PSIT
Kanpur

Machine Learning Techniques [KCS-055]

Module- 1

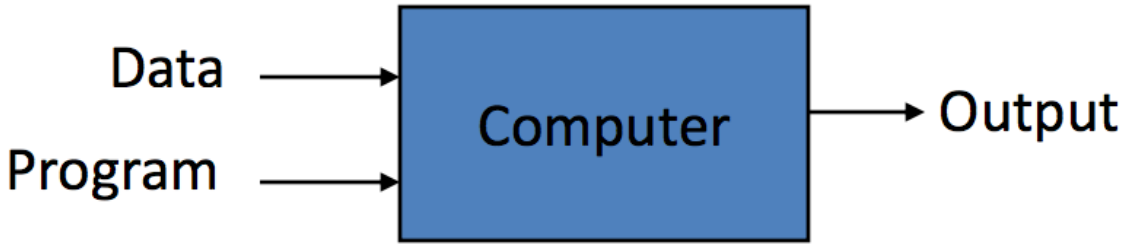
Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs** that can access data and use it learn for themselves.

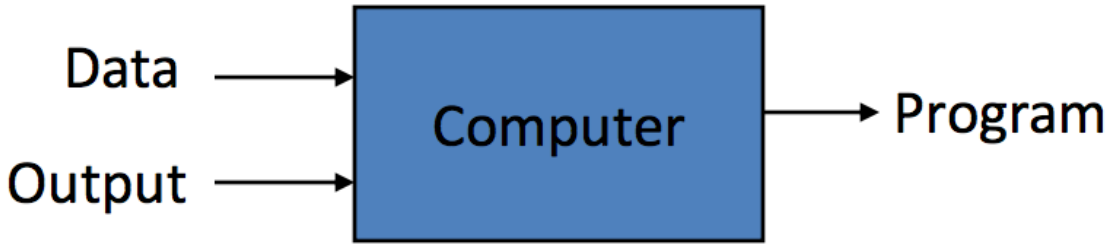
The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.

Traditional Learning v/s Machine Learning

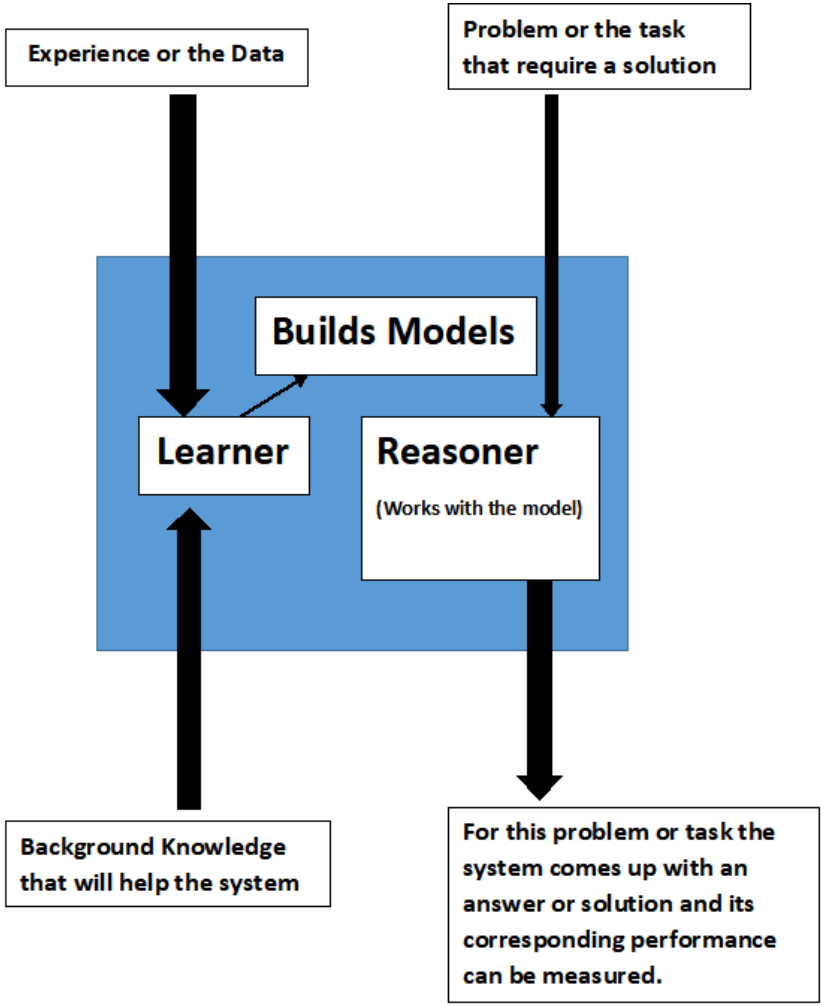
Traditional Programming



Machine Learning



Learning System



Example of Machine Learning

What is Machine Learning?
Have you ever shopped online? So while checking for a product, did you noticed when it recommends for a product similar to what you are looking for? or did you noticed “the person bought this product also bought this” combination of products. How are they doing this recommendation? This is machine learning.

Example of Machine Learning

Did you ever get a call from any bank or finance company asking you to take a loan or an insurance policy? What do you think, do they call everyone? No, they call only a few selected customers who they think will purchase their product. How do they select? This is target marketing and can be applied using Clustering. This is machine learning.

Types of Learning

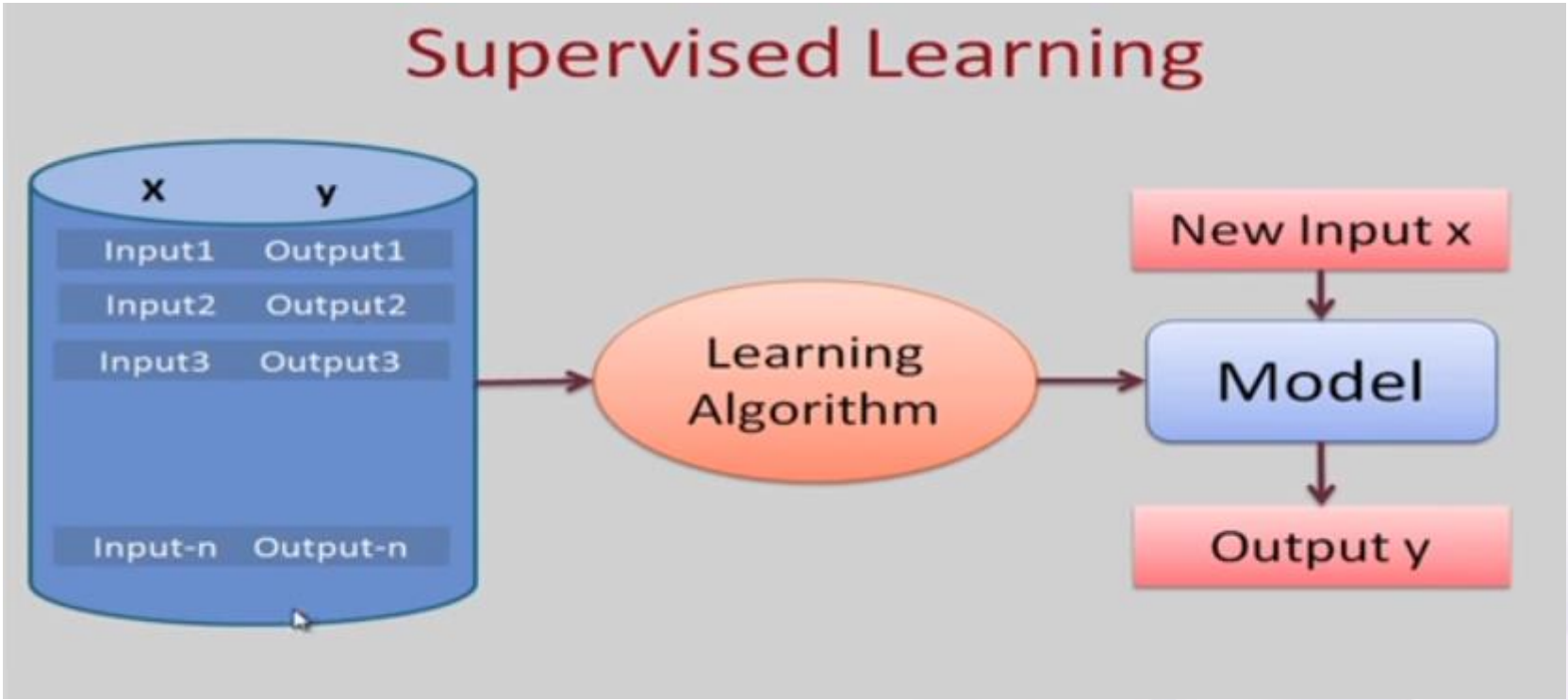
Machine learning is sub-categorized to three types:

- Supervised Learning – Train Me!
- Unsupervised Learning – I am self sufficient in learning
- Reinforcement Learning – My life My rules! (Hit & Trial)

What is Supervised Learning?

Supervised Learning is the one, where you can consider the learning is guided by a teacher. We have a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making a prediction or decision when new data is given to it.

Block Diagram of Supervised Learning

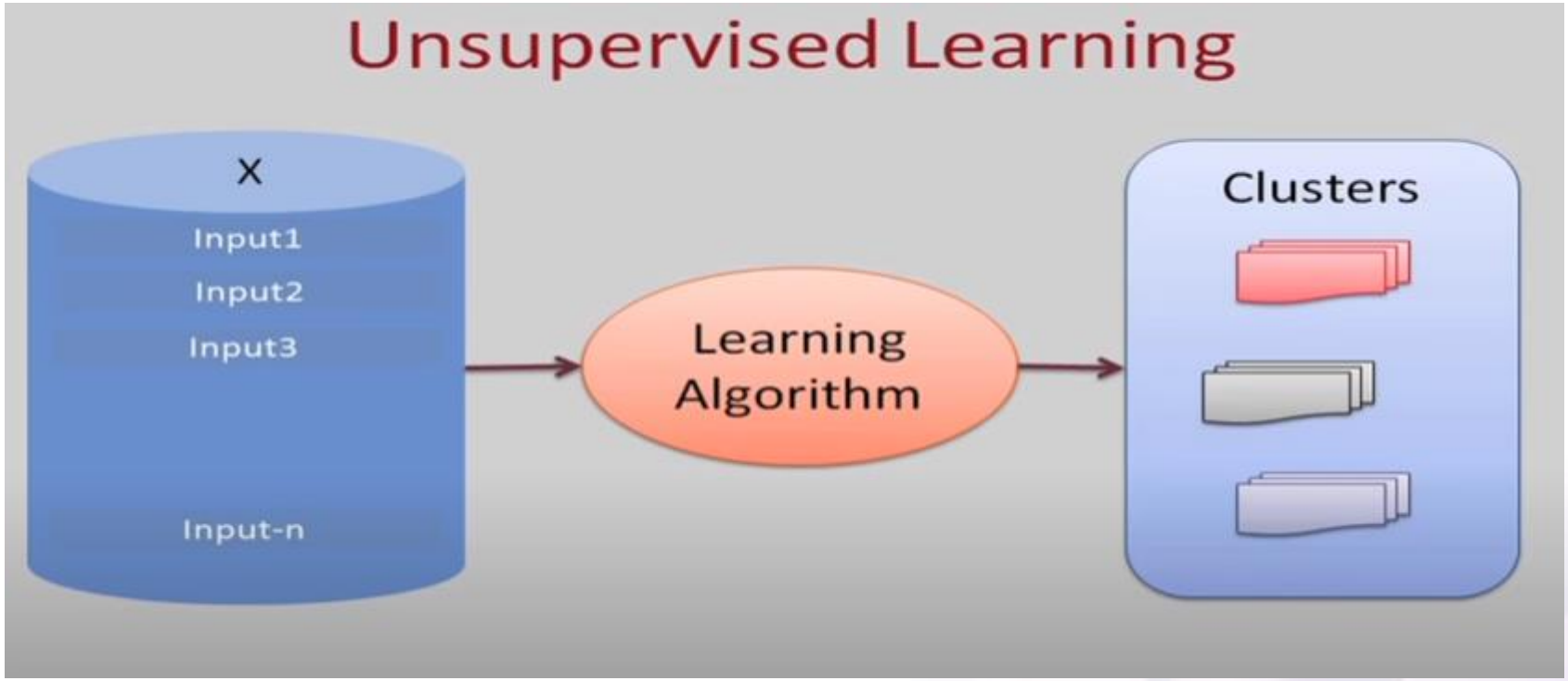


What is Unsupervised Learning?

The model learns through observation and finds structures in the data. Once the model is given a dataset, it automatically finds patterns and relationships in the dataset by creating clusters in it. What it cannot do is add labels to the cluster, like it cannot say this a group of apples or mangoes, but it will separate all the apples from mangoes.

Suppose we presented images of apples, bananas and mangoes to the model, so what it does, based on some patterns and relationships it creates clusters and divides the dataset into those clusters. Now if a new data is fed to the model, it adds it to one of the created clusters.

Block Diagram of Unsupervised Learning



What is Reinforcement Learning?

It is the ability of an agent to interact with the environment and find out what is the best outcome. It follows the concept of hit and trial method. The agent is rewarded or penalized with a point for a correct or a wrong answer, and on the basis of the positive reward points gained the model trains itself. And again once trained it gets ready to predict the new data presented to it.

Block Diagram of Reinforcement Learning

Supervised Learning vs Unsupervised Learning

	SUPERVISED LEARNING	UNSUPERVISED LEARNING
Input Data	Uses Known and Labeled Data as input	Uses Unknown Data as input
Computational Complexity	Very Complex	Less Computational Complexity
Real Time	Uses off-line analysis	Uses Real Time Analysis of Data
Number of Classes	Number of Classes are known	Number of Classes are not known
Accuracy of Results	Accurate and Reliable Results	Moderate Accurate and Reliable Results

Well-Posed Learning Problems

- *Definition*:

  A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

Well-Posed Learning Problems : Examples

- A checkers learning problem
  - Task *T* : playing checkers
  - Performance measure *P* : percent of games won against opponents
  - Training experience *E* : playing practice games against itself

- A handwriting recognition learning problem
  - Task *T* : recognizing and classifying handwritten words within images
  - Performance measure *P* : percent of words correctly classified
  - Training experience *E* : a database of handwritten words with given classifications

Well-Posed Learning Problems : Examples (cont.)

- A robot driving learning problem
  - Task *T* : driving on public four-lane highways using vision sensors
  - Performance measure *P* : average distance traveled before an error (as judged by human overseer)
  - Training experience *E* : a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

- Choosing the Training Experience
- Choosing the Target Function
- Choosing a Representation for the Target Function
- Choosing a Function Approximation Algorithm
- The Final Design

Choosing the Training Experience

- Whether the training experience provides direct or indirect feedback regarding the choices made by the performance system:
- Example:
  - Direct training examples in learning to play checkers consist of individual checkers board states and the correct move for each.
  - Indirect training examples in the same game consist of the move sequences and final outcomes of various games played in which information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost – *credit assignment problem.*

Choosing the Training Experience (cont.)

- The degree to which the learner controls the sequence of training examples:
- Example:
  - The learner might rely on the teacher to select informative board states and to provide the correct move for each
  - The learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move. Or the learner may have complete control over the board states and (indirect) classifications, as it does when it learns by playing against itself with no teacher present.

Choosing the Training Experience (cont.)

- How well it represents the distribution of examples over which the final system performance P must be measured: In general learning is most reliable when the training examples follow a distribution similar to that of future test examples.
- Example:
  - If the training experience in play checkers consists only of games played against itself, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion. (Note however that the most current theory of machine learning rests on the crucial assumption that the distribution of training examples is identical to the distribution of test examples)

Choosing the Target Function

- To determine what type of knowledge will be learned and how this will be used by the performance program:
- Example:
  - In play checkers, it needs to learn to choose the best move among those legal moves: *ChooseMove*: $B \rightarrow M$, which accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

Choosing the Target Function (cont.)

- Since the target function such as *ChooseMove* turns out to be very difficult to learn given the kind of indirect training experience available to the system, an alternative target function is then an evaluation function that assigns a numerical score to any given board state, $V: B \rightarrow R$.

Choosing a Representation for the Target Function

- Given the ideal target function V, we choose a representation that the learning system will use to describe V' that it will learn:
- Example:
  - In play checkers,

    $V'(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - where w_i is the numerical coefficient or weight to determine the relative importance of the various board features and x_i is the number of i-th objects on the board.

Choosing a Function Approximation Algorithm

- Each training example is given by $<b, V_{train}(b)>$ where $V_{train}(b)$ is the training value for a board b.
- Estimating Training Values:

$$V_{train}(b) \leftarrow V'(Successor(b)).$$

- Adjusting the weights: To specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{<b, V_{train}(b)>\}$, which minimizes the squared error E between the training values and the values predicted by the hypothesis V'

- $$E = \sum_{<b, V\text{train}(b)> \in \text{training examples}} (V_{\text{train}}(b) - V'(b))^2$$

Choosing a Function Approximation Algorithm (cont.)

- $E = \sum_{<b, V\text{train}(b)> \in \text{training examples}} (V_{\text{train}}(b) - V'(b))^2$
- To minimize *E*, the following rule is used:

LMS weight update rule

For each training example $<b, V_{train}(b)>$

Use the current weights to calculate $V'(b)$

For each weight w_i *, update it as*

$$w_i \leftarrow w_i + \eta (V_{train}(b) - V'(b)) x_i$$

The Final Design

- Performance System: To solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and a trace of its solution (game history) as output.

- Critic: To take as input the history or trace of the game and produce as output a set of training examples of the target function.

The Final Design (cont.)

- Generalizer: To take as input the training examples and produce an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

The Final Design (cont.)

- Experiment Generator: To take as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The Final Design (cont.)

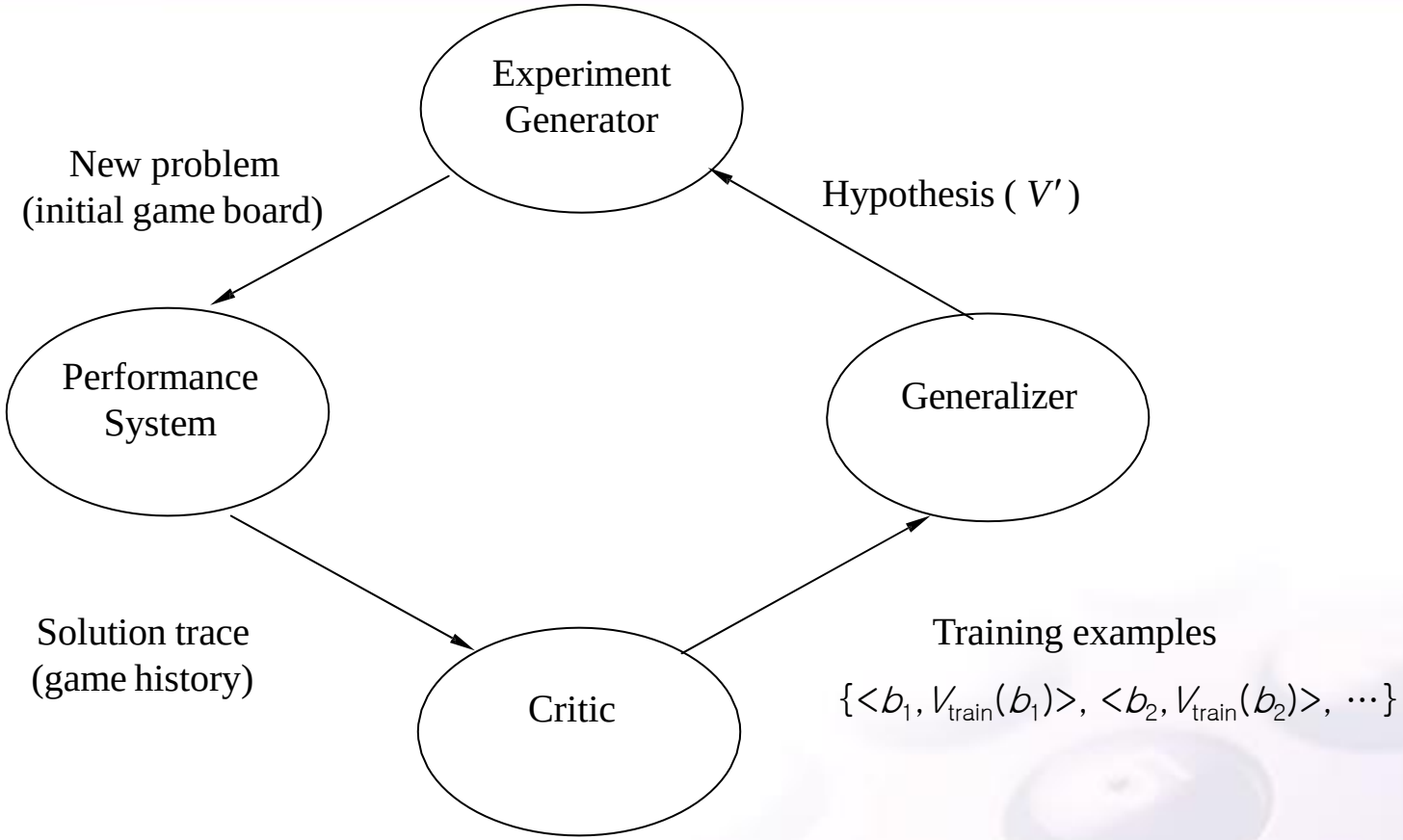
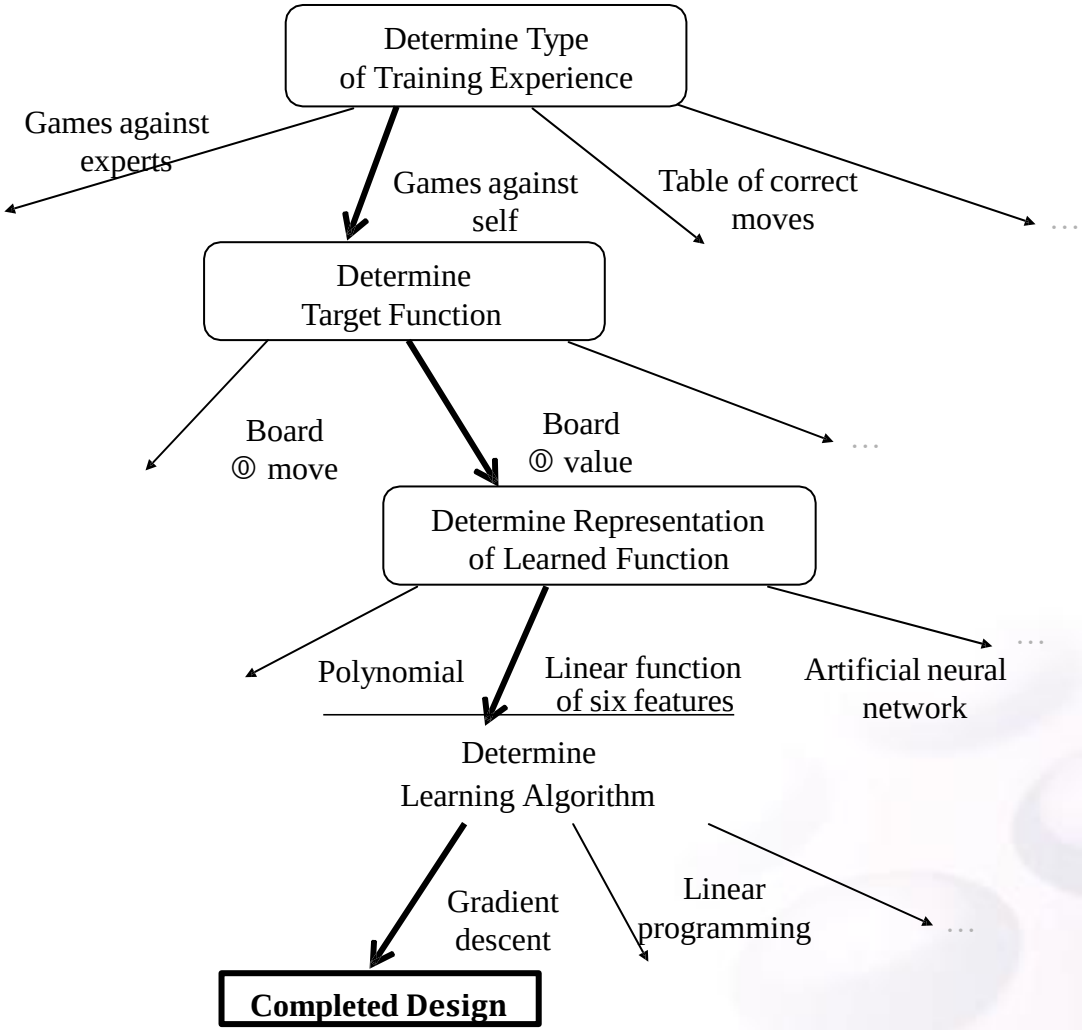


Figure 1 Final design of the checkers learning program

Choices in Designing the Checkers Learning Problem



History of ML

1950s: Samuel's Checker-Playing Program
1960s: Neural Network: Rosenblatt's Perceptron (Inventor of ANN)
Pattern Recognition
Minsky & Papert Prove Limitations of Perceptron
1970s: Symbolic Concept Introduction
Expert Systems & Knowledge Acquisition Bottleneck
Quinlan's ID3
NLP
1980s: Advanced Decision Trees & Rule Learning
Focus on experimental methodology
Resurgence of Neural Network

History of ML

90s: ML & Statistics

- Support Vector Machines
- Data Mining
- Adaptive Agents & Web Applications
- Text Learning
- Reinforcement Learning
- Ensembles
- Bayes Net Learning

1994: Self Driving Cars Road Test

1997: Deep Blue defeated Garry Kasparov in Chess Exhibition Match.

2009: Google Builds Self Driving Cars

2011: Watson wins Jeopardy

2014: Human Vision surpasses by ML systems

History of ML

Machine Learning the Game of Checkers

Arthur Samuel of IBM developed a computer program for playing checkers in the 1950s. Since the program had a very small amount of computer memory available, Samuel initiated what is called alpha-beta pruning. His design included a scoring function using the positions of the pieces on the board. The scoring function attempted to measure the chances of each side winning. The program chooses its next move using a minimax strategy, which eventually evolved into the minimax algorithm.

Samuel also designed a number of mechanisms allowing his program to become better. In what Samuel called rote learning, his program recorded/remembered all positions it had already seen and combined this with the values of the reward function. Arthur Samuel first came up with the phrase “Machine Learning” in 1952.

History of ML : The Perceptron

In 1957, Frank Rosenblatt – at the Cornell Aeronautical Laboratory – combined Donald Hebb's model of brain cell interaction with Arthur Samuel's Machine Learning efforts and created the perceptron. The perceptron was initially planned as a machine, not a program. The software, originally designed for the IBM 704, was installed in a custom-built machine called the Mark 1 perceptron, which had been constructed for image recognition. This made the software and the algorithms transferable and available for other machines.

Described as the first successful neuro-computer, the Mark I perceptron developed some problems with broken expectations. Although the perceptron seemed promising, it could not recognize many kinds of visual patterns (such as faces), causing frustration and stalling neural network research. It would be several years before the frustrations of investors and funding agencies faded. Neural network/Machine Learning research struggled until a resurgence during the 1990s.

History of ML

Multilayers Provide the Next Step

In the 1960s, the discovery and use of multilayers opened a new path in neural network research. It was discovered that providing and using two or more layers in the perceptron offered significantly more processing power than a perceptron using one layer. Other versions of neural networks were created after the perceptron opened the door to "layers" in networks, and the variety of neural networks continues to expand. The use of multiple layers led to feedforward neural networks and backpropagation.

Backpropagation, developed in the 1970s, allows a network to adjust its hidden layers of neurons/nodes to adapt to new situations. It describes "the backward propagation of errors," with an error being processed at the output and then distributed backward through the network's layers for learning purposes. Backpropagation is now being used to train deep neural networks.

History of ML

An Artificial Neural Network (ANN) has hidden layers which are used to respond to more complicated tasks than the earlier perceptrons could. ANNs are a primary tool used for Machine Learning. Neural networks use input and output layers and, normally, include a hidden layer (or layers) designed to transform input into data that can be used the by output layer. The hidden layers are excellent for finding patterns too complex for a human programmer to detect, meaning a human could not find the pattern and then teach the device to recognize it.

History of ML

The Nearest Neighbor Algorithm

In 1967, the nearest neighbor algorithm was conceived, which was the beginning of basic pattern recognition. This algorithm was used for mapping routes and was one of the earliest algorithms used in finding a solution to the traveling salesperson's problem of finding the most efficient route. Using it, a salesperson enters a selected city and repeatedly has the program visit the nearest cities until all have been visited. Marcello Pelillo has been given credit for inventing the "nearest neighbor rule."

Machine Learning and Artificial Intelligence take Separate Paths

In the late 1970s and early 1980s, Artificial Intelligence research had focused on using logical, knowledge-based approaches rather than algorithms. Additionally, neural network research was abandoned by computer science and AI researchers. This caused a schism between Artificial Intelligence and Machine Learning. Until then, Machine Learning had been used as a training program for AI.

The Machine Learning industry, which included a large number of researchers and technicians, was reorganized into a separate field and struggled for nearly a decade. The industry goal shifted from training for Artificial Intelligence to solving practical problems in terms of providing services. Its focus shifted from the approaches inherited from AI research to methods and tactics used in probability theory and statistics.

Machine Learning and Artificial Intelligence take Separate Paths (continued)

During this time, the ML industry maintained its focus on neural networks and then flourished in the 1990s. Most of this success was a result of Internet growth, benefiting from the ever-growing availability of digital data and the ability to share its services by way of the Internet.

History of ML (Boosting)

“Boosting” was a necessary development for the evolution of Machine Learning. Boosting algorithms are used to reduce bias during supervised learning and include ML algorithms that transform weak learners into strong ones. The concept of boosting was first presented in a 1990 paper titled “The Strength of Weak Learnability,” by Robert Schapire. Schapire states, “A set of weak learners can create a single strong learner.” Weak learners are defined as classifiers that are only slightly correlated with the true classification (still better than random guessing). By contrast, a strong learner is easily classified and well-aligned with the true classification.

Most boosting algorithms are made up of repetitive learning weak classifiers, which then add to a final strong classifier. After being added, they are normally weighted in a way that evaluates the weak learners’ accuracy. Then the data weights are “re-weighted.” Input data that is misclassified gains a higher weight, while data classified correctly loses weight.

Boosting

This environment allows future weak learners to focus more extensively on previous weak learners that were misclassified.
The basic difference between the various types of boosting algorithms is “the technique” used in weighting training data points. AdaBoost is a popular Machine Learning algorithm and historically significant, being the first algorithm capable of working with weak learners. More recent algorithms include BrownBoost, LPBoost, MadaBoost, TotalBoost, xgboost, and LogitBoost. A large number boosting algorithms work within the AnyBoost framework.

History of ML

Speech Recognition

Currently, much of speech recognition training is being done by a Deep Learning technique called Long Short-Term Memory (LSTM), a neural network model described by Jürgen Schmidhuber and Sepp Hochreiter in 1997. LSTM can learn tasks that require memory of events that took place thousands of discrete steps earlier, which is quite important for speech.

Around the year 2007, Long Short-Term Memory started outperforming more traditional speech recognition programs. In 2015, the Google speech recognition program reportedly had a significant performance jump of 49 percent using a CTC-trained LSTM.

History of ML

Facial Recognition Becomes a Reality

In 2006, the *Face Recognition Grand Challenge* – a National Institute of Standards and Technology program – evaluated the popular face recognition algorithms of the time. 3D face scans, iris images, and high-resolution face images were tested. Their findings suggested the new algorithms were ten times more accurate than the facial recognition algorithms from 2002 and 100 times more accurate than those from 1995. Some of the algorithms were able to outperform human participants in recognizing faces and could uniquely identify identical twins.

In 2012, Google's X Lab developed an ML algorithm that can autonomously browse and find videos containing cats. In 2014, Facebook developed DeepFace, an algorithm capable of recognizing or verifying individuals in photographs with the same accuracy as humans.

Machine Learning at Present

Recently, Machine Learning was defined by Stanford University as "the science of getting computers to act without being explicitly programmed." Machine Learning is now responsible for some of the most significant advancements in technology, such as the new industry of self-driving vehicles. Machine Learning has prompted a new array of concepts and technologies, including supervised and unsupervised learning, new algorithms for robots, the Internet of Things, analytics tools, chatbots, and more.

Machine Learning at Present

Listed below are seven common ways the world of business is currently using Machine Learning:

Analyzing Sales Data: Streamlining the data

Real-Time Mobile Personalization: Promoting the experience

Fraud Detection: Detecting pattern changes

Product Recommendations: Customer personalization

Learning Management Systems: Decision-making programs

Dynamic Pricing: Flexible pricing based on a need or demand

Natural Language Processing: Speaking with humans

Machine Learning models have become quite adaptive in continuously learning, which makes them increasingly accurate the longer they operate. ML algorithms combined with new computing technologies promote scalability and improve efficiency. Combined with business analytics, Machine Learning can resolve a variety of organizational complexities. Modern ML models can be used to make predictions ranging from outbreaks of disease to the rise and fall of stocks.

Artificial Neural Networks

Your first step in Deep Learning.

Deep Learning is the most exciting and powerful branch of Machine Learning. It's a technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It’s achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

Artificial Neural Network

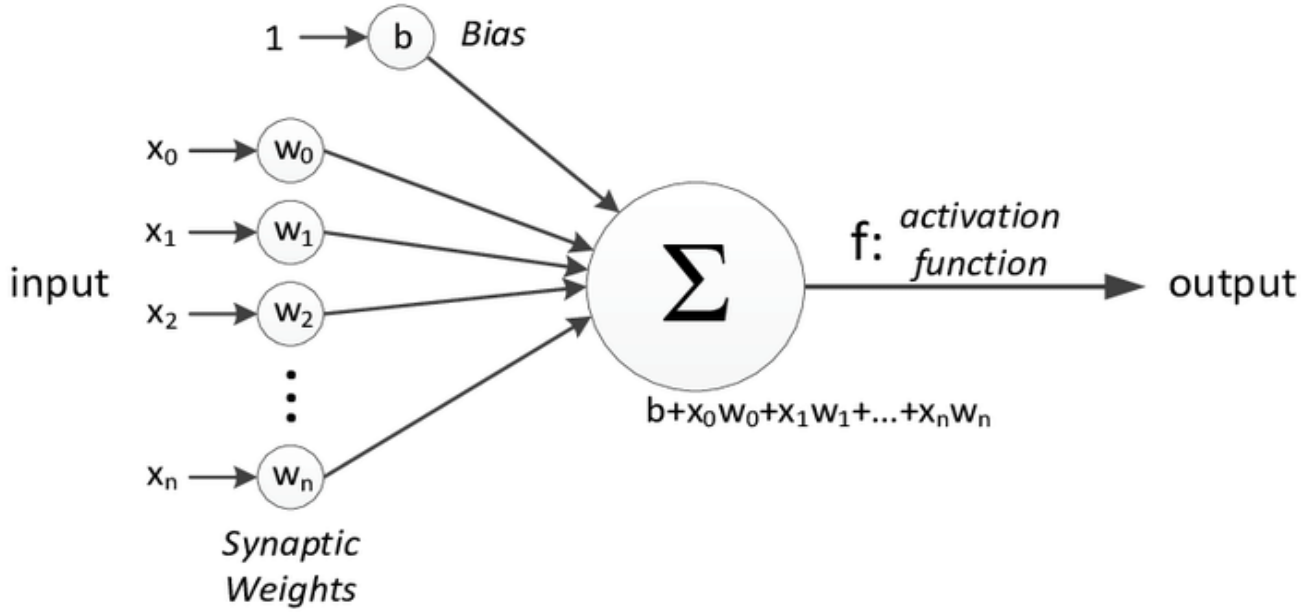
Deep Learning models can be used for a variety of complex tasks:
Artificial Neural Networks(ANN) for Regression and classification
Convolutional Neural Networks(CNN) for Computer Vision
Recurrent Neural Networks(RNN) for Time Series analysis
Self-organizing maps for Feature extraction
Deep Boltzmann machines for Recommendation systems
Auto Encoders for Recommendation systems

Artificial Neural Network : Definition

Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem

Perceptron

The following diagram represents the general model of ANN which is inspired by a biological neuron. It is also called Perceptron.
A single layer neural network is called a Perceptron. It gives a single output.



Explanation of Perceptron

In the above figure, for one single observation, x0, x1, x2, x3...x(n) represents various inputs(independent variables) to the network. Each of these inputs is multiplied by a connection weight or synapse. The weights are represented as w0, w1, w2, w3….w(n) . **Weight shows the strength of a particular node.**
b is a bias value. A bias value allows you to shift the activation function up or down.
In the simplest case, these products are summed, fed to a transfer function (activation function) to generate a result, and this result is sent as output.
Mathematically, x1.w1 + x2.w2 + x3.w3 ..... xn.wn = $\sum$ xi.wi
Now activation function is applied $\phi(\sum$ xi.wi)

Activation function

The Activation function is important for an ANN to learn and make sense of something really complicated. Their main purpose is to convert an input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer in the stack.

Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The motive is to introduce non-linearity into the output of a neuron.

If we do not apply activation function then the output signal would be simply linear function(one-degree polynomial). Now, a linear function is easy to solve but they are limited in their complexity, have less power. Without activation function, our model cannot learn and model complicated data such as images, videos, audio, speech, etc.

Now the question arises why do we need Non-Linearity?

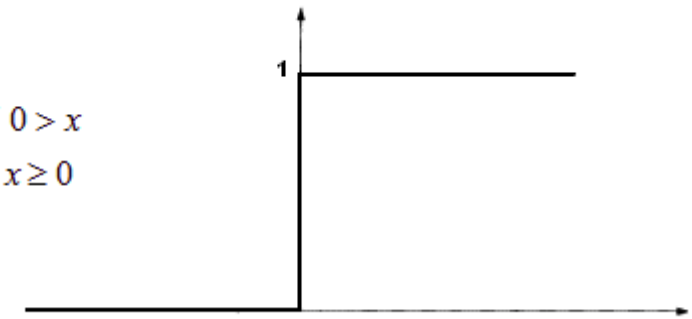
Non-Linear functions are those which have a degree more than one and they have a curvature. Now we need a neural network to learn and represent almost anything and any arbitrary complex function that maps an input to output.

Neural Network is considered **“Universal Function Approximators”.** It means they can learn and compute any function at all.

Types of Activation Functions:

1. **Threshold Activation Function — (Binary step function)**
   A Binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



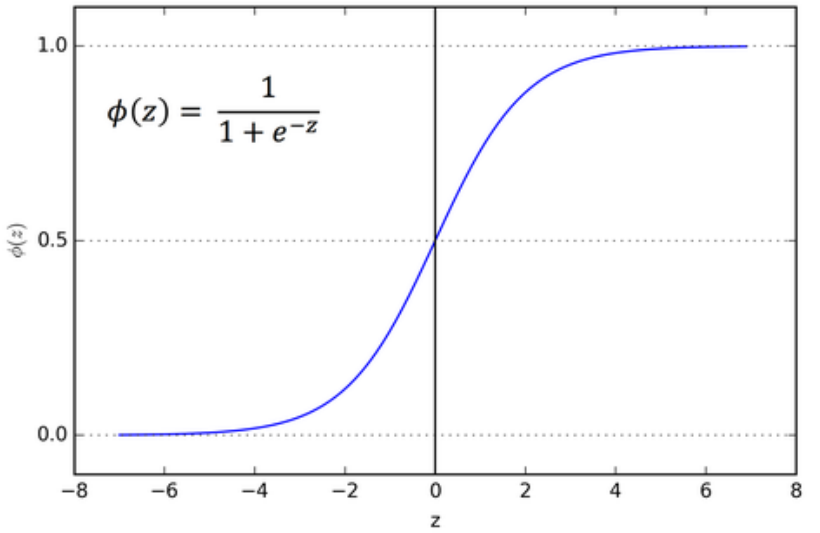
A Binary step function

Activation function A = “activated” if Y > threshold
else not or A=1 if y>threshold 0 otherwise.
The problem with this function is for creating a binary classifier (1 or 0), but if you want multiple such neurons to be connected to bring in more classes, Class1, Class2, Class3, etc. In this case, all neurons will give 1, so we cannot decide.

Types of Activation Functions:

Sigmoid Activation Function — (Logistic function)

A Sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve which ranges between 0 and 1, therefore it is used for models where we need to predict the probability as an output.



Sigmoid curve

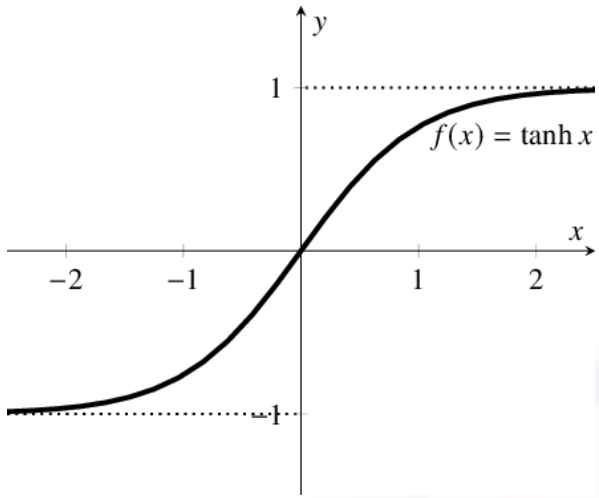
Types of Activation Functions:

The Sigmoid function is differentiable, means we can find the slope of the curve at any 2 points.
The drawback of the Sigmoid activation function is that it can cause the neural network to get stuck at training time if strong negative input is provided.

Types of Activation Functions:

Hyperbolic Tangent Function — (tanh)

It is similar to Sigmoid but better in performance. It is nonlinear in nature, so great we can stack layers. The function ranges between (-1,1).



Hyperbolic tangent function

Types of Activation Functions:

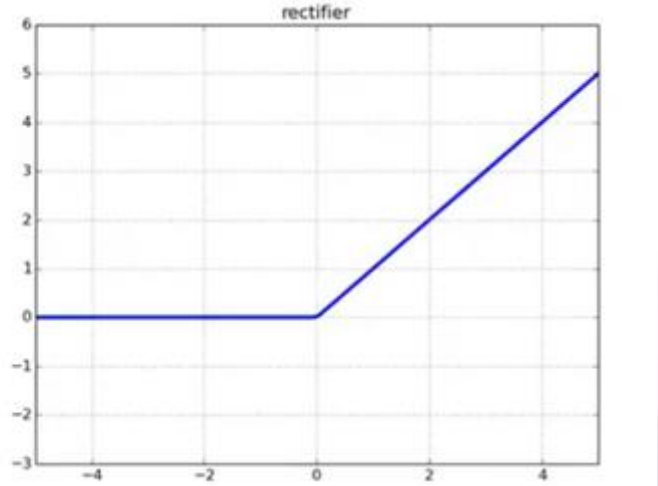
The main advantage of this function is that strong negative inputs will be mapped to negative output and only zero-valued inputs are mapped to near-zero outputs.,So less likely to get stuck during training.

Types of Activation Functions:

Rectified Linear Units — (ReLu)

ReLu is the most used activation function in CNN and ANN which ranges from zero to infinity.[0,∞)

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \Rightarrow 0 \end{cases}$$



ReLu

Types of Activation Functions:

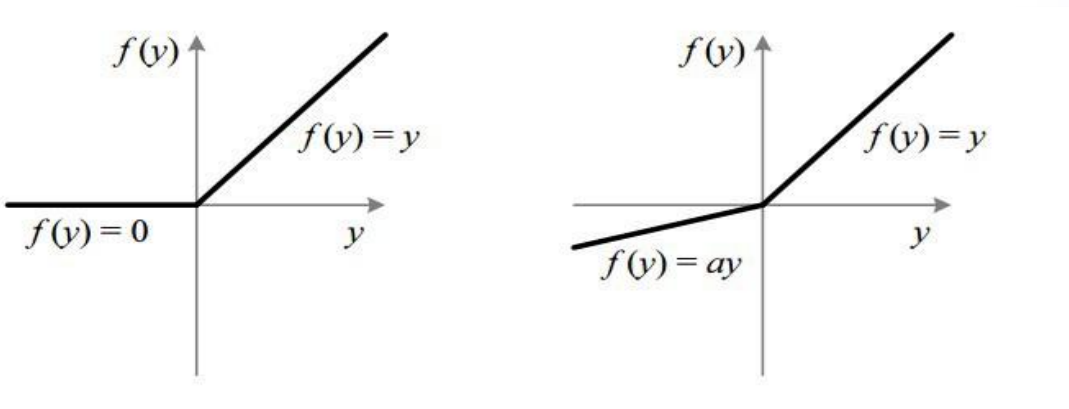
It gives an output ‘x’ if x is positive and 0 otherwise. It looks like having the same problem of linear function as it is linear in the positive axis. Relu is non-linear in nature and a combination of ReLu is also non-linear. In fact, it is a good approximator and any function can be approximated with a combination of Relu.

ReLu is 6 times improved over hyperbolic tangent function.

It should only be applied to hidden layers of a neural network. So, for the output layer use softmax function for classification problem and for regression problem use a Linear function.

Types of Activation Functions:

Here one problem is some gradients are fragile during training and can die. It causes a weight update which will make it never activate on any data point again. Basically ReLu could result in dead neurons.
To fix the problem of dying neurons, **Leaky ReLu** was introduced. So, Leaky ReLu introduces a small slope to keep the updates alive.
Leaky ReLu ranges from -∞ to +∞.



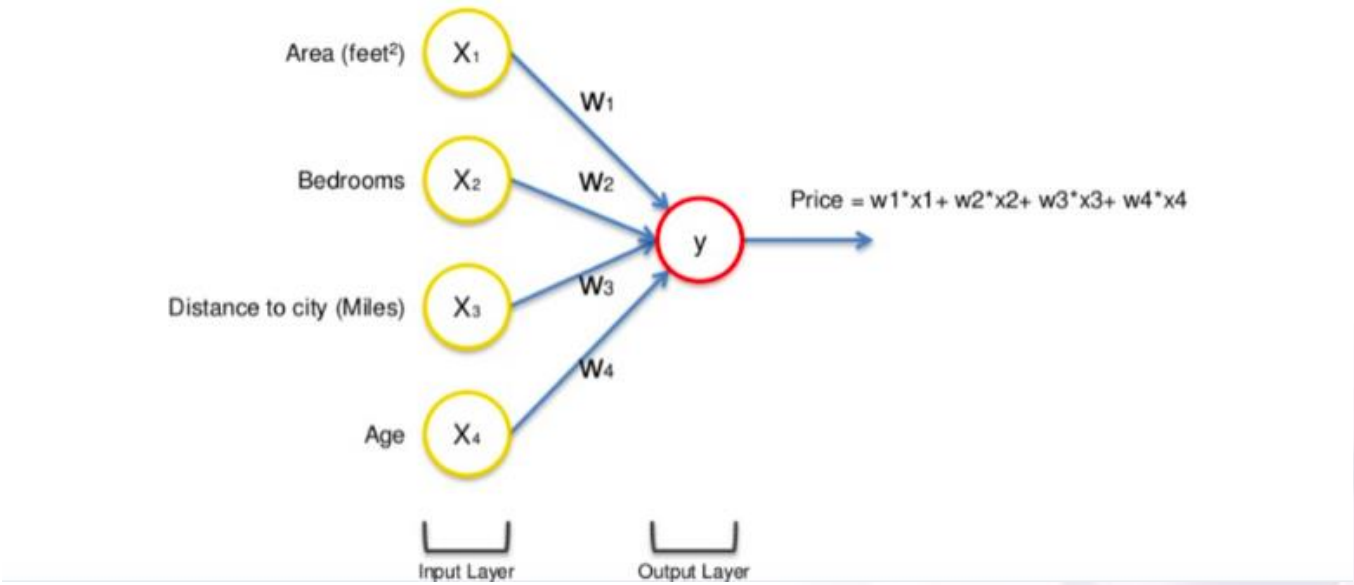
ReLu vs Leaky ReLu

Types of Activation Functions:

Leak helps to increase the range of the ReLu function. Usually, the value of a = 0.01 or so.
When a is not 0.01, then it is called Randomized ReLu.

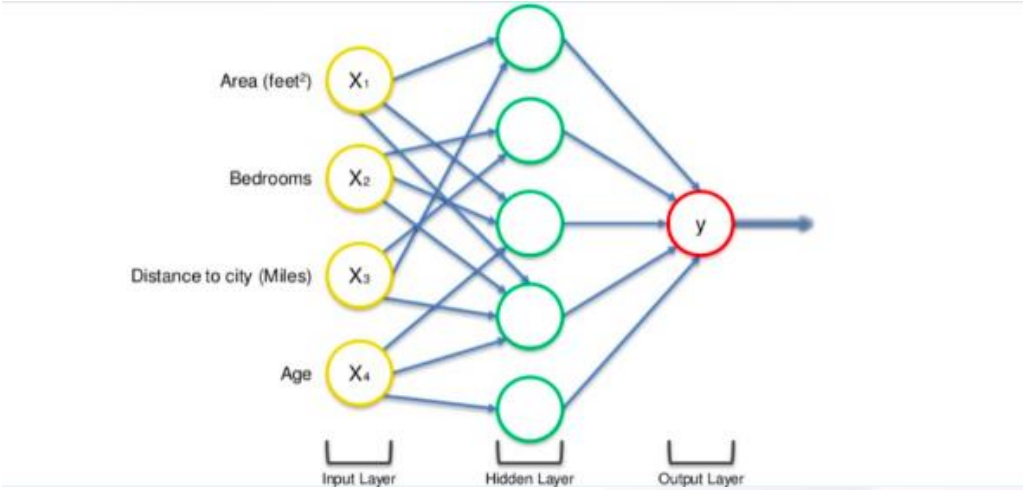
How does the Neural network work?

Let us take the example of the price of a property and to start with we have different factors assembled in a single row of data: Area, Bedrooms, Distance to city and Age.



How does the Neural network work?

The input values go through the weighted synapses straight over to the output layer. All four will be analyzed, an activation function will be applied, and the results will be produced.
This is simple enough but there is a way to amplify the power of the Neural Network and increase its accuracy by the addition of a hidden layer that sits between the input and output layers.



A neural network with a hidden layer(only showing non-0 values)

How does the Neural network work?

Now in the above figure, all 4 variables are connected to neurons via a synapse. However, not all of the synapses are weighted. they will either have a 0 value or non-0 value.
here, the non-0 value → indicates the importance
0 value → They will be discarded.

Let's take the example of Area and Distance to City are non-zero for the first neuron, which means they are weighted and matter to the first neuron. The other two variables, Bedrooms and Age aren't weighted and so are not considered by the first neuron.

You may wonder why that first neuron is only considering two of the four variables. In this case, it is common on the property market that larger homes become cheaper the further they are from the city. That's a basic fact. So what this neuron may be doing is looking specifically for properties that are large but are not so far from the city.

How does the Neural network work?

Now, this is where the power of neural networks comes from. There are many of these neurons, each doing similar calculations with different combinations of these variables.

Once this criterion has been met, the neuron applies the activation function and do its calculations. The next neuron down may have weighted synapses of Distance to the city and, Bedrooms.
This way the neurons work and interact in a very flexible way allowing it to look for specific things and therefore make a comprehensive search for whatever it is trained for.

How do Neural networks learn?

Looking at an analogy may be useful in understanding the mechanisms of a neural network. Learning in a neural network is closely related to how we learn in our regular lives and activities — we perform an action and are either accepted or corrected by a trainer or coach to understand how to get better at a certain task. Similarly, neural networks require a trainer in order to describe what should have been produced as a response to the input. Based on the difference between the actual value and the predicted value, an error value also called **Cost Function** is computed and sent back through the system.

Cost Function: One half of the squared difference between actual and output value.

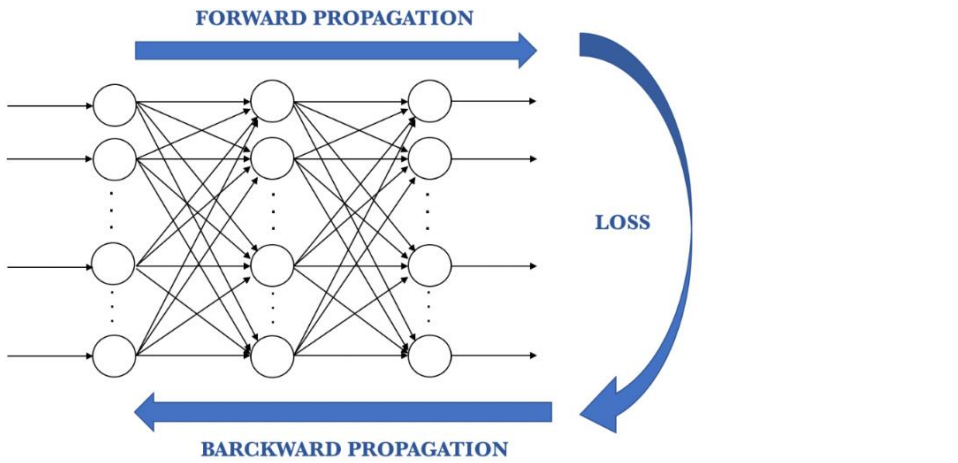
How do Neural networks learn?

For each layer of the network, the cost function is analyzed and used to adjust the threshold and weights for the next input. Our aim is to minimize the cost function. The lower the cost function, the closer the actual value to the predicted value. In this way, the error keeps becoming marginally lesser in each run as the network learns how to analyze values.

We feed the resulting data back through the entire neural network. The weighted synapses connecting input variables to the neuron are the only thing we have control over.

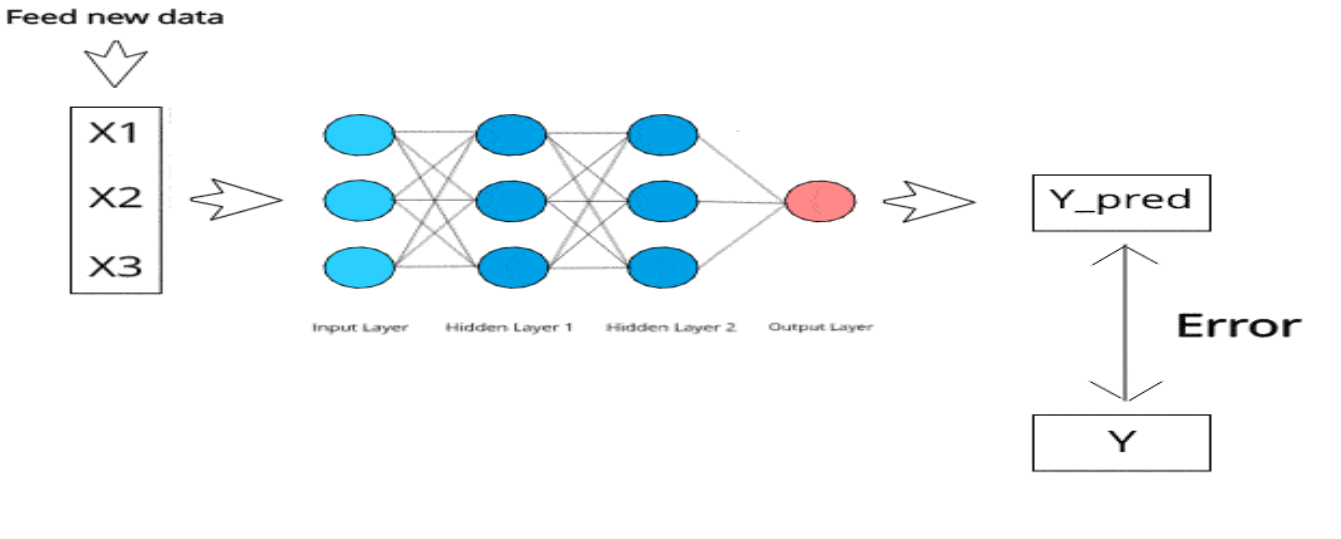
How do Neural networks learn?

As long as there exists a disparity between the actual value and the predicted value, we need to adjust those weights. Once we tweak them a little and run the neural network again, A new Cost function will be produced, hopefully, smaller than the last.
We need to repeat this process until we scrub the cost function down to as small as possible.



How do Neural networks learn?

The procedure described above is known as **Back-propagation** and is applied continuously through a network until the error value is kept at a minimum.



Back-propagation

How do Neural networks learn?

There are basically 2 ways to adjust weights: —
1. Brute-force method
2. Batch-Gradient Descent

Brute-force method

Best suited for the single-layer feed-forward network. Here you take a number of possible weights. In this method, we want to eliminate all the other weights except the one right at the bottom of the U-shaped curve.

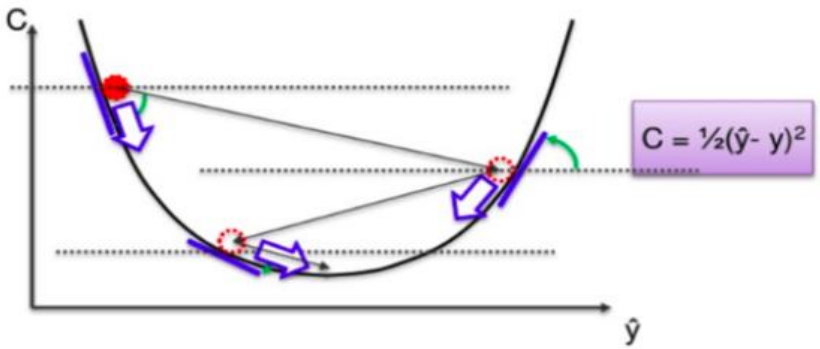
Optimal weight can be found using simple elimination techniques. This process of elimination work if you have one weight to optimize. What if you have complex NN with many numbers of weights, then this method fails because of the **Curse of Dimensionality.**

The alternative approach that we have is called Batch Gradient Descent.

How do Neural networks learn?

Batch-Gradient Descent

It is a first-order iterative optimization algorithm and its responsibility is to find the minimum cost value(loss) in the process of training the model with different weights or updating weights.



Gradient Descent

How do Neural networks learn?

In Gradient Descent, instead of going through every weight one at a time, and ticking every wrong weight off as you go, we instead look at the angle of the function line.

If slope → Negative, that means yo go down the curve.

If slope → Positive, Do nothing

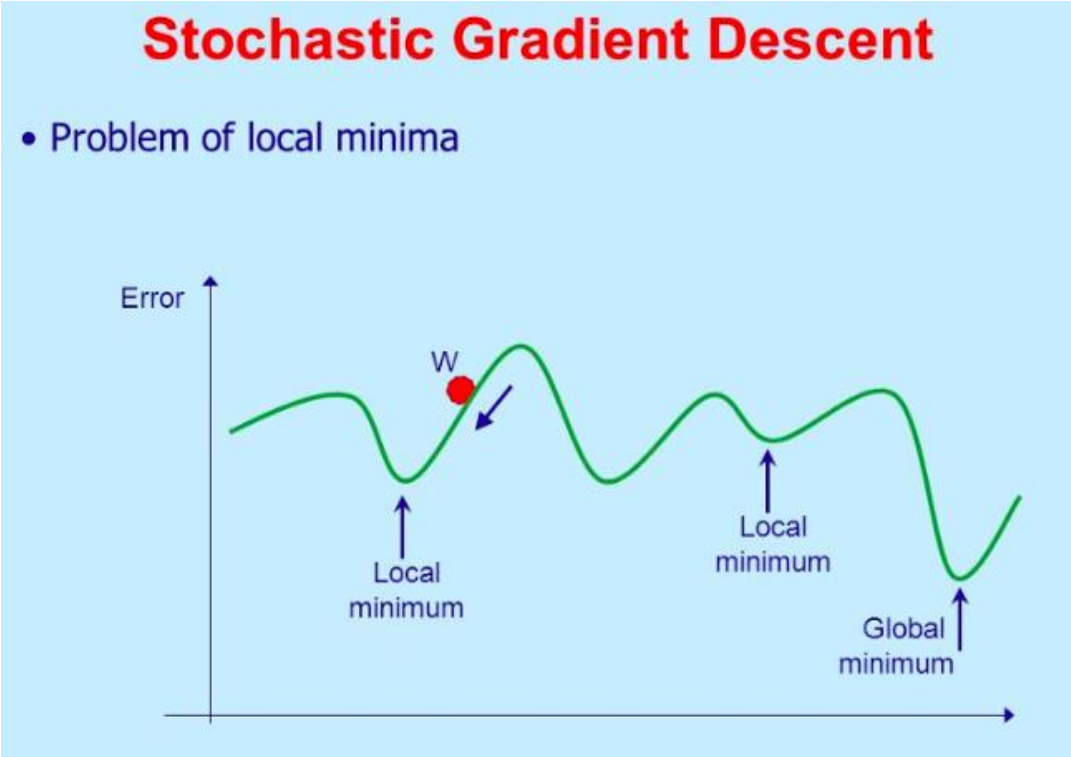
This way a vast number of incorrect weights are eliminated. For instance, if we have 3 million samples, we have to loop through 3 million times. So basically you need to calculate each cost 3 million times.

How do Neural networks learn?

Stochastic Gradient Descent(SGD)

Gradient Descent works fine when we have a convex curve just like in the above figure. But if we don't have a convex curve, Gradient Descent fails.

The word ‘*stochastic*‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.



Stochastic Gradient Descent

How do Neural networks learn?

In SGD, we take one row of data at a time, run it through the neural network then adjust the weights. For the second row, we run it, then compare the Cost function and then again adjusting weights. And so on…
SGD helps us to avoid the problem of local minima. It is much faster than Gradient Descent because it is running each row at a time and it doesn't have to load the whole data in memory for doing computation. One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

Training ANN with Stochastic Gradient Descent

Step-1 → Randomly initialize the weights to small numbers close to 0 but not 0.
Step-2 → Input the first observation of your dataset in the input layer, each feature in one node.
Step-3 → **Forward-Propagation**: From left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted value.
Step-4 → Compare the predicted result to the actual result and measure the generated error(Cost function).
Step-5 → **Back-Propagation**: from right to left, the error is backpropagated. Update the weights according to how much they are responsible for the error. The learning rate decides how much we update weights.

Training ANN with Stochastic Gradient Descent

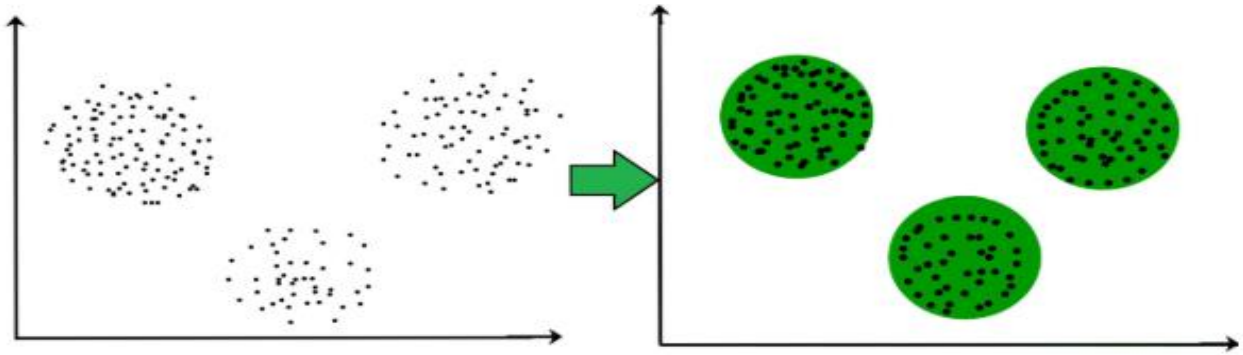
Step-6 → Repeat step-1 to 5 and update the weights after each observation(Reinforcement Learning)
Step-7 → When the whole training set passed through the ANN, that makes and epoch. Redo more epochs.

Clustering in Machine Learning

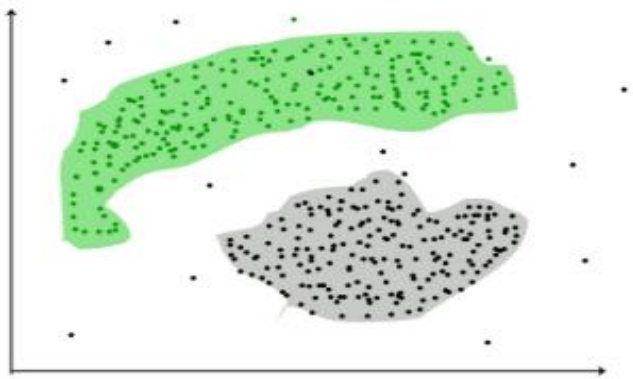
It is basically a type of *unsupervised learning method* . An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples. **Clustering** is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Example of Clustering

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



It is not necessary for clusters to be a spherical. Such as :



Why Clustering ?

DBSCAN: Density-based Spatial Clustering of Applications with Noise
These data points are clustered by using the basic concept that the data point lies within the given constraint from the cluster centre. Various distance methods and techniques are used for calculation of the outliers.

Why Clustering ?
Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods :

Density-Based Methods : These methods consider the clusters as the dense region having some similarity and different from the lower dense region of the space. These methods have good accuracy and ability to merge two clusters.Example *DBSCAN (Density-Based Spatial Clustering of Applications with Noise)* , *OPTICS (Ordering Points to Identify Clustering Structure)* etc.

Hierarchical Based Methods : The clusters formed in this method forms a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

- **Agglomerative** (*bottom up approach*)
- **Divisive** (*top down approach*)

examples *CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies)* etc.

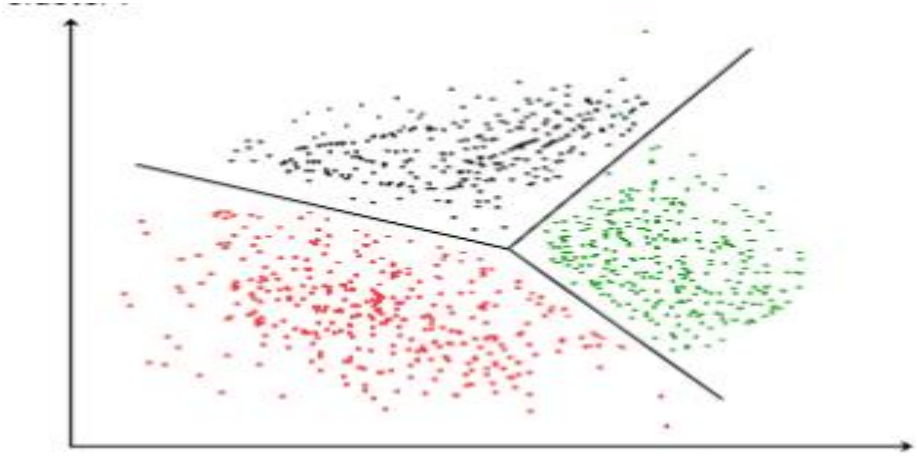
Other Methods

Partitioning Methods : These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means*, *CLARANS (Clustering Large Applications based upon Randomized Search)* etc.

Grid-based Methods : In this method the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operation done on these grids are fast and independent of the number of data objects example *STING (Statistical Information Grid)*, *wave cluster*, *CLIQUE (CLustering In Quest)* etc.

Clustering Algorithms :

K-means clustering algorithm – It is the simplest unsupervised learning algorithm that solves clustering problem.K-means algorithm partition n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster .



Applications of Clustering in different fields

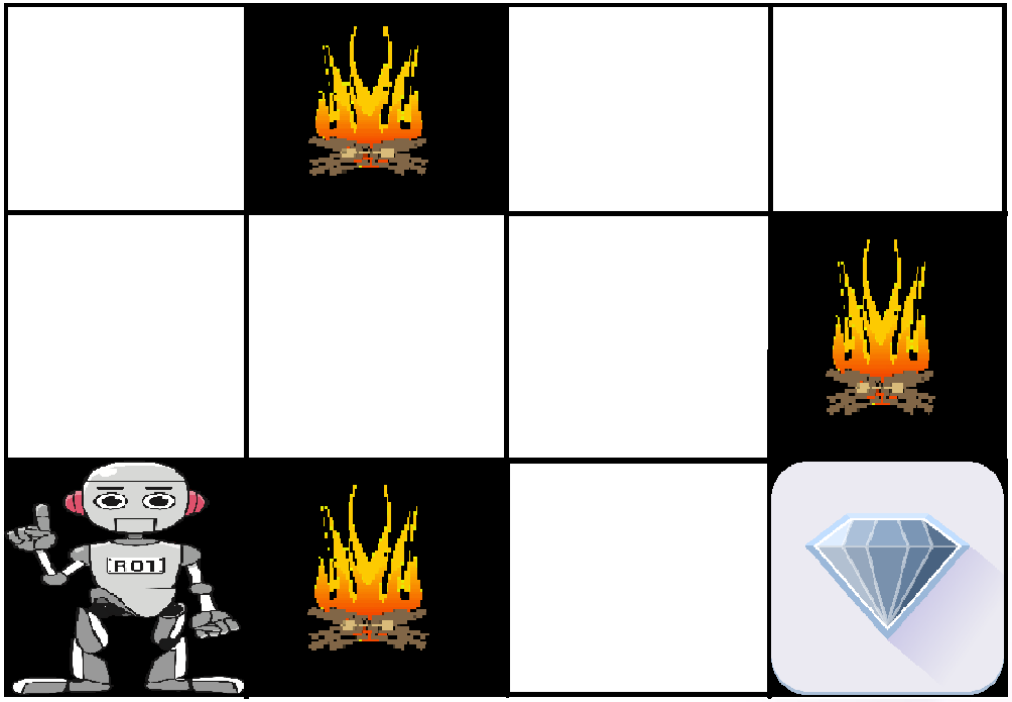
Marketing : It can be used to characterize & discover customer segments for marketing purposes.
Biology : It can be used for classification among different species of plants and animals.
Libraries : It is used in clustering different books on the basis of topics and information.
Insurance : It is used to acknowledge the customers, their policies and identifying the frauds.
City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones.

Reinforcement Learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Example: The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.

Example: Reinforcement Learning



The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fire. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

Input: The input should be an initial state from which the model will start
Output: There are many possible output as there are variety of solution to a particular problem
Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
The model keeps continues to learn.
The best solution is decided based on the maximum reward.

Supervised Learning vs Reinforcement Learning

REINFORCEMENT LEARNING	SUPERVISED LEARNING
Reinforcement learning is all about making decisions sequentially. In simple words we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	Supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game	Example: Object recognition

Types of Reinforcement: There are two types of Reinforcement:

Positive –
Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior. Advantages of reinforcement learning are:

- Maximizes Performance
- Sustain Change for a long period of time

Disadvantages of reinforcement learning:

- Too much Reinforcement can lead to overload of states which can diminish the results

Negative –
Negative Reinforcement is defined as strengthening of a behavior because a negative condition is stopped or avoided. Advantages of reinforcement learning:

- Increases Behavior
- Provide defiance to minimum standard of performance

Disadvantages of reinforcement learning:

- It Only provides enough to meet up the minimum behavior

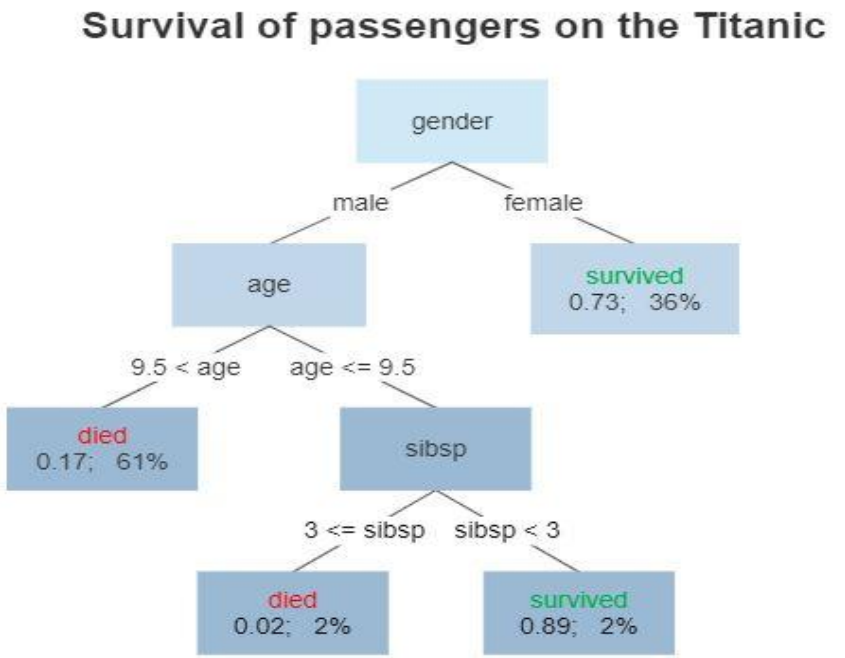
Various Practical applications of Reinforcement Learning –

RL can be used in robotics for industrial automation.
RL can be used in machine learning and data processing
RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.
RL can be used in large environments in the following situations:
A model of the environment is known, but an analytic solution is not available;
Only a simulation model of the environment is given (the subject of simulation-based optimization)
The only way to collect information about the environment is to interact with it.

Decision Tree Learning

Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called **classification trees**; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making.

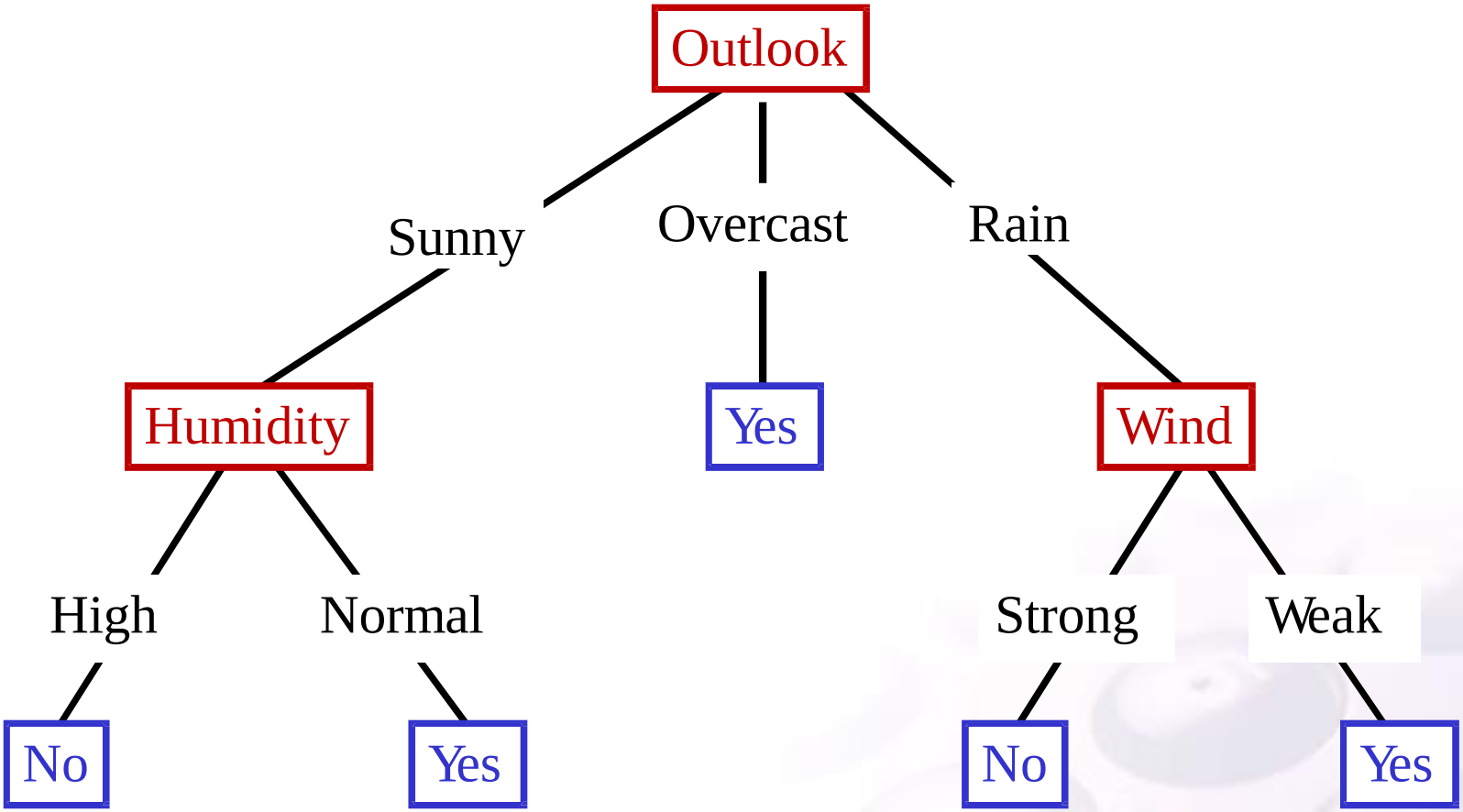


A tree showing survival of passengers on the Titanic ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of survival and the percentage of observations in the leaf. Summarizing: Your chances of survival were good if you were (i) a female or (ii) a male younger than 9.5 years with strictly less than 3 siblings.

Decision Tree Learning

- **Decision tree learning** is a method for approximating discrete-valued target functions.
- The learned function is represented by **a decision tree**.
  - A learned decision tree can also be re-represented as a set of if-then rules.
- Decision tree learning is one of the most widely used and practical methods for inductive inference.
- It is robust to noisy data and capable of learning disjunctive expressions.
- Decision tree learning method searches a completely expressive hypothesis .
  - Avoids the difficulties of restricted hypothesis spaces.
  - Its inductive bias is a preference for small trees over large trees.
- The decision tree algorithms such as ID3, C4.5 are very popular inductive inference algorithms, and they are sucessfully applied to many leaning tasks.

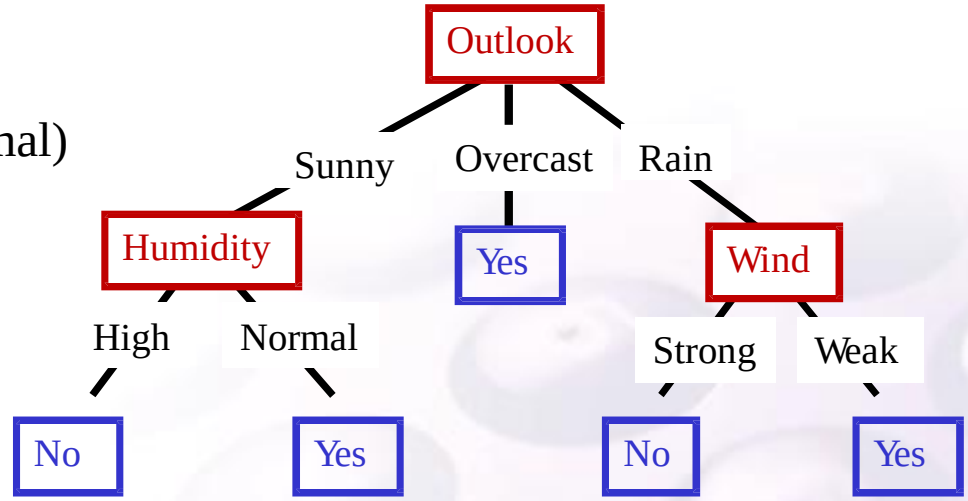
Decision Tree for PlayTennis



Decision Tree

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and
- The tree itself is a disjunction of these conjunctions.

$$(\text{Outlook} = \text{Sunny} \wedge \text{Humidity} = \text{Normal}) \vee (\text{Outlook} = \text{Overcast}) \vee (\text{Outlook} = \text{Rain} \wedge \text{Wind} = \text{Weak})$$



Decision Tree

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance.
- Each branch descending from a node corresponds to one of the possible values for the attribute.
- Each leaf node assigns a classification.
- The instance

  (Outlook=Sunny, Temperature=Hot, Humidity=High, Wind=Strong)

  is classified as a negative instance.

When to Consider Decision Trees

- Instances are represented by attribute-value pairs.
  - Fixed set of attributes, and the attributes take a small number of disjoint possible values.
- The target function has discrete output values.
  - Decision tree learning is appropriate for a boolean classification, but it easily extends to learning functions with more than two possible output values.
- Disjunctive descriptions may be required.
  - decision trees naturally represent disjunctive expressions.
- The training data may contain errors.
  - Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
- The training data may contain missing attribute values.
  - Decision tree methods can be used even when some training examples have unknown values.
- Decision tree learning has been applied to problems such as learning to classify
  - medical patients by their disease,
  - equipment malfunctions by their cause, and
  - loan applicants by their likelihood of defaulting on payments.

Top-Down Induction of Decision Trees -- ID3

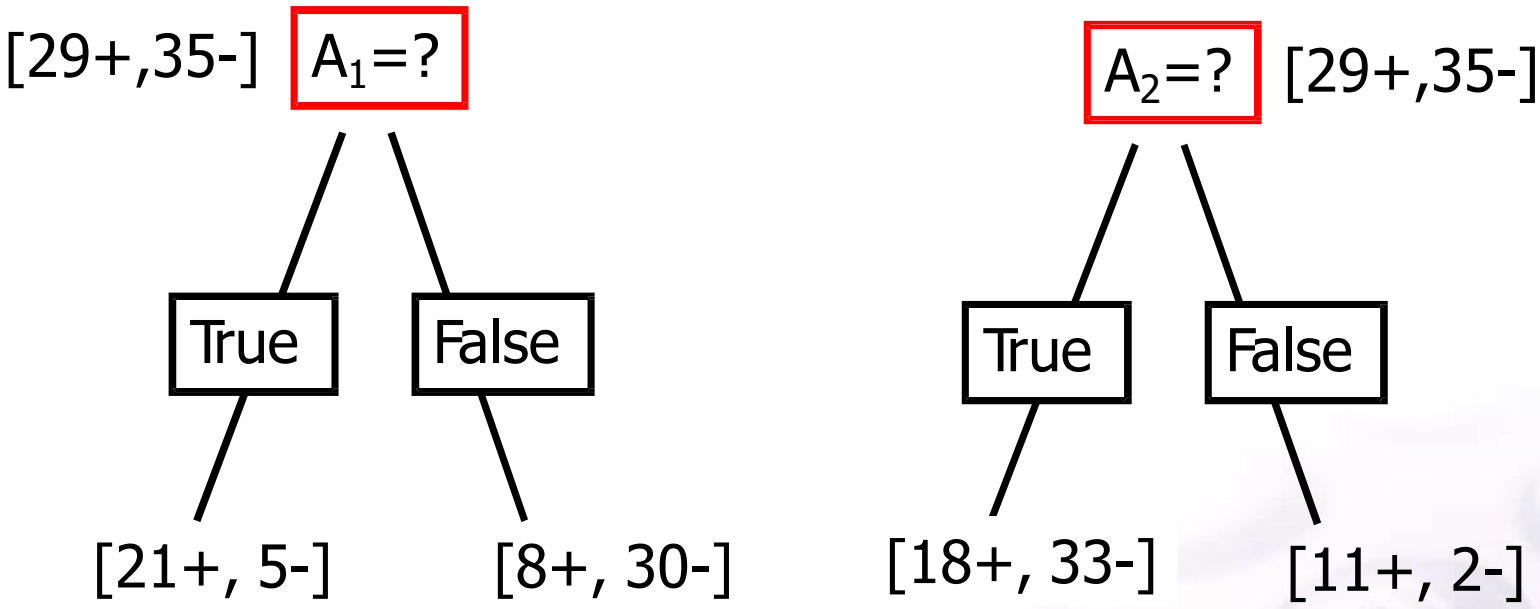
1. A ← the **"best"** decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A create new descendant *node*
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) STOP,

   else iterate over new leaf nodes.

Which Attribute is ”best”?

- We would like to select the attribute that is most useful for classifying examples.
- ***Informution gain*** measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses this *information gain* measure to select among the candidate attributes at each step while growing the tree.
- In order to define information gain precisely, we use a measure commonly used in information theory, called ***entropy***
- ***Entropy*** characterizes the (im)purity of an arbitrary collection of examples.

Which Attribute is ”best”?



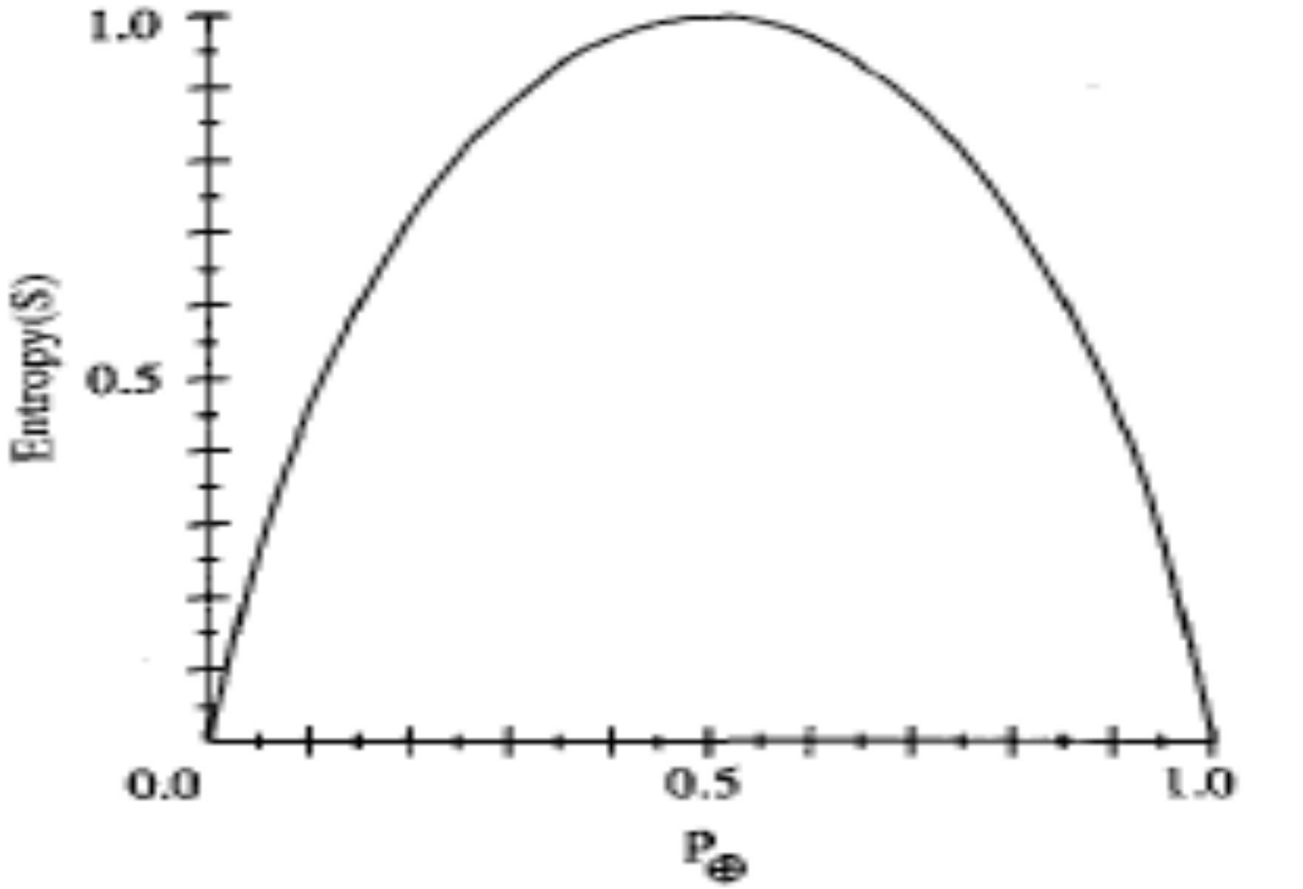
Entropy

- Given a collection S, containing positive and negative examples of some target concept, the ***entropy of S*** relative to this boolean classification is:

$$\mathbf{Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-}$$

- **S** is a sample of training examples
- $\mathbf{p_+}$ is the proportion of positive examples
- $\mathbf{p_-}$ is the proportion of negative examples

Entropy



Entropy

Entropy([9+,5-] = $-(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$

Entropy([12+,4-] = $-(12/16)\log_2(12/16) - (4/16)\log_2(4/16) = 0.811$

Entropy([12+,5-] = $-(12/17)\log_2(12/17) - (5/17)\log_2(5/17) = 0.874$

Entropy([8+,8-] = $-(8/16)\log_2(8/16) - (8/16)\log_2(8/16) = 1.0$

Entropy([8+,0-] = $-(8/8)\log_2(8/8) - (0/8)\log_2(0/8) = 0.0$

Entropy([0+,8-] = $-(0/8)\log_2(0/8) - (8/8)\log_2(8/8) = 0.0$

- It is assumed that $\log_2(0)$ is 0

Entropy – Information Theory

- Entropy(S)= expected number of bits needed to encode class (+ or -) of randomly drawn members of S (under the optimal, shortest length-code)
  - if p+ is 1, the receiver knows the drawn example will be positive, so no message need be sent, and the entropy is zero.
  - if p+ is 0.5, one bit is required to indicate whether the drawn example is positive or negative.
  - if p+ is 0.8, then a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to collections of positive examples and longer codes to less likely negative examples.
- Information theory optimal length code assign $\mathbf{-\log_2 p}$ bits to messages having probability p.
- So the expected number of bits to encode (+ or -) of random member of S:

$$- p_+ \log_2 p_+ - p_- \log_2 p_-$$

Entropy – Non-Boolean Target Classification

- If the target attribute can take on c different values, then the entropy of S relative to this c-wise classification is defined as

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

- p_i is the proportion of S belonging to class i.
- The logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits.
- If the target attribute can take on c possible values, the entropy can be as large as $\log_2 c$.

Information Gain

- ***entropy*** is a measure of the impurity in a collection of training examples
- ***information gain*** is a measure of the effectiveness of an attribute in classifying the training data.
- ***information gain*** measures the expected reduction in entropy by partitioning the examples according to an attribute.

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} (|S_v| / |S|) \text{Entropy}(S_v)$$

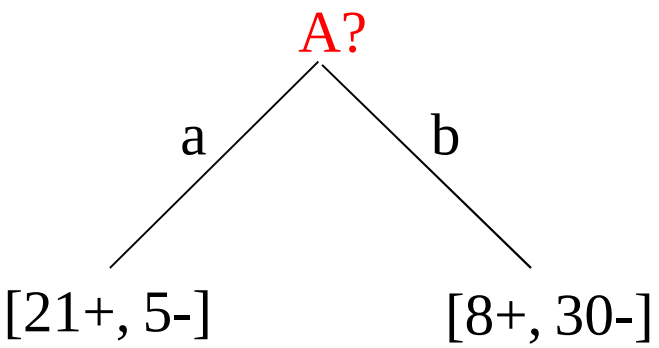
- S – a collection of examples
- A – an attribute
- Values(A) – possible values of attribute A
- S_v – the subset of S for which attribute A has value v

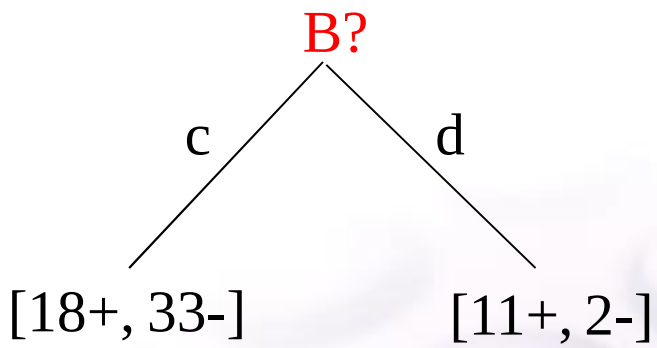
Information Gain

$$
\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &\leftarrow [6+, 2-] \\
S_{Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Weak}) \\
&\quad - (6/14) Entropy(S_{Strong}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}
$$

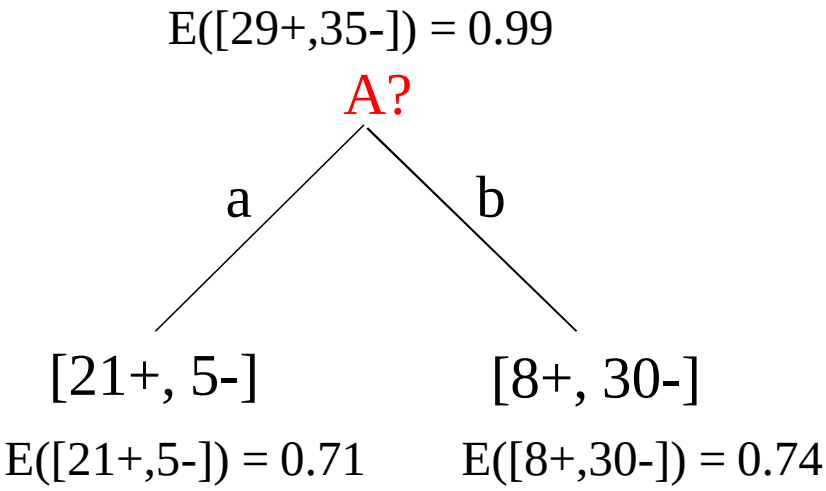
Which attribute is the best classifier?

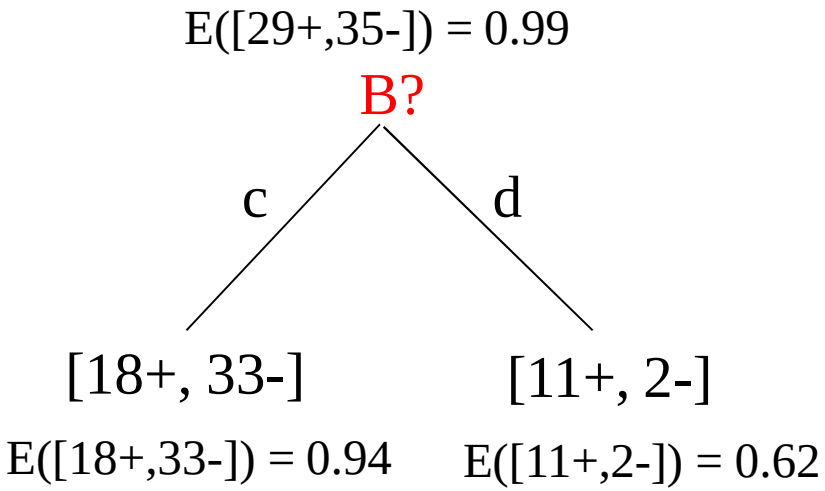
- S: [29+,35-] Attributes: A and B
- possible values for A: a,b possible values for B: c,d
- Entropy([29+,35-]) = $-29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$





Which attribute is the best classifier?





Gain(S,A) = Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
= **0.27**

Gain(S,B) = Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
= **0.12**

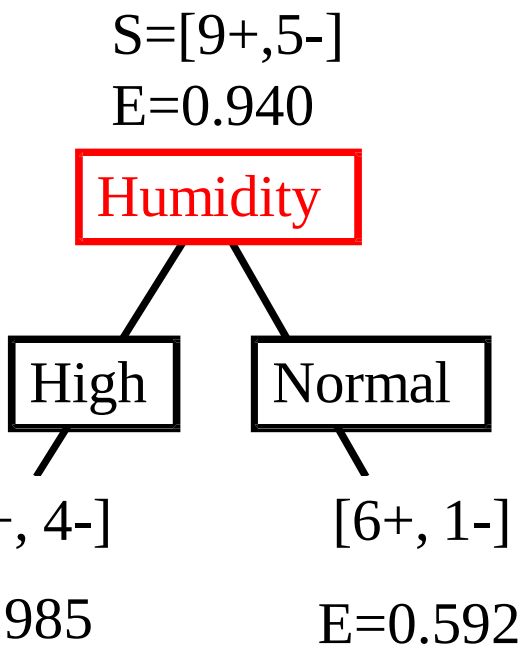
A provides greater information gain than B.
A is a better classifier than B.

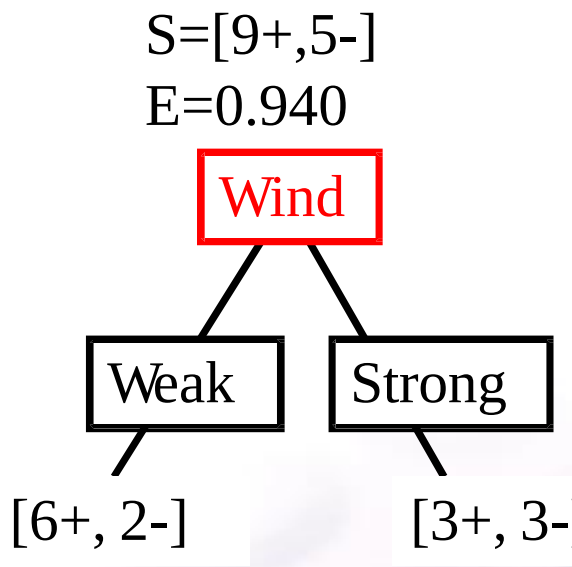
ID3 - Training Examples – [9+,5-]

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 – Selecting Next Attribute

Entropy([9+,5-] = $-(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$

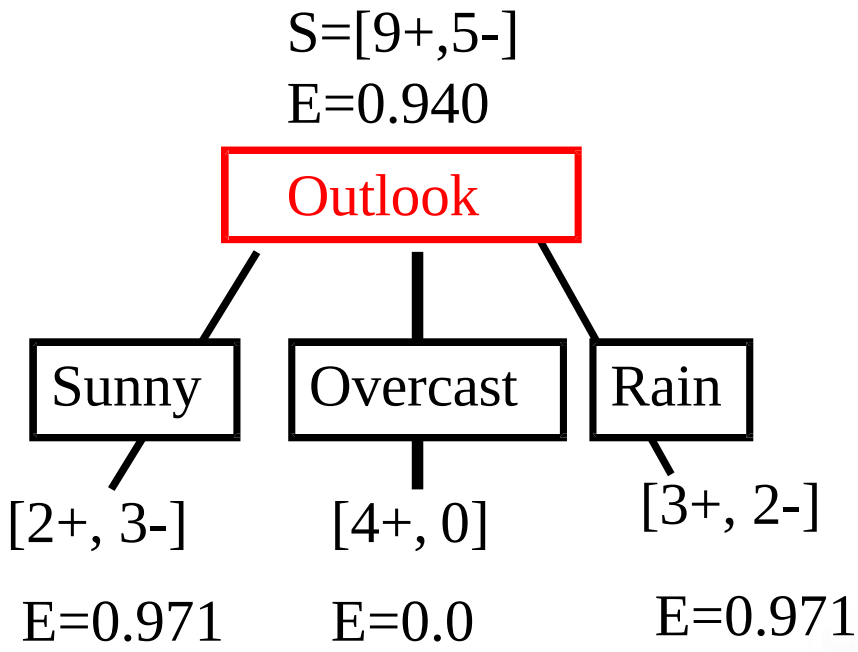




Gain(S,Humidity) =
0.940-(7/14)*0.985-(7/14)*0.592
= **0.151**

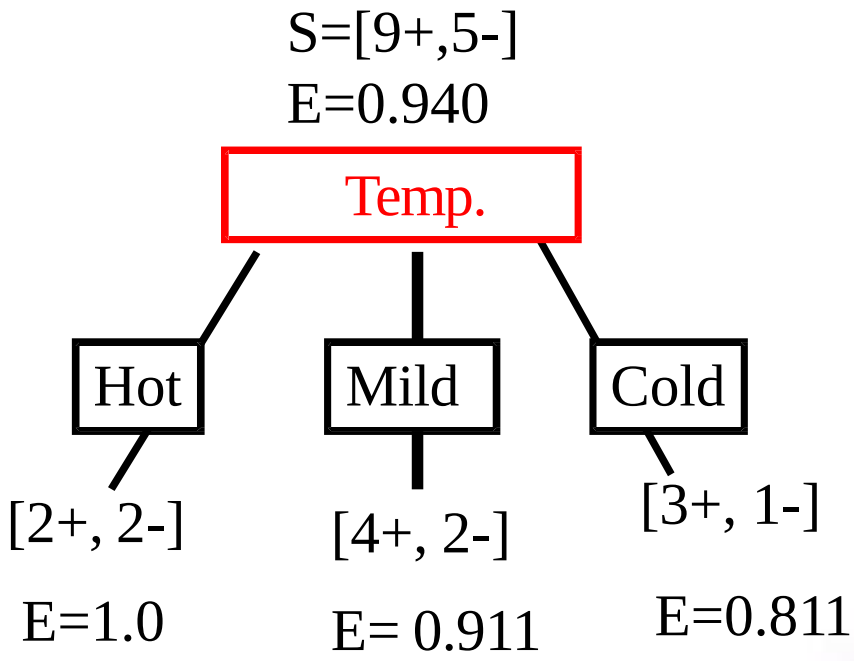
Gain(S,Wind) =
0.940-(8/14)*0.811-(6/14)*1.0
= **0.048**

ID3 – Selecting Next Attribute



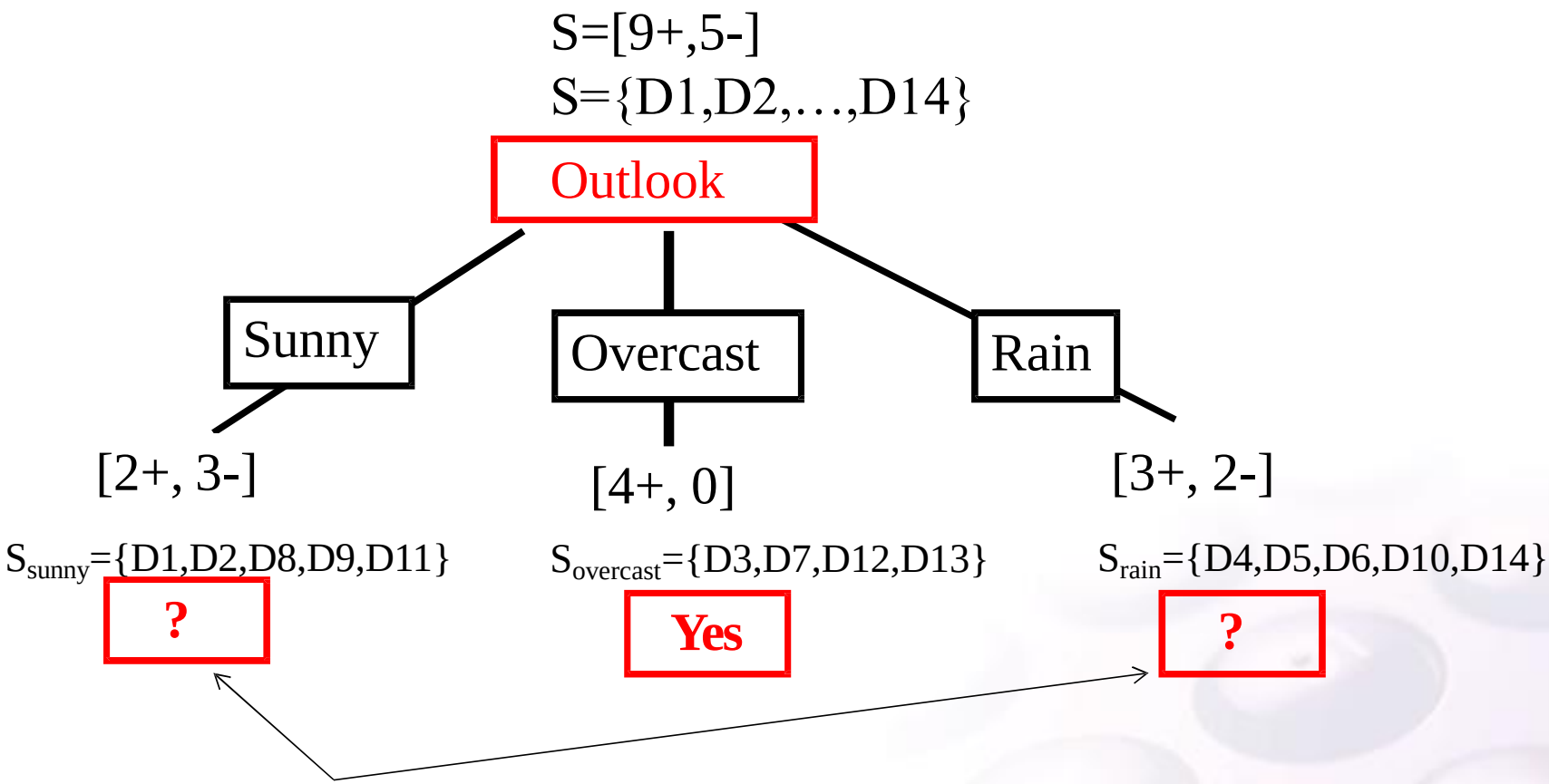
Gain(S,Outlook) =
0.940-(5/14)*0.971 -(4/14)*0.0 -(5/14)*0.0971
= **0.247**

ID3 – Selecting Next Attribute



Gain(S,Outlook) =
0.940-(4/14)*1.0 - (6/14)*0.911 - (4/14)*0.811
= **0.029**

Best Attribute - Outlook



Which attribute should be tested here?

ID3 - S_{sunny}

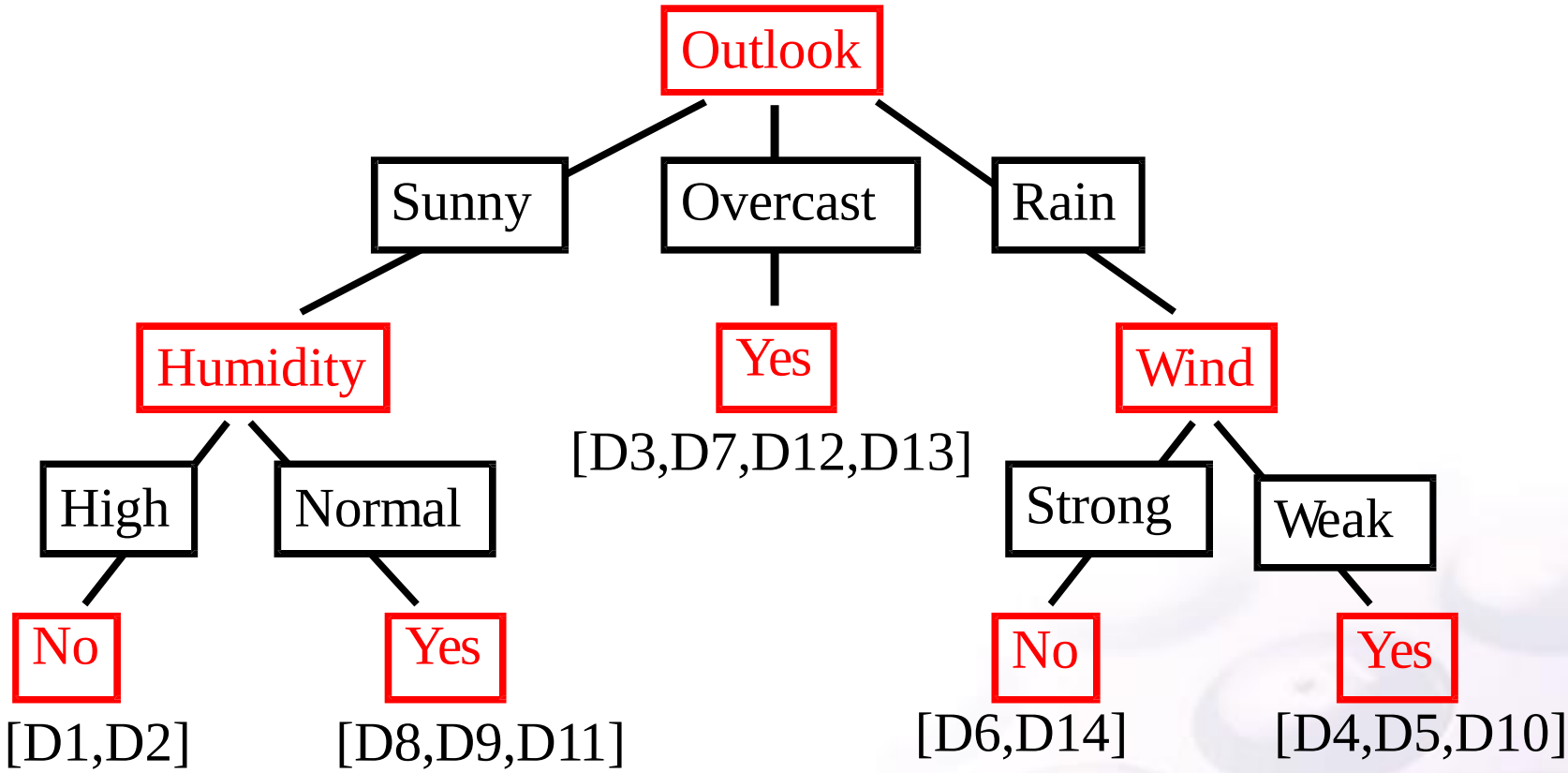
$Gain(S_{sunny}, Humidity) = 0.970-(3/5)0.0 – 2/5(0.0) = \mathbf{0.970}$

$Gain(S_{sunny}, Temp.) = 0.970-(2/5)0.0 –2/5(1.0)-(1/5)0.0 = 0.570$

$Gain(S_{sunny}, Wind) = 0.970= -(2/5)1.0 – 3/5(0.918) = 0.019$

So, Hummudity will be selected

ID3 - Result


Outlook
Sunny
Overcast
Rain
Humidity
Yes
[D3,D7,D12,D13]
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes
[D1,D2]
[D8,D9,D11]
[D6,D14]
[D4,D5,D10]

ID3 - Algorithm

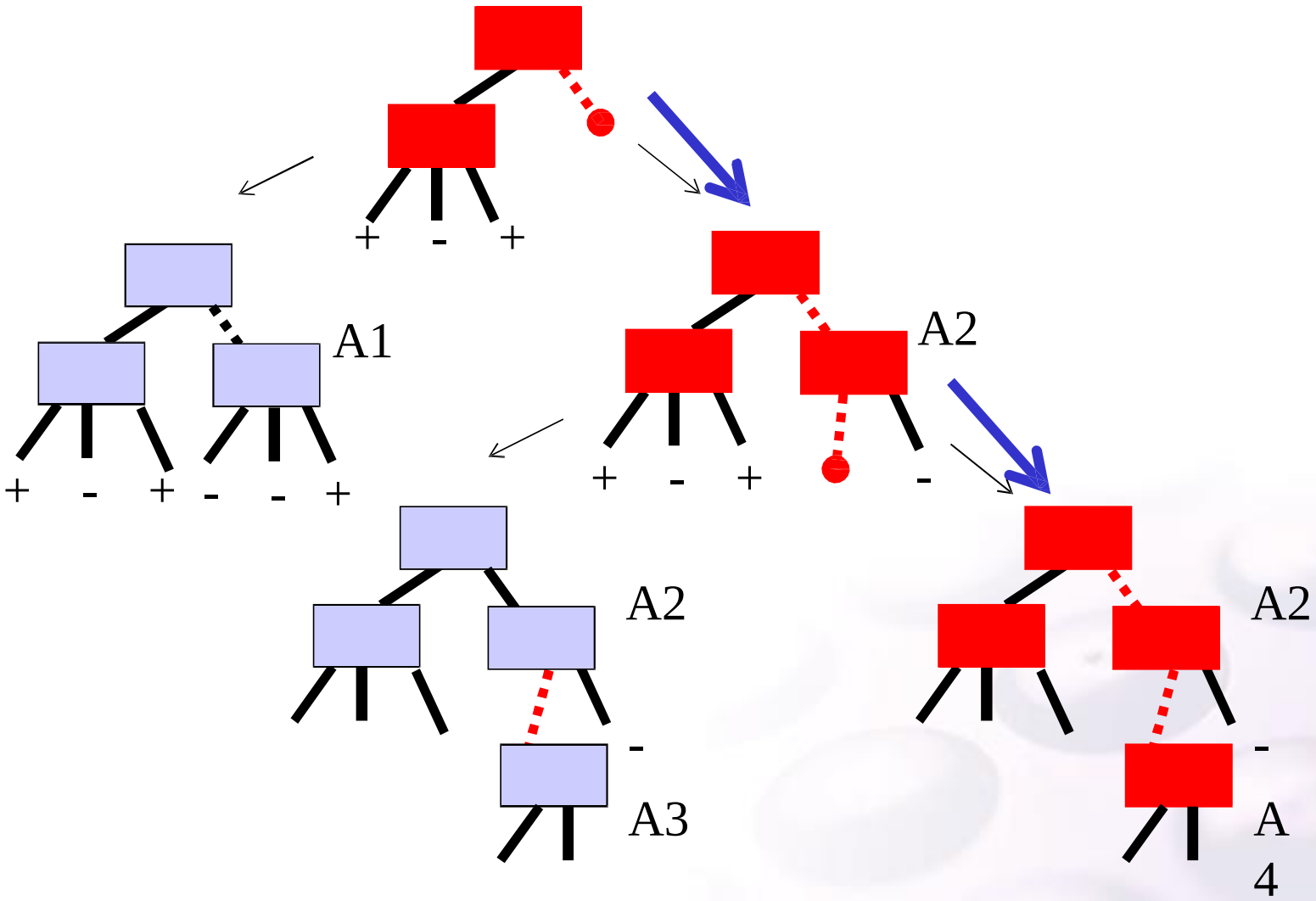
ID3(*Examples*, *TargetAttribute*, *Attributes*)

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = -
- If *Attributes* is empty, Return the single-node tree Root, with label = most common value of *TargetAttribute* in *Examples*
- Otherwise Begin
  - A ← the attribute from *Attributes* that best classifies *Examples*
  - The decision attribute for *Root* ←A
  - For each possible value, vi, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = vi
    - Let $Examples_{vi}$ be the subset of *Examples* that have value vi for A
    - If $Examples_{vi}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *TargetAttribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{vi}$, *TargetAttribute*, *Attributes* – {A})
- End
- Return *Root*

Hypothesis Space Search in Decision Tree Learning (ID3)

- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a simple-to complex, hill-climbing search through this hypothesis space,
- Begins with the empty tree, then considers progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data.
- The information gain measure guides the hill-climbing search.

Hypothesis Space Search in Decision Tree Learning (ID3)



ID3 - Capabilities and Limitations

- ID3's hypothesis space of all decision trees is a **complete** space of finite discrete-valued functions.
  - Every finite discrete-valued function can be represented by some decision tree.
  - Target function is surely in the hypothesis space.
- ID3 maintains only a single current hypothesis, and outputs only a single hypothesis.
  - ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.
  - ID3 cannot determine how many alternative decision trees are consistent with the available training data.
- No backtracking on selected attributes (greedy search)
  - Local minimal (suboptimal splits)
- Statistically-based search choices
  - Robust to noisy data

Inductive Bias in ID3

- ID3 search strategy
  - selects in favor of shorter trees over longer ones,
  - selects trees that place the attributes with highest information gain closest to the root.
  - because ID3 uses the information gain heuristic and a hill climbing strategy, it does not always find the shortest consistent tree, and it is biased to favor trees that place attributes with high information gain closest to the root.

Inductive Bias of ID3:

- Shorter trees are preferred over longer trees.
- Trees that place high information gain attributes close to the root are preferred over those that do not.

Inductive Bias in ID3 - Occam's Razor

OCCAM'S RAZOR: Prefer the simplest hypothesis that fits the data.

Why prefer short hypotheses?

Argument in favor:

- Fewer short hypotheses than long hypotheses
- A short hypothesis that fits the data is unlikely to be a coincidence
- A long hypothesis that fits the data might be a coincidence

Argument opposed:

- There are many ways to define small sets of hypotheses
- What is so special about small sets based on *size* of hypothesis

Inductive Bias in ID3 – Restriction Bias and Preference Bias

- ID3 searches a complete hypothesis space
  - It searches **incompletely** through this space, from simple to complex hypotheses, until its termination condition is met
  - Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy.
  - Its hypothesis space introduces no additional bias.
- Candidate Elimanation searches an incomplete hypothesis space
  - It searches this space completely, finding every hypothesis consistent with the training data.
  - Its inductive bias is solely a consequence of the expressive power of its hypothesis representation.
  - Its search strategy introduces no additional bias.

Inductive Bias in ID3 – Restriction Bias and Preference Bias

- The inductive bias of ID3 is a preference for certain hypotheses over others, with no hard restriction on the hypotheses that can be eventually enumerated.

  ➔ **preference bias** (***search bias***).
- The inductive bias of Candidate Elimination is in the form of a categorical restriction on the set of hypotheses considered.

  ➔ **restriction bias** (***language bias***)

- A *preference bias* is more desirable than a *restriction bias*, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.

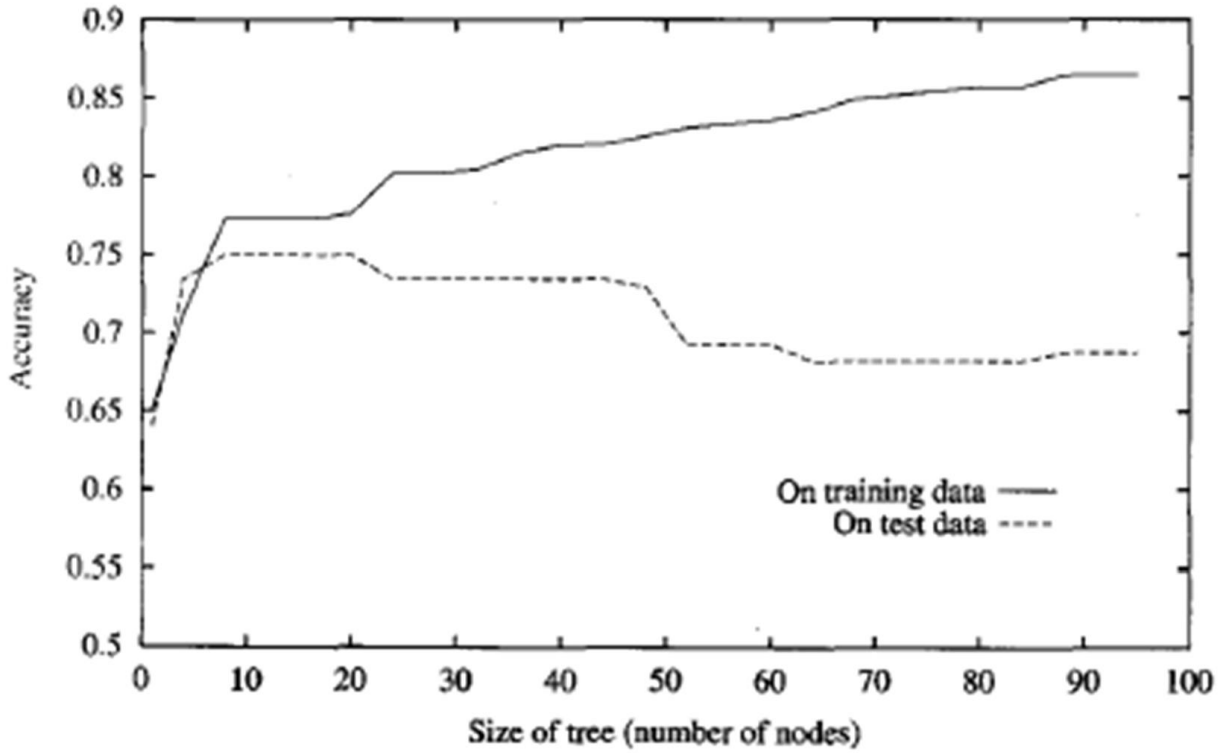
Overfitting

Given a hypothesis space H, a hypothesis $h \in H$ is said to ***OVERFIT*** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

Reasons for overfitting:

- Errors and noise in training examples
- Coincidental regularities (especially small number of examples are associated with leaf nodes).

Overfitting



- As ID3 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically.
- However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases.

Avoid Overfitting

How can we avoid overfitting?

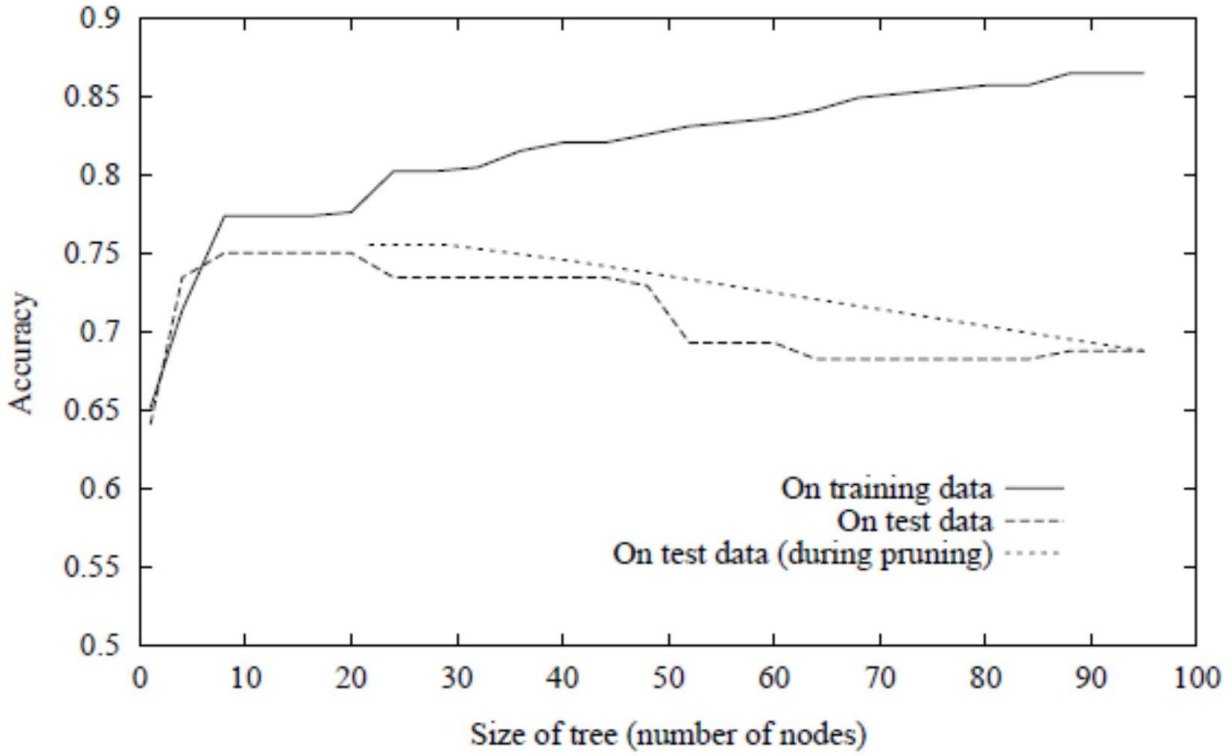
- Stop growing when data split not statistically significant
  - stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- Grow full tree then post-prune
  - allow the tree to overfit the data, and then post-prune the tree.

- The correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size.
  - Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
  - Use all the available data for training, but apply a statistical test to estimate whether expanding a particular node is likely to produce an improvement beyond the training set. (chi-square test)

Avoid Overfitting - Reduced-Error Pruning

- Split data into *training* and *validation* set
- Do until further pruning is harmful:
  - Evaluate impact on *validation* set of pruning each possible node (plus those below it)
  - Greedily remove the one that most improves the *validation* set accuracy
- Pruning of nodes continues until further pruning is harmful (i.e., decreases accuracy of the tree over the *validation* set).
- Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available.
  - The major drawback of this approach is that when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.

Effect of Reduced Error Pruning

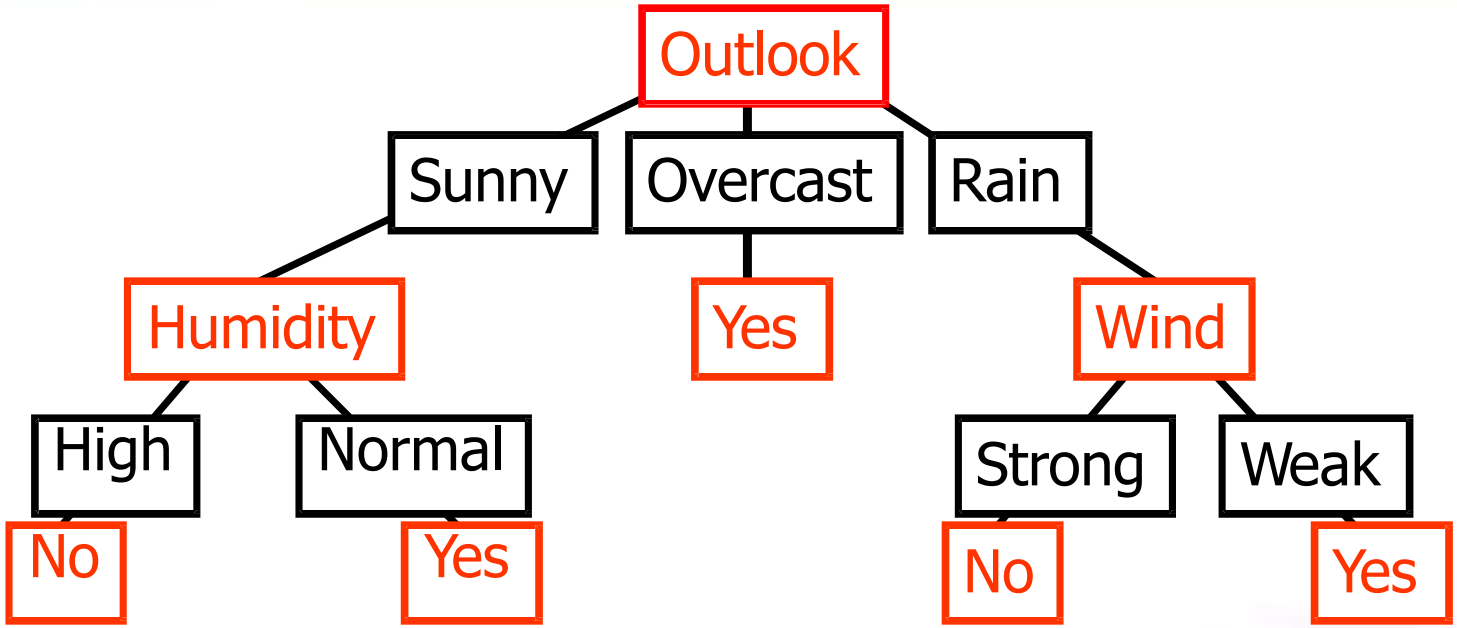


- the accuracy increases over the test set as nodes are pruned from the tree.
- the validation set used for pruning is distinct from both the training and test sets.

Rule-Post Pruning

- Rule-Post Prunning is another sucessful method for finding high accuracy hypotheses.
- It is used by C4.5 learning algorithm (an extension of ID3).
- *Steps of Rule-Post Pruning:*
  - Infer the decision tree from the training set.
  - Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
  - Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
  - Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Converting a Decision Tree to Rules



R1: If (Outlook=Sunny) $\wedge$ (Humidity=High) Then PlayTennis=No
R2: If (Outlook=Sunny) $\wedge$ (Humidity=Normal) Then PlayTennis=Yes
R3: If (Outlook=Overcast) Then PlayTennis=Yes
R4: If (Outlook=Rain) $\wedge$ (Wind=Strong) Then PlayTennis=No
R5: If (Outlook=Rain) $\wedge$ (Wind=Weak) Then PlayTennis=Yes

Pruning Rules

- Each rule is pruned by removing any antecedent (precondition).
  - Ex. Prune R1 by removing (Outlook=Sunny) or (Humidity=High)
- Select whichever of the pruning steps produced the greatest improvement in estimated rule accuracy.
  - Then, continue with other preconditions.
  - No pruning step is performed if it reduces the estimated rule accuracy.
- In order to estimate rule accuracy:
  - use a validation set of examples disjoint from the training set
  - evaluate performance based on the training set itself(using statistical techniques). C4.5 uses this approach.

Why Convert The Decision Tree To Rules Before Pruning?

- Converting to rules improves readability.
  - Rules are often easier for to understand.
- Distinguishing different contexts in which a node is used
  - separate pruning decision for each path
- No difference for root/inner
  - no bookkeeping on how to reorganize tree if root node is pruned

Continuous-Valued Attributes

- ID3 is restricted to attributes that take on a discrete set of values.
- Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals
- For a continuous-valued attribute A that is, create a new boolean attribute Ac, that is true if A < c and false otherwise.
  - Select c using information gain
  - Sort examples according to the continuous attributeA,
  - Then identify adjacent examples that differ in their target classification
  - Generate candidate thresholds midway between corresponding values of A.
  - The value of c that maximizes information gain must always lie at a boundary.
  - These candidate thresholds can then be evaluated by computing the information gain associated with each.

Continuous-Valued Attributes - Example

Temperature:	40	48	60	72	80	90
PlayTennis :	No	No	Yes	Yes	Yes	No

Two candidate thresholds: (48+60)/2=54 (80+90)/2=85

Check the information gain for new boolean attributes:

Temperature$_{>54}$ Temperature$_{>85}$

Use these new new boolean attributes same as other discrete valued attributes.

Alternative Selection Measures

- Information gain measure favors attributes with many values
  - separates data into small subsets
  - high gain, poor prediction
- Ex. Date attribute has many values, and may separate training examples into very small subsets (even singleton sets – perfect partitions)
  - Information gain will be very high for Date attribute.
  - Perfect partition ➔maximum gain : Gain(S,Date) = Entropy(S) – 0 = Entropy(S) because $\log_2 1$ is 0.
  - It has high information gain, but very poor predictor for unseen data.
- There are alternative selection measures such as *GainRatio* measure based on *SplitInformation*

Split Information

- The *gain ratio* measure penalizes attributes with many values (such as Date) by incorporating a term, called *split informution*

$$\text{SplitInformation}(S,A) = -\sum_{i=1}^{c} (|Si| / |S|) \log_2 (|Si| / |S|)$$

- Split information for boolean attributes is 1 ($= \log_2 2$),

  GainRatio(S,A) = Gain(S,A) / SplitInformation(S,A)

- Split information fot attributes for n values is $\log_2 n$
  - *SplitInformation* term discourages the selection of attributes with many uniformly distributed values.

Practical Issues on Split Information

- Some value ‘rules’
  - |Si| close to |S|
  - SplitInformation 0 or very small
  - GainRatio undefined or very large
- Apply heuristics to select attributes
  - compute Gain first
  - compute GainRatio only when Gain large enough (above average Gain)

Missing Attribute Values

- The available data may be missing values for some attributes.
- It is common to estimate the missing attribute value based on other examples for which this attribute has a known value.
- Assume that an example (with classification c) in S has a missing value for attribute A.
  - Assign the most common value of A in S.
  - Assign the most common value of in the examples having c classification in S.
  - Or, use probabilty value for each possible attribute value.

Attributes with Differing Costs

- Measuring attribute costs something
  - prefer cheap ones if possible
  - use costly ones only if good gain
  - introduce cost term in selection measure
  - no guarantee in finding optimum, but give bias towards cheapest
- Example applications
  - robot & sonar: time required to position
  - medical diagnosis: cost of a laboratory test

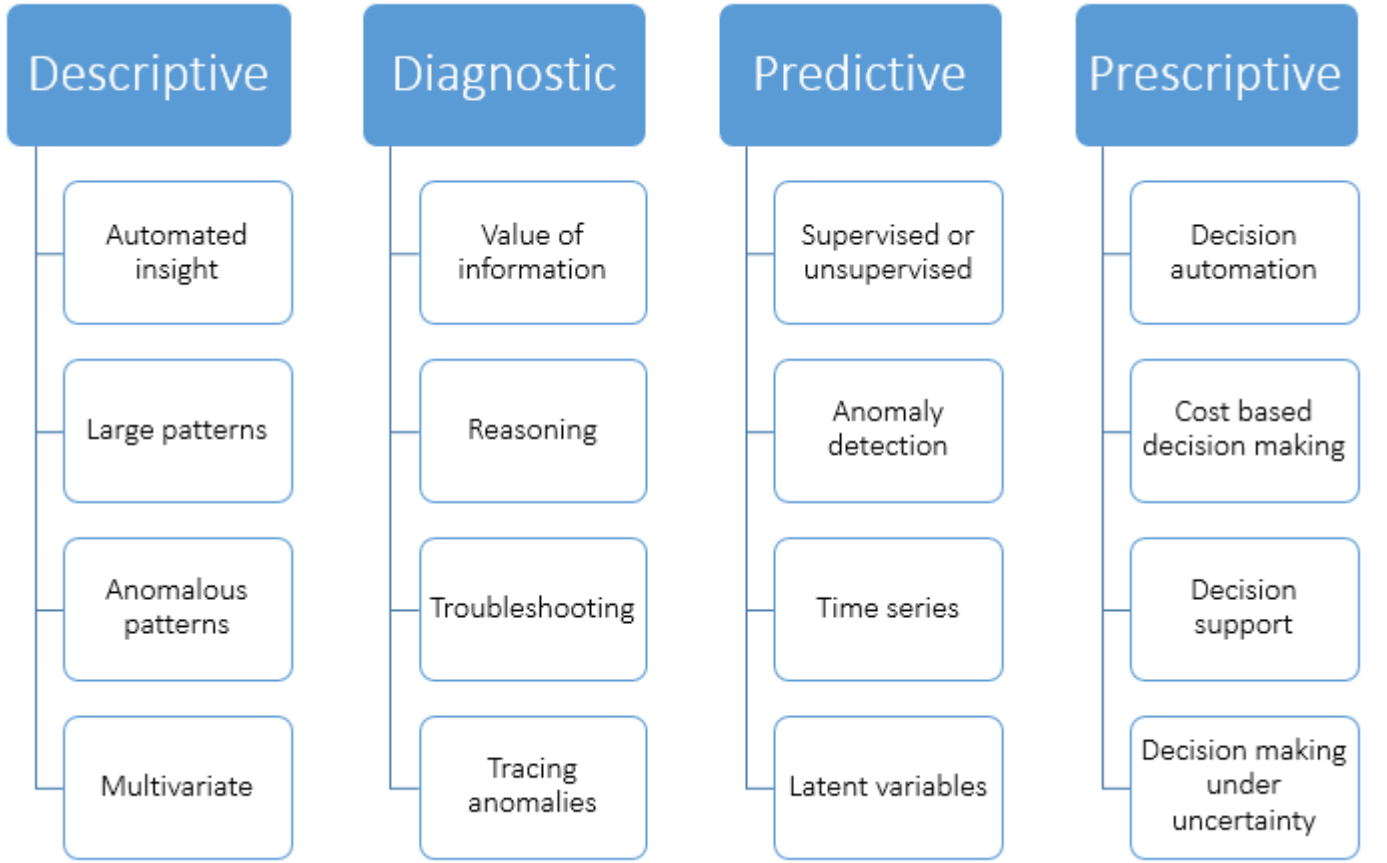
Main Points with Decision Tree Learning

- Decision tree learning provides a practical method for concept learning and for learning other discrete-valued functions.
  - decision trees are inferred by growing them from the root downward, greedily selecting the next best attribute.
- ID3 searches a complete hypothesis space.
- The inductive bias in ID3 includes a *preference* for smaller trees.
- Overfitting training data is an important issue in decision tree learning.
  - Pruning decision trees or rules are important.
- A large variety of extensions to the basic ID3 algorithm has been developed. These extensions include methods for
  - post-pruning trees, handling real-valued attributes, accommodating training examples with missing attribute values, using attribute selection measures other than information gain, and considering costs associated with instance attributes.

Bayesian Networks

Bayesian networks are a type of **Probabilistic Graphical Model** that can be used to build models from data and/or expert opinion.
They can be used for a wide range of tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction and decision making under uncertainty. Figure below shows these capabilities in terms of the four major analytics disciplines, **Descriptive analytics**, **Diagnostic analytics**, **Predictive analytics** and **Prescriptive analytics**.

Descriptive, diagnostic, predictive & prescriptive analytics with Bayesian networks



Descriptive, diagnostic, predictive & prescriptive analytics with Bayesian networks

Bayesian Networks

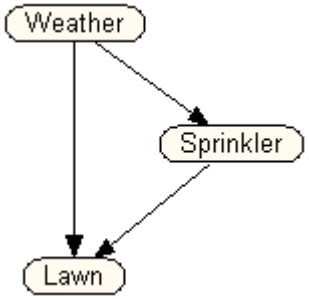
They are also commonly referred to as **Bayes nets**, **Belief networks** and sometimes **Causal networks**.

A Bayes net is a model. It reflects the states of some part of a world that is being modeled and it describes how those states are related by probabilities. The model might be of your house, or your car, your body, your community, an ecosystem, a stock-market, etc. Absolutely anything can be modeled by a Bayes net. All the possible states of the model represent all the possible worlds that can exist, that is, all the possible ways that the parts or states can be configured. The car engine can be running normally or giving trouble. It's tires can be inflated or flat. Your body can be sick or healthy, and so on.

Bayesian Networks

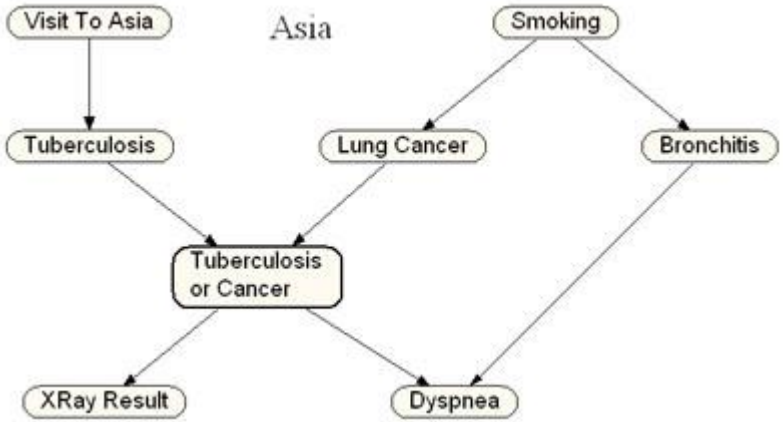
So where do the probabilities come in? Well, typically some states will tend to occur more frequently when other states are present. Thus, if you are sick, the chances of a runny nose are higher. If it is cloudy, the chances of rain are higher, and so on.
Here is a simple Bayes net that illustrates these concepts. In this simple world, let us say the weather can have three states: sunny, cloudy, or rainy, also that the grass can be wet or dry, and that the sprinkler can be on or off. Now there are some causal links in this world. If it is rainy, then it will make the grass wet directly. But if it is sunny for a long time, that too can make the grass wet, indirectly, by causing us to turn on the sprinkler.

Bayesian Networks



When actual probabilities are entered into this net that reflect the reality of real weather, lawn, and sprinkler-use-behavior, such a net can be made to answer a number of useful questions, like, "if the lawn is wet, what are the chances it was caused by rain or by the sprinkler", and "if the chance of rain increases, how does that affect my having to budget time for watering the lawn".
Here is another simple Bayes net called **Asia**. It is an example which is popular for introducing Bayes nets and is from Lauritzen&Spiegelhalter88. Note, it is for example purposes only, and should not be used for real decision making.

Bayesian Networks

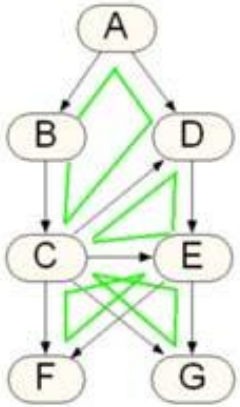


It is a simplified version of a network that could be used to diagnose patients arriving at a clinic. Each node in the network corresponds to some condition of the patient, for example, "Visit to Asia" indicates whether the patient recently visited Asia. The arrows (also called links) between any two nodes indicate that there are probability relationships that are know to exist between the states of those two nodes. Thus, smoking increases the chances of getting lung cancer and of getting bronchitis. Both lung cancer and bronchitis increase the chances of getting dyspnea (shortness of breath). Both lung cancer and tuberculosis, but not usually bronchitis, can cause an abnormal lung x-ray. And so on.

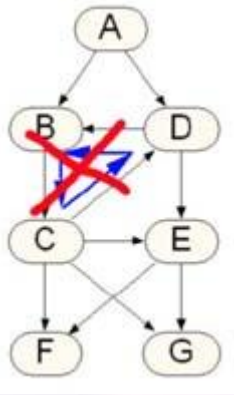
Bayesian Networks

The direction of the link arrows roughly corresponds to "causality". That is the nodes higher up in the diagram tend to influence those below rather than, or, at least, more so than the other way around.

In a Bayes net, the links may form loops, but they may not form cycles. This is not an expressive limitation; it does not limit the modeling power of these nets. It only means we must be more careful in building our nets. In the left diagram below, there are numerous loops. These are fine. In the right diagram, the addition of the link from D to B creates a cycle, which is not permitted.



A valid Bayes net



Not a Bayes net

Bayesian Networks

The key advantage of not allowing cycles it that it makes possible very fast update algorithms, since there is no way for probabilistic influence to "cycle around" indefinitely.
To diagnose a patient, values could be entered for some of nodes when they are known. This would allow us to re-calculate the probabilities for all the other nodes. Thus if we take a chest x-ray and the x-ray is abnormal, then the chances of the patient having TB or lung-cancer rise. If we further learn that our patient visited Asia, then the chances that they have tuberculosis would rise further, and of lung-cancer would drop (since the X-ray is now better explained by the presence of TB than of lung-cancer).

Support Vector Machines

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line.

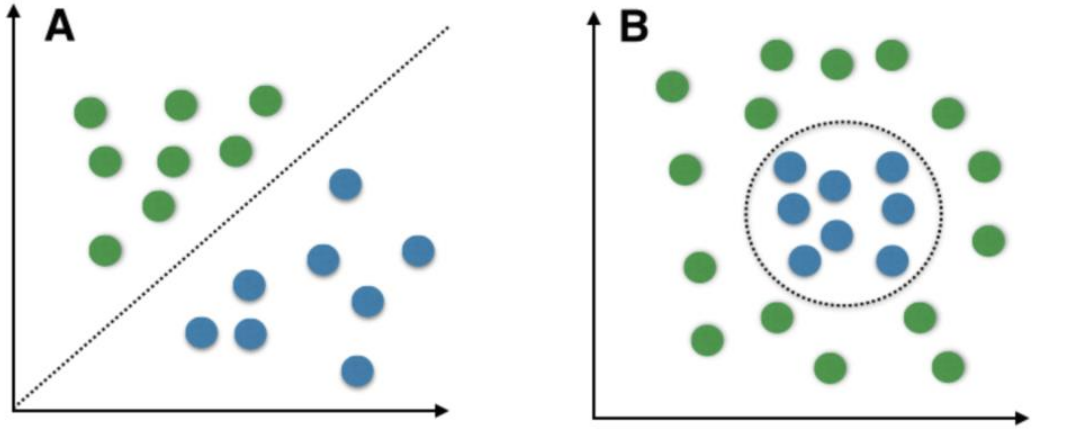
In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

Support Vector Machine for Multi-Class Problems

To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :
The data point belongs to that class OR
The data point does not belong to that class.
For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the ‘mango’ class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM.

SVM for complex (Non Linearly Separable)

SVM works very well without any modifications for linearly separable data. **Linearly Separable Data** is any data that can be plotted in a graph and can be separated into classes using a straight line.



A: Linearly Separable Data B: Non-Linearly Separable Data

Kernelized SVM

We use **Kernelized SVM** for non-linearly separable data. Say, we have some non-linearly separable data in one dimension. We can transform this data into two-dimensions and the data will become linearly separable in two dimensions. This is done by mapping each 1-D data point to a corresponding 2-D ordered pair.
So for any non-linearly separable data in any dimension, we can just map the data to a higher dimension and then make it linearly separable. This is a very powerful and general transformation.
A **kernel** is nothing a measure of similarity between data points.
The **kernel function** in a kernelized SVM tell you, that given two data points in the original feature space, what the similarity is between the points in the newly transformed feature space.

Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. **Each generation consist of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy –

Individual in population compete for resources and mate

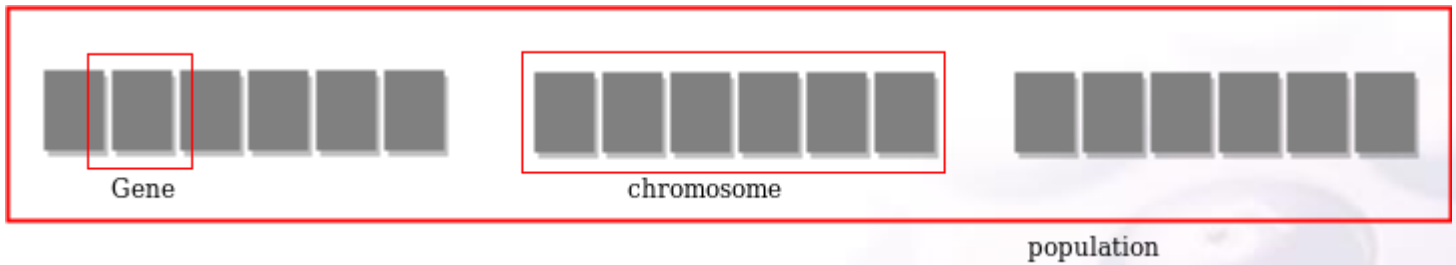
Those individuals who are successful (fittest) then mate to create more offspring than others

Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.

Thus each successive generation is more suited for their environment.

Search space

The population of individuals are maintained within search space. Each individual represent a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores. The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die. Each new generation has on average more "better genes" than the individual (solution) of previous generations. Thus each new generations have better **"partial solutions"** than previous generations. Once the offsprings produced having no significant difference than offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

Issues in Machine Learning

- What algorithms exist for learning general target functions from specific training examples ?
- How does the number of training examples influence accuracy ?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples ?

Issues in Machine Learning (cont.)

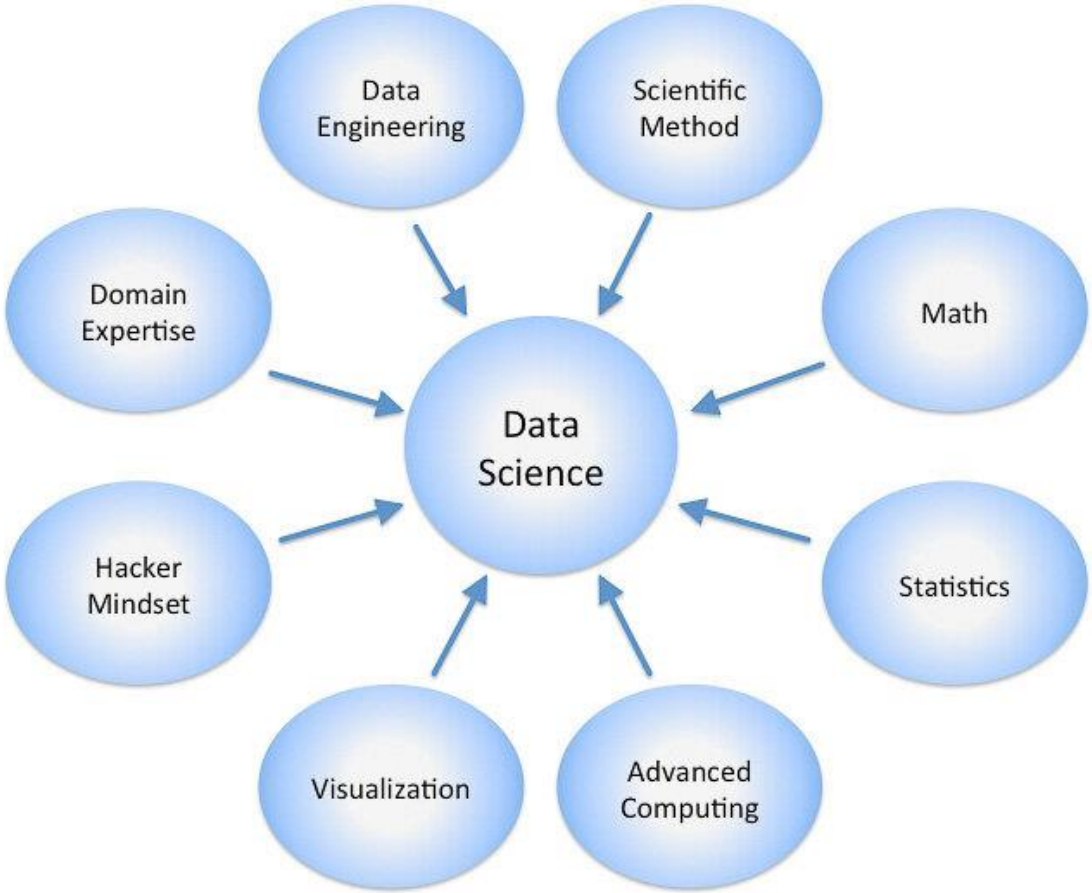
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem ?
- What is the best way to reduce the learning task to one or more function approximation problems ?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function ?

Data Science

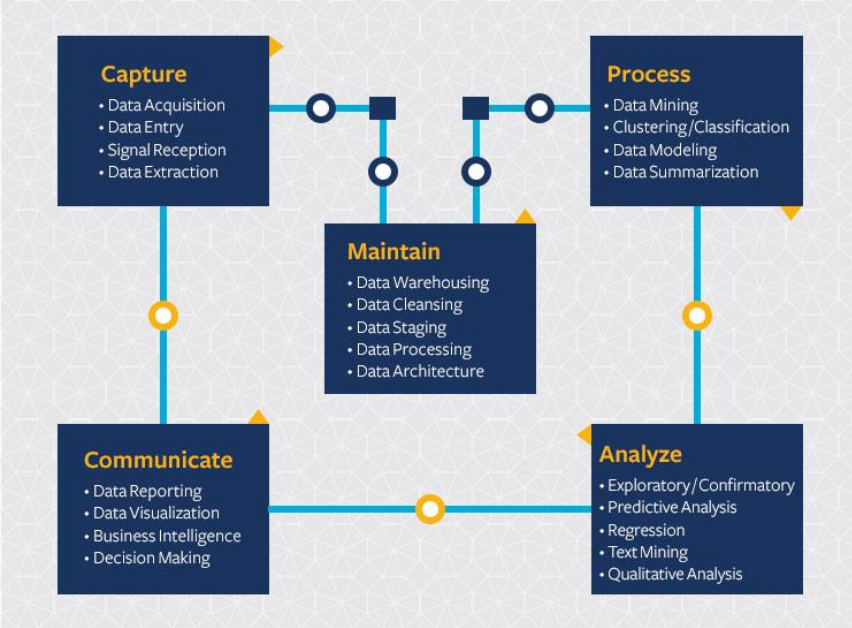
Data science is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from many structural and unstructured data. Data science is related to data mining, machine learning and big data.

Data science continues to evolve as one of the most promising and in-demand career paths for skilled professionals. Today, successful data professionals understand that they must advance past the traditional skills of analyzing large amounts of data, data mining, and programming skills. In order to uncover useful intelligence for their organizations, data scientists must master the full spectrum of the data science life cycle and possess a level of flexibility and understanding to maximize returns at each phase of the process.

Common Disciplines of a Data Scientist



The Data Science Life Cycle



The image represents the five stages of the data science life cycle: ***Capture****, (data acquisition, data entry, signal reception, data extraction);* ***Maintain*** *(data warehousing, data cleansing, data staging, data processing, data architecture);* ***Process*** *(data mining, clustering/classification, data modeling, data summarization);* ***Analyze*** *(exploratory/confirmatory, predictive analysis, regression, text mining, qualitative analysis);* ***Communicate*** *(data reporting, data visualization, business intelligence, decision making).*

Data Science vs Machine Learning

Data Science vs. Machine Learning

Subject	Data Science	Machine Learning
Scope	Create Insights from data, dealing with all real-world complexities	Accurately classify or predict outcomes for new data points by learning patterns from historical data, using mathematical models.
Input Data	Most of the input data is generated as human consumable data which is to be read or analyzed by humans like tabular data or images	Input data for ML will be transformed specifically for algorithms used. Feature scaling, word embedding or adding polynomial features are some examples.
System Complexity	Components for handling unstructured raw data coming.	Major complexity is with algorithms and mathematical concepts behind that.
Preferred Skillset	Domain expertise, ETL and data profiling, strong SQL, Visualization	Strong maths understanding, Python/R programming, Data wrangling with SQL model-specific visualization

THANK YOU