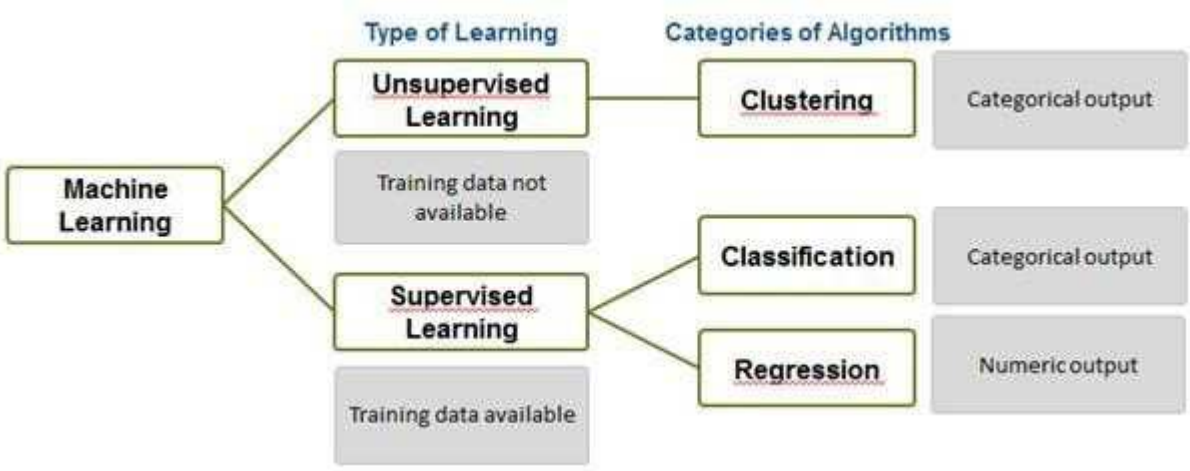
Unit-4

Machine Learning : Supervised and unsupervised learning, Decision trees, Statistical learning models, Learning with complete data - Naive Bayes models, Learning with hidden data – EM algorithm, Reinforcement learning,

4.1 Machine Learning

Machine learning is a scientific discipline that *deals with the construction and study of algorithms that can learn from data. Such algorithms operate by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions.* Example applications include *spam filtering, optical character recognition (OCR), search engines and computer vision.*



Types of problems/tasks

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. These are:

- **Supervised learning**. The computer is presented with example *inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.*
- **Unsupervised learning**, *no labels are given to the learning algorithm, leaving it on its own to find structure in its input*. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end.
- In **reinforcement learning**, a computer program interacts with a *dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not*. Another example is learning to play a game by playing against an opponent.

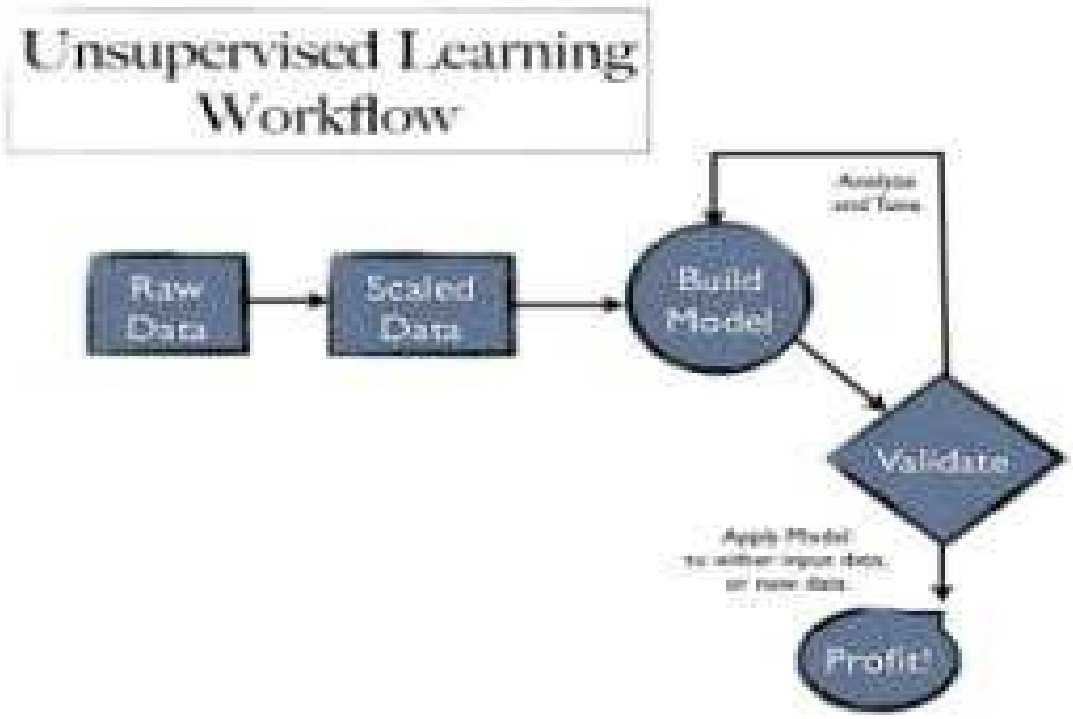
Approaches

1. **Decision tree learning**
   Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value.

2. **Association rule learning**
   Association rule learning is a method for discovering interesting relations between variables in large databases.
3. **Artificial neural networks**
   An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is inspired by the structure and functional aspects of biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.
4. **Inductive logic programming**
   Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses.
5. **Support vector machines**
   Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.
6. **Clustering**
   Cluster analysis is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to some pre designated criterion or criteria, while observations drawn from different clusters are dissimilar.
7. **Bayesian networks**
   A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG).
   For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.
8. **Reinforcement learning**
   Reinforcement learning is concerned with how an *agent* has to take *actions* in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states.
   Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.
9. **Genetic algorithm**
   A genetic algorithm (GA) is a search heuristic that mimics the process of natural selection, and uses methods such as mutation and crossover to generate new genotype in the hope of finding good solutions to a given problem.

4.2 Unsupervised learning

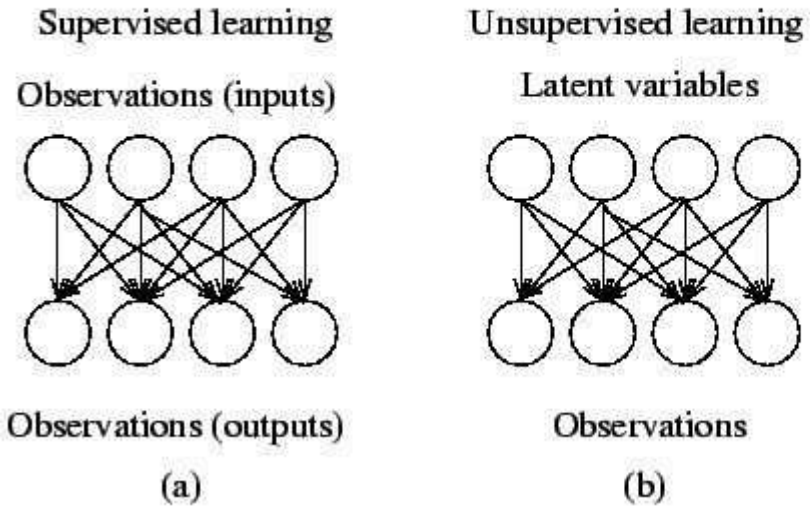
In machine learning, the problem of unsupervised learning is that of *trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution*. This distinguishes unsupervised learning from supervised learning and reinforcement learning.
Unsupervised learning is closely related to the *problem of density estimation in statistics.* However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data.



Approaches to unsupervised learning include:

- clustering (e.g., k-means, mixture models, hierarchical clustering),
- hidden Markov models,
- blind signal separation using feature extraction techniques for dimensionality reduction (e.g., principal component analysis, independent component analysis, non-negative matrix factorization, singular value decomposition).

Figure 2: The causal structure of (a) supervised and (b) unsupervised learning. In supervised learning, one set of observations, called inputs, is assumed to be the cause of another set of observations, called outputs, while in unsupervised learning all observations are assumed to be caused by a set of latent variables.



4.3 Supervised learning

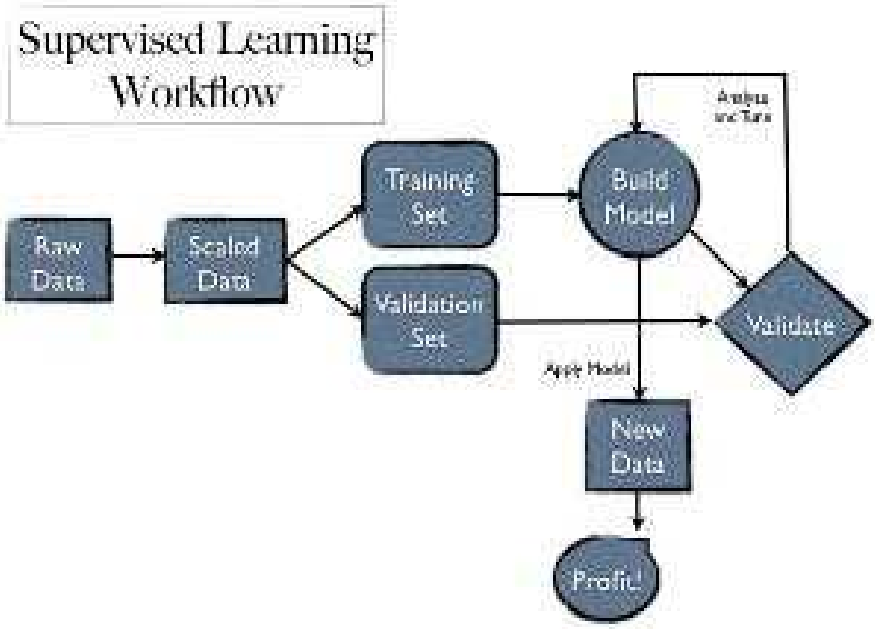
Supervised learning is the machine learning task of *inferring a function from labeled training data.* The training data consist of a set of *training examples*. In supervised learning, each example *is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal).* A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples

In order to solve a given problem of supervised learning, one has to perform the following steps:

1. **Determine the type of training examples**. Before doing anything else, the user should decide what kind of data is to be used as a training set. In the case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
2. **Gather a training set.** The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
3. **Determine the input feature representation of the learned function**. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object.
4. **Determine the structure of the learned function and corresponding learning algorithm**. For example, the engineer may choose to use support vector machines or decision trees.
5. **Complete the design**. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters.

These parameters may be adjusted by optimizing performance on a subset (called a *validation* set) of the training set, or via cross-validation.

6. **Evaluate the accuracy of the learned function**. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.



4.4 Decision tree

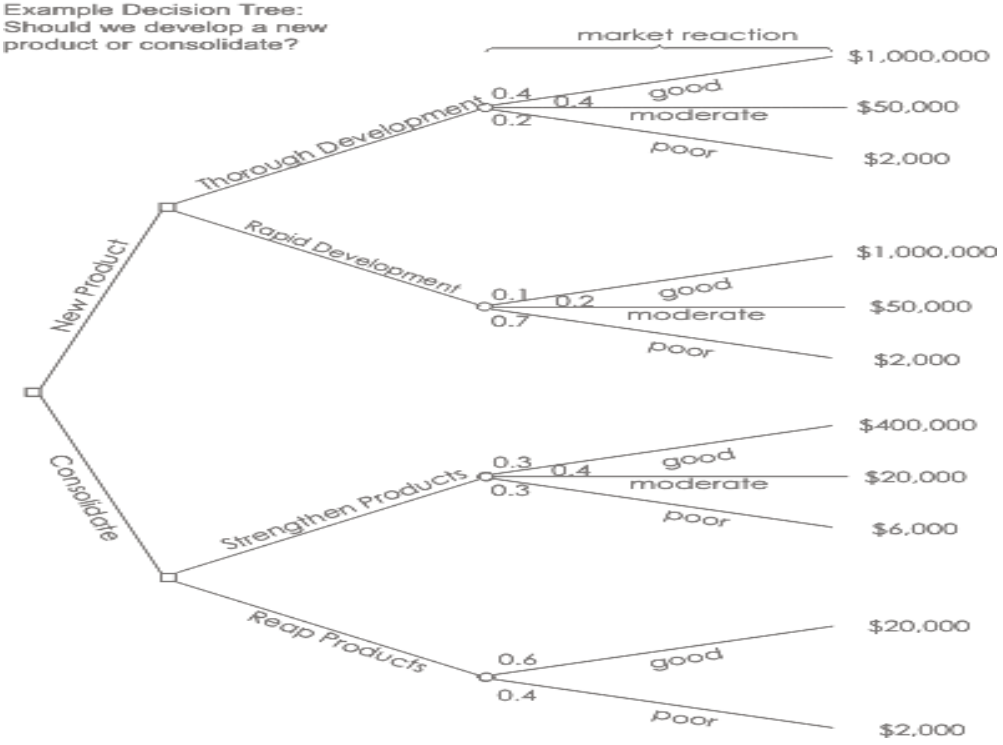
Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning.

Tree models where the target variable can take a finite 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.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making.

Decision tree learning is one of the most successful techniques for supervised classification learning.

Figure 2
Example Decision Tree:
Should we develop a new product or consolidate?



Calculating The Value of Uncertain Outcome Nodes

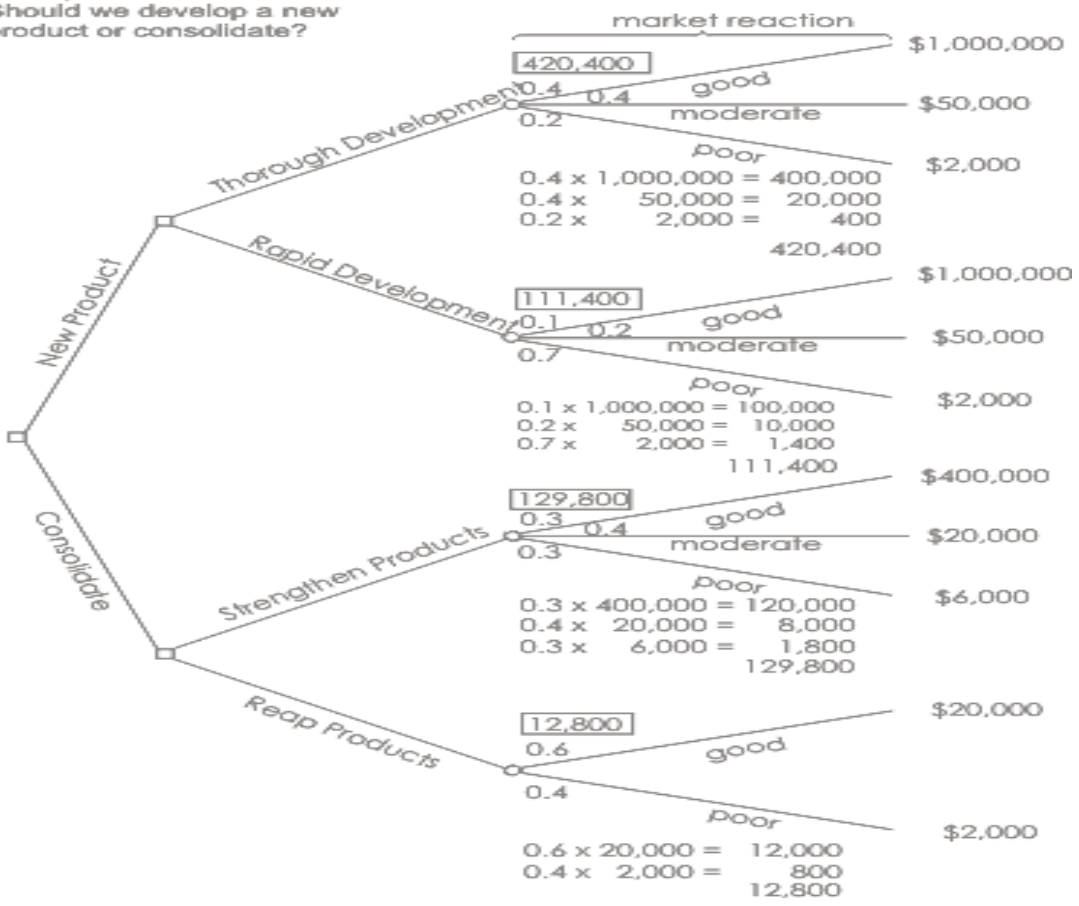
Where you are calculating the value of uncertain outcomes (circles on the diagram), do this by multiplying the value of the outcomes by their probability. The total for that node of the tree is the total of these values.

In the example in Figure 2, the value for "new product, thorough development" is:

0.4 (probability good outcome) x $1,000,000 (value) =	$400,000
0.4 (probability moderate outcome) x $50,000 (value) =	$20,000
0.2 (probability poor outcome) x $2,000 (value) =	$400
TOTAL	**$420,400**

Figure 3 shows the calculation of uncertain outcome nodes:

Figure 3
Example Decision Tree:
Should we develop a new product or consolidate?



Note that the values calculated for each node are shown in the boxes.

Calculating the Value of Decision Nodes

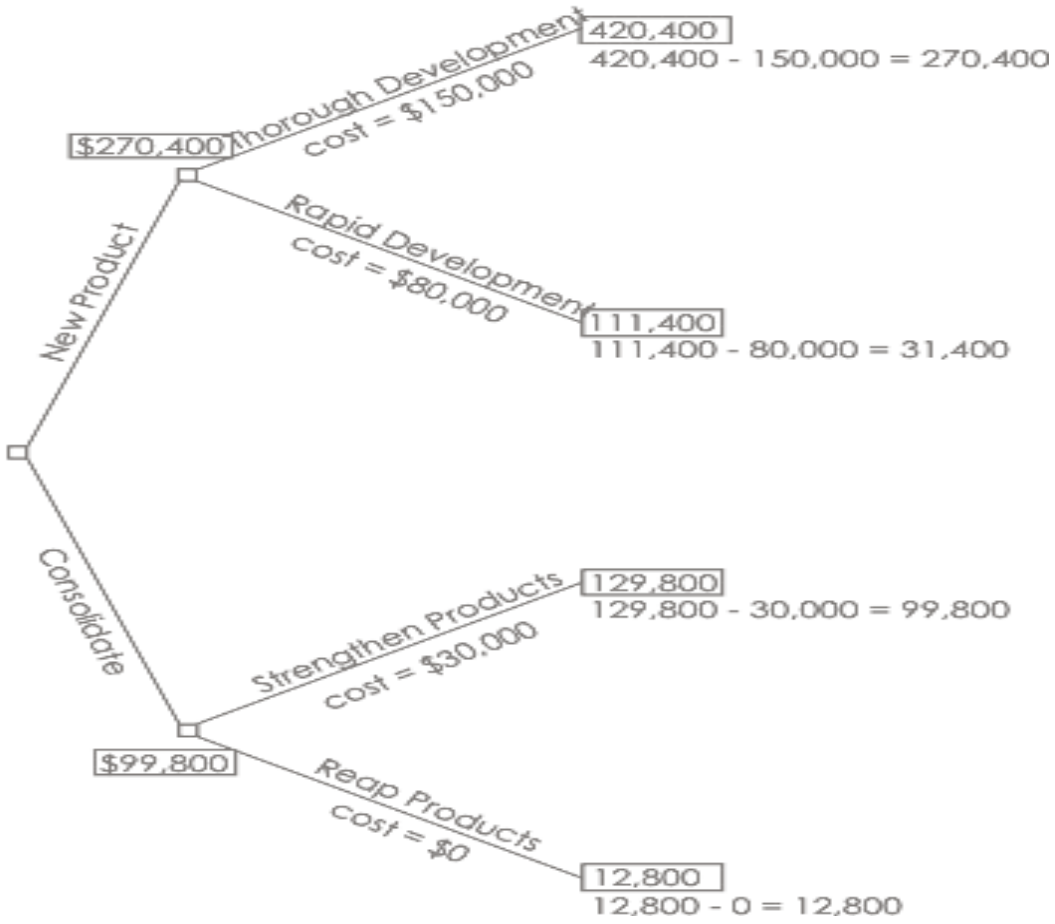
When you are evaluating a decision node, write down the cost of each option along each decision line. Then subtract the cost from the outcome value that you have already calculated. This will give you a value that represents the benefit of that decision.

Note that amounts already spent do not count for this analysis – these are 'sunk costs' and (despite emotional counter-arguments) should not be factored into the decision.

When you have calculated these decision benefits, choose the option that has the largest benefit, and take that as the decision made. This is the value of that decision node.

Figure 4 shows this calculation of decision nodes in our example:

Figure 4:
Example Decision Tree:
Should we develop a new product or consolidate?



In this example, the benefit we previously calculated for 'new product, thorough development' was $420,400. We estimate the future cost of this approach as $150,000. This gives a net benefit of $270,400.

The net benefit of 'new product, rapid development' was $31,400. On this branch we therefore choose the most valuable option, 'new product, thorough development', and allocate this value to the decision node.

Decision tree advantages

- **Simple to understand and interpret.** People are able to understand decision tree models after a brief explanation.
- **Requires little data preparation.** Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.

- **Able to handle both numerical and categorical data.** Other techniques are usually specialized in analyzing datasets that have only one type of variable. (For example, relation rules can be used only with nominal variables while neural networks can be used only with numerical variables.)
- **Uses a white box model.** If a given situation is observable in a model the explanation for the condition is easily explained by Boolean logic. (An example of a black box model is an artificial neural network since the explanation for the results is difficult to understand.)
- **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
- **Robust.** Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

- **Performs well with large datasets.** Large amounts of data can be analyzed using standard computing resources in reasonable time.

Limitations

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
- Decision-tree learners can create over-complex trees that do not generalize well from the training data. (This is known as over fitting.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- For data including categorical variables with different numbers of levels, information gain in decision trees is biased in favor of those attributes with more levels. However, the issue of biased predictor selection is avoided by the Conditional Inference approach.

4.5 Statistical learning Models

Statistical learning theory is a *framework for machine learning drawing from the fields of statistics and functional analysis.* Statistical learning theory *deals with the problem of finding a predictive function based on data.* Statistical learning theory has led to successful applications in fields such as *computer vision, speech recognition, bioinformatics* and baseball. It is the theoretical framework underlying support vector machines.

Let us consider a *very* simple example. Our favorite Surprise candy comes in two flavors: cherry (yum) and lime (ugh). The candy manufacturer has a peculiar sense of humor and wraps each piece of candy in the same opaque wrapper, regardless of flavor. The candy is sold in very large bags, of which there are known to be five kinds—again, indistinguishable from the outside:

h1: 100% cherry
h2: 75% cherry + 25% lime
h3: 50% cherry + 50% lime

h4: 25% cherry + 75% lime
h5: 100% lime

Given a new bag of candy, the random variable H (for *hypothesis*) denotes the type of the bag, with possible values h1 through h5. H is not directly observable, of course. As the pieces of candy are opened and inspected, data are revealed—D1, D2, : : :, DN, where each Di is a random variable with possible values cherry and lime.

Bayesian learning simply calculates the probability of each hypothesis, given the data, and makes predictions on that basis. That is, the predictions are made by using *all* the hypotheses, weighted by their probabilities, rather than by using just a single ‑best‖ hypothesis. Let **D** represent all the data, with observed value **d**; then the probability of each hypothesis is obtained by Bayes' rule:

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i)P(h_i) \ . \qquad (20.1)$$

Now, suppose we want to make a prediction about an unknown quantity X. Then we have

$$\mathbf{P}(X|\mathbf{d}) = \sum_i \mathbf{P}(X|\mathbf{d}, h_i)\mathbf{P}(h_i|\mathbf{d}) = \sum_i \mathbf{P}(X|h_i)P(h_i|\mathbf{d}) \ , \qquad (20.2)$$

The key quantities in the Bayesian approach are the **hypothesis prior**, $P(h_i)$, and the **likelihood** of the data under each hypothesis, $P(\mathbf{d}|h_i)$.

A very common approximation—one that is usually adopted in science—is to make predictions based on a single *most probable* hypothesis—that is, an hi that maximizes P(hi|**d**). This is often called a **maximum a posteriori** or MAP (pronounced ‑em-ay-pee‖) hypothesis. Predictions made according to an MAP hypothesis h_{MAP} are approximately Bayesian to the extent that $\mathbf{P}(X|\mathbf{d}) \approx \mathbf{P}(X|h_{MAP})$.

4.6 Learning with complete data- Naive Bayes models

A parameter learning task involves *finding the numerical parameters for a probability model whose structure is fixed.* For example, we might be interested in *learning the conditional probabilities in a Bayesian network with a given structure. Data are complete when each data point contains values for every variable in the probability model being learned*. Complete data greatly simplify the problem of learning the parameters of a complex model.

Maximum-likelihood parameter learning: Discrete models

Suppose we buy a bag of lime and cherry candy from a new manufacturer whose lime–cherry proportions are completely unknown—that is, the fraction could be anywhere between 0 and 1. In that case, the hypothesis is h_θ. Now suppose we unwrap N candies, of which c are cherries and l=N - c are limes. The likelihood of this particular data set is

The **parameter** in this case, which we call θ, is the proportion of cherry candies, and the

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell \ .$$

The maximum-likelihood hypothesis is given by the value of θ that maximizes this expression. The same value is obtained by maximizing the **log likelihood**,

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = c \log\theta + \ell \log(1-\theta) \ .$$

To find the maximum-likelihood value of , we differentiate L with respect to and set the resulting expression to zero:

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \qquad \Rightarrow \qquad \theta = \frac{c}{c+\ell} = \frac{c}{N} \ .$$

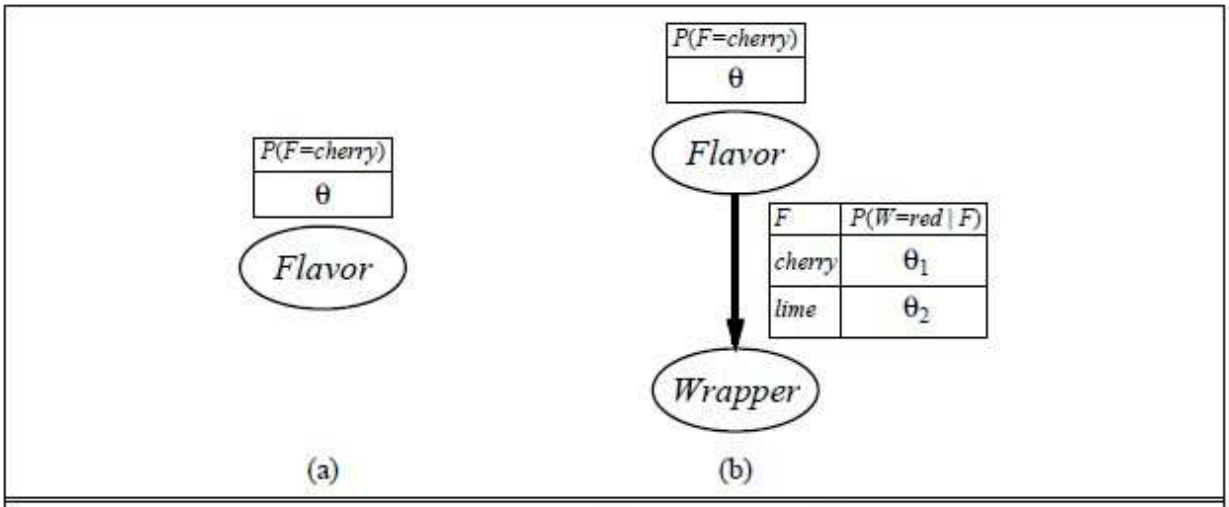


Figure 20.2 (a) Bayesian network model for the case of candies with an unknown proportion of cherries and limes. (b) Model for the case where the wrapper color depends (probabilistically) on the candy flavor.

- The example also illustrates a significant problem with maximum-likelihood learning in general: *when the data set is small enough that some events have not yet been observed—for instance, no cherry candies—the maximum likelihood hypothesis assigns zero probability to those events.*
- The most important point is that, *with complete data, the maximum-likelihood parameter learning problem for a Bayesian network decomposes into separate learning problems, one for each parameter*. The second point is that the parameter values for a variable, given its parents, are just the observed frequencies of the variable values for each setting of the parent values.

Naive Bayes models

In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" Variables X_i are the leaves. The model is "naïve" because it assumes that the attributes are

conditionally independent of each other, given the class. Assuming Boolean variables, the parameters are

$$\theta = P(C = true), \theta_{i1} = P(X_i = true|C = true), \theta_{i2} = P(X_i = true|C = false).$$

With observed attribute values $x_1; : : : ; x_n$, the probability of each class is given by

$$\mathbf{P}(C|x_1, \ldots, x_n) = \alpha\, \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C)\ .$$

Maximum-likelihood parameter learning: Continuous models

Let us begin with a very simple case: learning the parameters of a Gaussian density function on a single variable. That is, the data are generated as follows:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\ .$$

The parameters of this model are the mean μ and the standard deviation σ. (Notice that the normalizing "constant" depends on σ, so we cannot ignore it.) Let the observed values be $x_1, \ldots, x_N$. Then the log likelihood is

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}\ .$$

Setting the derivatives to zero as usual, we obtain

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2}\sum_{j=1}^{N}(x_j - \mu) = 0 \quad &\Rightarrow \quad \mu &= \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(x_j - \mu)^2 = 0 \quad &\Rightarrow \quad \sigma &= \sqrt{\frac{\sum_j (x_j-\mu)^2}{N}}\ . \end{aligned} \tag{20.4}$$

That is, the maximum-likelihood value of the mean is the sample average and the maximum-likelihood value of the standard deviation is the square root of the sample variance. Again, these are comforting results that confirm "commonsense" practice.

To learn the conditional distribution P(Y | X), we can maximize the conditional likelihood

$$P(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}\ .$$

Here, the parameters are θ_1, θ_2, and σ. The data are a collection of (x_j, y_j) pairs, as illustrated in Figure 20.4. Using the usual methods (Exercise 20.6), we can find the maximum-likelihood values of the parameters. Here, we want to make a different point. If we consider just the parameters θ_1 and θ_2 that define the linear relationship between x and y, it becomes clear that maximizing the log likelihood with respect to these parameters is the same as *minimizing* the numerator in the exponent of Equation (20.5):

$$E = \sum_{j=1}^{N} (y_j - (\theta_1 x_j + \theta_2))^2 .$$

The quantity $(y_j - (\theta_1 x_j + \theta_2))$ is the **error** for (x_j, y_j)—that is, the difference between the actual value y_j and the predicted value $(\theta_1 x_j + \theta_2)$—so E is the well-known **sum of squared errors**. This is the quantity that is minimized by the standard **linear regression** procedure.

IN Now we can understand why: minimizing the sum of squared errors gives the maximum-likelihood straight-line model, *provided that the data are generated with Gaussian noise of fixed variance.*

4.7 Learning with Hidden Variables: The EM Algorithm

Many real-world problems have **hidden variables** (sometimes called **latent variables**) which are not observable in the data that are available for learning. For example, medical records often include the observed symptoms, the treatment applied, and perhaps the outcome of the treatment, but they seldom contain a direct observation of the disease itself!6 One might ask, ‑If the disease is not observed, why not construct a model without it?‖

Assume that each variable has three possible values (e.g., none, moderate, and severe). Removing the hidden variable from the network in (a) yields the network in (b); the total number of parameters increases from 78 to 708. Thus, *latent variables can dramatically reduce the number of parameters required to specify a Bayesian network.* This, in turn, can dramatically reduce the amount of data needed to learn the parameters.

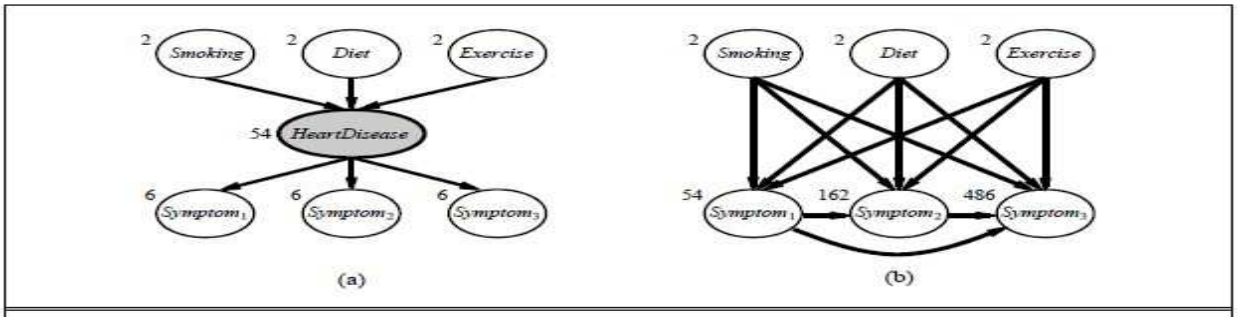


Figure 20.7 (a) A simple diagnostic network for heart disease, which is assumed to be a hidden variable. Each variable has three possible values and is labeled with the number of independent parameters in its conditional distribution; the total number is 78. (b) The equivalent network with *HeartDisease* removed. Note that the symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters.

The EM Algorithm

The basic idea of EM in this context is to *pretend that we know the parameters of the model and then to infer the probability that each data point belongs to each component. After that, we refit the components to the data, where each component is fitted to the entire data set with each point weighted by the probability that it belongs to that component. The process iterates until convergence.*

Essentially, we are ―completing‖ the data by inferring probability distributions over the hidden variables—which component each data point belongs to—based on the current model. For the mixture of Gaussians, we initialize the mixture model parameters arbitrarily and then iterate the following two steps:

1. E-step: Compute the probabilities $p_{ij} = P(C = i|\mathbf{x}_j)$, the probability that datum $\mathbf{x}_j$ was generated by component i. By Bayes' rule, we have $p_{ij} = \alpha P(\mathbf{x}_j|C = i)P(C = i)$. The term $P(\mathbf{x}_j|C = i)$ is just the probability at $\mathbf{x}_j$ of the ith Gaussian, and the term $P(C = i)$ is just the weight parameter for the ith Gaussian. Define $p_i = \sum_j p_{ij}$.
2. M-step: Compute the new mean, covariance, and component weights as follows:

$$\boldsymbol{\mu}_i \leftarrow \sum_j p_{ij}\mathbf{x}_j / p_i$$

$$\boldsymbol{\Sigma}_i \leftarrow \sum_j p_{ij}\mathbf{x}_j\mathbf{x}_j^\top / p_i$$

$$w_i \leftarrow p_i \,.$$

The E-step, or *expectation* step, can be viewed as computing the expected values p_{ij} of the hidden **indicator variables** Z_{ij}, where Z_{ij} is 1 if datum $\mathbf{x}_j$ was generated by the ith component and 0 otherwise. The M-step, or *maximization* step, finds the new values of the parameters that maximize the log likelihood of the data, given the expected values of the hidden indicator variables.

The general form of the EM algorithm

We have seen several instances of the EM algorithm. Each involves computing expected values of hidden variables for each example and then recomputing the parameters, using the expected values as if they were observed values. Let $\mathbf{x}$ be all the observed values in all the examples, let $\mathbf{Z}$ denote all the hidden variables for all the examples, and let θ be all the parameters for the probability model. Then the EM algorithm is

$$\theta^{(i+1)} = \operatorname*{argmax}_{\theta} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z}|\mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z}|\theta) \,.$$

The E-step is the computation of the summation, which is the expectation of the log likelihood of the ―completed‖ data with respect to the distribution P(**Z**=**z**|**x**; $\theta^{(i)}$), which is the posterior over the hidden variables, given the data. The M-step is the maximization of this expected log likelihood with respect to the parameters. For mixtures of Gaussians, the hidden variables are the Z_{ij}'s, where Z_{ij} is 1 if example j was generated by component i.

4.8 Reinforcement learning

Machine learning in general can be classified into three categories:

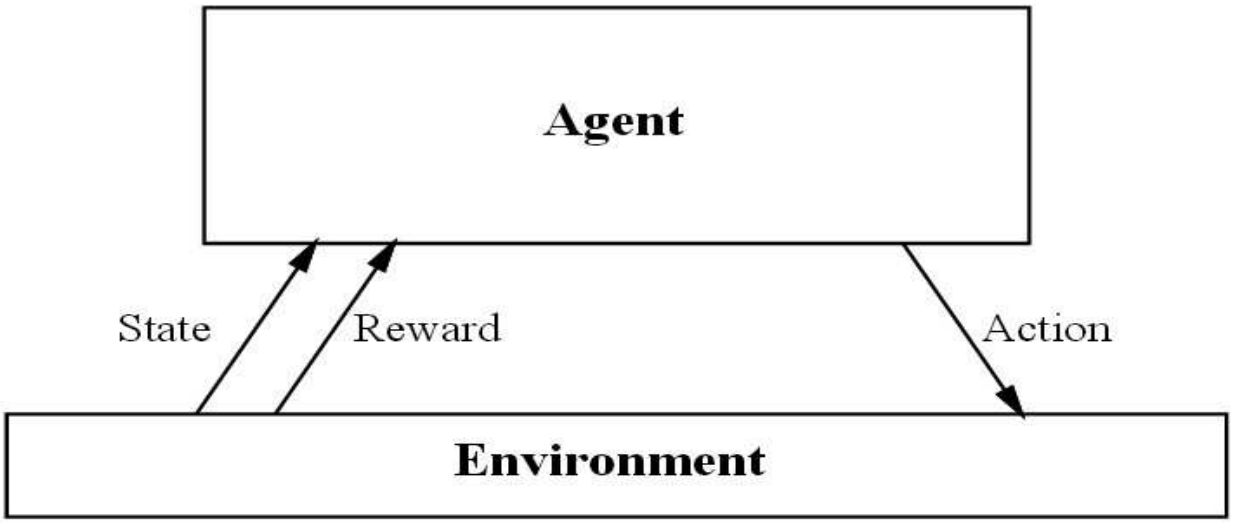
1) Supervised learning (SL). These are learning in which you know the input as well as the output.

2) Unsupervised learning (USL). These are learning in which you know the input but not the output.

3) Reinforcement learning (RL). This kind of learning falls between the first two categories, in which you have the input, but not output, instead you have "critic" -- whether the classifier's output is correct or wrong. RL is often the issue of an agent acting in an environment that tries to achieve the best performance over time by trial and error.

Reinforcement Learning is a type of *Machine Learning that* allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

Reinforcement Learning allows the machine or software agent to learn its behavior based on feedback from the environment. This behavior can be learnt once and for all, or keep on adapting as time goes by. If the problem is modeled with care, some Reinforcement Learning algorithms can converge to the global optimum; this is the ideal behavior that maximizes the reward.



Limitations...

1. It is often too memory expensive to store values of each state, since the problems can be pretty complex. Solving this involves looking into value approximation techniques, such

as *Decision Trees* or *Neural Networks*. There are many consequence of introducing these imperfect value estimations, and research tries to minimize their impact on the quality of the solution.

2. **M**oreover, problems are also generally very modular; similar behaviors reappear often, and modularity can be introduced to avoid learning everything all over again. Hierarchical approaches are common-place for this, but doing this automatically is proving a challenge.
3. Due to limited perception, it is often impossible to fully determine the current state. This also affects the performance of the algorithm, and much work has been done to compensate this *Perceptual Aliasing*.

Applications...

- **I**n practice, this ranges from controlling robotic arms to find the most efficient motor combination, to robot navigation where collision avoidance behavior can be learnt by negative feedback from bumping into obstacles.
- Logic games are also well-suited to Reinforcement Learning, as they are traditionally defined as a sequence of decisions: games such as poker, back-gammon, othello, chess have been tackled more or less succesfully.