action
3. The tree will determine a target function *f: X → Y where*
   - *X* = {feature vectors}
   - *Y* = {class values/labels/actions}
   - *f: X → Y* target function

Decision Tree Definition

- Set of possible instances X = {feature vectors}
- Set of possible labels Y = {class values}
- Unknown target function $f: X \rightarrow Y$
- Set of function hypotheses $H = \{ h \mid h : f: X \rightarrow Y \}$

Input: Training examples of unknown target function *f*
$\{ (\boldsymbol{X_i}, \boldsymbol{Y_i}) \}_{i=1}^{n} = \{ (\boldsymbol{X_1}, \boldsymbol{Y_1}), \ldots, (\boldsymbol{X_n}, \boldsymbol{Y_n}) \}$

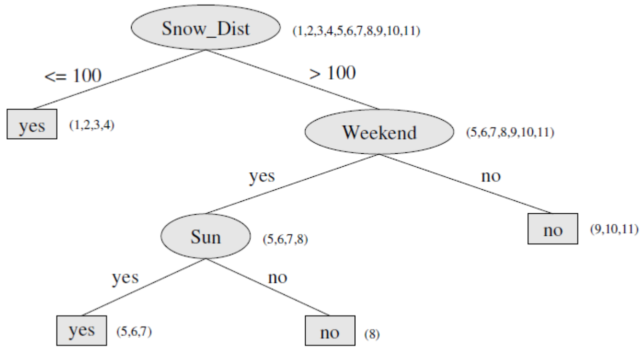
Output: Hypothesis h $\in$ H that **best approximates** f

Example toy problem:

How well can a set of features explain whether someone should go skiing?

Variable	Value	Description
Ski (goal variable)	yes, no	Should I drive to the nearest ski resort with enough snow?
Sun (feature)	yes, no	Is there sunshine today?
Snow_Dist (feature)	≤ 100, > 100	Distance to the nearest ski resort with good snow conditions (over/under 100 km)
Weekend (feature)	yes, no	Is it the weekend today?

Decision Tree Example



Decision tree for the skiing classification problem.

Algorithm 1:
Exhaustive Search of all possible Decision Trees

- Create all trees and search for the one with the smallest number of errors?
- Number of trees grows exponentially, thus computation is unacceptably wrong in many cases

Algorithm 2:
Greedy approach:

- Need a metric for choices for nodes (features in constructing the tree)
  - **Information gain** can be used as a measure of how far a feature moves toward a solution.
  - Information gain can be derived from **Entropy**
- Decide how to construct the tree, e.g. top-down or bottom-up or??

Entropy: H(probability distribution)

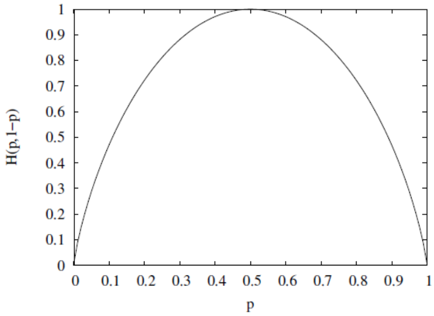
- Entropy is a measure of the "disorder" of a system. In information theory can be defined as the number of bits required to encode an event. The higher the uncertainty of an outcome then the higher the entropy
- Given a probability distribution (p_1, p_2, …, p_n)
  - The higher the uncertainty about the outcome, the more bits needed to encode an event
  - $\log_2 1/p_i$ = number of bits needed to encode the probability of the event
- The entropy H of a probability distribution is defined as:

$$H(\mathbf{p}) = H(p_1, \ldots, p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i.$$

(note: since the p_i are are less than 1 the logs will be negative hence the minus sign in the expression makes the measure positive.)

Basic Entropy curve

$$H(\mathbf{p}) = H(p_1, p_2) = H(p_1, 1-p_1) = -(p_1 \log_2 p_1 + (1-p_1)\log_2(1-p_1))$$



H(p1,p2)=H(1,0)=H(0,1)=0
$\log_2 1 = 0$

H(p1,p2)=H(1/2,1/2) =1
$\log_2 1/2 = - \log_2 2 = - 1$

Back to our example: Data

The rows 6 and 7 are inconsistent with an exhaustive model being correct on 100% of the data.

Day	*Snow_Dist*	*Weekend*	*Sun*	*Skiing*
1	≤ 100	yes	yes	yes
2	≤ 100	yes	yes	yes
3	≤ 100	yes	no	yes
4	≤ 100	no	yes	yes
5	> 100	yes	yes	yes
6	> 100	yes	yes	yes
7	> 100	yes	yes	no
8	> 100	yes	no	no
9	> 100	no	yes	no
10	> 100	no	yes	no
11	> 100	no	no	no

Entropy as a metric for information content

- Data set D with probability distribution p: H(D) = H(p):
- Information content I(D) from the data set D is the opposite of uncertainty, thus

$$I(D) := 1 - H(D)$$

- The attribute with the highest information gain will be chosen as the root – first decision node.
- Information Gain: $G(D,A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D)$
- Training data set S = (yes,yes,yes,yes,yes,yes,no,no,no,no,no) with the estimated probabilities p(yes,no) = $(p_1, p_2) = ({}^{6}/_{11}), ({}^{5}/_{11})$);
- H(D): 0.994

Entropy as a metric for information content

- Information Gain

algebraic manipulation

$$G(D,A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} I(D_i) - I(D) = H(D) - \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i)$$

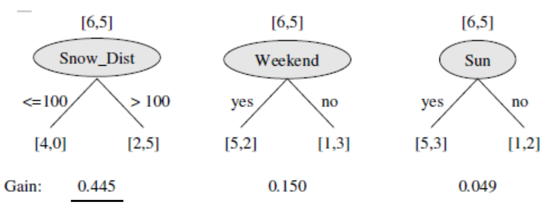
- Training data set S = (yes,yes,yes,yes,yes,yes,no,no,no,no,no) has estimated probabilities p(yes,no) = $(p_1, p_2) = (({}^{6}/_{11}), ({}^{5}/_{11})$);
- The initial situation is
  H(D) = H((6⁄11), (5⁄11))= H: 0.994
  $\rightarrow$ Information content is I(D) = .006

Computing the root node

$$\begin{aligned} G(D,\text{Snow\_Dist}) &= H(D) - \left(\frac{4}{11}H(D_{\leq 100}) + \frac{7}{11}H(D_{>100})\right) \\ &= 0.994 - \left(\frac{4}{11}\cdot 0 + \frac{7}{11}\cdot 0.863\right) = 0.445 \end{aligned}$$

Similarly: $G(D,\text{Weekend}) = 0.150$ $G(D,\text{Sun}) = 0.049$

Thus Snow_Dist becomes the root node

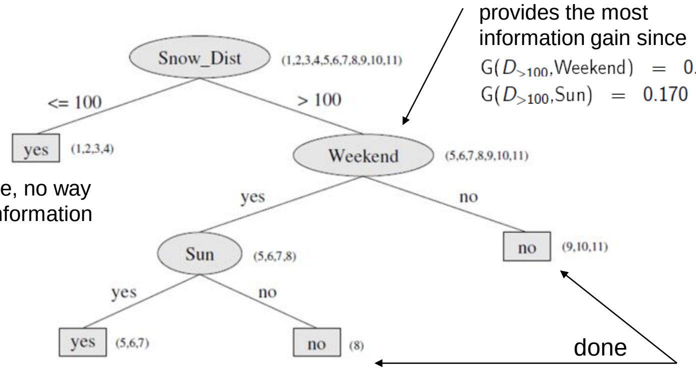


Working down the tree

Repeat the calculations here to find the best attribute for splitting this distribution. *Weekend* provides the most information gain since

$$G(D_{>100},\text{Weekend}) = 0.292$$
$$G(D_{>100},\text{Sun}) = 0.170$$

I = 1 done, no way to gain information



Decision tree for the skiing classification problem.

Decision tree pruning helps combat overfitting
(Eliminating nodes that are not clearly relevant – Occam, Einstein)

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

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

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

Broadening the applicability of decision trees

- Decision trees can be made more widely useful by handling the following complications:
  - Missing data
  - Continuous and multivalued input attributes
  - Continuous-valued output attribute

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

Supervised Learning

- Training set of examples of input output (N)
  - $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$,
  - $y = f(x)$
- function h is hypothesis about the world, approximates the true function f
  - drawn from a hypothesis space H of possible functions
  - h Model of the data, drawn from a model class H
- Consistent hypothesis: an h such that each x_i in the training set has $h(x_i) = y_i$.
- look for a best-fit function for which each $h(x_i)$ is close to y_i
- The true measure of a hypothesis, depends on how well it handles inputs it has not yet seen. Eg: a second sample of (x_i, y_i)
- h generalizes well if it accurately predicts the outputs of the test set

Model Selection and Optimization

- Task of finding a good hypothesis as two subtasks:
  - Model selection: model selection chooses a good hypothesis space
  - Optimization (training) finds the best hypothesis within that space.
- A training set to create the hypothesis, and a test set to evaluate it.
- Error rate: the proportion of times that $h(x) \neq y$ for an (x, y)
- Three data sets are needed:
  - A training set to train candidate models.
  - A validation set, also known as a development set or dev set, to evaluate the candidate models and choose the best one.
  - A test set to do a final unbiased evaluation of the best model.
- When insufficient amount of data to create three sets: k-fold cross-validation – see text

End of Slides used

Example problem: Restaurant waiting

The problem of deciding whether to wait for a table at a restaurant.

- For this problem the **output, y**, is a Boolean variable that we will call **WillWait**.
- The input, x, is a vector of ten attribute values, each of which has discrete values:
  - **Alt**ernate: whether there is a suitable alternative restaurant nearby.
  - **Bar**: whether the restaurant has a comfortable bar area to wait in.
  - **Fri**/Sat: true on Fridays and Saturdays.
  - **Hun**gry: whether we are hungry right now.
  - **Pat**rons: how many people in the restaurant (values are None, Some, and Full).
  - **Price**: the restaurant's price range ($, $$, $$$).
  - **Rain**ing: whether it is raining outside.
  - **Res**ervation: whether we made a reservation.
  - **Type**: the kind of restaurant (French, Italian, Thai, or burger).
  - Wait**Est**imate: host's wait estimate: 0–10, 10–30, 30–60, or >60minutes

Finding the highest probability solution

- Determine how probable a hypothesis is not just if possible
- hypothesis h∗ that is most probable given the data:
  - $h* = \text{argmax}_{h \in H} P(h|data)$
- By Bayes' rule this is equivalent to

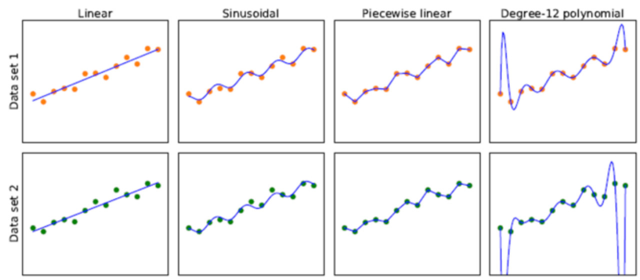
  $$h* = \text{argmax} \ P(h|data) \ P(h)$$

Supervised Learning

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

Supervised Learning

- 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).



Data for restaurant domain

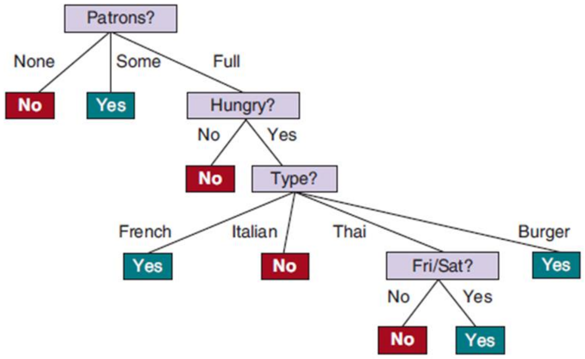
Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$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

Decision Trees

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

- Boolean decision tree is equivalent to a logical statement of the form:
- Output ⇔ (Path1 ∨ Path2 ∨⋯)

Splitting the examples by testing on attributes.



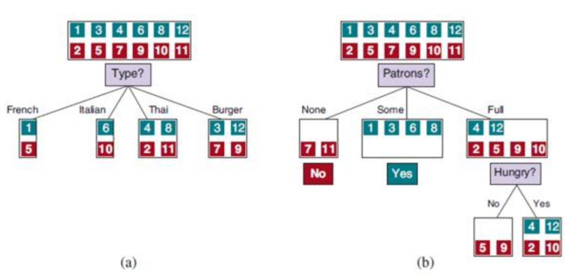
A decision tree learning algorithm

- The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly. (leaf value
- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
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 ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
     tree ← a new decision tree with root test A
     for each value v of A do
        exs ← {e : e ∈ 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
```

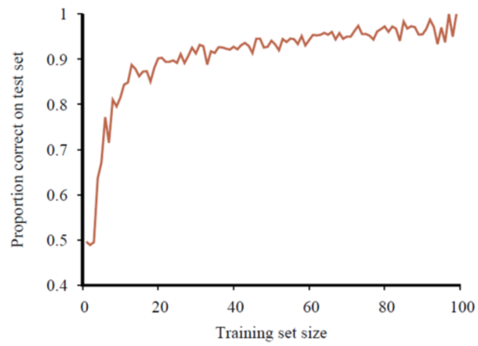

Splitting the examples by testing on attributes.

- At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. Yes No
  - Splitting on Type brings us no nearer to distinguishing between positive and negative examples.
  - Splitting on Patrons does a good job of separating positive and negative examples. After splitting on Patrons, Hungry is a fairly good second test

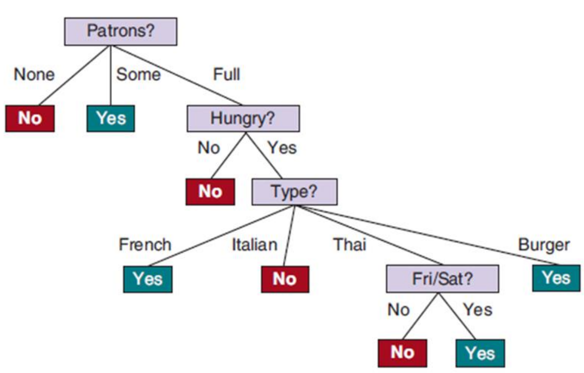


Testing results

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



Splitting the examples by testing on attributes.



Decision Trees:
Choosing attribute tests

- **Entropy**: measure of the uncertainty of a random variable;
  - the more information, the less entropy
  - fundamental quantity in information theory
- In general, the entropy of a random variable V with values v_k having probability

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

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

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

Entropy before split

Entropy after split