

Vyasapuri, Bandlaguda, Post:Keshavgi
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



Department of Computer Science and Engineering (AIML)

(R18)
Machine Learning
Lecture Notes

B. Tech III YEAR – I SEM

Prepared by

Mrs.Swapna
(Professor&HOD-CSM)
Dept. CSE(AIML)

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



Syllabus

MACHINE LEARNING

B.Tech. III Year I Sem.

L T P C
3 0 0 3

Prerequisites:

1. Data Structures
2. Knowledge on statistical methods

Course Objectives:

- This course explains machine learning techniques such as decision tree learning, Bayesian learning etc.
- To understand computational learning theory.
- To study the pattern comparison techniques.

Course Outcomes:

- Understand the concepts of computational intelligence like machine learning
- Ability to get the skill to apply machine learning techniques to address the real time problems in different areas
- Understand the Neural Networks and its usage in machine learning application.

UNIT - I

Introduction - Well-posed learning problems, designing a learning system, Perspectives and issues in machine learning

Concept learning and the general to specific ordering – introduction, a concept learning task, concept learning as search, find-S: finding a maximally specific hypothesis, version spaces and the candidate elimination algorithm, remarks on version spaces and candidate elimination, inductive bias.

Decision Tree Learning – Introduction, decision tree representation, appropriate problems for decision tree learning, the basic decision tree learning algorithm, hypothesis space search in decision tree learning, inductive bias in decision tree learning, issues in decision tree learning.

UNIT - II

Artificial Neural Networks-1– Introduction, neural network representation, appropriate problems for neural network learning, perceptions, multilayer networks and the back-propagation algorithm.

Artificial Neural Networks-2- Remarks on the Back-Propagation algorithm, An illustrative example: face recognition, advanced topics in artificial neural networks.

Evaluation Hypotheses – Motivation, estimation hypothesis accuracy, basics of sampling theory, a general approach for deriving confidence intervals, difference in error of two hypotheses, comparing learning algorithms.

UNIT - III

Bayesian learning – Introduction, Bayes theorem, Bayes theorem and concept learning, Maximum Likelihood and least squared error hypotheses, maximum likelihood hypotheses for predicting probabilities, minimum description length principle, Bayes optimal classifier, Gibbs algorithm, Naïve Bayes classifier, an example: learning to classify text, Bayesian belief networks, the EM algorithm.

Computational learning theory – Introduction, probably learning an approximately correct hypothesis, sample complexity for finite hypothesis space, sample complexity for infinite hypothesis spaces, the

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



mistake bound model of learning.

Instance-Based Learning- Introduction, k -nearest neighbour algorithm, locally weighted regression, radial basis functions, case-based reasoning, remarks on lazy and eager learning.

UNIT- IV

Genetic Algorithms – Motivation, Genetic algorithms, an illustrative example, hypothesis space search, genetic programming, models of evolution and learning, parallelizing genetic algorithms.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



Learning Sets of Rules – Introduction, sequential covering algorithms, learning rule sets: summary, learning First-Order rules, learning sets of First-Order rules: FOIL, Induction as inverted deduction, inverting resolution.

Reinforcement Learning – Introduction, the learning task, Q-learning, non-deterministic, rewards and actions, temporal difference learning, generalizing from examples, relationship to dynamic programming.

UNIT - V

Analytical Learning-1- Introduction, learning with perfect domain theories: PROLOG-EBG, remarks on explanation-based learning, explanation-based learning of search control knowledge.

Analytical Learning-2-Using prior knowledge to alter the search objective, using prior knowledge to augment search operators.

Combining Inductive and Analytical Learning – Motivation, inductive-analytical approaches to learning, using prior knowledge to initialize the hypothesis.

TEXT BOOK:

1. Machine Learning – Tom M. Mitchell, - MGH

REFERENCE BOOK:

2. Machine Learning: An Algorithmic Perspective, Stephen Marshland, Taylor & Francis.

INTRODUCTION

Computer machinery and intelligence:

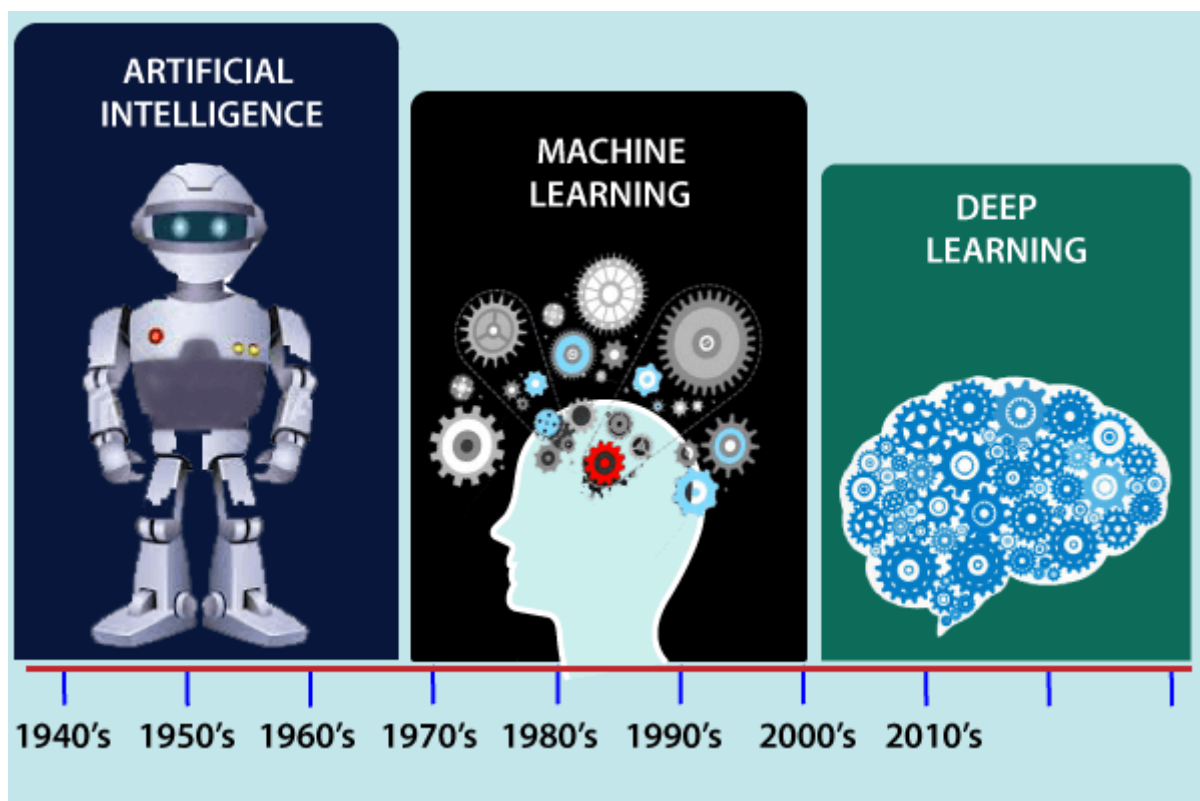
- **1950:** In 1950, Alan Turing published a seminal paper, "Computer Machinery and Intelligence," on the topic of artificial intelligence. In his paper, he asked, "Can machines think?"

Machine intelligence in Games:

- **1952:** Arthur Samuel, who was the pioneer of machine learning, created a program that helped an IBM computer to play a checkers game. It performed better more it played.

1959: In 1959, the term "Machine Learning" was first coined by **Arthur Samuel**.

The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence, and stated that it gives computers the ability to learn without being explicitly programmed



Ever since computers were invented, we have wondered whether they might be made to learn.

If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization

Some successful applications of machine learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

Classification of Machine Learning

machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**

1. Supervised Learning

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.

The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.

The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning

Supervised learning, as the name indicates, has the presence of a supervisor as a teacher. Basically supervised learning is when we teach or train the machine using data that is well labeled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

- If the shape of the object is rounded and has a depression at the top, is red in color, then it will be labeled as **–Apple**.
- If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as **–Banana**.

Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.

Since the machine has already learned the things from previous data and this time has to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category. Thus the machine learns the things from training data(basket containing fruits) and then applies the knowledge to test data(new fruit).

Supervised learning is classified into two categories of algorithms:

- **Classification:** A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”.
- **Regression:** A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Supervised learning deals with or learns with “labeled” data. This implies that some data is already tagged with the correct answer.

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. In unsupervised learning, we don't have a predetermined result.

when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values.

It can be further classified into two categories of algorithms

- **Clustering**
- **Association**

Structure the input data into new features or a group of objects with similar patterns

Reinforcement learning: When you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can accompany an example with positive or negative feedback according to the solution the algorithm proposes comes under the category of Reinforcement learning, which is connected to applications for which the algorithm must make decisions (so the product is prescriptive, not just descriptive, as in unsupervised learning), and the decisions bear consequences. In the human world, it is just like learning by trial and error.

Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves.

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance

Terminologies of Machine Learning

- **Model**

A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called **hypothesis**.

- **Feature**

A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc.**

Note: Choosing informative, discriminating and independent features is a crucial step for effective algorithms. We generally employ a **feature extractor** to extract the relevant features from the raw data.

- **Target (Label)**

A target variable or label is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.

- **Training**

The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.

- **Prediction**

Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).



Basic Difference in ML and Traditional Programming?

- **Traditional Programming :** We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning :** We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.

In 1997, Tom Mitchell gave a “well-posed” mathematical and relational definition that “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

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.

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

1. **Checkers game:** A computer program that learns to play *checkers* might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself.

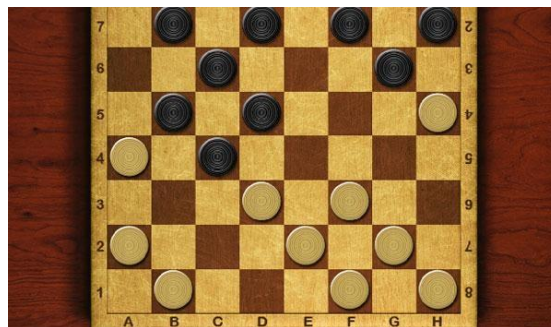


Fig: Checker game board

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

2. 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

3. A robot driving learning problem:

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error.
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

DESIGNING A LEARNING SYSTEM

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
 1. Estimating training values
 2. Adjusting the weights
5. The Final Design

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides *direct or indirect feedback* regarding the choices made by the performance system.

For example, in checkers game:

In learning to play checkers, the system might learn from *direct training examples* consisting of *individual checkers board states* and *the correct move for each*.

Indirect training examples consisting of the *move sequences* and *final outcomes* of various games played. The 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.

Here the learner faces an additional problem of *credit assignment*, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.

Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. The degree to which the *learner controls the sequence of training examples*

For example, in checkers game:

The learner might depend on the *teacher* to select informative board states and to provide the correct move for each.

Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with *no teacher present*.

3. How well it represents the *distribution of examples* over which the final system performance P must be measured

For example, in checkers game:

In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.

It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. *Choosing the Target Function*

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let's consider a checkers-playing program that can generate the legal moves from any board state.

The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let *ChooseMove* be the target function and the notation is

***ChooseMove* : $B \rightarrow M$**

which indicate that this function 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.

ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an *evaluation function* that assigns a *numerical score* to any given board state

Let the target function V and the notation

$$V : B \rightarrow R$$

which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V , then it can easily use it to select the best move from any current board position.

Let us define the target value $V(b)$ for an arbitrary board state b in B , as follows:

- If b is a final board state that is won, then $V(b) = 100$
- If b is a final board state that is lost, then $V(b) = -100$
- If b is a final board state that is drawn, then $V(b) = 0$

3. Choosing a Representation for the Target Function

Let's choose a simple representation - for any given board state, the function c will be calculated as a linear combination of the following board features:

- x_1 : the number of black pieces on the board
- x_2 : the number of red pieces on the board
- x_3 : the number of black kings on the board
- x_4 : the number of red kings on the board
- x_5 : the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x_6 : the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

4. Choosing a Function Approximation Algorithm

In order to learn the target function f we require a set of training examples, each describing a specific board state b and the training value $V_{\text{train}}(b)$ for b .

Each training example is an ordered pair of the form $(b, V_{\text{train}}(b))$.

For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{\text{train}}(b)$ is therefore $+100$.

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{\text{train}}(b)$ for any intermediate board state b to be $\hat{V}(\text{Successor}(b))$

Where ,

- \hat{V} is the learner's current approximation to V
- $\text{Successor}(b)$ denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

$$V_{\text{train}}(b) \leftarrow \hat{V}(\text{Successor}(b))$$

2. Adjusting the weights

Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{\text{train}}(b))\}$

A first step is to define what we mean by the bestfit to the training data.

One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

$$E \equiv \sum_{\langle b, V_{\text{train}}(b) \rangle \in \text{training examples}} (V_{\text{train}}(b) - \hat{V}(b))^2$$

Several algorithms are known for finding weights of a linear function that minimize E . One such algorithm is called the *least mean squares, or LMS training rule*. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS weight update rule :- For each training example $(b, V_{\text{train}}(b))$
Use the current weights to calculate $\hat{V}(b)$
For each weight w_i , update it as

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

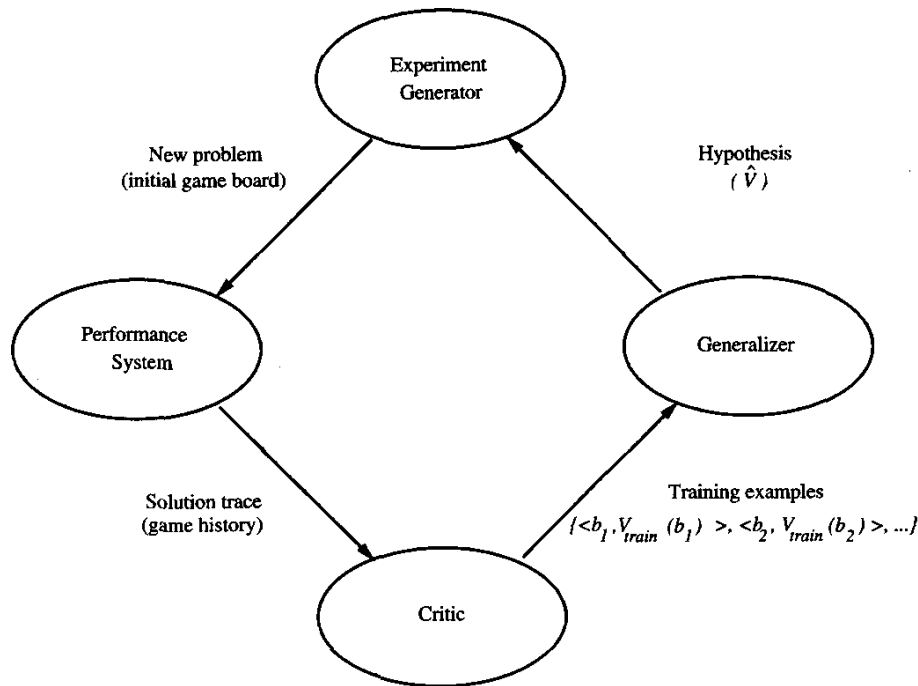
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error $(V_{\text{train}}(b) - \hat{V}(b))$ is zero, no weights are changed.
- When $(V_{\text{train}}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each w_i is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. The Final Design

The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems



1. **The Performance System** is the module that must 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 produces a trace of its solution (game history) as output.
2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function
3. **The Generalizer** takes as input the training examples and produces 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.
4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Issues in Machine Learning

The field of machine learning, and much of this book, is concerned with answering questions such as the following

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- 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? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Issues in Machine Learning

Although machine learning is being used in every industry and helps organizations make more informed and data-driven choices that are more effective than classical methodologies, it still has so many problems that cannot be ignored. Here are some common issues in Machine Learning that professionals face to inculcate ML skills and create an application from scratch.

1. Inadequate Training Data

The major issue that comes while using machine learning algorithms is the lack of quality as well as quantity of data. Although data plays a vital role in the processing of machine learning algorithms, many data scientists claim that inadequate data, noisy data, and unclean data are extremely exhausting the machine learning algorithms. For example, a simple task requires thousands of sample data, and an advanced task such as speech or image recognition needs millions of sample data examples. Further,

data quality is also important for the algorithms to work ideally, but the absence of data quality is also found in Machine Learning applications. Data quality can be affected by some factors as follows:

- **Noisy Data-** It is responsible for an inaccurate prediction that affects the decision as well as accuracy in classification tasks.
- **Incorrect data-** It is also responsible for faulty programming and results obtained in machine learning models. Hence, incorrect data may affect the accuracy of the results also.
- **Generalizing of output data-** Sometimes, it is also found that generalizing output data becomes complex, which results in comparatively poor future actions.

2. Poor quality of data

As we have discussed above, data plays a significant role in machine learning, and it must be of good quality as well. Noisy data, incomplete data, inaccurate data, and unclean data lead to less accuracy in classification and low-quality results. Hence, data quality can also be considered as a major common problem while processing machine learning algorithms.

3. Non-representative training data

To make sure our training model is generalized well or not, we have to ensure that sample training data must be representative of new cases that we need to generalize. The training data must cover all cases that are already occurred as well as occurring.

Further, if we are using non-representative training data in the model, it results in less accurate predictions. A machine learning model is said to be ideal if it predicts well for generalized cases and provides accurate decisions. If there is less training data, then there will be a sampling noise in the model, called the non-representative training set. It won't be accurate in predictions. To overcome this, it will be biased against one class or a group.

Hence, we should use representative data in training to protect against being biased and make accurate predictions without any drift.

4. Overfitting and Underfitting

Overfitting is one of the most common issues faced by Machine Learning engineers and data scientists. Whenever a machine learning model is trained with a huge amount of data, it starts capturing noise and inaccurate data into the training data set. It negatively affects the performance of the model. Let's

understand with a simple example where we have a few training data sets such as 1000 mangoes, 1000 apples, 1000 bananas, and 5000 papayas. Then there is a considerable probability of identification of an apple as papaya because we have a massive amount of biased data in the training data set; hence prediction got negatively affected.

Underfitting:

Under fitting is just the opposite of over fitting. Whenever a machine learning model is trained with fewer amounts of data, and as a result, it provides incomplete and inaccurate data and destroys the accuracy of the machine learning model. Under fitting occurs when our model is too simple to understand the base structure of the data, is less in quantity.

5. Monitoring and maintenance

As we know that generalized output data is mandatory for any machine learning model; hence, regular monitoring and maintenance become compulsory for the same. Different results for different actions require data change; hence editing of codes as well as resources for monitoring them also become necessary.

6. Getting bad recommendations

A machine learning model operates under a specific context which results in bad recommendations and concept drift in the model. Let's understand with an example where at a specific time customer is looking for some gadgets, but now customer requirement changed over time but still machine learning model showing same recommendations to the customer while customer expectation has been changed. This incident is called a Data Drift. It generally occurs when new data is introduced or interpretation of data changes. However, we can overcome this by regularly updating and monitoring data according to the expectations.

7. Lack of skilled resources

Although Machine Learning and Artificial Intelligence are continuously growing in the market, still these industries are fresher in comparison to others. The absence of skilled resources in the form of manpower is also an issue. Hence, we need manpower having in-depth knowledge of mathematics, science, and technologies for developing and managing scientific substances for machine learning.

8. Customer Segmentation

Customer segmentation is also an important issue while developing a machine learning algorithm. To identify the customers who paid for the recommendations shown by the model and who don't even check them. Hence, an algorithm is necessary to recognize the customer behavior and trigger a relevant recommendation for the user based on past experience.

9. Process Complexity of Machine Learning

The machine learning process is very complex, which is also another major issue faced by machine learning engineers and data scientists. However, Machine Learning and Artificial Intelligence are very new technologies but are still in an experimental phase and continuously being changing over time. There is the majority of hits and trial experiments; hence the probability of error is higher than expected. Further, it also includes analyzing the data, removing data bias, training data, applying complex mathematical calculations, etc., making the procedure more complicated and quite tedious.

10. Data Bias

Data Biasing is also found a big challenge in Machine Learning. These errors exist when certain elements of the dataset are heavily weighted or need more importance than others. Biased data leads to inaccurate results, skewed outcomes, and other analytical errors. However, we can resolve this error by determining where data is actually biased in the dataset. Further, take necessary steps to reduce it.

11. Lack of Explainability

This basically means the outputs cannot be easily comprehended as it is programmed in specific ways to deliver for certain conditions. Hence, a lack of explainability is also found in machine learning algorithms which reduce the credibility of the algorithms.

12. Slow implementations and results

This issue is also very commonly seen in machine learning models. However, machine learning models are highly efficient in producing accurate results but are time-consuming. Slow programming, excessive requirements' and overloaded data take more time to provide accurate results than expected. This needs continuous maintenance and monitoring of the model for delivering accurate results.

13. Irrelevant features

Although machine learning models are intended to give the best possible outcome, if we feed garbage data as input, then the result will also be garbage. Hence, we should use relevant features in our training

sample. A machine learning model is said to be good if training data has a good set of features or less to no irrelevant features.

CONCEPT LEARNING

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

Definition: Concept learning - Inferring a Boolean-valued function from training examples of its input and output

A CONCEPT LEARNING TASK

Consider the example task of learning the target concept "Days on which *Aldo* enjoys his favorite water sport"

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Table: Positive and negative training examples for the target concept *EnjoySport*.

The task is to learn to predict the value of *EnjoySport* for an arbitrary day, based on the values of its other attributes?

What hypothesis representation is provided to the learner?

- Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- Let each hypothesis be a vector of six constraints, specifying the values of the six

Vyasapuri, Bandlaguda, Post:Keshavgi
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



attributes *Sky, AirTemp, Humidity, Wind, Water, and Forecast.*

For each attribute, the hypothesis will either

- Indicate by a "?" that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a "Φ" that no value is acceptable

If some instance x satisfies all the constraints of hypothesis h , then h classifies x as a positive example ($h(x) = 1$).

The hypothesis that *PERSON* enjoys his favorite sport only on cold days with high humidity is represented by the expression

(?, Cold, High, ?, ?, ?)

The most general hypothesis-that every day is a positive example-is represented by

(?, ?, ?, ?, ?, ?)

The most specific possible hypothesis-that no day is a positive example-is represented by

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

- The set of items over which the concept is defined is called the *set of instances*, which is denoted by X .

Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

- The concept or function to be learned is called the *target concept*, which is denoted by c . c can be any Boolean valued function defined over the instances X

$c: X \rightarrow \{0, 1\}$

Example: The target concept corresponds to the value of the attribute *EnjoySport* (i.e., $c(x) = 1$ if *EnjoySport* = Yes, and $c(x) = 0$ if *EnjoySport* = No).

- Instances for which $c(x) = 1$ are called *positive examples*, or members of the target concept.
- Instances for which $c(x) = 0$ are called *negative examples*, or non-members of the target concept.
- The ordered pair $(x, c(x))$ to describe the training example consisting of the instance x and its target *concept value* $c(x)$.

Vyasapuri, Bandlaguda, Post:Keshavgi
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



- D to denote the set of available training examples

- The symbol H to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis h in H represents a Boolean-valued function defined over X

$$h: X \rightarrow \{0, 1\}$$

The goal of the learner is to find a hypothesis h such that $h(x) = c(x)$ for all x in X .

-
- Given:
 - Instances X : Possible days, each described by the attributes
 - *Sky* (with possible values Sunny, Cloudy, and Rainy),
 - *AirTemp* (with values Warm and Cold),
 - *Humidity* (with values Normal and High),
 - *Wind* (with values Strong and Weak),
 - *Water* (with values Warm and Cool),
 - *Forecast* (with values Same and Change).
 - Hypotheses H : Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), " Φ " (no value is acceptable), or a specific value.
 - Target concept c : *EnjoySport* : $X \rightarrow \{0, 1\}$
 - Training examples D : Positive and negative examples of the target function
 - Determine:
 - A hypothesis h in H such that $h(x) = c(x)$ for all x in X .

Table: The *EnjoySport* concept learning task.

The inductive learning hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.

Example:

Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp*, *Humidity*, *Wind*, *Water*, *Forecast* each have two possible values, the instance space X contains exactly

$$3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96 \text{ distinct instances } v$$

$$5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120 \text{ distinct hypotheses within } H. (\text{specific and general})$$

$$1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973. \text{ Semantically distinct hypotheses (only general +1 specific)}$$

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
 - For each attribute constraint a_i in h
 - If the constraint a_i is satisfied by x
 - Then do nothing
 - Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

To illustrate this algorithm, assume the learner is given the sequence of training examples from the *EnjoySport* task

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- The first step of FIND-S is to initialize h to the most specific hypothesis in H

$h - (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

- Consider the first training example

$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$

Observing the first training example, it is clear that hypothesis h is too specific. None of the " \emptyset " constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

$h_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle$

- Consider the second training example

$x_2 = \langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle, +$

The second training example forces the algorithm to further generalize h , this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

$h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

- Consider the third training example

$x_3 = \langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle, -$

Upon encountering the third training the algorithm makes no change to h . The FIND-S algorithm simply ignores every negative example.

$h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$

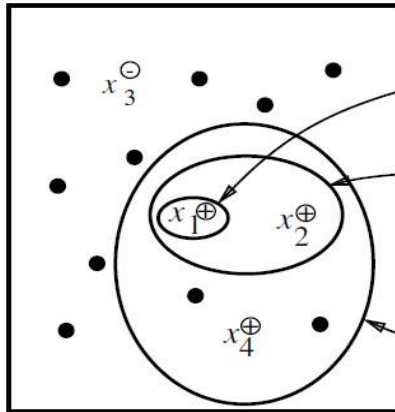
- Consider the fourth training example

$x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

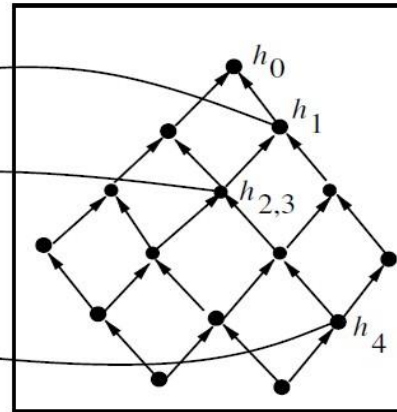
The fourth example leads to a further generalization of h

$h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

Instances X



Hypotheses H



Specific

General

$x_1 = \langle \text{Sunny Warm Normal Strong Warm Same} \rangle, +$
 $x_2 = \langle \text{Sunny Warm High Strong Warm Same} \rangle, +$
 $x_3 = \langle \text{Rainy Cold High Strong Warm Change} \rangle, -$
 $x_4 = \langle \text{Sunny Warm High Strong Cool Change} \rangle, +$

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
 $h_1 = \langle \text{Sunny Warm Normal Strong Warm Sam} \rangle$
 $h_2 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$
 $h_3 = \langle \text{Sunny Warm ? Strong Warm Same} \rangle$
 $h_4 = \langle \text{Sunny Warm ? Strong ? ?} \rangle$

The key property of the FIND-S algorithm

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H , and provided the training examples are correct.

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

Limitations of Find-S Algorithm

There are a few limitations of the Find-S algorithm listed down below:

1. There is no way to determine if the hypothesis is consistent throughout the data.
2. Inconsistent training sets can actually mislead the Find-S algorithm, since it ignores the negative examples.

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all *hypotheses consistent with the training examples*

Representation

Definition: consistent- A hypothesis h is **consistent** with a set of training examples D if and only if $h(x) = c(x)$ for each example $(x, c(x))$ in D .

$$\text{Consistent}(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) h(x) = c(x)$$

Note difference between definitions of *consistent* and *satisfies*

- An example x is said to *satisfy* hypothesis h when $h(x) = 1$, regardless of whether x is a positive or negative example of the target concept.
- An example x is said to *consistent* with hypothesis h iff $h(x) = c(x)$

Definition: version space- The **version space**, denoted $VS_{H, D}$ with respect to hypothesis space H and training examples D , is the subset of hypotheses from H consistent with the training examples in D

$$VS_{H, D} \equiv \{h \in H \mid \text{Consistent}(h, D)\}$$

A More Compact Representation for Version Spaces

The version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

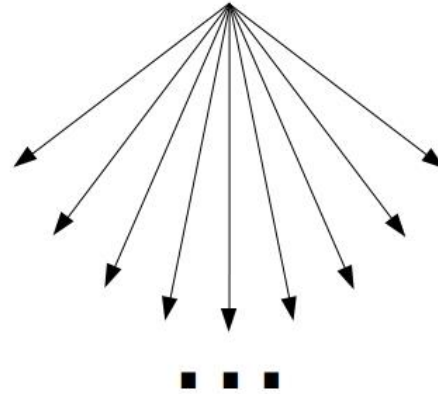
Definition: The **general boundary** G , with respect to hypothesis space H and training data D , is the set of maximally general members of H consistent with D

$$G \equiv \{g \in H \mid \text{Consistent}(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge \text{Consistent}(g', D)]\}$$

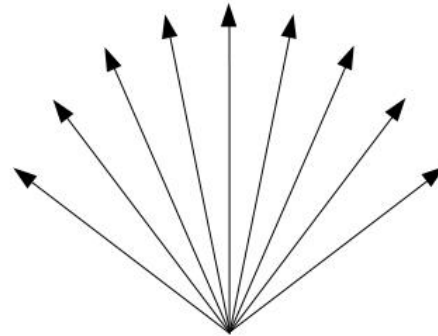
Definition: The **specific boundary** S , with respect to hypothesis space H and training data D , is the set of minimally general (i.e., maximally specific) members of H consistent with D .

$$S \equiv \{s \in H \mid \text{Consistent}(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge \text{Consistent}(s', D)]\}$$

The specific boundary $S_0 < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$



The whole space H is between the boundaries



The general boundary $G_0 < ?, ?, ?, ?, ?, ? >$

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINATION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d , do

- If d is a positive example
 - Moves from Specific to general
 - Remove from G any hypothesis inconsistent with d
 - For each hypothesis s in S that is not consistent with d
 - Remove s from S
 - Add to S all minimal generalizations h of s such that
 - h is consistent with d , and some member of G is more general than h
 - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
 - Moves from general to specific
 - Remove from S any hypothesis inconsistent with d
 - For each hypothesis g in G that is not consistent with d
 - Remove g from G
 - Add to G all minimal specializations h of g such that
 - h is consistent with d , and some member of S is more specific than h
 - Remove from G any hypothesis that is less general than another hypothesis in G

CANDIDATE- ELIMINATION algorithm using version spaces

An Illustrative Example

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

CANDIDATE-ELIMINATION algorithm begins by initializing the version space to the set of all hypotheses in H ;

Initializing the G boundary set to contain the most general hypothesis in H

$$G_0 \langle ?, ?, ?, ?, ?, ? \rangle$$

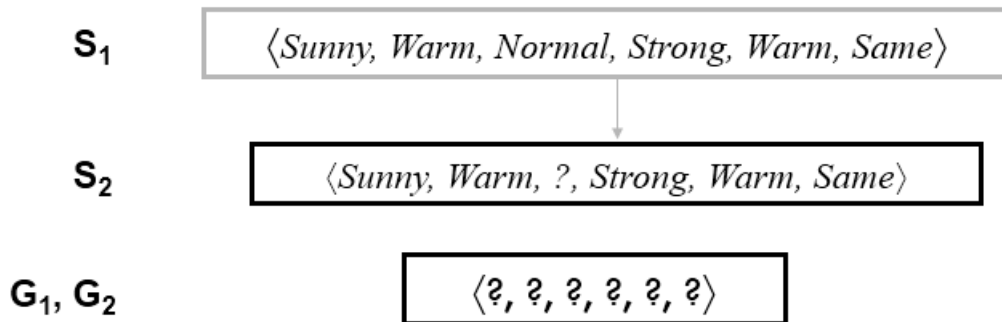
Initializing the S boundary set to contain the most specific (least general) hypothesis

$$S_0 \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

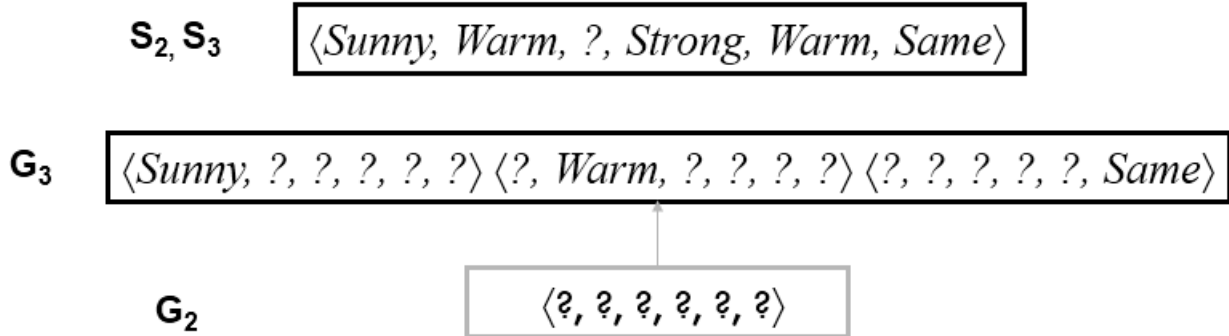
- When the second training example is observed, it has a similar effect of generalizing S

For training example d ,

$$\langle \text{Sunny, Warm, High, Strong, Warm, Same} \rangle +$$



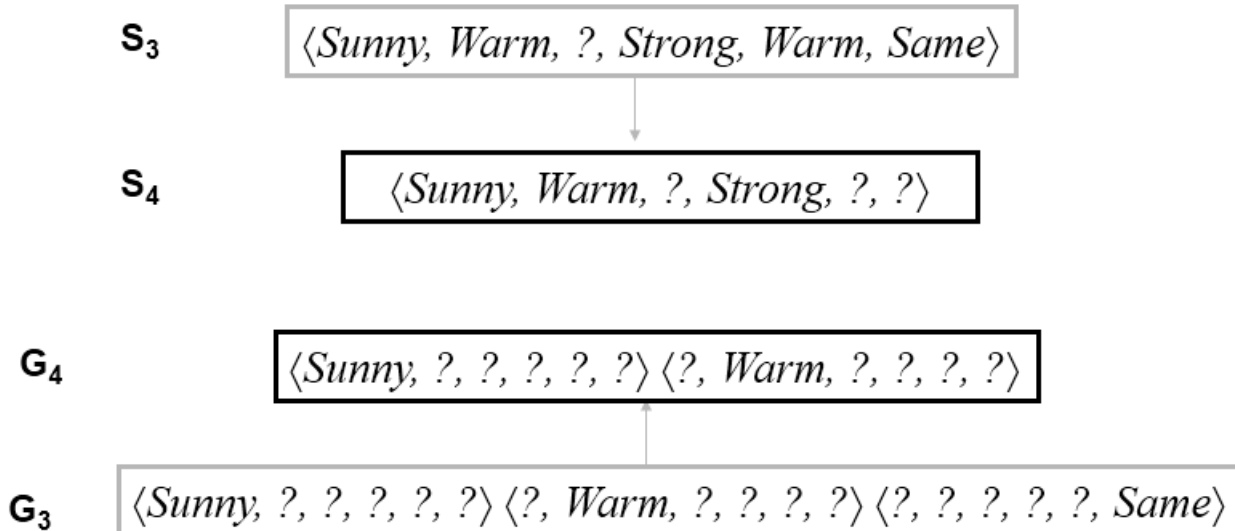
For training example d, $\langle \text{Rainy, Cold, High, Strong, Warm, Change} \rangle -$



Given that there are six attributes that could be specified to specialize G_2 , why are there only three new hypotheses in G_3 ?

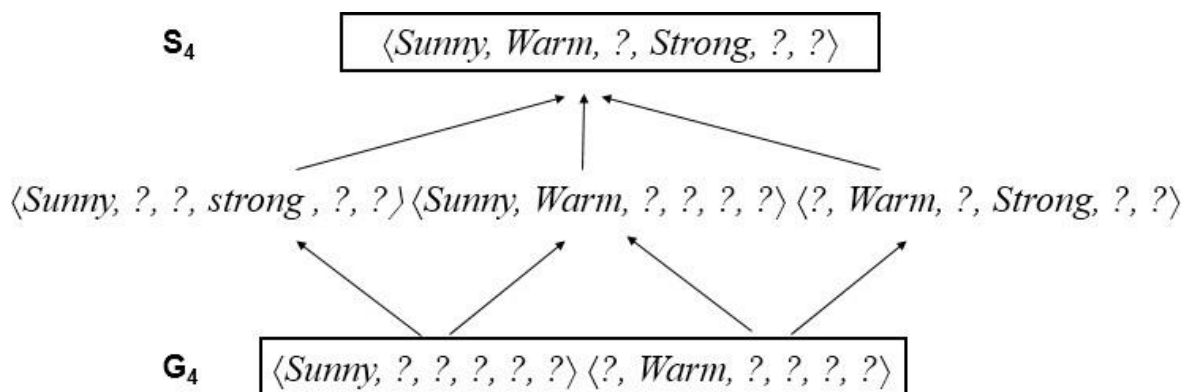
- Consider the fourth training example.

For training example d, $\langle \text{Sunny, Warm, High, Strong, Cool Change} \rangle +$



- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

After processing these four examples, the boundary sets S_4 and G_4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



INDUCTIVE BIAS: Inductive bias refers to the restrictions that are imposed by the assumptions made in the learning method.

The fundamental questions for inductive inference

1. What if the target concept is not contained in the hypothesis space how the output is predicted
2. Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis by making more generalize
3. How does the size of this hypothesis space influence the ability of the algorithm to generalize and make predictions
4. How does the size of the hypothesis space influence the number of training examples that must be observed and how given hypothesis is correct prediction

These fundamental questions are examined in the context of the CANDIDATE-ELIMINATION algorithm

A Biased Hypothesis Space:

Which are more Specific-

Deals only with Postive consistent data

- Suppose the target concept is not contained in the hypothesis space H , then obvious solution is to enrich the hypothesis space to include every possible hypothesis.
- Consider the *EnjoySport* example in which the hypothesis space is restricted to include only conjunctions of attribute values. Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as
"Sky = Sunny or Sky = Cloudy."
- The following three training examples of disjunctive hypothesis, the algorithm would find that there are zero hypotheses in the version space

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

- If Candidate Elimination algorithm is applied, then it end up with empty Version Space. After first two training example

$$S = \langle ? \text{ Warm Normal Strong Cool Change} \rangle$$

- This new hypothesis is overly general and it incorrectly covers the third negative training example.

An Unbiased Learner :

Which are more Generalized

Have solution for negative inconsistent data

- The solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space capable of representing every teachable concept that is representing every possible subset of the instances X.
- The set of all subsets of a set X is called the power set of X
 - In the *EnjoySport* learning task the size of the instance space X of days described by the six attributes is 96 instances.
 - Thus, there are 2^{96} distinct target concepts that could be defined over this instance space and learner might be called upon to learn.

Example:

Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp*, *Humidity*, *Wind*, *Water*, *Forecast* each have two possible values, the instance space X contains exactly

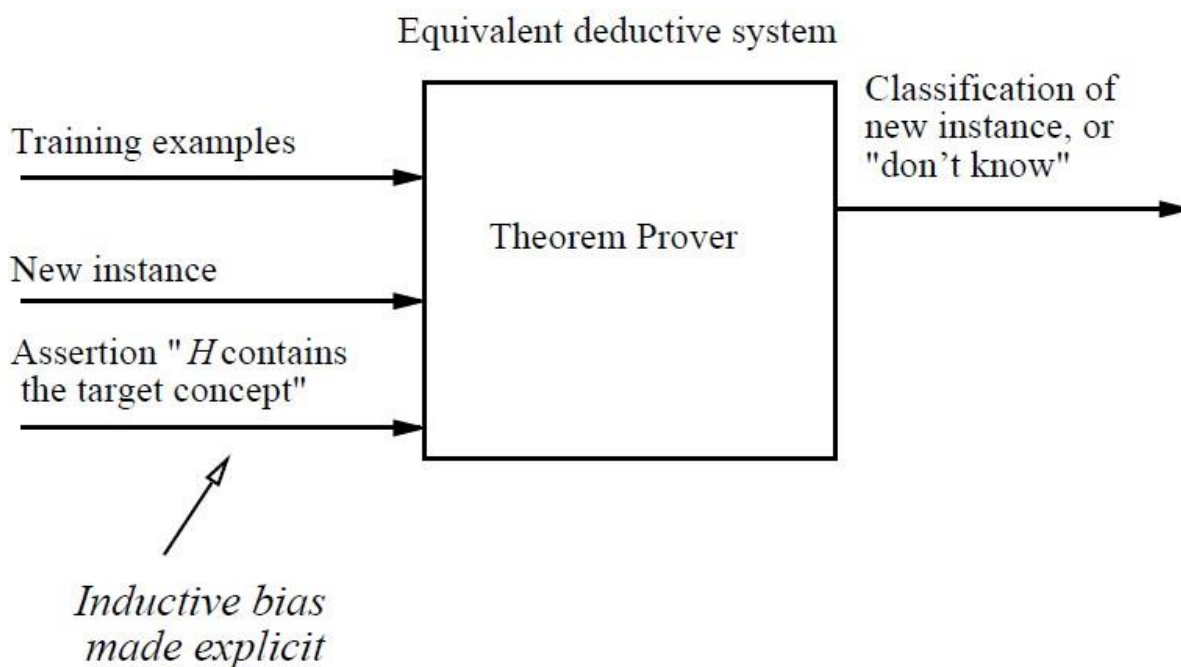
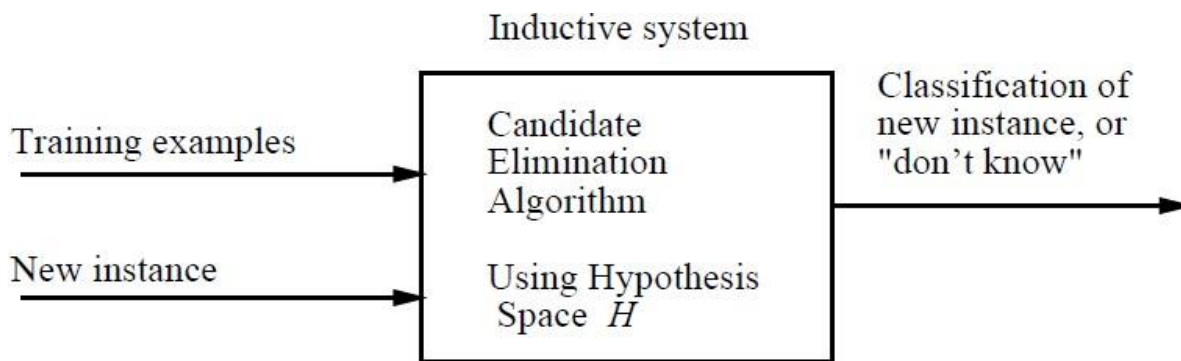
$$3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96 \text{ distinct instances } \vee$$

$$5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120 \text{ distinct hypotheses within H. (specific and general)}$$

$$1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973. \text{ Semantically distinct hypotheses (only general +1 specific)}$$

The below figure explains Inductive and deductive system

- Modelling inductive systems by equivalent deductive systems.
- The input-output behavior of the CANDIDATE-ELIMINATION algorithm using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertion "H contains the target concept." This assertion is therefore called the inductive bias of the CANDIDATE-ELIMINATION algorithm.
- Characterizing inductive systems by their inductive bias allows modelling them by their equivalent deductive systems. This provides a way to compare inductive systems according to their policies for generalizing beyond the observed training data.



An Example for general Mathematic resdaoning for inductive and deductive system

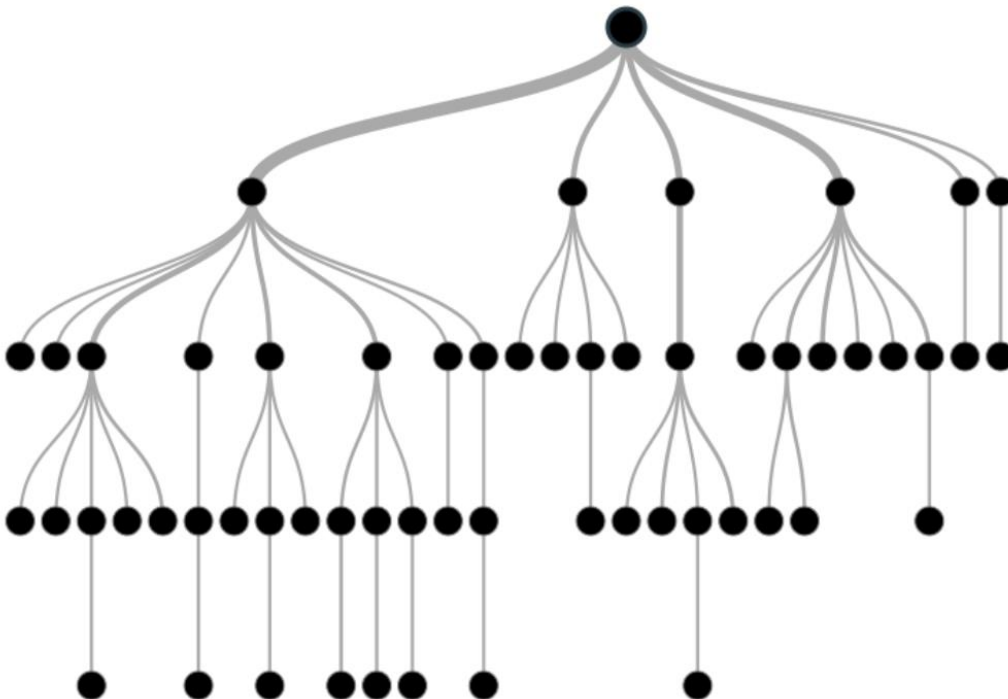
Inductive Reasoning: Maximilian is a shelter dog. He is happy. All shelter dogs are happy.
 Deductive Reasoning: Maximillian is a shelter dog. All shelter dogs are happy.

DECISION TREE LEARNING

What is a Decision Tree

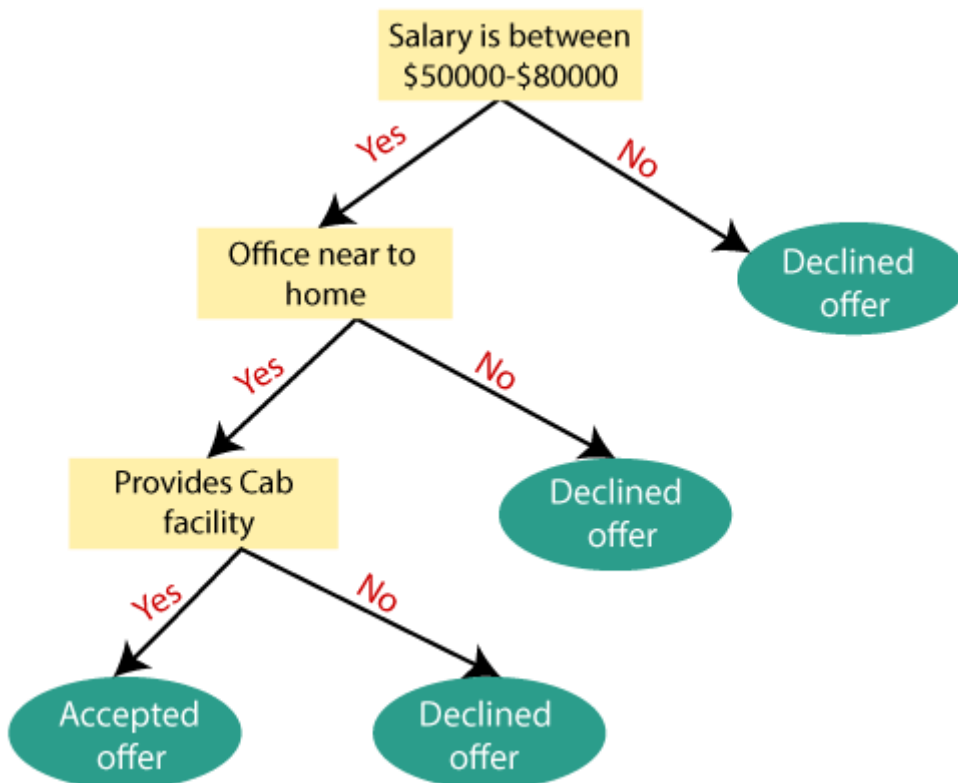
- ID3 stands for Iterative Dichotomiser 3
- ID3 is a precursor to the C4.5 Algorithm.
- The ID3 algorithm was invented by Ross Quinlan in 1975
- Used to generate a decision tree from a given data set by employing a top-down, greedy search, to test each attribute at every node of the tree.
- The resulting tree is used to classify future samples.

It is a tool that has applications spanning several different areas. Decision trees can be used for classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.



- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- **Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.

- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(s) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree through information gain

Information Gain

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- **Information gain**, is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, $\text{Gain}(S, A)$ of an attribute A , relative to a collection of examples S , is defined as

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

Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

- Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)]

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm(not in syllabus)**
- For more class labels, the computational complexity of the decision tree may increase.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

To define information gain, we begin by defining a measure called entropy. *Entropy measures the impurity of a collection of examples.*

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

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where,

p_{+} is the proportion of positive examples in S
 p_{-} is the proportion of negative examples in S.

Example:

Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is

$$\begin{aligned} \text{Entropy}([9+, 5-]) &= -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) \\ &= 0.940 \end{aligned}$$

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

DECISION TREE REPRESENTATION

- 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, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

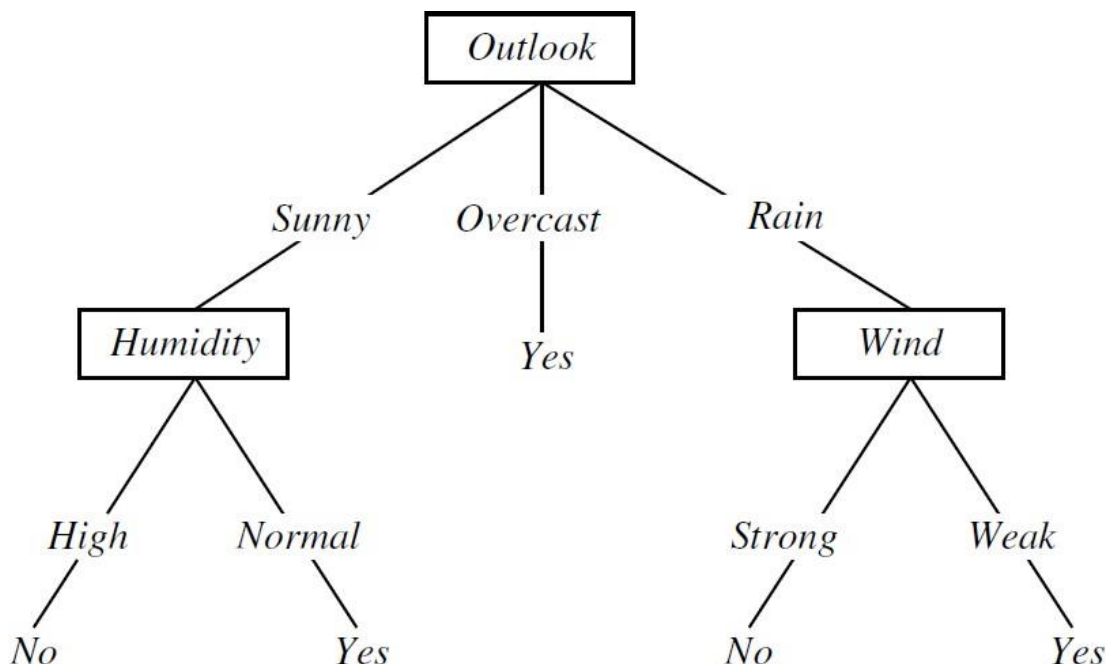


FIGURE: A decision tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

An Illustrative Example

- To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.
- Here the target attribute *PlayTennis*, which can have values *yes* or *no* for different days.
- Consider the first step through the algorithm, in which the topmost node of the decision tree is created.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain.

What are the steps in ID3 algorithm?

The steps in ID3 algorithm are as follows:

1. Calculate entropy for dataset.
2. For each attribute/feature.
 - 2.1. Calculate entropy for all its categorical values.
 - 2.2. Calculate information gain for the feature.
3. Find the feature with maximum information gain.
4. Repeat it until we get the desired tree.

Complete entropy of dataset is:

$$\begin{aligned}
 H(S) &= - p(\text{yes}) * \log_2(p(\text{yes})) - p(\text{no}) * \log_2(p(\text{no})) \\
 &= - (9/14) * \log_2(9/14) - (5/14) * \log_2(5/14) \\
 &= - (-0.41) - (-0.53) \\
 &= 0.94
 \end{aligned}$$

First Attribute – Outlook

Categorical values - sunny, overcast and rain

$$H(\text{Outlook}=\text{sunny}) = -(2/5) * \log_2(2/5) - (3/5) * \log_2(3/5) = 0.971$$

$$H(\text{Outlook}=\text{rain}) = -(3/5) * \log_2(3/5) - (2/5) * \log_2(2/5) = 0.971$$

$$H(\text{Outlook}=\text{overcast}) = -(4/4) * \log_2(4/4) - 0 = 0$$

Average Entropy Information for Outlook -

$$\begin{aligned}
 I(\text{Outlook}) &= p(\text{sunny}) * H(\text{Outlook}=\text{sunny}) + p(\text{rain}) * H(\text{Outlook}=\text{rain}) + p(\text{overcast}) * \\
 &H(\text{Outlook}=\text{overcast}) \\
 &= (5/14) * 0.971 + (5/14) * 0.971 + (4/14) * 0 \\
 &= 0.693
 \end{aligned}$$

$$\begin{aligned}
 \text{Information Gain} &= H(S) - I(\text{Outlook}) \\
 &= 0.94 - 0.693 \\
 &= 0.247
 \end{aligned}$$

Second Attribute - Temperature

Categorical values - hot, mild, cool

$$H(\text{Temperature}=\text{hot}) = -(2/4) * \log_2(2/4) - (2/4) * \log_2(2/4) = 1$$

$$H(\text{Temperature}=\text{cool}) = -(3/4) * \log_2(3/4) - (1/4) * \log_2(1/4) = 0.811$$

$$H(\text{Temperature}=\text{mild}) = -(4/6) * \log(4/6) - (2/6) * \log(2/6) = 0.9179$$

Average Entropy Information for Temperature -

$$\begin{aligned} I(\text{Temperature}) &= p(\text{hot}) * H(\text{Temperature}=\text{hot}) + p(\text{mild}) * H(\text{Temperature}=\text{mild}) + \\ & p(\text{cool}) * H(\text{Temperature}=\text{cool}) \\ &= (4/14) * 1 + (6/14) * 0.9179 + (4/14) * 0.811 \\ &= 0.9108 \end{aligned}$$

$$\begin{aligned} \text{Information Gain} &= H(S) - I(\text{Temperature}) \\ &= 0.94 - 0.9108 \\ &= 0.0292 \end{aligned}$$

Third Attribute - Humidity

Categorical values - high, normal

$$H(\text{Humidity}=\text{high}) = -(3/7) * \log(3/7) - (4/7) * \log(4/7) = 0.983$$

$$H(\text{Humidity}=\text{normal}) = -(6/7) * \log(6/7) - (1/7) * \log(1/7) = 0.591$$

Average Entropy Information for Humidity -

$$\begin{aligned} I(\text{Humidity}) &= p(\text{high}) * H(\text{Humidity}=\text{high}) + p(\text{normal}) * H(\text{Humidity}=\text{normal}) \\ &= (7/14) * 0.983 + (7/14) * 0.591 \\ &= 0.787 \end{aligned}$$

$$\begin{aligned} \text{Information Gain} &= H(S) - I(\text{Humidity}) \\ &= 0.94 - 0.787 \\ &= 0.153 \end{aligned}$$

Fourth Attribute - Wind

Categorical values - weak, strong

$$H(\text{Wind}=\text{weak}) = -(6/8) * \log(6/8) - (2/8) * \log(2/8) = 0.811$$

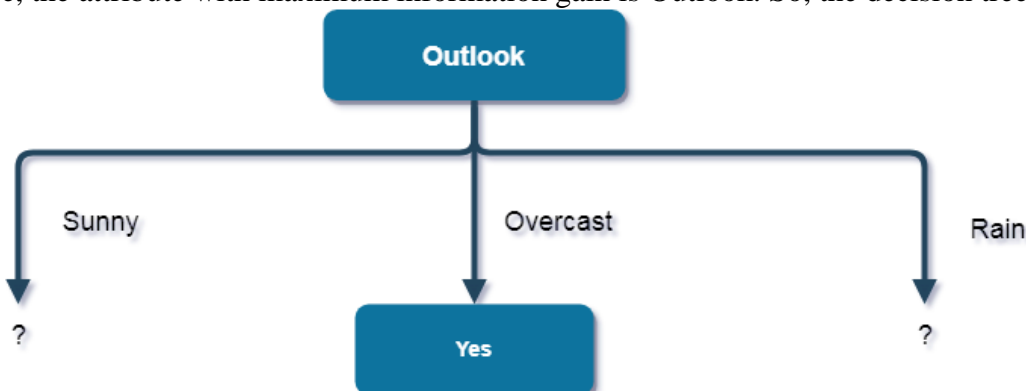
$$H(\text{Wind}=\text{strong}) = -(3/6) * \log(3/6) - (3/6) * \log(3/6) = 1$$

Average Entropy Information for Wind -

$$\begin{aligned} I(\text{Wind}) &= p(\text{weak}) * H(\text{Wind}=\text{weak}) + p(\text{strong}) * H(\text{Wind}=\text{strong}) \\ &= (8/14) * 0.811 + (6/14) * 1 \\ &= 0.892 \end{aligned}$$

$$\begin{aligned} \text{Information Gain} &= H(S) - I(\text{Wind}) \\ &= 0.94 - 0.892 \\ &= 0.048 \end{aligned}$$

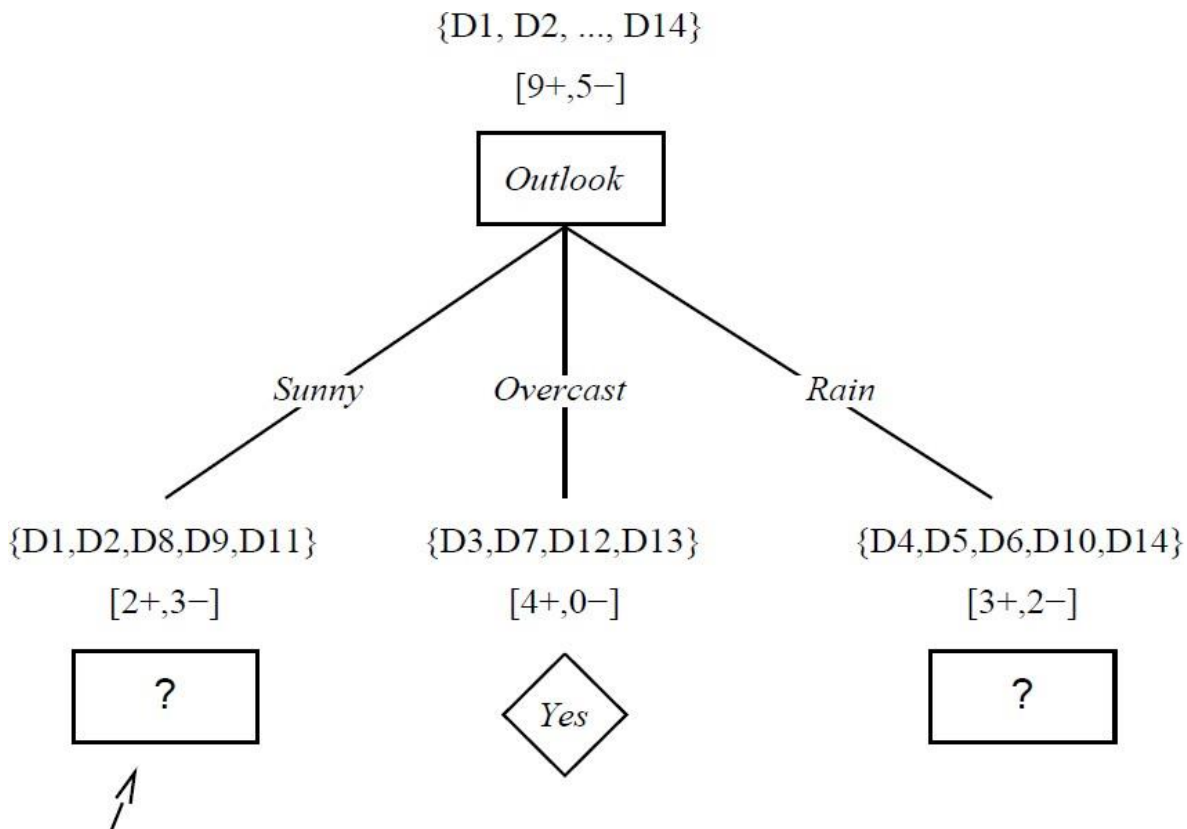
Here, the attribute with maximum information gain is Outlook. So, the decision tree built so far -



- when Outlook == overcast, it is of pure class(Yes).
 Now, we have to repeat same procedure for the data with rows consist of Outlook value as Sunny and then for Outlook value as Rain. The information gain values for all four attributes are

$$\begin{aligned} \text{Gain}(S, \text{Outlook}) &= 0.246 \\ \text{Gain}(S, \text{Humidity}) &= 0.151 \\ \text{Gain}(S, \text{Wind}) &= 0.048 \\ \text{Gain}(S, \text{Temperature}) &= 0.029 \end{aligned}$$

- According to the information gain measure, the **Outlook** attribute provides the best prediction of the target attribute, **PlayTennis**, over the training examples. Therefore, **Outlook** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.



Which attribute should be tested here?

finding the best attribute for splitting the data with Outlook=Sunny

Complete entropy of Sunny is -

$$\begin{aligned}H(S) &= - p(\text{yes}) * \log_2(p(\text{yes})) - p(\text{no}) * \log_2(p(\text{no})) \\ &= - (2/5) * \log_2(2/5) - (3/5) * \log_2(3/5) \\ &= 0.971\end{aligned}$$

First Attribute - Temperature

Categorical values - hot, mild, cool

$$H(\text{Sunny, Temperature=hot}) = -0 - (2/2) * \log(2/2) = 0$$

$$H(\text{Sunny, Temperature=cool}) = -(1) * \log(1) - 0 = 0$$

$$H(\text{Sunny, Temperature=mild}) = -(1/2) * \log(1/2) - (1/2) * \log(1/2) = 1$$

Average Entropy Information for Temperature -

$$\begin{aligned}I(\text{Sunny, Temperature}) &= p(\text{Sunny, hot}) * H(\text{Sunny, Temperature=hot}) + p(\text{Sunny, mild}) * H(\text{Sunny, Temperature=mild}) + p(\text{Sunny, cool}) * H(\text{Sunny, Temperature=cool}) \\ &= (2/5) * 0 + (1/5) * 0 + (2/5) * 1 \\ &= 0.4\end{aligned}$$

$$\begin{aligned}\text{Information Gain} &= H(\text{Sunny}) - I(\text{Sunny, Temperature}) \\ &= 0.971 - 0.4 \\ &= 0.571\end{aligned}$$

Second Attribute - Humidity

Categorical values - high, normal

$$H(\text{Sunny, Humidity=high}) = -0 - (3/3) * \log(3/3) = 0$$

$$H(\text{Sunny, Humidity=normal}) = -(2/2) * \log(2/2) - 0 = 0$$

Average Entropy Information for Humidity -

$$\begin{aligned}I(\text{Sunny, Humidity}) &= p(\text{Sunny, high}) * H(\text{Sunny, Humidity=high}) + p(\text{Sunny, normal}) * H(\text{Sunny, Humidity=normal}) \\ &= (3/5) * 0 + (2/5) * 0 \\ &= 0\end{aligned}$$

$$\begin{aligned}\text{Information Gain} &= H(\text{Sunny}) - I(\text{Sunny, Humidity}) \\ &= 0.971 - 0 \\ &= 0.971\end{aligned}$$

Third Attribute – Wind

Categorical values - weak, strong

$$H(\text{Sunny}, \text{Wind}=\text{weak}) = -(1/3) * \log(1/3) - (2/3) * \log(2/3) = 0.918$$

$$H(\text{Sunny}, \text{Wind}=\text{strong}) = -(1/2) * \log(1/2) - (1/2) * \log(1/2) = 1$$

Average Entropy Information for Wind -

$$I(\text{Sunny}, \text{Wind}) = p(\text{Sunny}, \text{weak}) * H(\text{Sunny}, \text{Wind}=\text{weak}) + p(\text{Sunny}, \text{strong}) * H(\text{Sunny}, \text{Wind}=\text{strong})$$

$$= (3/5) * 0.918 + (2/5) * 1$$

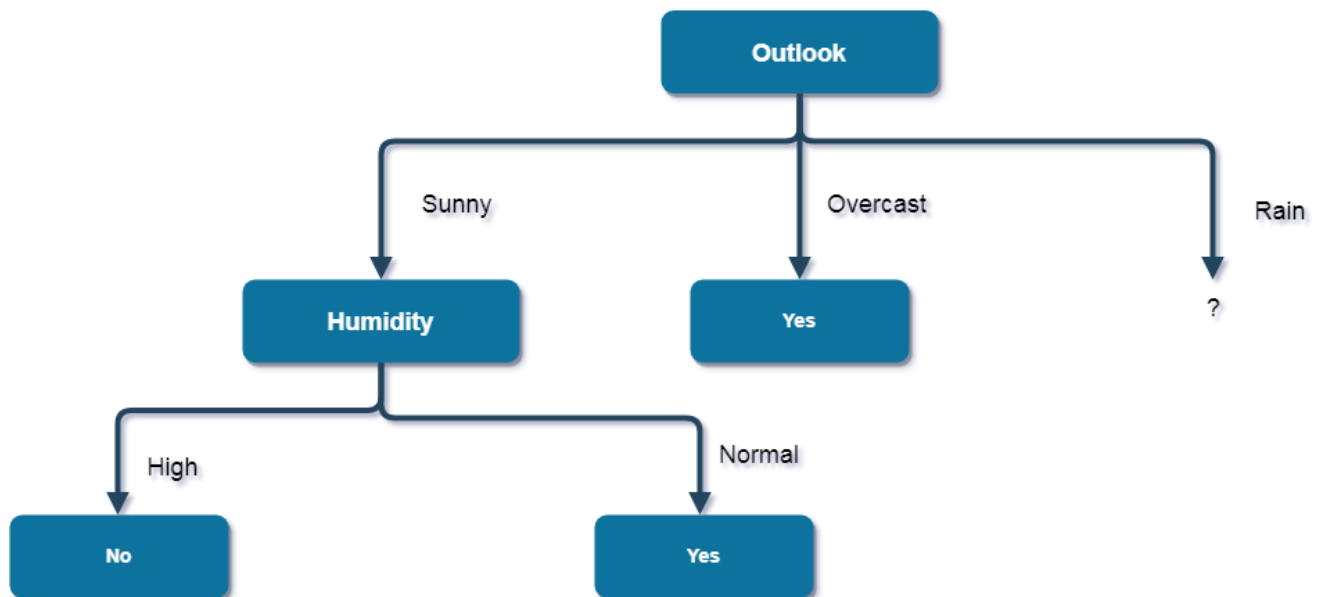
$$= 0.9508$$

$$\text{Information Gain} = H(\text{Sunny}) - I(\text{Sunny}, \text{Wind})$$

$$= 0.971 - 0.9508$$

$$= 0.0202$$

Here, the attribute with maximum information gain is Humidity. So, the decision tree built so far -



Here, when Outlook = Sunny and Humidity = High, it is a pure class of category "no". And When Outlook = Sunny and Humidity = Normal, it is again a pure class of category "yes". Therefore, we don't need to do further calculations.

Complete entropy of Rain is -

$$H(S) = - p(\text{yes}) * \log_2(p(\text{yes})) - p(\text{no}) * \log_2(p(\text{no}))$$

$$= - (3/5) * \log(3/5) - (2/5) * \log(2/5)$$

$$= 0.971$$

First Attribute - Temperature

First Attribute - Temperature

Categorical values - mild, cool

$$H(\text{Rain, Temperature=cool}) = - (1/2) * \log(1/2) - (1/2) * \log(1/2) = 1$$

$$H(\text{Rain, Temperature=mild}) = - (2/3) * \log(2/3) - (1/3) * \log(1/3) = 0.918$$

Average Entropy Information for Temperature -

$$I(\text{Rain, Temperature}) = p(\text{Rain, mild}) * H(\text{Rain, Temperature=mild}) + p(\text{Rain, cool}) * H(\text{Rain, Temperature=cool})$$

$$= (2/5) * 1 + (3/5) * 0.918$$

$$= 0.9508$$

$$\text{Information Gain} = H(\text{Rain}) - I(\text{Rain, Temperature})$$

$$= 0.971 - 0.9508$$

$$= 0.0202$$

Second Attribute - Wind

Categorical values - weak, strong

$$H(\text{Wind=weak}) = - (3/3) * \log(3/3) - 0 = 0$$

$$H(\text{Wind=strong}) = 0 - (2/2) * \log(2/2) = 0$$

Average Entropy Information for Wind -

$$I(\text{Wind}) = p(\text{Rain, weak}) * H(\text{Rain, Wind=weak}) + p(\text{Rain, strong}) * H(\text{Rain, Wind=strong})$$

$$= (3/5) * 0 + (2/5) * 0$$

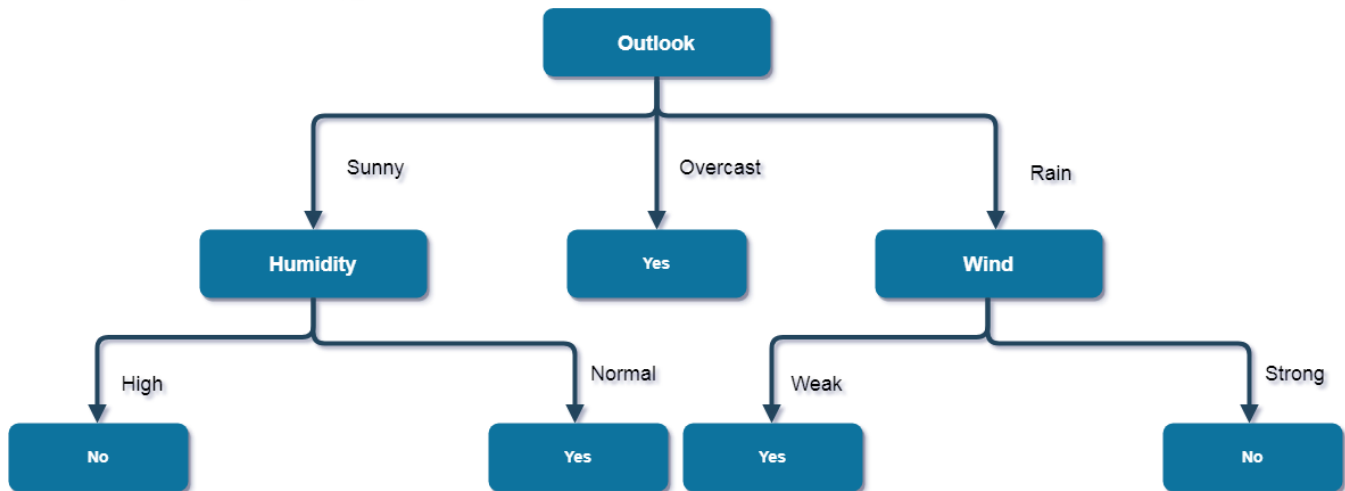
$$= 0$$

$$\text{Information Gain} = H(\text{Rain}) - I(\text{Rain, Wind})$$

$$= 0.971 - 0$$

$$= 0.971$$

Here, the attribute with maximum information gain is Wind. So, the decision tree built so far -



Here, when Outlook = Rain and Wind = Strong, it is a pure class of category "no". And When Outlook = Rain and Wind = Weak, it is again a pure class of category "yes".

And this is our final desired tree for the given dataset.

REAL time implementation of Algorithm

ID3 Algorithm is used to build a Decision Tree to predict the weather.

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- 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, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data

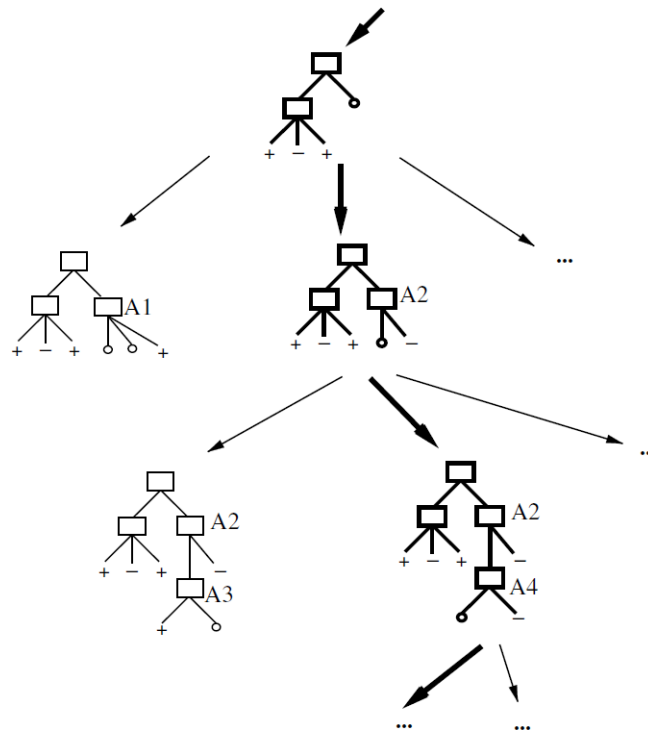


Figure: Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain.

ID3 in terms of its search space and search strategy, there are some insight into its capabilities and limitations

1. *ID3's hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree*

ID3 avoids one of the major risks of methods that search incomplete hypothesis spaces: that the hypothesis space might not contain the target function.

- ID3 maintains only a single current hypothesis as it searches through the space of decision trees.*

For example, with the earlier version space candidate elimination method, which maintains the set of all hypotheses consistent with the available training examples. But ID3 maintains Single set of Hypothesis formed

- ID3 in its pure form performs no backtracking in its search. Once it selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice.*
In the case of ID3, corresponds to the decision tree it selects along the single search path it explores.
- ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis.*

Why Prefer Short trees

Hypotheses proposed by

Theory of Occam's razor

- Occam's razor: is the problem-solving principle that the simplest solution tends to be the right one. When presented with competing hypotheses to solve a problem, **one should select the solution with the fewest assumptions.**
- Occam's razor: **“Prefer the simplest hypothesis that fits the data”.**

INDUCTIVE BIAS IN DECISION TREE LEARNING

Inductive bias is the set of assumptions that, together with the training data, can also deductively justify the classifications assigned by the learner to future instances

Given a collection of training examples, there are typically many decision trees consistent with these examples. Which of these decision trees does ID3 choose?

ID3 search strategy

- Selects in favour of shorter trees over longer ones
- Selects trees that place the attributes with highest information gain closest to the root.

Approximate inductive bias of ID3: Shorter trees are preferred over larger trees

- Consider an algorithm that begins with the empty tree and searches breadth first through progressively more complex trees.
- First considering all trees of depth 1, then all trees of depth 2, etc.
- Once it finds a decision tree consistent with the training data, it returns the smallest consistent tree at that search depth (e.g., the tree with the fewest nodes).
- Let us call this breadth-first search algorithm BFS-ID3.
- BFS-ID3 finds a shortest decision tree and thus exhibits the bias "shorter trees are preferred over longer trees.

A closer approximation to the 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.

- ID3 can be viewed as an efficient approximation to BFS-ID3, using a greedy heuristic search to attempt to find the shortest tree without conducting the entire breadth-first search through the hypothesis space.
- Because ID3 uses the information gain heuristic and a hill climbing strategy, it exhibits a more complex bias than BFS-ID3.

Restriction Biases and Preference Biases

Difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATION Algorithm.

ID3:

- 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.

Preference bias – The inductive bias of ID3 is a preference for certain hypotheses over others (e.g., preference for shorter hypotheses over larger hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is called a **preference bias or a search bias**.

Id3 inductive bias is called as Preference bias or search bias

CANDIDATE-ELIMINATION Algorithm:

- It searches this space completely, finding every hypothesis consistent with the training data.
- Its inductive bias is solely a consequence of the expressive power $2^{(96)}$ of its hypothesis representation.

Restriction bias – The bias of the CANDIDATE ELIMINATION algorithm is in the form of a categorical restriction on the set of hypotheses considered. This form of bias is typically called a **restriction bias or a language bias**.

Candidate Elimination Algorithm inductive bias is called as Preference bias or search bias

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



ESTD : 2001

Which type of inductive bias is preferred in order to generalize beyond the training data, a preference bias or restriction 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.
- In contrast, a restriction bias that strictly limits the set of potential hypotheses is generally less desirable, because it introduces the possibility of excluding the unknown target function altogether.

ISSUES IN DECISION TREE LEARNING

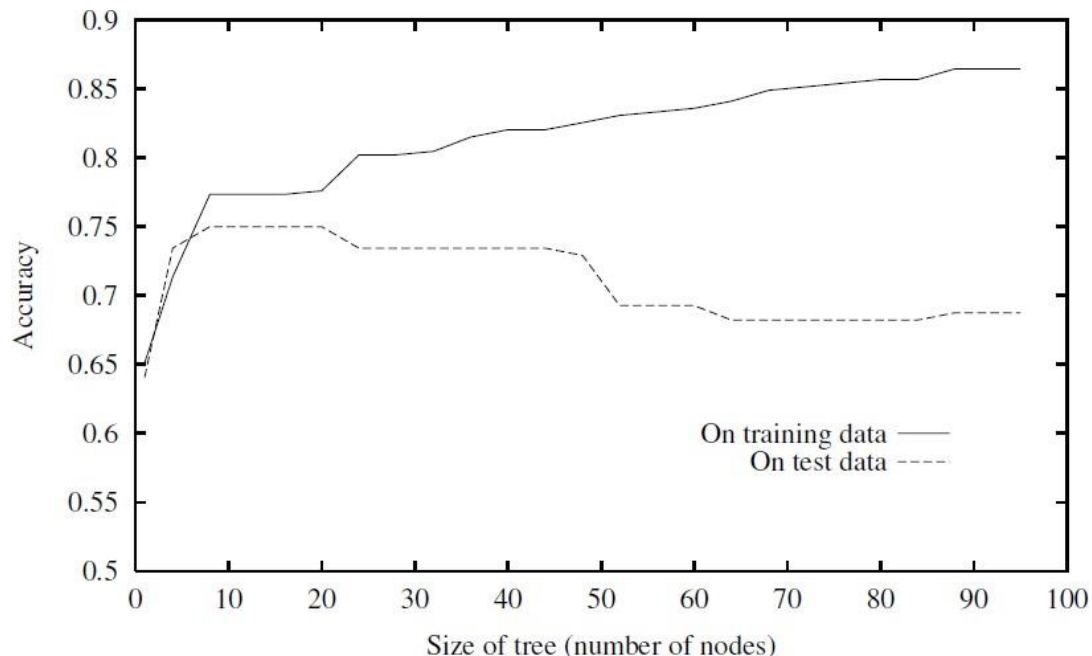
Issues in learning decision trees include

1. Avoiding Overfitting the Data
 - Reduced error pruning
 - Rule post-pruning
2. Incorporating Continuous-Valued Attributes
3. Alternative Measures for Selecting Attributes
4. Handling Training Examples with Missing Attribute Values
5. Handling Attributes with Differing Costs

1. Avoiding Overfitting the Data

- The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.
- **Definition - Overfit:** 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.

The below figure illustrates the impact of overfitting in a typical application of decision tree learning.



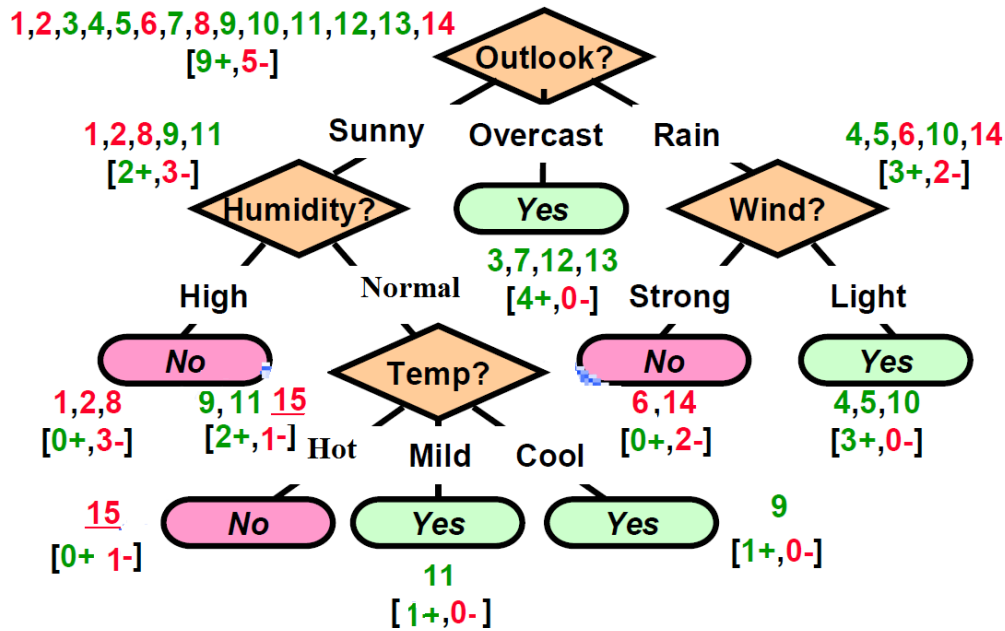
- *The horizontal axis* of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree.
- *The solid line* shows the accuracy of the decision tree over the training examples. The broken line shows accuracy measured over an independent set of test example
- The accuracy of the tree over the training examples increases monotonically as the tree is grown. The accuracy measured over the independent test examples first increases, then decreases.

How can it be possible for tree h to fit the training examples better than h' , but for it to perform more poorly over subsequent examples?

1. Overfitting can occur when the training examples contain random errors or noise
2. When small numbers of examples are associated with leaf nodes.

Noisy Training Example

- Example 15: <Sunny, Hot, Normal, Strong, ->
- Example is noisy because the correct label is +
- Previously constructed tree misclassifies it



Approaches to avoiding overfitting in decision tree learning

- Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

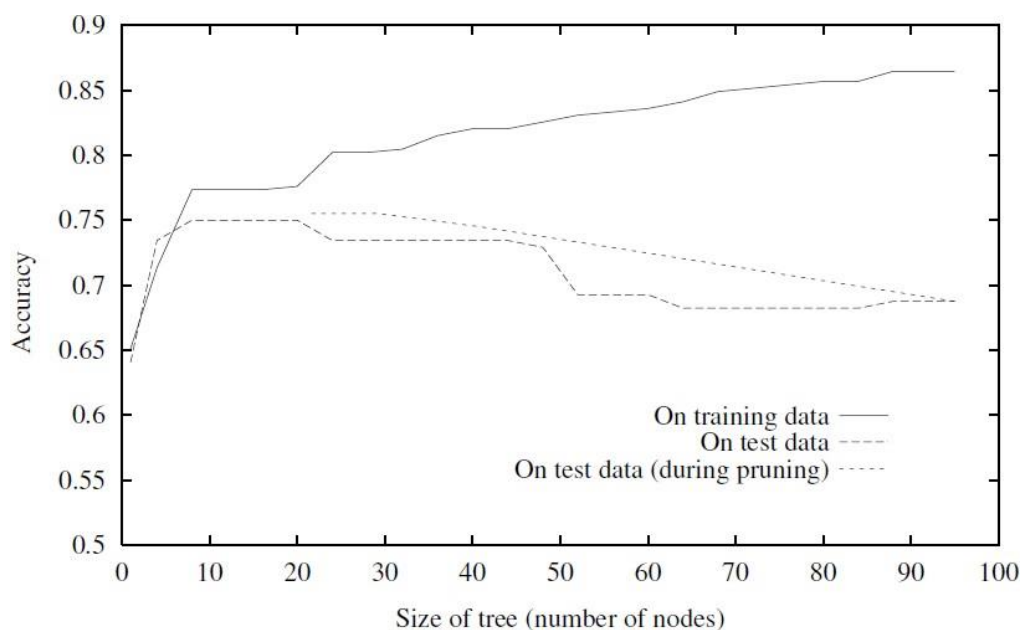
- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
- Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length

$$\text{MDL} - \text{Minimize} : \text{size}(\text{tree}) + \text{size}(\text{misclassifications}(\text{tree}))$$

Reduced-Error Pruning

- Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning
- **Pruning** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set.
- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in below figure



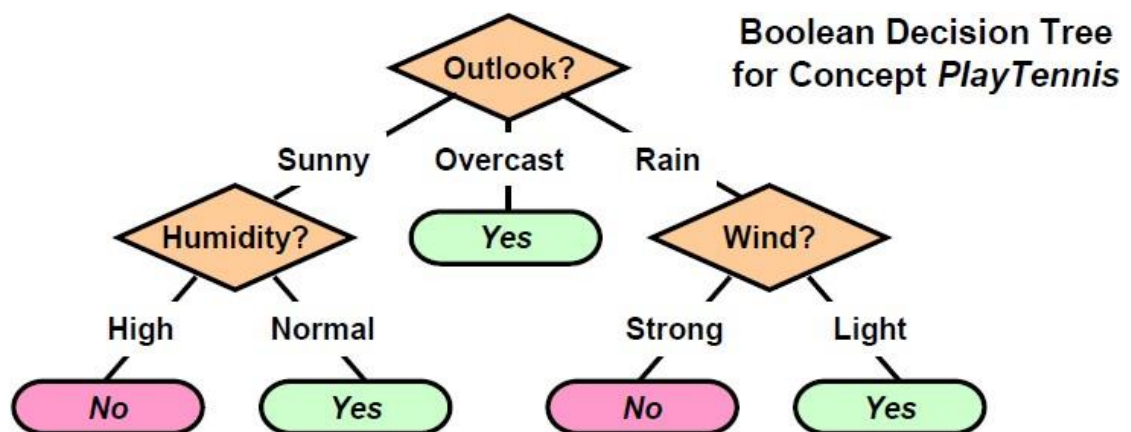
- The additional line in figure shows accuracy over the test examples as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- The available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets.

Rule Post-Pruning

Rule post-pruning is successful method for finding high accuracy hypotheses

- Rule post-pruning involves the following steps:
- Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
- 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 into Rules



Example

- IF (*Outlook = Sunny*) \wedge (*Humidity = High*) THEN *PlayTennis = No*
- IF (*Outlook = Sunny*) \wedge (*Humidity = Normal*) THEN *PlayTennis = Yes*
- ...

For example, consider the decision tree. The leftmost path of the tree in below figure is translated into the rule.

IF (Outlook = Sunny) ^ (Humidity = High)
THEN PlayTennis = No

Given the above rule, rule post-pruning would consider removing the preconditions
(Outlook = Sunny) and (Humidity = High)

- It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step.
- No pruning step is performed if it reduces the estimated rule accuracy.

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

1. Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high \equiv Temp > 35° C, med \equiv 10° C < Temp \leq 35° C, low \equiv Temp \leq 10° C}

2. Using thresholds for splitting nodes

e.g., $A \leq a$ produces subsets $A \leq a$ and $A > a$

What threshold-based Boolean attribute should be defined based on Temperature?

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

- Pick a threshold, c , that produces the greatest information gain
- In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: $(48 + 60)/2$, and $(80 + 90)/2$.
- The information gain can then be computed for each of the candidate attributes, Temperature $>_{54}$, and Temperature $>_{85}$ and the best can be selected (Temperature $>_{54}$)

3. Alternative Measures for Selecting Attributes

- The problem is if attributes with many values, Gain will select it ?
- Example: consider the attribute Date, which has a very large number of possible values. (e.g., March 4, 1979).
- If this attribute is added to the PlayTennis data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data.
- This decision tree with root node Date is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

One Approach: Use GainRatio instead of Gain

The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

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

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

Where, S_i is subset of S , for which attribute A has value v_i

4. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes

Example: Medical diagnosis

- <Fever = true, Blood-Pressure = normal, ..., Blood-Test = ?, ...>
- Sometimes values truly unknown, sometimes low priority (or cost too high)

Strategies for dealing with the missing attribute value

- If node n test A, assign most common value of A among other training examples sorted to node n
- Assign most common value of A among other training examples with same target value
- Assign a probability p_i to each of the possible values v_i of A rather than simply assigning the most common value to $A(x)$

5. Handling Attributes with Differing Costs

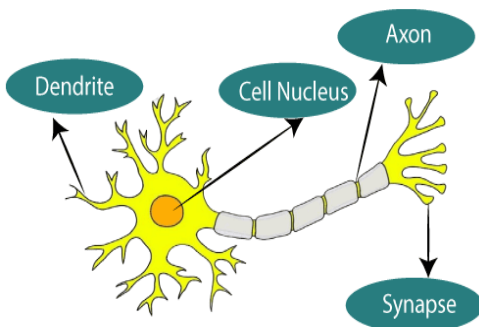
- In some learning tasks the instance attributes may have associated costs.
- For example: In learning to classify medical diseases, the patients described in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc.
- These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort
- Decision trees use low-cost attributes where possible, and depends only on high-cost attributes only when needed to produce reliable and accurate classifications

Unit 2

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

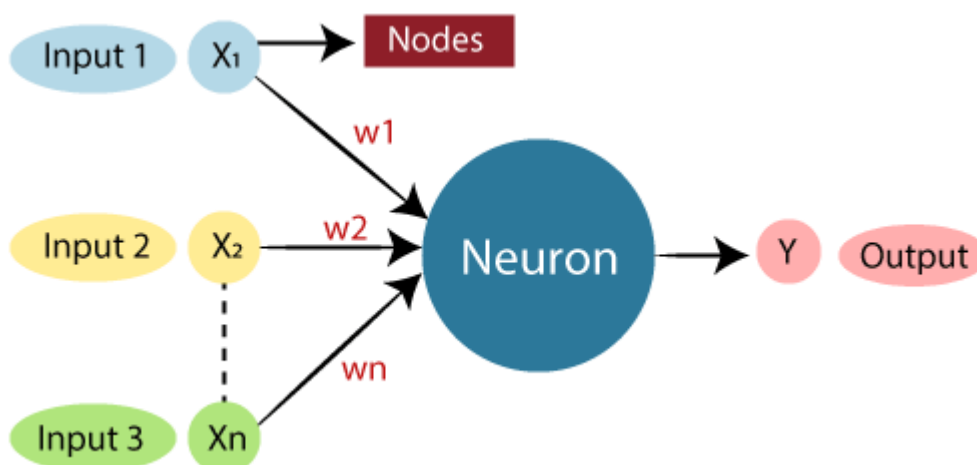
What is Artificial Neural Network

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

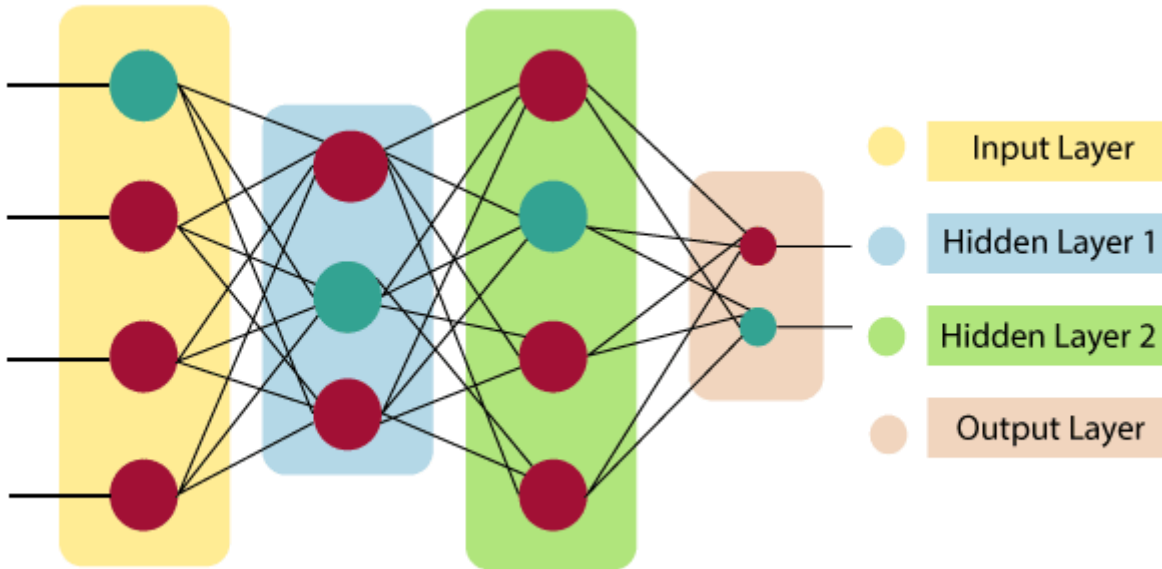
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^n W_i * X_i + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Advantages of Artificial Neural Network (ANN)

Parallel processing capability:

Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Capability to work with incomplete knowledge:

After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having fault tolerance:

Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.

Disadvantages of Artificial Neural Network:

Assurance of proper network structure:

There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

Unrecognized behavior of the network:

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results.

PERCEPTRON

- One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.

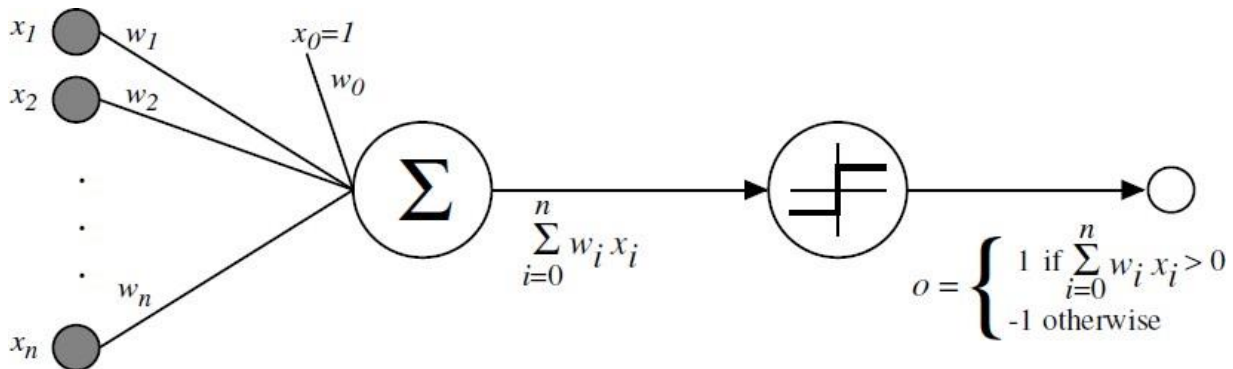


Figure: A perceptron

- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- Given inputs \mathbf{x} through \mathbf{x}_n , the output $\mathbf{O}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ computed by the perceptron is

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + \dots + w_nx_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where, each w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output.
- $-w_0$ is a threshold that the weighted combination of inputs $w_1x_1 + \dots + w_nx_n$ must surpass in order for the perceptron to output a 1.

Representational Power of Perceptrons

- The perceptron can be viewed as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points)
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in below figure

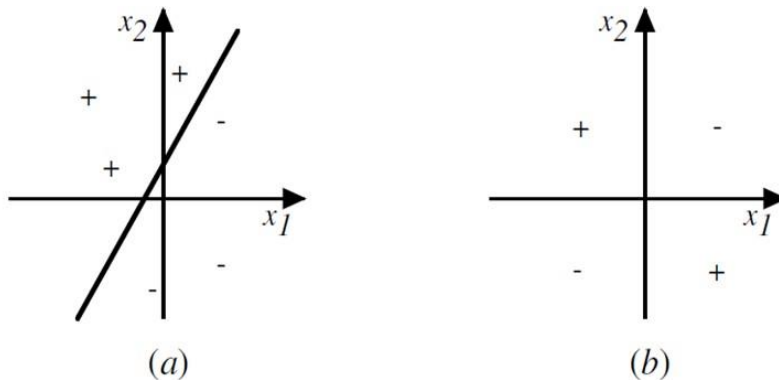
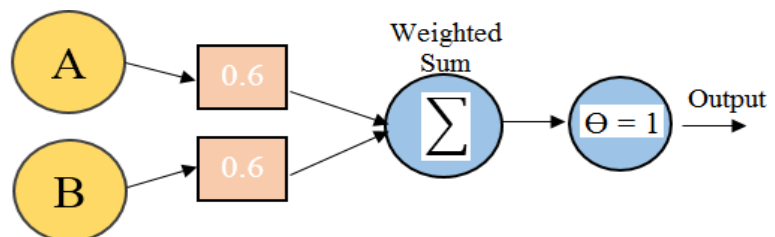


Figure : The decision surface represented by a two-input perceptron.
 (a) A set of training examples and the decision surface of a perceptron that classifies them correctly. (b) A set of training examples that is not linearly separable.
 x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND (\sim AND), and NOR (\sim OR)

Example: Representation of AND functions

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1



If $A=0$ & $B=0 \rightarrow 0*0.6 + 0*0.6 = 0$.

This is not greater than the threshold of 1, so the output = 0.

If $A=0$ & $B=1 \rightarrow 0*0.6 + 1*0.6 = 0.6$.

This is not greater than the threshold, so the output = 0.

If $A=1$ & $B=0 \rightarrow 1*0.6 + 0*0.6 = 0.6$.

This is not greater than the threshold, so the output = 0.

If $A=1$ & $B=1 \rightarrow 1*0.6 + 1*0.6 = 1.2$.

This exceeds the threshold, so the output = 1.

Drawback of perceptron

- The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable

The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = \eta(t - o)x_i$$

Here,

t is the target output for the current training example

o is the output generated by the perceptron

η is a positive constant called the *learning rate*

Drawback:

The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.

Problem 1

Truth Table of OR Logical GATE is,

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Weights $w_1 = 0.6$, $w_2 = 0.6$, Threshold = 1 and Learning Rate $n = 0.5$ are given

For Training Instance 1: A=0, B=0 and Target = 0

$$w_i \cdot x_i = 0 \cdot 0.6 + 0 \cdot 0.6 = 0$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

$$w_i \cdot x_i = 0 \cdot 0.6 + 1 \cdot 0.6 = 0.6$$

This is not greater than the threshold of 1, so the output = 0. Here the target does not match with calculated output.

$$w_i = w_i + n(t - o)x_i$$

$$w_1 = 0.6 + 0.5(1 - 0)0 = 0.6$$

$$w_2 = 0.6 + 0.5(1 - 0)1 = 1.1$$

Now,

Weights $w_1 = 0.6$, $w_2 = 1.1$, Threshold = 1 and Learning Rate $n = 0.5$ are given

For Training Instance 1: A=0, B=0 and Target = 0

$$w_i \cdot x_i = 0 \cdot 0.6 + 0 \cdot 1.1 = 0$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

$$w_i \cdot x_i = 0 \cdot 0.6 + 1 \cdot 1.1 = 1.1$$

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

For Training Instance 3: A=1, B=0 and Target = 1

$$w_i \cdot x_i = 1 \cdot 0.6 + 0 \cdot 1.1 = 0.6$$

This is not greater than the threshold of 1, so the output = 0. Here the target does not match with calculated output.

$$w_i = w_i + n(t - o)x_i$$

$$w_1 = 0.6 + 0.5(1 - 0)1 = 1.1$$

$$w_2 = 1.1 + 0.5(1 - 0)0 = 1.1$$

Now,

Weights $w_1 = 1.1$, $w_2 = 1.1$, Threshold = 1 and Learning Rate $n = 0.5$ are given

For Training Instance 1: A=0, B=0 and Target = 0

$$w_i \cdot x_i = 0 \cdot 2.2 + 0 \cdot 1.1 = 0$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 1

$$w_i \cdot x_i = 0 \cdot 1.1 + 1 \cdot 1.1 = 1.1$$

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

For Training Instance 3: A=1, B=0 and Target = 1

$$w_i.x_i = 1*1.1 + 0*1.1 = 1.1$$

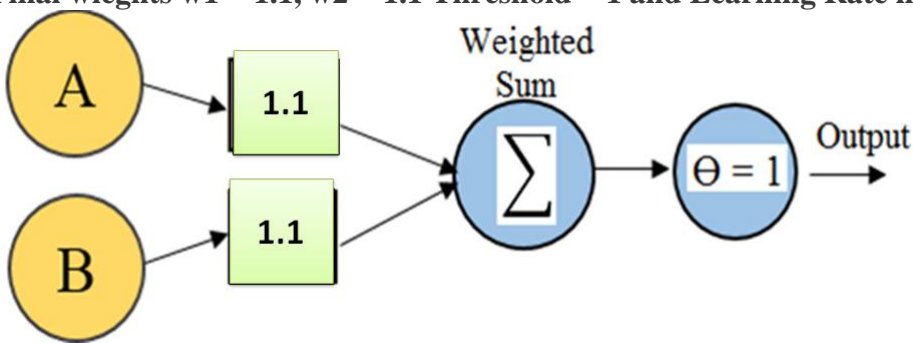
This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

For Training Instance 4: A=1, B=1 and Target = 1

$$w_i.x_i = 1*1.1 + 1*1.1 = 2.2$$

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

Final weights $w_1 = 1.1$, $w_2 = 1.1$ Threshold = 1 and Learning Rate $n = 0.5$.



Problem 2

Truth Table of AND Logical GATE is,

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Weights $w_1 = 1.2$, $w_2 = 0.6$, Threshold = 1 and Learning Rate $n = 0.5$ are given

For Training Instance 1: A=0, B=0 and Target = 0

$$w_i.x_i = 0*1.2 + 0*0.6 = 0$$

This is not greater than the threshold of 1, so the output = 0, Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 0

$$w_i \cdot x_i = 0 \cdot 1.2 + 1 \cdot 0.6 = 0.6$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=1, B=0 and Target = 0

$$w_i \cdot x_i = 1 \cdot 1.2 + 0 \cdot 0.6 = 1.2$$

This is greater than the threshold of 1, so the output = 1. Here the target does not match with the calculated output.

Hence we need to update the weights.

$$w_i = w_i + n(t - o)x_i$$

$$w_1 = 1.2 + 0.5(0 - 1)1 = 0.7$$

$$w_2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

Now,

After updating weights are $w_1 = 0.7$, $w_2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

$w_1 = 0.7$, $w_2 = 0.6$ Threshold = 1 and Learning Rate $n = 0.5$

For Training Instance 1: A=0, B=0 and Target = 0

$$w_i \cdot x_i = 0 \cdot 0.7 + 0 \cdot 0.6 = 0$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 2: A=0, B=1 and Target = 0

$$w_i \cdot x_i = 0 \cdot 0.7 + 1 \cdot 0.6 = 0.6$$

This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 3: A=1, B=0 and Target = 0

$$w_i \cdot x_i = 1 \cdot 0.7 + 0 \cdot 0.6 = 0.7$$

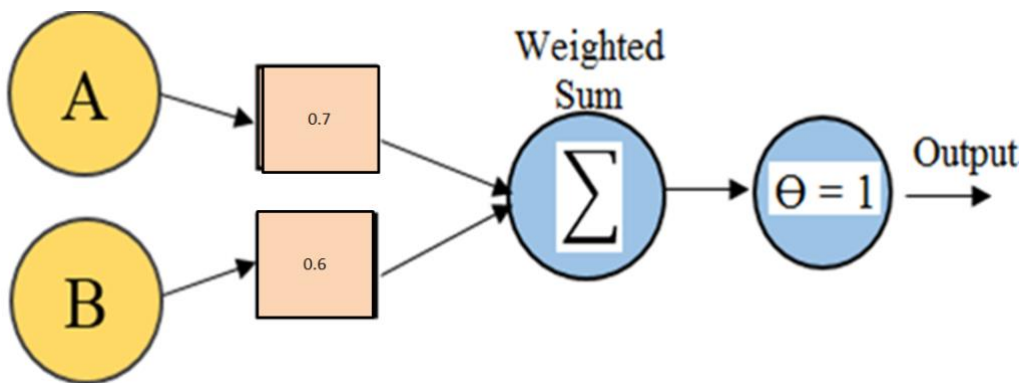
This is not greater than the threshold of 1, so the output = 0. Here the target is same as calculated output.

For Training Instance 4: A=1, B=1 and Target = 1

$$w_i \cdot x_i = 1 \cdot 0.7 + 1 \cdot 0.6 = 1.3$$

This is greater than the threshold of 1, so the output = 1. Here the target is same as calculated output.

Hence the final weights are $w_1 = 0.7$ and $w_2 = 0.6$, Threshold = 1 and Learning Rate $\eta = 0.5$.



An Example for NEURAL NETWORK REPRESENTATIONS in Real Time

- A prototypical example of ANN learning is provided by author Pomerleau's system ALVINN, (Autonomous Land Vehicle In a Neural Network) which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways
- The input to the neural network is a 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network output is the direction in which the vehicle is steered



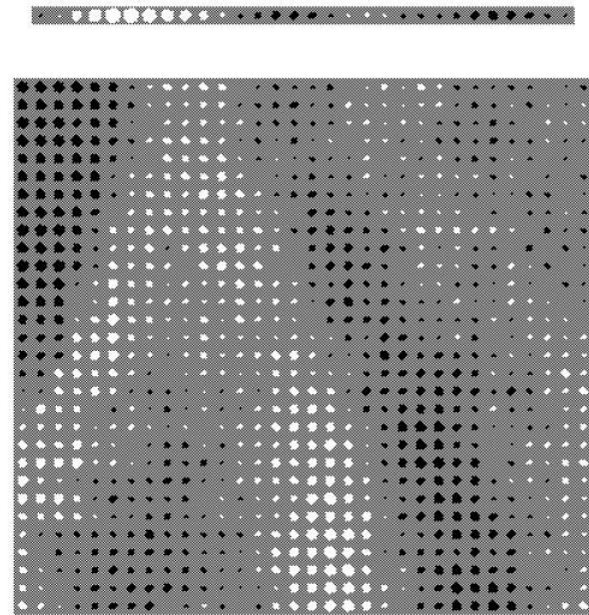
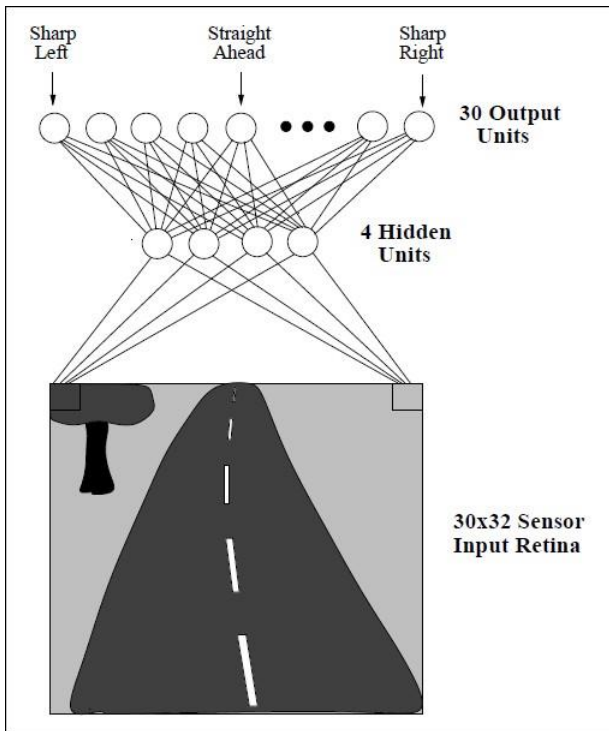


Figure: Neural network learning to steer an autonomous vehicle.

- Figure illustrates the neural network representation.
- The network is shown on the left side of the figure, with the input camera image depicted below it.
- Each node (i.e., circle) in the network diagram corresponds to the output of a single network unit, and the lines entering the node from below are its inputs.
- There are four units that receive inputs directly from all of the 30 x 32 pixels in the image. These are called "hidden" units because their output is available only within the network and is not available as part of the global network output. Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs
- These hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.
- The diagrams on the right side of the figure depict the learned weight values associated with one of the four hidden units in this ANN.
- The large matrix of black and white boxes on the lower right depicts the weights from the 30 x 32 pixel inputs into the hidden unit. Here, a white box indicates a positive weight, a black box a negative weight, and the size of the box indicates the weight magnitude.

APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

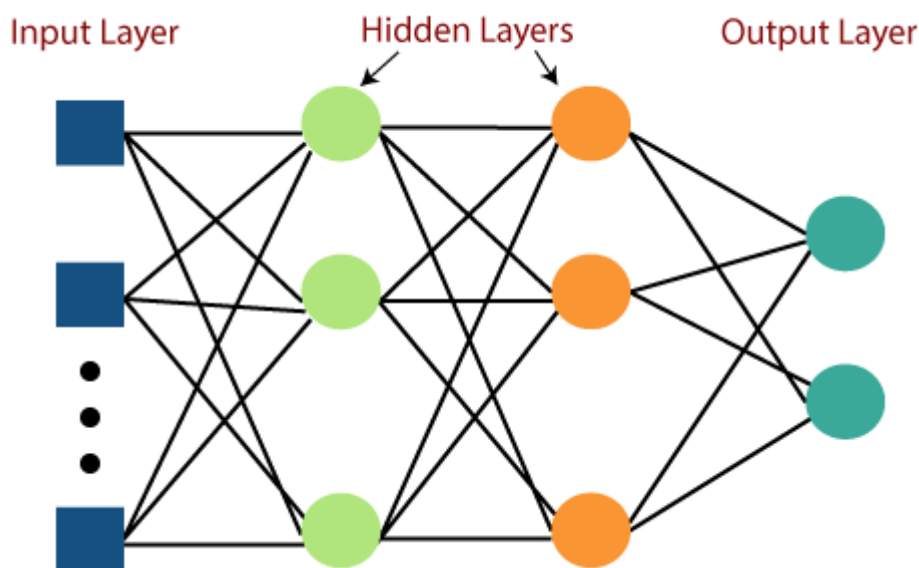
ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

ANN is appropriate for problems with the following characteristics:

1. Instances which are represented by many attribute-value pairs.
2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
3. The training examples may contain errors.
4. Long training times are acceptable to train.
5. Fast evaluation of the learned target function it can also support fast testing process.

Multi-layer Perceptron in TensorFlow

Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron. TensorFlow is a very popular deep learning framework released by, and this notebook will guide to build a neural network with this library. If we want to understand what is a Multi-layer perceptron, The pictorial representation of multi-layer perceptron learning is as shown below-



MLP networks are used for supervised learning format. A typical learning algorithm for MLP networks is also called **back propagation's algorithm**.

A multilayer perceptron (MLP) is a feed forward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers. MLP uses back propagation for training the network. MLP is a deep learning method.

MULTILAYER NETWORKS AND THE BACKPROPAGATION ALGORITHM example

Multilayer networks learned by the BACKPROPAGATION algorithm are capable of expressing a rich variety of nonlinear decision surfaces.

Consider the example:

- Here the speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h_d" (i.e., "hid," "had," "head," "hood," etc.).
- The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The network prediction is the output whose value is highest.
- The plot on the right illustrates the highly nonlinear decision surface represented by the learned network. Points shown on the plot are test examples distinct from the examples used to train the network.

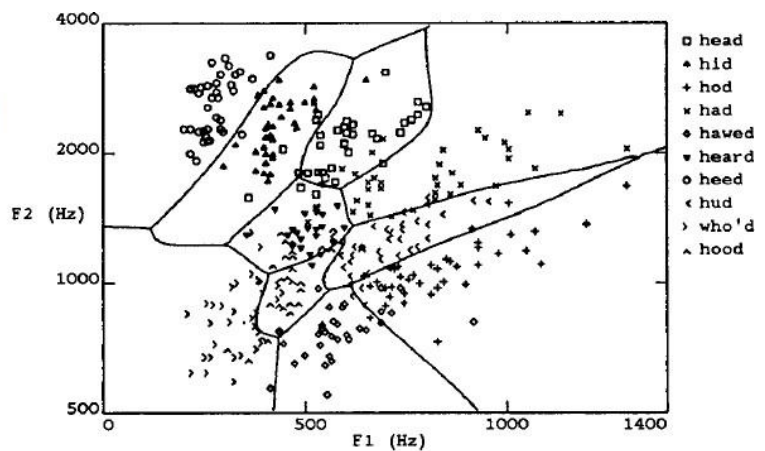
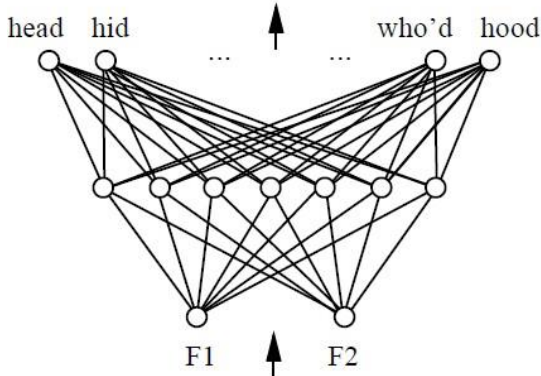


Figure: Decision regions of a multilayer feedforward network.

What is back propagation?

We can define the back propagation algorithm as an algorithm that trains some given feed-forward Neural Network for a given input pattern where the classifications are known to us. At the point when every passage of the example set is exhibited to the network, the network looks at its yield reaction to the example input pattern. After that, the comparison done between output response and expected output with the error value is measured. Later, we adjust the connection weight based upon the error value measured.

In simple terms, after each feed-forward passes through a network, this algorithm does the backward pass to adjust the model's parameters based on weights and biases. A typical supervised learning algorithm attempts to find a function that maps input data to the right output. Back propagation works with a multi-layered neural network and learns internal representations of input to output mapping.

How does back propagation work?

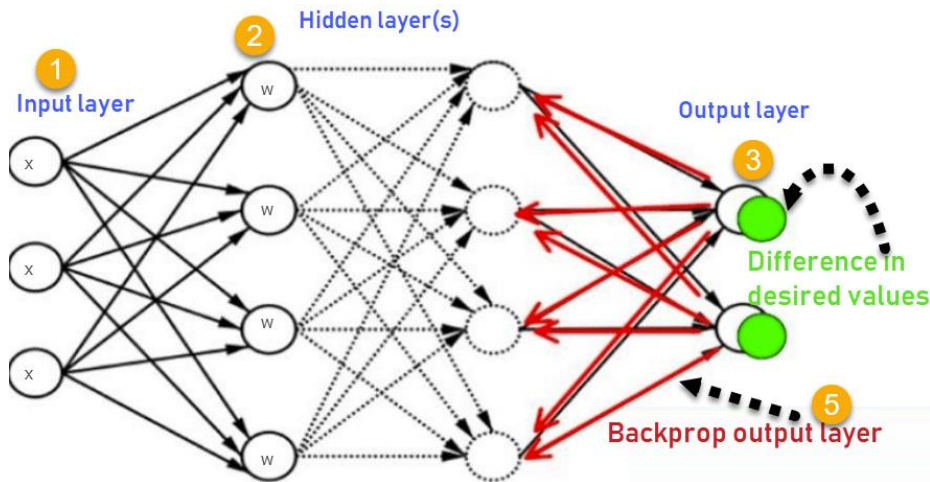
Let us take a look at how back propagation works. It has four layers: input layer, hidden layer, hidden layer II and final output layer.

So, the main three layers are:

1. Input layer
2. Hidden layer

3. Output layer

Each layer has its own way of working and its own way to take action such that we are able to get the desired results and correlate these scenarios to our conditions. Let us discuss other details needed to help summarizing this algorithm.



This image summarizes the functioning of the backpropagation approach.

1. Input layer receives x

2. Input is modeled using weights w

3. Each hidden layer calculates the output and data is ready at the output layer

4. Difference between actual output and desired output is known as the error

5. Go back to the hidden layers and adjust the weights so that this error is reduced in future runs

This process is repeated till we get the desired output. The training phase is done with supervision. Once the model is stable, it is used in production.

Why do we need back propagation?

Back propagation has many advantages, some of the important ones are listed below -

- Back propagation is fast, simple and easy to implement
- There are no parameters to be tuned
- Prior knowledge about the network is not needed thus becoming a flexible method
- This approach works very well in most cases

Feed forward network

Feed forward networks are also called MLN i.e Multi-layered Networks. They are known as feed-forward because the data only travels forward in NN through input node, hidden layer and finally to the output nodes. It is the simplest type of artificial neural network.

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.
- Back propagation algorithm can be quite sensitive to noisy data

Gradient Descent

A gradient measures how much the output of a function changes if you change the inputs a little bit."
—Lex Fridman (MIT)

- The key idea behind the delta rule is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

To understand the delta training rule, consider the task of training a threshold perception. That is, a linear unit for which the output O is given by

$$O = w_0 + w_1x_1 + \dots + w_nx_n$$
$$O(\vec{x}) = (\vec{w} \cdot \vec{x}) \quad \text{equ. (1)}$$

To derive a weight learning rule for linear units, specify a measure for the *training error* of a hypothesis (weight vector), relative to the training examples.

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad \text{equ. (2)}$$

Where,

- D is the set of training examples,
- t_d is the target output for training example d ,
- o_d is the output of the linear unit for training example d
- $E(\vec{w})$ is simply half the squared difference between the target output t_d and the linear unit output o_d , summed over all training examples.

Gradient Descent Error Estimation in 3 dimensional plane

A gradient simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning.

Visualizing the Hypothesis Space

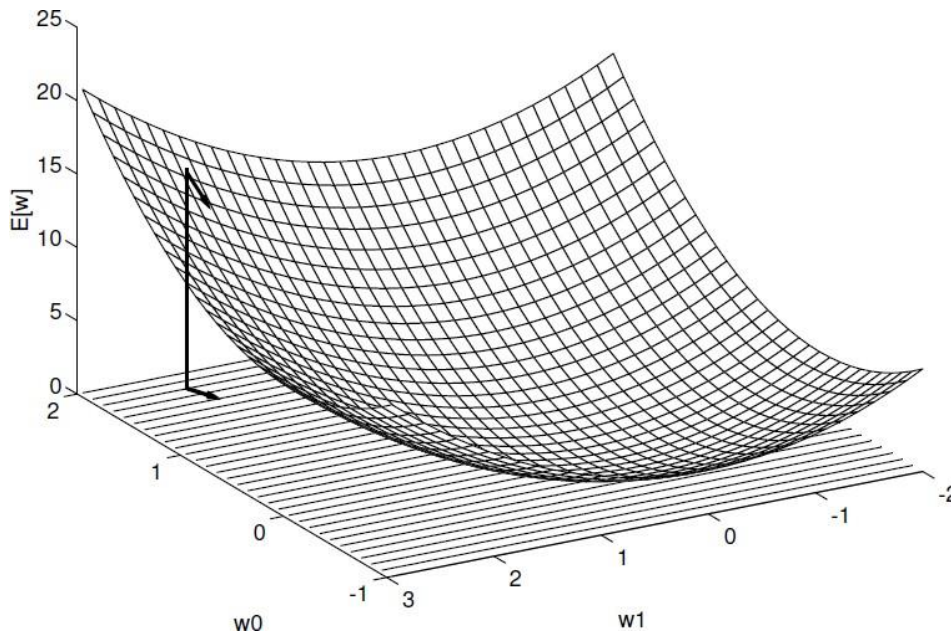
- To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated E values as shown in below figure.
- Here the axes w_0 and w_1 represent possible values for the two weights of a simple linear unit. The w_0, w_1 plane therefore represents the entire hypothesis space.
- The vertical axis indicates the error E relative to some fixed set of training examples.
- The arrow shows the negated gradient at one particular point, indicating the direction in the w_0, w_1 plane producing steepest descent along the error surface.
- The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU,Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956,ISO 9001:2015 Certified



ESTD : 2001



- Given the way in which we chose to define E , for linear units this error surface must always be parabolic with a single global minimum.

Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.

At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

Types of Gradient Descent

There are three popular types of gradient descent that mainly differ in the amount of data they use:

BATCH GRADIENT DESCENT

Batch gradient descent, also called vanilla gradient descent, calculates the error for each example within the training dataset, but only after all training examples have been evaluated does the model get updated. This whole process is like a cycle and it's called a training epoch.

Some advantages of batch gradient descent are its computational efficient, it produces a stable error gradient and a stable convergence. Some disadvantages are the stable error gradient can sometimes result in a state of convergence that isn't the best the model can achieve. It also requires the entire training dataset be in memory and available to the algorithm.

STOCHASTIC GRADIENT DESCENT

By contrast, stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for whole training data set example one by one. Depending on the problem, this can make SGD faster than batch gradient descent.

The frequent updates, however, are more computationally expensive than the batch gradient descent approach.

Additionally, the frequency of those updates can result in noisy gradients, which may cause the error rate to jump around instead of slowly decreasing.

MINI-BATCH GRADIENT DESCENT

Mini-batch gradient descent is the go-to method since it's a combination of the concepts of SGD and batch gradient descent. It simply splits the training dataset into small batches and performs an update for each of those batches.

This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Derivation of the Gradient Descent Rule

How to calculate the direction of steepest descent along the error surface?

The direction of steepest can be found by computing the derivative of E with respect to each component of the vector \vec{w} . This vector derivative is called the gradient of E with respect to \vec{w} , written as

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right] \quad \text{equ. (3)}$$

The gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

Where,

$$\Delta \vec{w} = -\eta \nabla E(\vec{w}) \quad \text{equ. (4)}$$

- Here η is a positive constant called the learning rate, which determines the stepsize in the gradient descent search.
- The negative sign is present because we want to move the weight vector in the direction that decreases E.

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \quad \text{equ. (5)}$$

This training rule can also be written in its component form

Calculate the gradient at each step. The vector derivatives that form the gradient can be obtained by differentiating E from Equation (2), as

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d})\end{aligned}\quad \text{equ. (6)}$$

Substituting Equation (6) into Equation (5) yields the weight update rule for gradient descent

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{i,d} \quad \text{equ. (7)}$$

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
 - Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do
$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$
 - For each linear unit weight w_i , Do
$$w_i \leftarrow w_i + \Delta w_i$$
-

To summarize, the gradient descent algorithm for training linear units is as follows:

- Pick an initial random weight vector.
- Apply the linear unit to all training examples, then compute Δw_i for each weight according to

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \quad \text{equ. (7)}$$

- Update each weight w_i by adding Δw_i , then repeat this process

Issues in Gradient Descent Algorithm

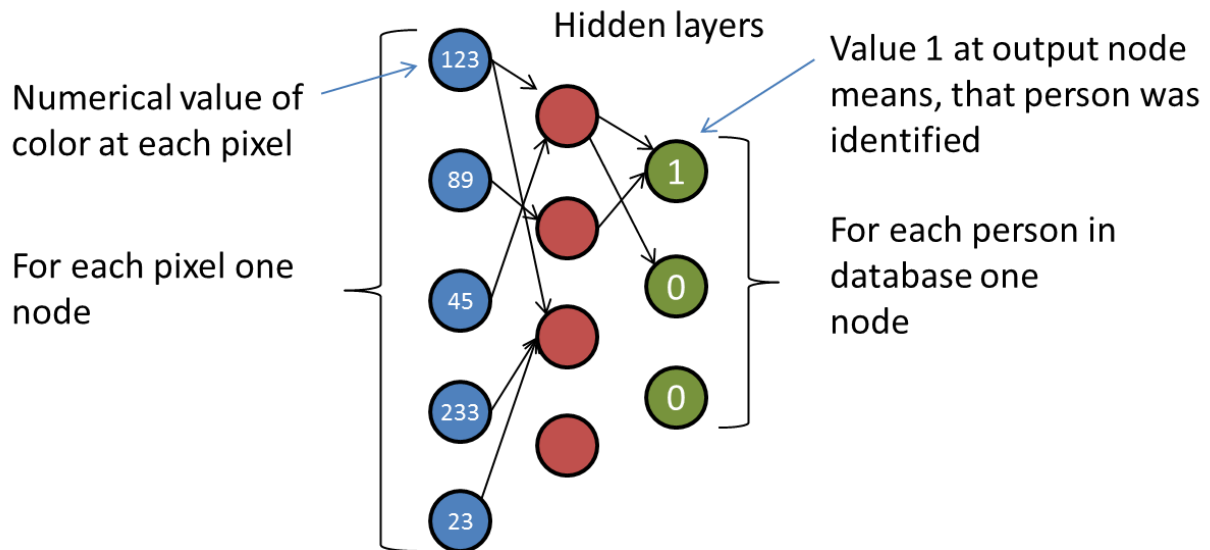
- Can veer off in the wrong direction due to frequent updates.
- Frequent updates are computationally expensive in process due to using all resources for processing one training sample at a time.

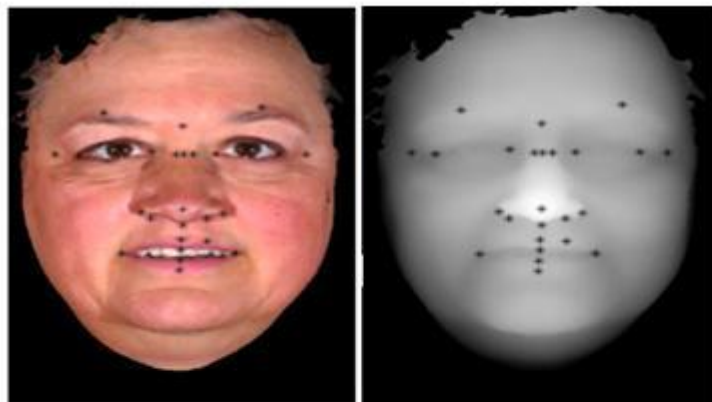
Example of Feed Forward Network and back propagation in Real time: Face recognition

Face recognition using neural network explains about concept of improving performance of detecting face by using neural technology. Fundamental part of face recognition is done through

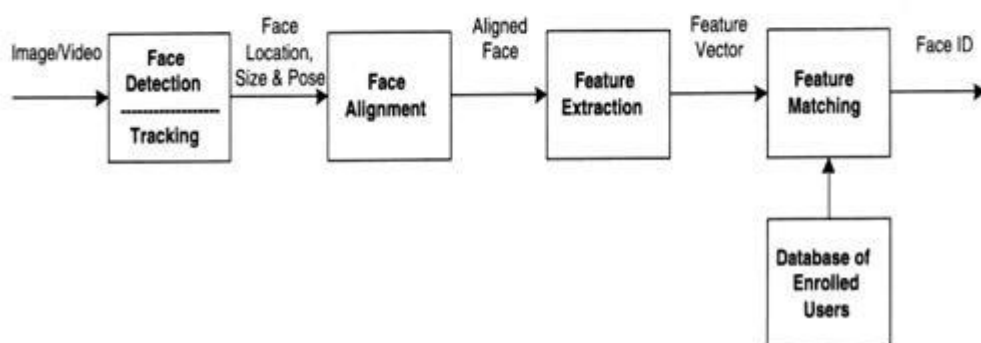
face detection system. Problems with face detection from arbitrary images are due to changes in skin color, quality of image position and orientation.

Different set of multilayer neural network work on detection of face and back propagation algorithm is used for error detection when the face is undetected by machine.





Face recognition processing



Face Recognition Tasks

The task of face recognition is broad and can be tailored to the specific needs of a prediction problem.

For example, in the 1995 paper titled "[Human and machine recognition of faces: A survey](#)," the authors describe three face recognition tasks:

- **Face Matching:** Find the best match for a given face.
- **Face Similarity:** Find faces that are most similar to a given face.
- **Face Transformation:** Generate new faces that are similar to a given face.

With **face detection**, you can get the information you need to perform tasks like embellishing selfies and portraits, or generating avatars from a user's photo.

Key capabilities

- **Recognize and locate facial features** Get the coordinates of the eyes, ears, cheeks, nose, and mouth of every face detected.
- **Get the contours of facial features** Get the contours of detected faces and their eyes, eyebrows, lips, and nose.
- **Recognize facial expressions** Determine whether a person is smiling or has their eyes closed.
- **Track faces across video frames** Get an identifier for each unique detected face. The identifier is consistent across invocations, so you can perform image manipulation on a particular person in a video stream.
- **Process video frames in real time** Face detection is performed on the device, and is fast enough to be used in real-time applications, such as video manipulation.

Face alignment

Face alignment is a **computer vision technology for identifying the geometric structure of human faces in digital images**. Given the location and size of a face, it automatically determines the shape of the face components such as eyes and nose.

Feature extraction refers to **the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set**.

Feature matching refers to **finding corresponding features from two similar images based on a search distance algorithm**. One of the images is considered the source and the other as target, and the feature matching technique is used to either find or derive and transfer attributes from source to target image

Advance Topics in neural network

1. Alternative Error Functions
2. Alternative Error Minimization Procedures
3. Recurrent Networks
4. Dynamically Modifying Network Structure

1.Alternative Error Functions

the basic BACKPROPAGATION algorithm defines E in terms of the sum of squared errors of the network, other definitions have been suggested in order to incorporate other constraints

into the weight-tuning rule. For each new definition of E a new weight-tuning rule for gradient descent must be derived. Examples of alternative definitions of E include a Adding a penalty term for weight magnitude to Adjust Weights to achieve Global minima point

the new penalty weight added is

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

Gamma is constant term and wji is new penalty weight adjusted for error reduction

2. Alternative Error Minimization Procedures

1. Weight-update method

Direction: choosing a direction in which to alter or converge the current weight vector (ex: the gradient in Backpropagation) which depends on Distance : choosing a distance to move (ex: the learning ratio η)

Ex : Line search method, Conjugate gradient method

Line search method is an iterative approach to find a local minimum of a multidimensional nonlinear function using the function's gradients. It computes a search direction and then finds an acceptable step length that Line search method can be categorized into exact and inexact methods.

Gradient descent is computationally efficient, provides a slow rate of convergence. This is where line search comes into place and **provides much better rate of convergence at a slight increase in computational and accuracy**

The conjugate gradient method is a mathematical technique that can be useful for the optimization of both linear and non-linear systems. This technique is generally used as an iterative algorithm, however, it can be used as a direct method, and it will produce a numerical solution. Generally this method is used for very large systems where it is not practical to solve with a direct method of Gradient Decent back propagation.

3.Recurrent Networks

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data which are dynamic in nature. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language

processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate.

Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output within a specific time step .

Types of neural Network

1. One to many network
2. One to one network
3. Many to one network
4. Many to many network

4. Dynamically Modified network structure

Dynamic Neural networks can be considered as the improvement of the static neural networks in which by adding more decision algorithms we can make neural networks learning dynamically from the input and generate better quality results.

Modifying the network structure in hidden layer by adding and pruning the structure for better results and accuracy.

REMARKS ON THE BACKPROPAGATION ALGORITHM

1. Convergence to Local Minima Global Minimum

- The BACKPROPAGATION multilayer networks is only guaranteed to converge toward some local minimum in E and not necessarily to the global minimum error.
- Despite the lack of assured convergence to the global minimum error, BACKPROPAGATION is a highly effective function approximation method in practice.
- To alleviate this convergence we have frequently update the weights
- Can use batch and stochastic gradient decent to improve the efficiency and reduction in error is possible in multilayer network.

2. Representational Power of Feedforward Networks

What set of functions can be represented by feed-forward networks?

The answer depends on the width and depth of the networks. There are three quite general results are known about which function classes can be described by which types of Networks

1. Boolean functions – Every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs
2. Continuous functions – Every bounded continuous function can be approximated with arbitrarily small error by a network with two layers of units
3. Arbitrary functions – Any function can be approximated to arbitrary accuracy by a network with three layers of units.

3. Hypothesis Space Search and Inductive Bias

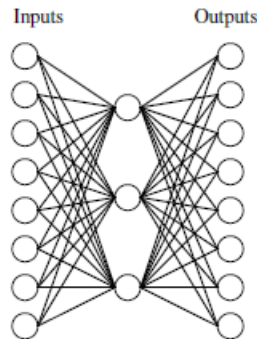
- Hypothesis space is the n -dimensional Euclidean space of the n network weights and hypothesis space is continuous.
- As it is continuous, E is differentiable with respect to the continuous parameters of the hypothesis, results in a well-defined error gradient that provides a very useful structure for organizing the search for the best hypothesis.
- It is difficult to characterize precisely the inductive bias of BACKPROPAGATION algorithm, because it depends on the interplay between the gradient descent and the way in which the weight space are adjusted to achieve global and local minima.
- However, one can roughly characterize it as smooth interpolation between different data nodes between input, output and hidden layer .

4. Hidden Layer Representations

BACKPROPAGATION can define new hidden layer features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function.

Consider example, the network shown in below Figure

A network:



Learned hidden layer representation:

Input	Hidden Values	Output
10000000	→ .89 .04 .08	→ 10000000
01000000	→ .01 .11 .88	→ 01000000
00100000	→ .01 .97 .27	→ 00100000
00010000	→ .99 .97 .71	→ 00010000
00001000	→ .03 .05 .02	→ 00001000
00000100	→ .22 .99 .99	→ 00000100
00000010	→ .80 .01 .98	→ 00000010
00000001	→ .60 .94 .01	→ 00000001

- Consider training the network shown in Figure to learn the simple target function $f(x) = x$, where x is a vector containing seven 0's and a single 1.
- The network must learn to reproduce the eight inputs at the corresponding eight output units. Although this is a simple function, the network in this case is constrained to use only three hidden units. Therefore, the essential information from all eight input units must be captured by the three learned hidden units.
- When BACKPROPAGATION applied to this task, using each of the eight possible vectors as training examples, it successfully learns the target function. By examining the hidden unit values generated by the learned network for each of the eight possible input vectors, it is easy to see that the learned encoding is similar to

the familiar standard binary encoding of eight values using three bits (e.g., 000,001,010, . . . , 111). The exact values of the hidden units for one typical run of shown in Figure.

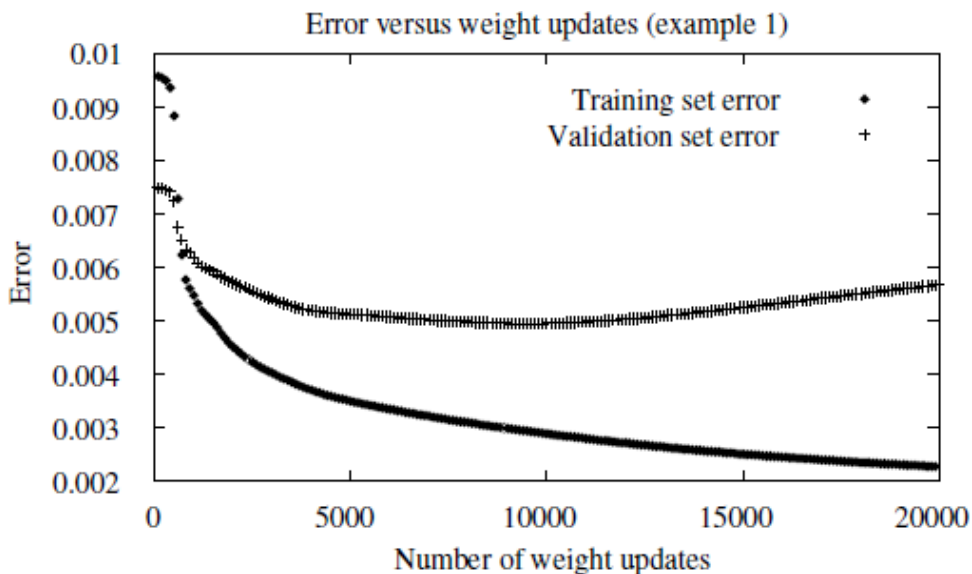
- This ability of multilayer networks to automatically discover useful representations at the hidden layers is a key feature of ANN learning

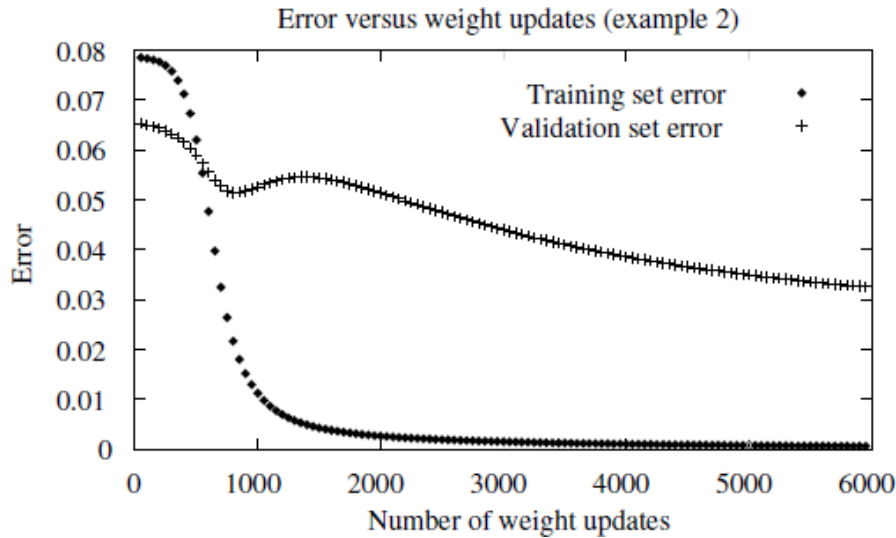
5. Generalization of weights reduces errors, Over fitting data makes the error reduction process complex, and Stopping Criterion when error is optimal when achieved global minima point.

What is an appropriate condition for terminating the weight update loop?

One choice is to continue training until the error E on the training examples falls below some predetermined threshold.

To see the dangers of minimizing the error over the training data, consider how the error E varies with the number of weight iterations





- Consider first the top plot in this figure. The lower of the two lines shows the monotonically decreasing error E over the training set, as the number of gradient descent iterations grows.
- The upper line shows the error E measured over a different validation set of examples, distinct from the training examples. This line measures the generalization accuracy of the network—the accuracy with which it fits examples beyond the training data.
- The generalization accuracy measured over the validation examples first decreases, then increases, even as the error over the training examples continues to decrease.

When the training data set increases over fitting increase the complexity of error and reduction of the error may take n number of iterations to modify the weights which becomes a tedious task.

Estimating Hypothesis Accuracy

A model is constructed based on hypothesis and estimating hypothesis based on accuracy which is best this is based on sample data or additional sample data which is taken into consideration in learning.

Different instances in the model are taken into consideration

Example in h_1 hypothesis is having 100 instances and h_2 hypothesis have 50 instances

The estimation is based on sample data and error frequency in different instances

Accuracy is based on Error Levels in the model

This is made clear by distinguishing between the true error of a model and the estimated or sample error.

- **Sample Error.** Estimate of error calculated on a sample data.
 - The sample error ($error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S , and the quantity $\delta(f(x), h(x))$ is 1 if error is identified

if $f(x) \neq h(x)$, and 0 no error identified.

- **True Error:** Estimation of Error over entire distribution
 - The true error ($error_D(h)$) of hypothesis h with respect to target function f and distribution D , is the probability that h will misclassify an instance drawn at random according to D .

$$error_D(h) \equiv \Pr_{x \in D} [f(x) \neq h(x)]$$

Confidence Intervals for Discrete-Valued Hypotheses

Suppose we wish to estimate the true error for some discrete valued hypothesis h , based on its observed sample error over a sample S , where

- The sample S contains n examples drawn independent of one another, and independent of h , according to the probability distribution D
- $n \geq 30$
- Hypothesis h commits r errors over these n examples (i.e., $error_S(h) = r/n$).

Under these conditions, statistical theory allows to make the following assertions:

1. Given no other information, the most probable value of $error_D(h)$ is $error_S(h)$
2. With approximately **95% probability**, the true error $error_D(h)$ lies in the interval

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Example:

Suppose the data sample S contains $n = 40$ examples and that hypothesis h commits $r = 12$ errors over this data.

- The **sample error** is $error_S(h) = r/n = 12/40 = 0.30$
- Given no other information, **true error** is $error_D(h) = error_S(h)$, i.e., $error_D(h) = 0.30$
- With the 95% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

$$= 0.30 \pm (1.96 * 0.07) \quad = 0.30 \pm 0.14$$

3. A different constant, **Z_N** , is used to calculate the **N% confidence interval**. The general expression for approximate N% confidence intervals for $error_D(h)$ is

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Where,

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

The above equation describes how to calculate the confidence intervals, or error bars, for estimates of error_D (h) that are based on error_S(h)

Example:

Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.

- The **sample error** is error_S(h) = r/n = 12/40 = 0.30
- With the 68% confidence interval estimate for error_D (h).

$$\begin{aligned} \text{error}_S(h) \pm 1.00 \sqrt{\frac{\text{error}_S(h)(1 - \text{error}_S(h))}{n}} \\ = 0.30 \pm (1.00 * 0.07) \\ = 0.30 \pm 0.07 \end{aligned}$$

Basics of Sampling Theory

Sampling theory is the field of statistics that is involved with the collection, analysis and interpretation of data gathered from random samples of a population under study.

The application of sampling theory is concerned not only with the proper selection of observations from the population that will constitute the random sample; it also involves the use of probability theory, along with prior knowledge about the population parameters, to analyze the data from the random sample and develop conclusions from the analysis

- A **random variable** can be viewed as the name of an experiment with a probabilistic outcome. its value is the outcome of the experiment.
- A **probability distribution** for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i , for each possible value y_i .
- The **expected value**, or **mean**, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent $E[Y]$.
- The **variance** of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The **standard deviation** of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used to represent the standard deviation of Y .
- The **Binomial distribution** gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p .
- The **Normal distribution** is a bell-shaped probability distribution that covers many natural phenomena.
- The **Central Limit Theorem** is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An **estimator** is a random variable Y used to estimate some parameter p of an underlying population.
- The **estimation bias** of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A **$N\%$ confidence interval** estimate for parameter p is an interval that includes p with probability $N\%$.

TABLE 5.2
 Basic definitions and facts from statistics.

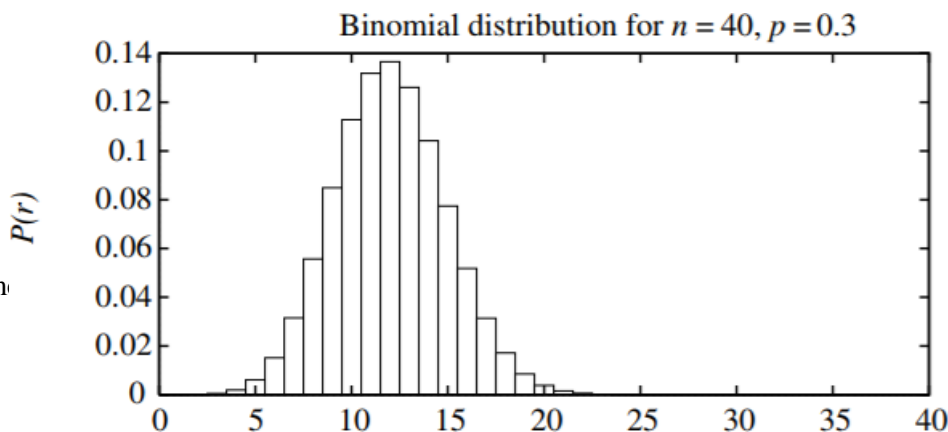
The Binomial Distribution

Consider the following problem for better understanding of Binomial Distribution

- Given a worn and bent coin and estimate the probability that the coin will turn up heads when tossed.
- Unknown probability of heads p . Toss the coin n times and record the number of times r that it turns up heads.

$$\text{Estimate of } p = r / n$$

- If the experiment were *rerun*, generating a new set of n coin tosses, we might expect the number of heads r to vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for p .
- The Binomial distribution describes for each possible value of r (i.e., from 0 to n), the probability of observing exactly r heads given a sample of n independent tosses of a



coin whose true probability of heads is p .

COMPARING LEARNING ALGORITHMS

Consider the case where we have two hypotheses h_1 and h_2 for some discrete-valued target function. Hypothesis h_1 has been tested on a sample S_1 containing n_1 randomly drawn examples, and h_2 has been tested on an independent sample S_2 containing n_2 examples drawn from the same distribution. Suppose we wish to estimate the difference d between the true errors of these two hypotheses.

$$d \equiv \text{error}_{\mathcal{D}}(h_1) - \text{error}_{\mathcal{D}}(h_2)$$

We will use the generic four-step procedure described at the beginning of Section 5.4 to derive a confidence interval estimate for d . Having identified d as the parameter to be estimated, we next define an estimator. The obvious choice for an estimator in this case is the difference between the sample errors, which we denote by \hat{d}

$$\hat{d} \equiv \text{error}_{S_1}(h_1) - \text{error}_{S_2}(h_2)$$

Which are important parameter of hypothesis testing ?

Null hypothesis :- In inferential statistics (make predictions (“inferences”) from that data.), the null hypothesis is a general statement or default position that there is no relationship between two measured phenomena, or no association among groups
In other words it is a basic assumption or made based on domain or problem knowledge.
Ex : a company production is = 50 unit/per day etc.

Alternative hypothesis :-

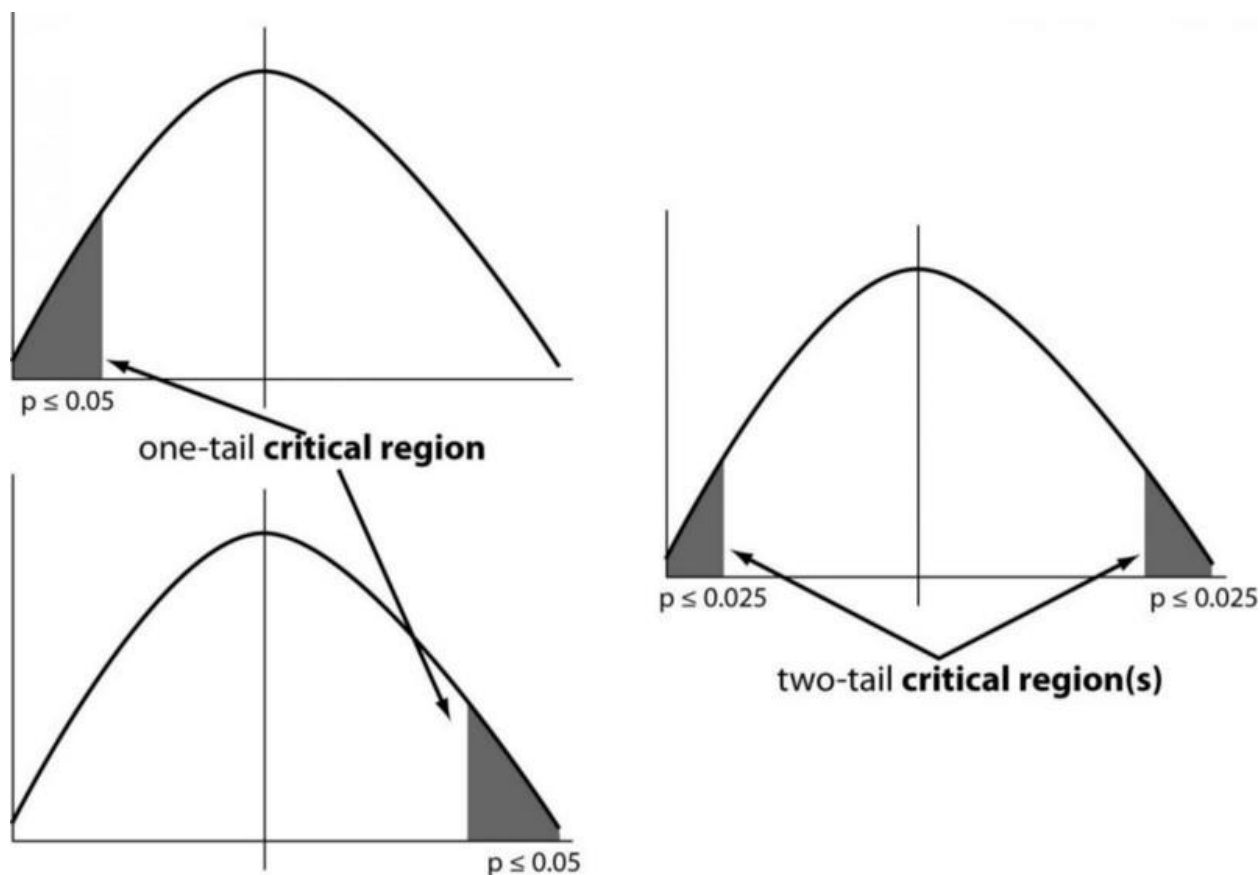
The alternative hypothesis is the hypothesis used in **hypothesis** testing that is contrary to the null hypothesis. It is usually taken to be that the observations are the result of a real effect (with some evidence)

Level of significance: Refers to the degree of significance in which we accept or reject the null-hypothesis. 100% accuracy is not possible for accepting or rejecting a hypothesis, so we therefore select a level of significance that is usually 5%.

This is normally denoted with alpha (maths symbol α) and generally it is 0.05 or 5% , which means your output should be 95% confident to give similar kind of result in each sample.

Type I error: When we reject the null hypothesis, although that hypothesis was true. Type I error is denoted by alpha. In hypothesis testing, the normal curve that shows the critical region is called the alpha region

Type II errors: When we accept the null hypothesis but it is false. Type II errors are denoted by beta. In Hypothesis testing, the normal curve that shows the acceptance region is called the beta region.



One tailed test :- A test of a statistical hypothesis , where the region of rejection is on only **one** side of the sampling distribution , is called a **one-tailed test**.

Two-tailed test :- A **two-tailed test** is a statistical **test** in which the critical area of a distribution is **two-sided** and tests whether a sample is greater than or less than a certain range of values. If the sample being tested falls into either of the critical areas, the alternative hypothesis is accepted instead of the null hypothesis.

Some of widely used hypothesis testing type(not in syllabus)

1. T Test

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU, Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956, ISO 9001:2015 Certified



2. Z Test
3. F- Test
4. ANOVA
5. Chi-Square Test

UNIT 3

Bayes Theorem provides a principled way for calculating a conditional probability.

Bayes Theorem is also widely used in the field of machine learning. Including its use in a probability framework for fitting a model to a training dataset, referred to as maximum a posteriori or MAP for short, and in developing models for classification predictive modeling problems such as the Bayes Optimal Classifier and Naive Bayes.

- **Joint Probability:** Probability of two (or more) simultaneous events, e.g. $P(A \text{ and } B)$ or $P(A, B)$. The conditional probability is the probability of one event given the occurrence of another event, often described in terms of events A and B from two dependent random variables e.g. X and Y.
- **Conditional Probability:** Probability of one (or more) event given the occurrence of another event, e.g. $P(A \text{ given } B)$ or $P(A | B)$.

The joint probability can be calculated using the conditional probability; for example:

- $P(A, B) = P(A | B) * P(B)$

This is called the product rule. Importantly, the joint probability is symmetrical, meaning that:

- $P(A, B) = P(B, A)$

The conditional probability can be calculated using the joint probability; for example:

- $P(A | B) = P(A, B) / P(B)$

An Alternate Way To Calculate Conditional Probability

The conditional probability can be calculated using the other conditional probability; for example:

- $P(A|B) = P(B|A) * P(A) / P(B)$

The reverse is also true; for example:

- $P(B|A) = P(A|B) * P(B) / P(A)$

Bayes theorem is a theorem in probability and statistics, named after the Reverend Thomas Bayes, that helps in determining the probability of an event that is based on some event that has already occurred. Bayes theorem has many applications such as bayesian interference, in the healthcare sector - to determine the chances of developing health problems with an increase in age and many others.

Bayes theorem, in simple words, determines the conditional probability of an event A given that event B has already occurred. Bayes theorem is also known as the Bayes Rule or Bayes Law.

It can be helpful to think about the calculation from these different perspectives and help to map your problem onto the equation.

Firstly, in general, the result $P(A|B)$ is referred to as the **posterior probability** and $P(A)$ is referred to as the **prior probability**.

- $P(A|B)$: Posterior probability.
- $P(A)$: Prior probability.

Sometimes $P(B|A)$ is referred to as the **likelihood** and $P(B)$ is referred to as the **evidence**.

- $P(B|A)$: Likelihood.
- $P(B)$: Evidence.

This allows Bayes Theorem to be restated as:

- Posterior = Likelihood * Prior / Evidence

Bayes' Formula (Conditional Probability)



$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Maximum a Posteriori (MAP) Hypothesis

- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data

D. Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis.

- Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided

$$\begin{aligned}h_{MAP} &= \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)\end{aligned}$$

- $P(D)$ can be dropped, because it is a constant independent of h

Notations

- $P(h)$ prior probability of h , reflects any background knowledge about the chance that h is correct
- $P(D)$ prior probability of D , probability that D will be observed
- $P(D|h)$ probability of observing D given a world in which h holds
- $P(h|D)$ posterior probability of h , reflects confidence that h holds after D has been observed

Maximum Likelihood (ML) Hypothesis

- In some cases, it is assumed that every hypothesis in H is equally probable a priori ($P(h_i) = P(h_j)$ for all h_i and h_j in H).
- In this case the below equation can be simplified and need only consider the term $P(D|h)$ to find the most probable hypothesis.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

the equation can be simplified

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D|h)$$

$P(D|h)$ is often called the likelihood of the data D given h , and any hypothesis that maximizes $P(D|h)$ is called a maximum likelihood (ML) hypothesis

Example

- Consider a medical diagnosis problem in which there are two alternative hypotheses:
 - (1) that the patient has particular form of cancer, and (2) that the patient does not. The available data is from a particular laboratory test with two possible outcomes: + (positive) and - (negative).
- We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease.
- The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result.
- The above situation can be summarized by the following probabilities:

$$\begin{aligned}
 P(cancer) &= .008 & P(\neg cancer) &= 0.992 \\
 P(\oplus|cancer) &= .98 & P(\ominus|cancer) &= .02 \\
 P(\oplus|\neg cancer) &= .03 & P(\ominus|\neg cancer) &= .97
 \end{aligned}$$

Suppose a new patient is observed for whom the lab test returns a positive (+) result. Should we diagnose the patient as having cancer or not?

$$\begin{aligned}
 P(\oplus|cancer)P(cancer) &= (.98).008 = .0078 \\
 P(\oplus|\neg cancer)P(\neg cancer) &= (.03).992 = .0298 \\
 \Rightarrow h_{MAP} &= \neg cancer
 \end{aligned}$$

The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum

$$P(cancer|\oplus) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

Faculty Name : Mrs Swapr $P(\neg cancer|\oplus) = \frac{0.0298}{0.0078 + 0.0298} = 0.79$

Result : The Patient is not Having Cancer

BAYES THEOREM AND CONCEPT LEARNING

What is the relationship between Bayes theorem and the problem of concept learning?

Since Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, and can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable.

Brute-Force Bayes Concept Learning

Consider the concept learning problem

- Assume the learner considers some finite hypothesis space H defined over the instance space X , in which the task is to learn some target concept $c : X \rightarrow \{0,1\}$.
- Learner is given some sequence of training examples $((x_1, d_1) \dots (x_m, d_m))$ where x_i is some instance from X and where d_i is the target value of x_i (i.e., $d_i = c(x_i)$).
- The sequence of target values are written as $D = (d_1 \dots d_m)$.

We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows:

BRUTE-FORCE MAP LEARNING algorithm:

1. For each hypothesis h in H , calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

Faculty Name : Mrs Swapna

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

Subject Name :ML

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for $P(h)$ and for $P(D|h)$?

Let's choose $P(h)$ and for $P(D|h)$ to be consistent with the following assumptions:

- The training data D is noise free (i.e., $d_i = c(x_i)$)
- The target concept c is contained in the hypothesis space H
- Do not have a priori reason to believe that any hypothesis is more probable than any other.

What values should we specify for $P(h)$?

- Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H .
- Assume the target concept is contained in H and require that these prior probabilities sum to 1.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

What choice shall we make for $P(D|h)$?

- $P(D|h)$ is the probability of observing the target values $D = (d_1 \dots d_m)$ for the fixed set of instances $(x_1 \dots x_m)$, given a world in which hypothesis h holds
- Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Given these choices for $P(h)$ and for $P(D|h)$ we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm.

Recalling Bayes theorem, we have

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Consider the case where h is inconsistent with the training data D

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

The posterior probability of a hypothesis inconsistent with D is zero

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|V S_{H,D}|}{|H|}} = \frac{1}{|V S_{H,D}|}$$

Consider the case where h is consistent with D

Where, $V S_{H,D}$ is the subset of hypotheses from H that are consistent with D

To summarize, Bayes theorem implies that the posterior probability $P(h|D)$ under our assumed $P(h)$ and $P(D|h)$ is

$$P(D|h) = \begin{cases} \frac{1}{|V S_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

Consider the problem of learning a *continuous-valued target function* such as neural network learning, linear regression, and polynomial curve fitting

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a *maximum likelihood (ML) hypothesis*

- Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X , i.e., $(\forall h \in H)[h : X \rightarrow \mathbb{R}]$ and training examples of the form $\langle x_i, d_i \rangle$
- The problem faced by L is to learn an unknown target function $f : X \rightarrow \mathbb{R}$
- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean ($d_i = f(x_i) + e_i$)
- Each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
 - Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.
 - It is assumed that the values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean.
- The task of the learner is to *output a maximum likelihood hypothesis* or a *MAP hypothesis assuming all hypotheses are equally probable a priori*.

Using the definition of h_{ML} we have

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} p(D|h)$$

Assuming training examples are mutually independent given h , we can write $P(D|h)$ as the product of the various $(d_i|h)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^m p(d_i|h)$$

Given the noise e_i obeys a Normal distribution with zero mean and unknown variance σ^2 , each d_i must also obey a Normal distribution around the true target value $f(x_i)$. Because we are writing the expression for $P(D|h)$, we assume h is the correct description of f .

Normal Probability Density Function

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

Hence, $\mu = f(x_i) = h(x_i)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2}$$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}$$

Maximize the less complicated logarithm, which is justified because of the monotonicity of function p

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^m \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^m -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^m \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, discard constants that are independent of h .

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^m (d_i - h(x_i))^2$$

Thus, above equation shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values d_i and the hypothesis predictions $h(x_i)$

MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

- Consider the setting in which we wish to learn a nondeterministic (probabilistic) function $f : X \rightarrow \{0, 1\}$, which has two discrete output values.
- We want a function approximator whose output is the probability that $f(x) = 1$. In other words, learn the target function $f' : X \rightarrow [0, 1]$ such that $f'(x) = P(f(x) = 1)$

How can we learn f' using a neural network?

- Use of brute force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x .

What criterion should we optimize in order to find a maximum likelihood hypothesis for f' in this setting?

- First obtain an expression for $P(D|h)$
- Assume the training data D is of the form $D = \{(x_1, d_1) \dots (x_m, d_m)\}$, where d_i is the observed 0 or 1 value for $f(x_i)$.
- Both x_i and d_i as random variables, and assuming that each training example is drawn independently, we can write $P(D|h)$ as

$$P(D|h) = \prod_{i=1}^m P(x_i, d_i | h) \quad \text{equ (1)}$$

Applying
 the product rule

$$P(D | h) = \prod_{i=1}^m P(d_i | h, x_i) P(x_i) \quad \text{equ (2)}$$

The probability $P(d_i|h, x_i)$

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \quad \text{equ (3)}$$

Re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \quad \text{equ (4)}$$

Equation (4) to substitute for $P(d_i | h, x_i)$ in Equation (5) to obtain

$$P(D|h) = \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i) \quad \text{equ (5)}$$

We write an expression for the maximum likelihood hypothesis

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h , so it can be dropped

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \quad \text{equ (6)}$$

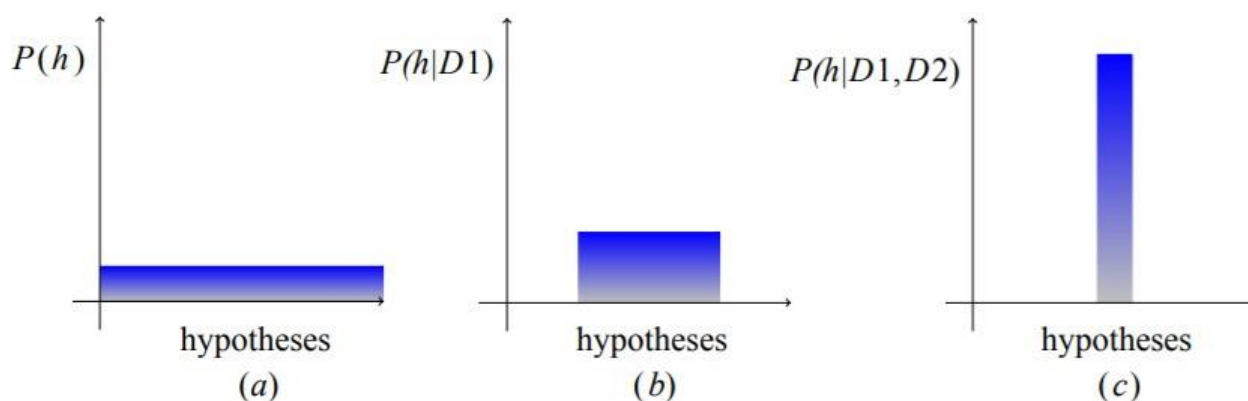
It easier to work with the log of the likelihood, yielding

$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \quad \text{equ (7)}$$

Equation (7) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting

The Evolution of Probabilities Associated with Hypotheses

- Figure (a) all hypotheses have the same probability.
- Figures (b) and (c), As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to 1 is shared equally among the remaining consistent hypotheses.



MAP Hypotheses and Consistent Learners

- A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.
- Every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H ($P(h_i) = P(h_j)$ for all i, j), and deterministic, noise free training data ($P(D|h) = 1$ if D and h are consistent, and 0 otherwise).

Example:

- FIND-S outputs a consistent hypothesis, it will output a MAP hypothesis under the probability distributions $P(h)$ and $P(D|h)$ defined above.
- Are there other probability distributions for $P(h)$ and $P(D|h)$ under which FIND-S outputs MAP hypotheses? Yes.

- Because FIND-S outputs a maximally specific hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favors more specific hypotheses.

Note

- Bayesian framework is a way to characterize the behavior of learning algorithms
- By identifying probability distributions $P(h)$ and $P(D|h)$ under which the output is a optimal hypothesis, implicit assumptions of the algorithm can be characterized (Inductive Bias)

MINIMUM DESCRIPTION LENGTH PRINCIPLE

- A Bayesian perspective on Occam's razor
- Motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory.

$$h_{MAP} = \underset{h \in H}{argmax} P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the \log_2

$$h_{MAP} = \underset{h \in H}{argmax} \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{argmin} -\log_2 P(D|h) - \log_2 P(h) \quad \text{equ (1)}$$

This equation (1) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data

- $-\log_2 P(h)$: the description length of h under the optimal encoding for the hypothesis space H , $L_{CH}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H .
- $-\log_2 P(D|h)$: the description length of the training data D given hypothesis

h , under the optimal encoding from the hypothesis space H : $L_{C_H}(D|h) = -\log_2 P(D|h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h .

- Rewrite Equation (1) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Where, C_H and $C_{D|h}$ are the optimal encodings for H and for D given h

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths of equ.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle:

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$$

Where, codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H , and if we choose C_2 to be the optimal encoding $C_{D|h}$, then

$$h_{MDL} = h_{MAP}$$

Bayes Optimal Classifier

It is described using the Bayes Theorem that provides a principled way for calculating a conditional probability. It is also closely related to the Maximum a Posteriori: a probabilistic framework referred to as MAP that finds the most probable hypothesis for a training dataset.

In practice, the Bayes Optimal Classifier is computationally expensive, if not intractable to calculate, and instead, simplifications such as the Gibbs algorithm and Naive Bayes can be used to approximate the outcome.

- Bayes Theorem provides a principled way for calculating conditional probabilities, called a posterior probability.
- Maximum a Posteriori is a probabilistic framework that finds the most probable hypothesis that describes the training dataset.
- Bayes Optimal Classifier is a probabilistic model that finds the most probable prediction using the training data and space of hypotheses to make a prediction for a new data instance.

To develop some intuitions consider a hypothesis space containing three hypotheses, h_1 , h_2 , and h_3 . Suppose that the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus, h_1 is the MAP hypothesis. Suppose a new instance x is encountered, which is classified positive by h_1 , but negative by h_2 and h_3 .

Taking all hypotheses into account, the probability that x is positive is .4 (the probability associated with h_1), and the probability that it is negative is therefore .6.

The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis. In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities.

If the possible classification of the new example can take on any value v_j from some set V , then the probability $P(v_j|D)$ that the correct classification for the new instance is v_j , is just

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

The optimal classification of the new instance is the value v_j , for which $P(v_j|D)$ is maximum.

Bayes optimal classification:

$$\operatorname{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \quad (6.18)$$

To illustrate in terms of the above example, the set of possible classifications of the new instance is $V = \{\oplus, \ominus\}$, and

$$\begin{aligned} P(h_1|D) &= .4, P(\ominus|h_1) = 0, P(\oplus|h_1) = 1 \\ P(h_2|D) &= .3, P(\ominus|h_2) = 1, P(\oplus|h_2) = 0 \\ P(h_3|D) &= .3, P(\ominus|h_3) = 1, P(\oplus|h_3) = 0 \end{aligned}$$

therefore

$$\begin{aligned} \sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) &= .4 \\ \sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) &= .6 \end{aligned}$$

and

$$\operatorname{argmax}_{v_j \in \{\oplus, \ominus\}} \sum_{h_i \in H} P_j(v_j|h_i)P(h_i|D) = \ominus$$

Any system that classifies new instances according to Equation (6.18) is called a *Bayes optimal classifier*, or Bayes optimal learner. No other classification method using the same hypothesis space and same prior knowledge can outperform this method on average. This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

Two of the most commonly used simplifications use a sampling algorithm for hypotheses, such as Gibbs sampling, or to use the simplifying assumptions of the Naive Bayes classifier.

1.Gibbs Algorithm. Randomly sample hypotheses biased on their posterior probability.

2.Naive Bayes. Assume that variables in the input data are conditionally independent.

1.Gibbs Algorithm

Gibbs sampling (also called alternating conditional sampling) is a Markov Chain Monte Carlo algorithm for high-dimensional data such as image processing and micro arrays.

It is called Monte Carlo because it draws samples from specified probability distributions the Markov chain comes from the fact that each sample is dependent on the previous sample. **Gibbs sampling is relatively easy to implement. However, it is less efficient than direct simulation from the distribution.**

An alternative, less optimal method is the Gibbs algorithm defined as follows:

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H .
2. Use h to predict the classification of the next instance x .

Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution. Surprisingly, it can be shown that under certain **conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier**

2.Naïve Bayes:

Assume that variables in the input data are conditionally independent.

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.**
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles.**

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes

11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	4

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	2	3	$5/14 = 0.35$
All	$4/14 = 0.29$	$10/14 = 0.71$	

Applying Bayes' theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{NO}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$$

So as we can see from the above calculation that $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

Hence on a Sunny day, Player can play the game.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Navie Bayers Text Classification:

$$v_{NB} = \underset{v_j \in \{\text{like}, \text{dislike}\}}{\operatorname{argmax}} P(v_j) \prod_{i=1}^n P(a_i | v_j)$$

$P(v_j)$ is the prior probability given the data belongs to that class

example : Target class : the data belongs to the class (y or no) or (like or Dislike)

$P(a|v_j)$ = Conditional probability or likelihood

ularity. Thus, the estimate for $P(w_k|v_j)$ will be

$$\frac{n_k + 1}{n + |Vocabulary|}$$

where n is the total number of word positions in all training examples whose target value is v_j , n_k is the number of times word w_k is found among these n word positions, and $|Vocabulary|$ is the total number of distinct words (and other tokens) found within the training data.

	docID	words in document	in c = China?
Training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$P(c) = \frac{3}{4} \quad P(\bar{c}) = \frac{1}{4}$$

$$P(\text{Chinese}|c) = \frac{(5 + 1)}{(8 + 6)} = \frac{6}{14} = \frac{3}{7} \quad P(\text{Toyko}|c) = P(\text{Japan}|c) = \frac{(0 + 1)}{(8 + 6)} = \frac{1}{14}$$

$$P(\text{Chinese}|\bar{c}) = \frac{(1 + 1)}{(3 + 6)} = \frac{2}{9} \quad P(\text{Toyko}|\bar{c}) = P(\text{Japan}|\bar{c}) = \frac{(1 + 1)}{(3 + 6)} = \frac{2}{9}$$

8=Total number of words in 3 docid 1,2, 3=8

6=Total number of unique words count like Chinese Beijing shanghai Tokyo japan maco

3= total number of word in doc id 4

$$P(c) = \frac{3}{4} \quad P(\bar{c}) = \frac{1}{4}$$

$$P(\text{Chinese}|c) = \frac{(5+1)}{(8+6)} = \frac{6}{14} = \frac{3}{7} \quad P(\text{Toyko}|c) = P(\text{Japan}|c) = \frac{(0+1)}{(8+6)} = \frac{1}{14}$$

$$P(\text{Chinese}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9} \quad P(\text{Toyko}|\bar{c}) = P(\text{Japan}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9}$$

$$P(c|d_5) \propto \frac{3}{4} \cdot \left(\frac{3}{7}\right)^3 \cdot \frac{1}{14} \cdot \frac{1}{14} \approx 0.0003$$

$$P(\bar{c}|d_5) \propto \frac{1}{4} \cdot \left(\frac{2}{9}\right)^3 \cdot \frac{2}{9} \cdot \frac{2}{9} \approx 0.0001$$

The classifier assigns the test document to $c = \text{China}$

Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network**, **belief network**, **decision network**, or **Bayesian model**.

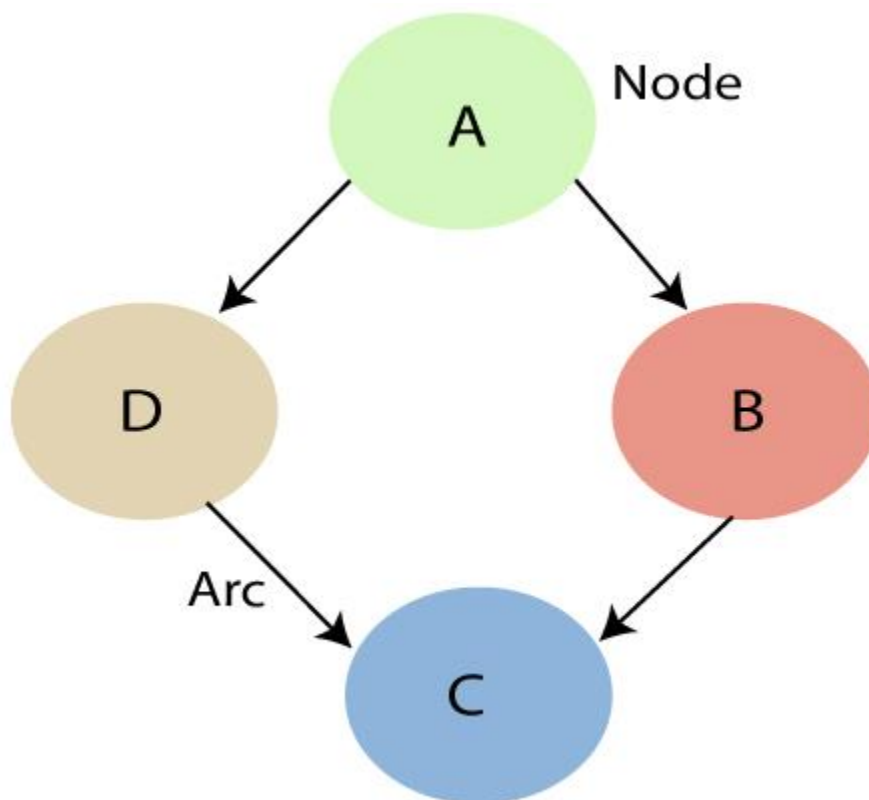
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction**, **anomaly detection**, **diagnostics**, **automated insight**, **reasoning**, **time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the

graph.

These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other

- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution $P(X_i | \text{Parent}(X_i))$, which determines the effect of the parent on that node.

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

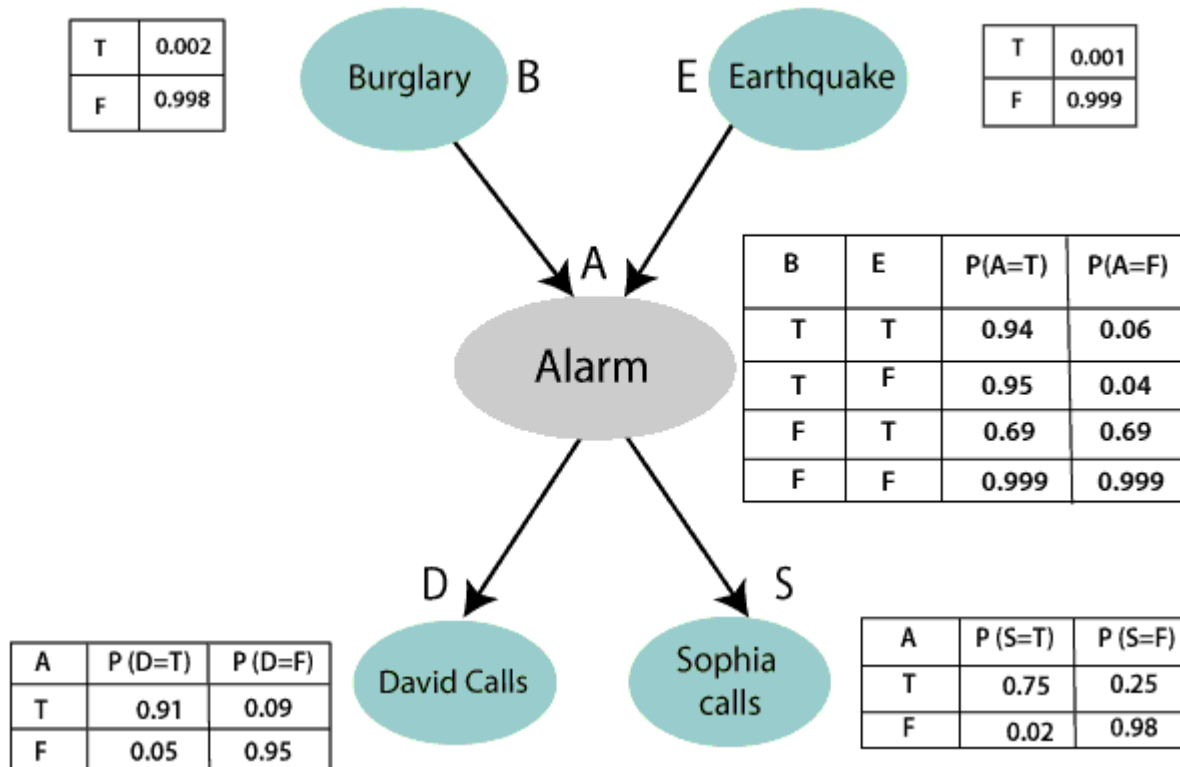
Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table

List of all events occurring in this network:

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**



$P(B = \text{True}) = 0.002$, which is the probability of burglary.

$P(B = \text{False}) = 0.998$, which is the probability of no burglary.

$P(E = \text{True}) = 0.001$, which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25

False	0.02	0.98
-------	------	------

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(\neg B) * P(\neg E).$$

$$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$$

$$= 0.00068045.$$

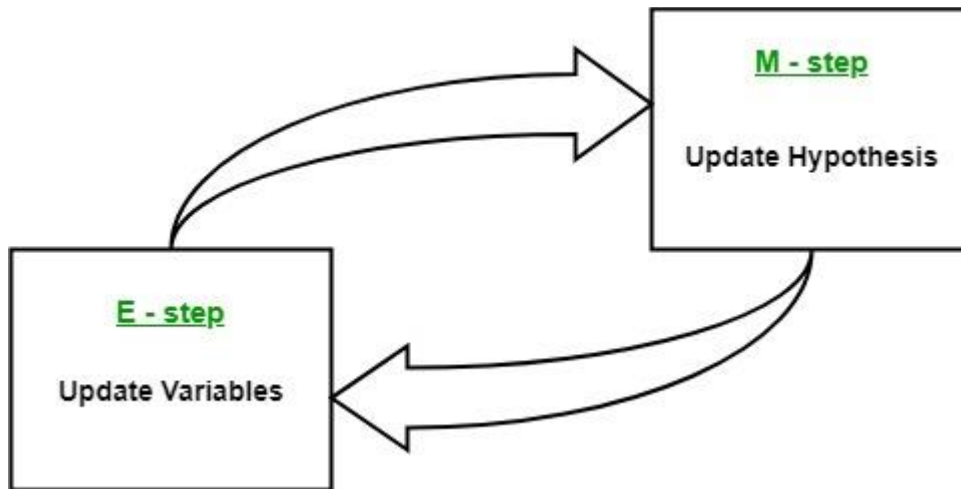
Expectation-Maximization Algorithm

Expectation-Maximization algorithm can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

Algorithm:

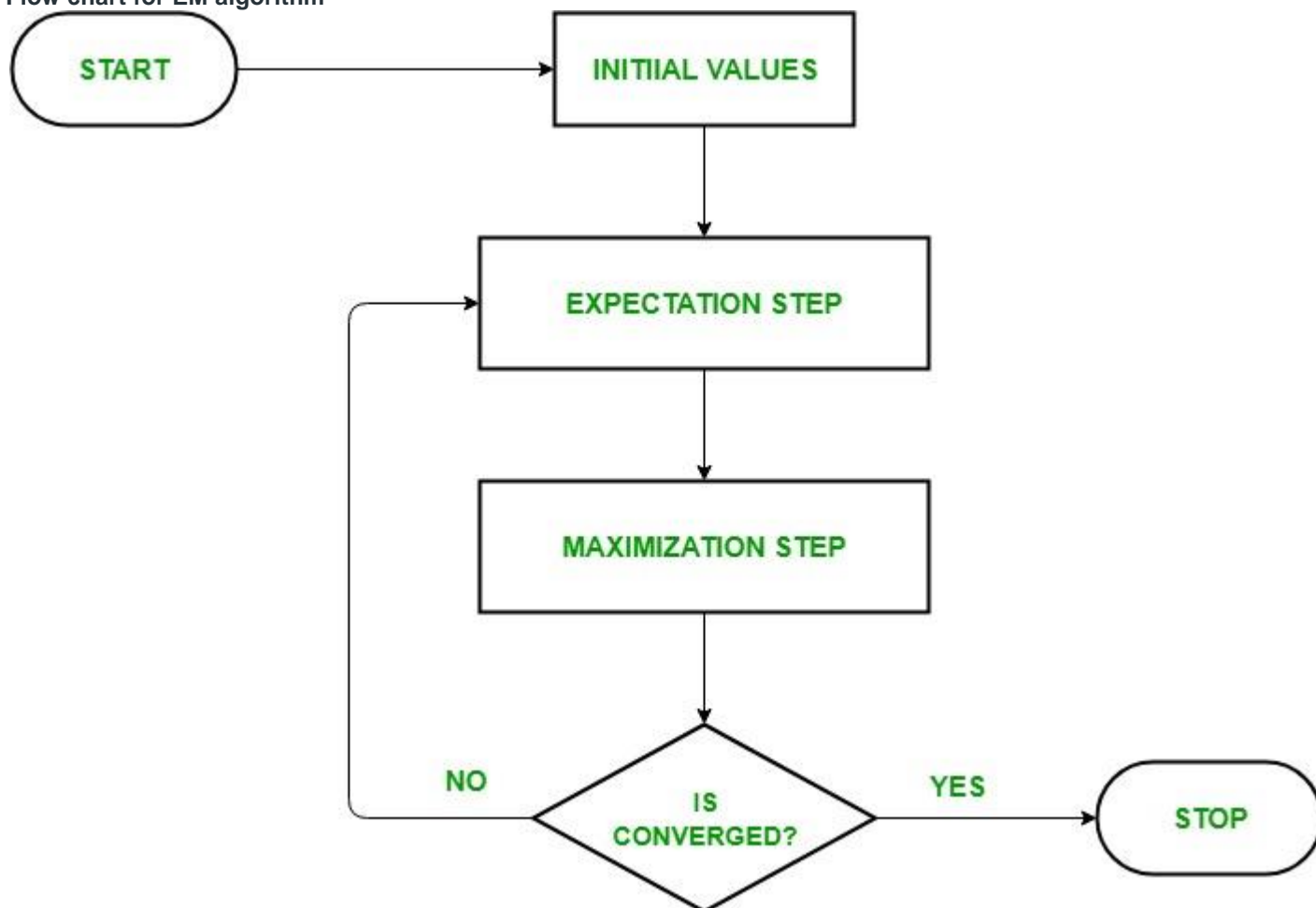
1. Given a set of incomplete data, consider a set of starting parameters.
2. **Expectation step (E – step):** Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. **Maximization step (M – step):** Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.



The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

- Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.
- The next step is known as “Expectation” – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.
- The next step is known as “Maximization”-step or *M-step*. In this step, we use the complete data generated in the preceding “Expectation” – step in order to update the values of the parameters. It is basically used to update the hypothesis.
- Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. “Expectation” – step and “Maximization” – step until the convergence occurs.

Flow chart for EM algorithm –



Usage of EM algorithm –

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm –

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.

Disadvantages of EM algorithm –

- It has slow convergence.
- It makes convergence to the local optima only.

Probably Approximately Correct (PAC) framework, we identify classes of hypotheses that can and cannot be learned from a polynomial number of training examples and we define a natural measure of complexity for hypothesis spaces that allows bounding the number of training examples required for learning. Within the mistake bound framework, we examine the number of training errors that will be made by a learner before it determines the correct hypothesis

we will be chiefly concerned with questions such as how many training examples are sufficient to successfully learn the target function, and how many mistakes will the learner make before succeeding. As we shall see, it is possible to set quantitative bounds on these measures, depending on attributes of the learning problem such as:

- 1) the size or complexity of the hypothesis space considered by the learner
- 2) the accuracy to which the target concept must be approximated
- 3) the probability that the learner will output a successful hypothesis
- 4) the manner in which training examples are presented to the learner

Goal of PAC is to answer questions such as:

1. Sample complexity. How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?
2. Computational complexity. How much computational effort is needed for a learner to converge (with high probability) to a successful hypothesis?
3. Mistake bound. How many training examples will the learner misclassify before converging to a successful hypothesis?

Error of a Hypothesis

Accuracy is based on Error Levels in the model

This is made clear by distinguishing between the true error of a model and the estimated or sample error.

- **Sample Error.** Estimate of error calculated on a sample data.
 - The sample error ($error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S , and the quantity $\delta(f(x), h(x))$ is 1 if error is identified

if $f(x) \neq h(x)$, and 0 no error identified.

Suppose the data sample S contains $n = 40$ examples and that hypothesis h commits $r = 12$ errors misclassify or mismatch over this data.

- The **sample error** is $error_S(h) = r/n = 12/40 = 0.30$

- **True Error:** Estimation of Error over entire distribution

- The true error ($error_D(h)$) of hypothesis h with respect to target function f and distribution D , is the probability that h will misclassify an instance drawn at random according to D .

$$error_D(h) \equiv \Pr_{x \in D} [f(x) \neq h(x)]$$

refer to unit 2 calculation of true error through confidence intervals

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Confidence Intervals for Discrete-Valued Hypotheses

Suppose we wish to estimate the true error for some discrete valued hypothesis h , based on its observed sample error over a sample S , where

- The sample S contains n examples drawn independent of one another, and independent of h , according to the probability distribution D
- $n \geq 30$
- Hypothesis h commits r errors over these n examples (i.e., $error_S(h) = r/n$).

Under these conditions, statistical theory allows to make the following assertions:

2. Given no other information, the most probable value of $error_D(h)$ is $error_S(h)$
3. With approximately **95% probability**, the true error $error_D(h)$ lies in the interval

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Example:

Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.

- The **sample error** is $error_S(h) = r/n = 12/40 = 0.30$
- Given no other information, **true error** is $error_D(h) = error_S(h)$, i.e., $error_D(h) = 0.30$
- With the 95% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

$$= 0.30 \pm (1.96 * 0.07) \quad = 0.30 \pm 0.14$$

True Error vs Sample Error

True Error

The true error represents the probability that a random sample from the population is misclassified.

True error is used to estimate the error of the population.

True error is difficult to calculate. It is estimated by the confidence interval range on the basis of Sample error.

The true error can be caused by poor data collection methods,

Sample Error

Sample Error represents the fraction of the sample which is misclassified.

Sample Error is used to estimate the errors of the sample.

Sample Error is easy to calculate. You just have to calculate the fraction of the sample that is misclassified.

Sampling error can be of type population-specific error (wrong people to survey), selection error, sample-frame

True Error

selection bias, or non-response bias.

Sample Error

error (wrong frame window selected for sample), and non-response error (when respondent failed to respond).

PROBABLY LEARNING AN APPROXIMATELY CORRECT HYPOTHESIS

We begin by specifying the problem setting that defines the PAC learning model, then consider the questions of how many training examples and how much computation are required in order to learn various classes of target functions within this PAC model.

PAC-learnability is largely determined by the number of training examples required by the learner. The growth in the number of required training examples with problem size, called the sample complexity of the learning problem

a general bound on the sample complexity for a very broad class of learners, called consistent learners. A learner is consistent if it outputs hypotheses that perfectly fit the training data.

The learner L considers some set H of possible hypotheses when attempting to learn the target concept. For example, H might be the set of all hypotheses describable by conjunctions of the attributes. After observing a sequence of training examples of the target concept c , L must output some hypothesis h from H , which is its estimate of c .

To be fair, we evaluate the success of L by the performance of h over new instances drawn randomly from X according to D (Training data), the same probability distribution used to generate the training data

Error of a Hypothesis

Figure 7.1 shows this definition of error in graphical form. The concepts c and h are depicted by the sets of instances within X that they label as positive. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree

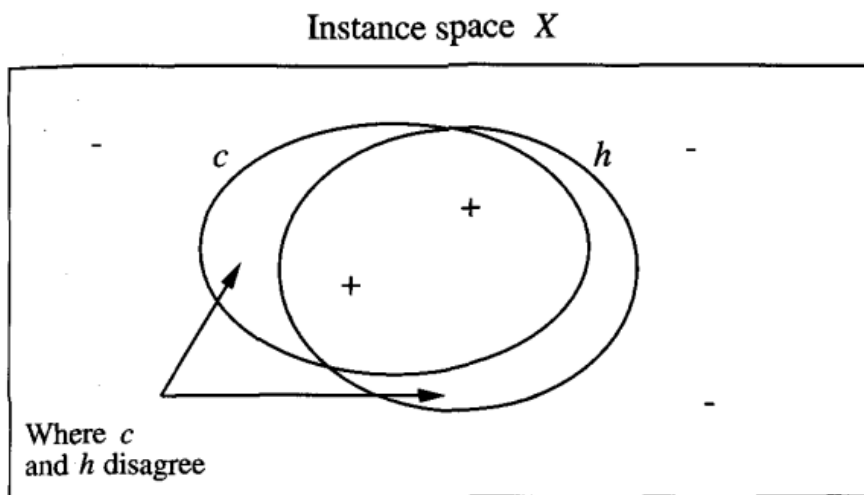


FIGURE 7.1

The error of hypothesis h with respect to target concept c . The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree on its classification. The $+$ and $-$ points indicate positive and negative training examples. Note h has a nonzero error with respect to c despite the fact that h and c agree on all five training examples observed thus far.

PAC Learnability

Our aim is to characterize classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation

First, unless we provide training examples corresponding to every possible instance in X (an unrealistic assumption), there may be multiple hypotheses consistent with the provided training examples, and the learner cannot be certain to pick the one corresponding to the target concept. Second, given that the training examples are drawn randomly, there will always be some nonzero probability that the training examples encountered by the learner will be misleading

To accommodate these two difficulties, we weaken our demands on the learner in two ways. First, we will not require that the learner output a zero error hypothesis—we will require only that its error be bounded by some constant, ϵ , that can be made arbitrarily small.

Definition: Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H . C is **PAC-learnable** by L using H if for all $c \in C$, distributions \mathcal{D} over X , ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n , and $size(c)$.

Our definition requires two things from L . First, L must, with arbitrarily high probability $(1 - \delta)$, output a hypothesis having arbitrarily low error (ϵ). Second, it must do so efficiently—in time that grows at most polynomially with $1/\epsilon$ and $1/\delta$, which define the strength of our demands on the output hypothesis, and with n and $size(c)$ that define the inherent complexity of the underlying instance space X and concept class C . Here, n is the size of instances in X . For example, if instances in X are conjunctions of k boolean features, then $n = k$. The second space parameter, $size(c)$, is the encoding length of c in C , assuming some representation for C . For example, if concepts in C are conjunctions of up to k boolean features, each described by listing the indices of the features in the conjunction, then $size(c)$ is the number of boolean features actually used to describe c .

SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACES

The growth in the number of required training examples with problem size, called the sample complexity of the learning problem

we present a general bound on the sample complexity for a very broad class of learners, called consistent learners. A learner is consistent if it outputs hypotheses that perfectly fit the training data

Find s Algorithm –biased Hypothesis

Candidate Algorithm –Unbiased hypothesis Restricted hypothesis

DecisonTree – preference Bias

The significance of the version space here is that every consistent learner outputs a hypothesis belonging to the version space, regardless of the instance space X , hypothesis space H , or training data D . The reason is simply that by definition the version space VSH,D contains every

consistent hypothesis in H . Therefore, to bound the number of examples needed by any consistent learner

Definition: Consider a hypothesis space H , target concept c , instance distribution \mathcal{D} , and set of training examples D of c . The version space $VS_{H,D}$ is said to be ϵ -exhausted with respect to c and \mathcal{D} , if every hypothesis h in $VS_{H,D}$ has error less than ϵ with respect to c and \mathcal{D} .

$$(\forall h \in VS_{H,D}) \text{error}_{\mathcal{D}}(h) < \epsilon$$

This definition is illustrated in Figure 7.2. The version space is ϵ -exhausted just in the case that all the hypotheses consistent with the observed training examples (i.e., those with zero training error) happen to have true error less than ϵ . Of course from the learner's viewpoint all that can be known is that these hypotheses fit the training data equally well—they all have zero training error. Only an observer who knew the identity of the target concept could determine with certainty whether the version space is ϵ -exhausted. Surprisingly, a probabilistic argument allows us to bound the probability that the version space will be ϵ -exhausted after a given number of training examples, even without knowing the identity of the target concept or the distribution from which training examples

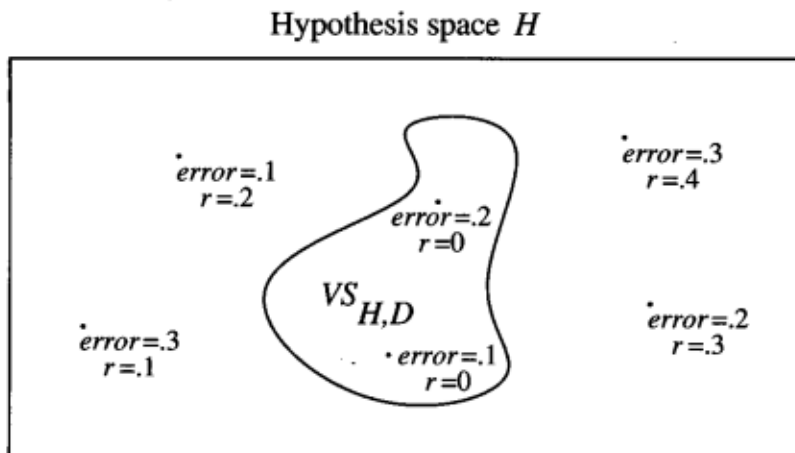


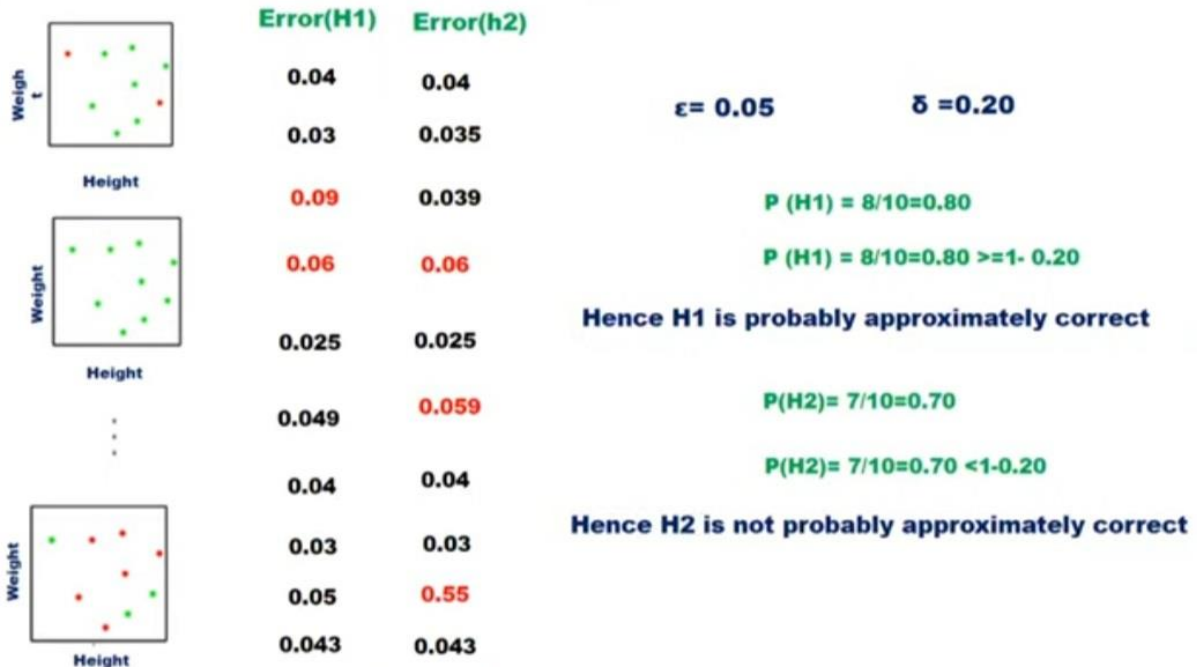
FIGURE 7.2

Exhausting the version space. The version space $VS_{H,D}$ is the subset of hypotheses $h \in H$, which have zero training error (denoted by $r = 0$ in the figure). Of course the true $\text{error}_{\mathcal{D}}(h)$ (denoted by error in the figure) may be nonzero, even for hypotheses that commit zero errors over the training data. The version space is said to be ϵ -exhausted when all hypotheses h remaining in $VS_{H,D}$ have $\text{error}_{\mathcal{D}}(h) < \epsilon$.

ϵ and δ parameters

ϵ gives an upper bound on the error in accuracy with which h approximated (accuracy: $1 - \epsilon$)

δ gives the probability of failure in achieving this accuracy (confidence : $1 - \delta$)



Theorem 7.1. ϵ -exhausting the version space. If the hypothesis space H is finite, and D is a sequence of $m \geq 1$ independent randomly drawn examples of some target concept c , then for any $0 \leq \epsilon \leq 1$, the probability that the version space $VS_{H,D}$ is not ϵ -exhausted (with respect to c) is less than or equal to

$$|H|e^{-\epsilon m}$$

Proof. Let h_1, h_2, \dots, h_k be all the hypotheses in H that have true error greater than ϵ with respect to c . We fail to ϵ -exhaust the version space if and only if at least one of these k hypotheses happens to be consistent with all m independent random training examples. The probability that any single hypothesis having true error greater than ϵ would be consistent with one randomly drawn example is at most $(1 - \epsilon)$. Therefore the probability that this hypothesis will be consistent with m independently drawn examples is at most $(1 - \epsilon)^m$. Given that we have k hypotheses with error greater than ϵ , the probability that at least one of these will be consistent with all m training examples is at most

$$k(1 - \epsilon)^m$$

And since $k \leq |H|$, this is at most $|H|(1 - \epsilon)^m$. Finally, we use a general inequality stating that if $0 \leq \epsilon \leq 1$ then $(1 - \epsilon) \leq e^{-\epsilon}$. Thus,

$$k(1 - \epsilon)^m \leq |H|(1 - \epsilon)^m \leq |H|e^{-\epsilon m}$$

which proves the theorem. □

SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

Sample Complexity Results for Infinite Hypothesis Spaces can be explained with concept of shattering coefficient

The Shattering Coefficient Let C be a concept class over an instance space X , i.e. a set of functions from X to $\{0, 1\}$ (where both C and X may be infinite).

Shattering a Set of Instances

To make this notion more precise, we first define the notion of *shattering* a set of instances. Consider some subset of instances $S \subseteq X$. For example, Figure 7.3 shows a subset of three instances from X . Each hypothesis h from H imposes some dichotomy on S ; that is, h partitions S into the two subsets $\{x \in S | h(x) = 1\}$ and $\{x \in S | h(x) = 0\}$. Given some instance set S , there are $2^{|S|}$ possible dichotomies, though H may be unable to represent some of these. We say that H shatters S if every possible dichotomy of S can be represented by some hypothesis from H .

Definition: A set of instances S is **shattered** by hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

Figure 7.3 illustrates a set S of three instances that is shattered by the hypothesis space. Notice that each of the 2^3 dichotomies of these three instances is covered by some hypothesis.

Note that if a set of instances is not shattered by a hypothesis space, then there must be some concept (dichotomy) that can be defined over the instances, but that cannot be represented by the hypothesis space. The ability of H to shatter

Instance space X

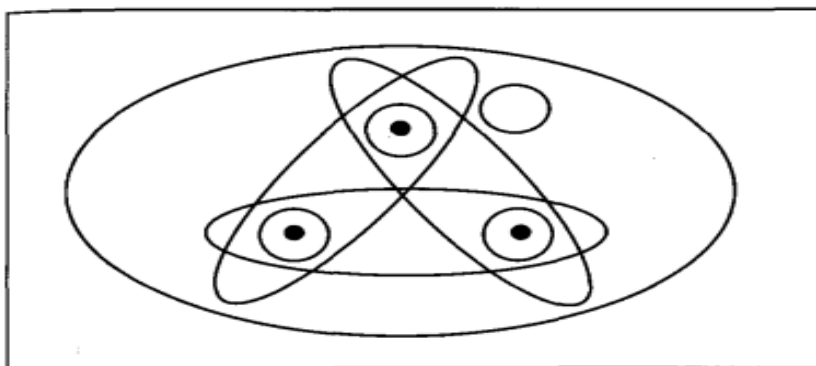


FIGURE 7.3

A set of three instances shattered by eight hypotheses. For every possible dichotomy of the instances, there exists a corresponding hypothesis.

a set of instances is thus a measure of its capacity to represent target concepts defined over these instances.

The Vapnik-Chervonenkis Dimension

The VC dimension quantifies the complexity of a hypothesis space, e.g. the models that could be fit given a representation and learning

The ability to shatter a set of instances is closely related to the inductive bias of a hypothesis space. Algorithm

Shatter or a **shattered set** in the case of a dataset, means points in the feature space can be selected or separated from each other using hypotheses in the space such that the labels of examples in the separate groups are correct

Definition: The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H . If arbitrarily large finite sets of X can be shattered by H , then $VC(H) \equiv \infty$.

Note that for any finite H , $VC(H) \leq \log_2 |H|$. To see this, suppose that $VC(H) = d$. Then H will require 2^d distinct hypotheses to shatter d instances. Hence, $2^d \leq |H|$, and $d = VC(H) \leq \log_2 |H|$.

THE MISTAKE BOUND MODEL OF LEARNING

the mistake bound model of learning, in which the learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis. As in the PAC setting, we assume the learner receives a sequence of training examples.

However, here we demand that upon receiving each example x , the learner must predict the target value $c(x)$, before it is shown the correct target value by the trainer. The question considered is "How many mistakes will the learner make in its predictions before it learns the target concept?" This question is significant in practical settings where learning must be done while the system is in actual use, rather than during some off-line training stage.

For example, if the system is to learn to predict which credit card purchases should be approved and which are fraudulent, based on data collected during use, then we are interested in minimizing the total number of mistakes it will make before converging to the correct target function. Here the total number of mistakes can be even more important than the total number of training examples.

This mistake bound learning problem may be studied in various specific settings. For example, we might count the number of mistakes made before PAC learning the target concept. In the examples below, we

consider instead the number of mistakes made before learning the target concept exactly. Learning the target concept exactly means converging to a hypothesis such that $(\forall x)h(x) = c(x)$.

Mistake Bound for the FIND-S Algorithm

To illustrate, consider again the hypothesis space H consisting of conjunctions of up to n boolean literals l_1, l_2, \dots, l_n , and their negations. Recall the FIND-S algorithm, which incrementally computes the maximally specific hypothesis consistent with the training examples. A straightforward implementation of FIND-S for the hypothesis space H is as follows

FIND-S:

- Initialize h to the most specific hypothesis $l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \dots l_n \wedge \neg l_n$
- For each positive training instance x
 - Remove from h any literal that is not satisfied by x
- Output hypothesis h .

FIND-S converges in the limit to a hypothesis that makes no errors, provided C , H and provided the training data is noise-free. FIND-S begins with the most specific hypothesis (which classifies every instance a negative example), then incrementally generalizes this hypothesis as needed to cover observed positive training examples. For the hypothesis representation used here, this generalization step consists of deleting unsatisfied literals.

Therefore, to calculate the number of mistakes it will make, we need only count the number of mistakes it will make misclassifying truly positive examples as negative.

Therefore, to calculate the number of mistakes it will make, we need only count the number of mistakes it will make misclassifying truly positive examples as negative.

How many such mistakes can occur before FIND-S learns c exactly? Consider the first positive example encountered by FIND-S. The learner will certainly make a mistake classifying this example, because its initial hypothesis labels every instance negative. However, the result will be that $1/2^n$ terms in its initial hypothesis will be eliminated, leaving only n terms. For each subsequent positive example that is mistakenly classified by the current hypothesis, at least one more of the remaining n terms must be eliminated from the hypothesis.

Therefore, the total number of mistakes can be at most $n + 1$. This number of mistakes will be required in the worst case, corresponding to learning the most general possible target concept .

INSTANCE BASED LEARNING

- Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.
- Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance
- Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified

The Machine Learning systems which are categorized as **instance-based learning** are the systems that learn the training examples by heart and then generalizes to new instances based on some similarity measure. It is called instance-based because it builds the hypotheses from the training instances. It is also known as **memory-based learning** or **lazy-learning**. The time complexity of this algorithm depends upon the size of training data.

***k*- NEAREST NEIGHBOR LEARNING**

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n -dimensional space R^n .

- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector $((a_1(x), a_2(x), \dots, a_n(x)))$
 Where, $a_r(x)$ denotes the value of the r^{th} attribute of instance x .

- Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$ Where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^n (a_r(x_i) - a_r(x_j))^2}$$

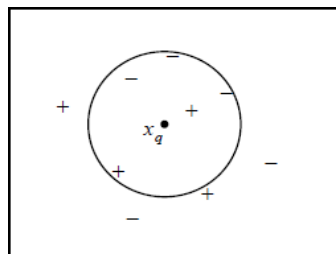
- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

Let us first consider learning **discrete-valued target functions** of

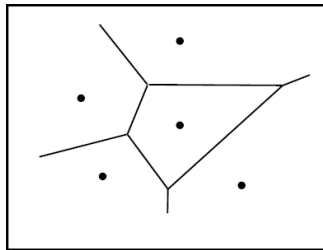
$$f : \mathbb{R}^n \rightarrow V.$$

the form Where, V is the finite set $\{v_1, \dots, v_s\}$

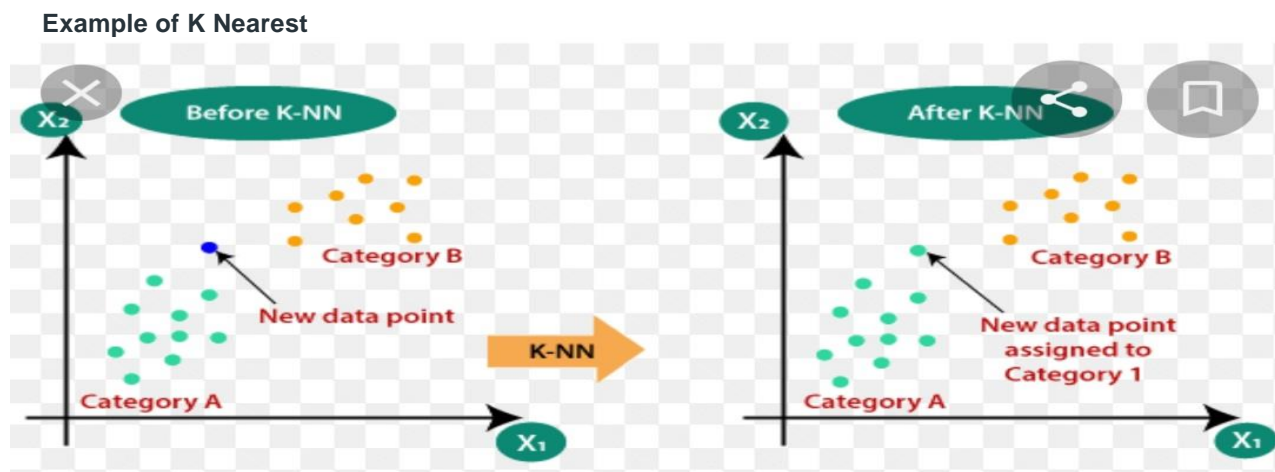
- The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q .
- If $k = 1$, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$. Where x_i is the training instance nearest to x_q .
- For larger values of k , the algorithm assigns the most common value among the k nearest training examples.
- Below figure illustrates the operation of the k -Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.



- The positive and negative training examples are shown by “+” and “-” respectively. A query point x_q is shown as well.
- The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.
- Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.



- For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the **Voronoi diagram** of the set of training example



Vyasapuri, Bandlaguda, Post:Keshavgi
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU, Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956, ISO 9001:2015 Certified



Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Example

Name	Acid Durability	Strength	Class
Type-1	7	7	Bad
Type-2	7	4	Bad
Type-3	3	4	Good
Type-4	1	4	Good

Test-Data → acid durability=3, and strength=7, class=?

Example of knn Problem

Similarity

- Calculated using distance measure like Euclidean

$$d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

Name	Acid Durability	Strength	Class	Distance
Type-1	7	7	Bad	Sqrt((7-3) ² +(7-7) ²)=4
Type-2	7	4	Bad	5
Type-3	3	4	Good	3
Type-4	1	4	Good	3.6

Rank these attributes

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

k=3

Name	Acid Durability	Strength	Class	Distance	Rank
Type-1	7	7	Bad	4	3
Type-2	7	4	Bad	5	4
Type-3	3	4	Good	3	1
Type-4	1	4	Good	3.6	2

Based on three neighbours, 2 Goods and 1 bad, majority → Good

The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is

given below:

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
 - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
 - Return

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

K-nearest neighbor alg for approximation discrete-valued target function.

$\delta(a,b) = 1$ if $a \neq b$
 $\delta(a,b) = 0$ if $a = b$.

$x_q = k$ nearest $k=3$.

According to ~~formula~~ equation $(C_1, f(x_1))$
 So, either 2 possibilities good or bad.

$(C_1, f(x_1)) + (C_1, f(x_2)) + (C_1, f(x_3))$

$x_1 = \text{Rank} = 1 = \text{good} = 3$
 $x_2 = \text{Rank} = 2 = \text{good} = 3, 6$
 $x_3 = \text{Rank} = 3 = \text{Bad} = 4.$

first possibilities: good
 $(C_1, 4) + (C_1, 4) + (C_1, B)$
 $1 + 1 + 0 = 2$

second possibility: bad
 $(B, f(x_1)) + (B, f(x_2)) + (B, f(x_3))$
 $(B, 4) + (B, 4) + (B, B)$
 $0 + 0 + 1 = 1$

Comparing $(1, 2) \rightarrow$ we take 2 as the target function is classified as good.

The K- Nearest Neighbor algorithm for approximation a real-valued target function is given below

Training algorithm:

- For each training example $(x, f(x))$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
 - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
 - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k f(x_i)}{k}$$

The k -nearest algorithm for approximations
of a real valued target function is given below
query instance x_q is classified

$$f(x_q) = \frac{\sum_{i=1}^k f(x_i)}{k}$$

According to given example $k = 3$

nearest distance are Rank 1 = 3
Rank 2 = 3.6
Rank 3 = 4

$$\frac{x_1 + x_2 + x_3}{3} = \frac{3 + 3.6 + 4}{3} = \frac{10.6}{3}$$

= 3.4, which is

nearest to Rank 2 -

and Rank 2 is classified as Good.

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
 - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
 - Return

$$\hat{f}(x_q) \leftarrow \operatorname{argmax}_{v \in V} \sum_{i=1}^k w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

o

Distance - weighted nearest neighbor
 Alg for approx discrete valued
 target function.

$$f(x_q) \leftarrow \operatorname{argmax}_{i=1}^k w_i \delta(y, f(x_i))$$

$$\text{where } w_i = \frac{1}{d(x_q, x_i)^2}$$

$$x_1 = \text{rank } 1 = 3 = \frac{1}{3} = 0.33 = w_1$$

$$x_2 = \text{rank } 2 = 3.6 = \frac{1}{3.6} = 0.27 = w_2$$

$$x_3 = \text{rank } 3 = 4 = \frac{1}{4} = 0.25 = w_3$$

first possibility for good

$$w_1(a, a) + w_2(a, a) + w_3(a, a)$$

$$w_1(a, a) + w_2(a, a) + w_3(a, b)$$

$$w_1(1) + w_2(1) + w_3(0)$$

$$= 0.33 + 0.27 = 0.60$$

second possibility - bad

$$w_1(b, a) + w_2(b, a) + w_3(b, a)$$

$$w_1(b, a) + w_2(b, a) + w_3(b, b)$$

$$w_1(0) + w_2(0) + w_3(0)$$

$$= 0.25(1) = 0.25 \Rightarrow$$

comparing (0.60, 0.25)
 upstroke
 ↓
 good.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU, Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956, ISO 9001:2015 Certified



Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

Training algorithm:

- For each training example $(x, f(x))$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
 - Let $x_1 \dots x_k$ denote the k instances from *training_examples* that are nearest to x_q
 - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Distance-weighted nearest neighbor
 Alg approx a Real valued target
 function

$$f(x_q) \leftarrow \frac{\sum w_i f(x_i)}{\sum w_i}$$

Let $w = \frac{1}{d(x_q, x_i)^2}$

for good

$$w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3)$$

$$w_1 (4,0,4) + w_2 (4,1,4) + w_3 (4,1,8)$$

$$0.33 + 0.27 + 0$$

$$= \frac{0.60}{0.33+0.27+0.25} = \frac{0.60}{0.85} = 0.70$$

for Bad

$$w_1 (8,4) + w_2 (8,4) + w_3 (8,8)$$

$$0 + 0 + 0.25$$

$$= \frac{0.25}{0.85} = 0.29$$

comparing both 4 and 8
 4 as more value than 8 so,
 target function is classified
 as good.

Regression Analysis in Machine learning

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price**, etc.

We can understand the concept of regression analysis using the below example:

Example: Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:

Advertisement	Sales
\$90	\$1000
\$120	\$1300
\$150	\$1800
\$100	\$1200
\$130	\$1380
\$200	??

Now, the company wants to do the advertisement of \$200 in the year 2019 **and wants to know the prediction about the sales for this year**. So to solve such type of prediction problems in machine learning, we need regression analysis.

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price**, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:

Mathematically, we can represent a linear regression as:

$$y = a_0 + a_1x + \epsilon$$

Y= Dependent Variable (Target Variable)

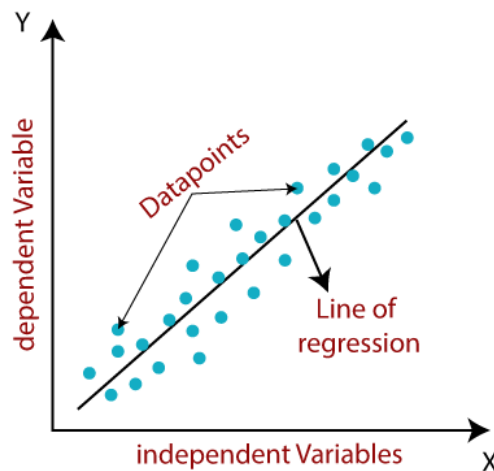
X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ϵ = random error

The values for x and y variables are training datasets for Linear Regression model representation.



for Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation

Yi = Actual value

($a_1x_i+a_0$)= Predicted value.

1.LOCALLY WEIGHTED REGRESSION

- The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.
- Given a new query instance x_q , the general approach in locally weighted regression is to construct an approximation \hat{f} that fits the training examples in the neighborhood surrounding x_q . This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.
- Consider locally weighted regression in which the target function f is

$$\hat{f}(x) = w_0 + w_1a_1(x) + \dots + w_na_n(x)$$

approximated near x_q using a linear function of the form

Where, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

Which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x))a_j(x)$$

Where, η is a constant learning rate

Need to modify this procedure to derive a local approximation rather than a global one.
The simple way is to redefine

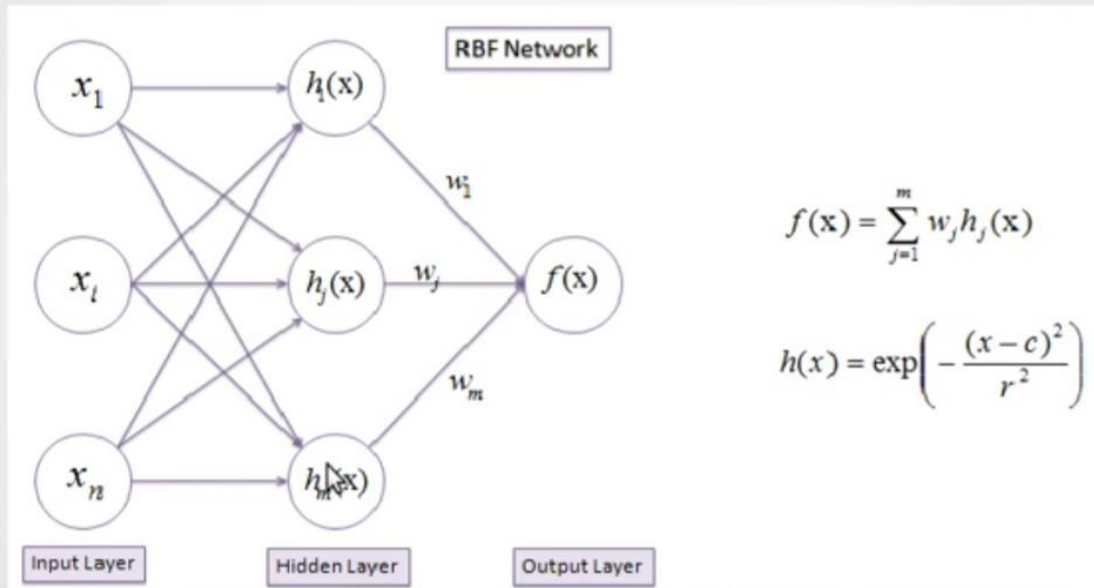
$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) (f(x) - \hat{f}(x)) a_j(x)$$

2.Radial basis functions

A radial basis function network is a type of supervised artificial neural network that uses supervised machine learning (ML) to function as a nonlinear classifier
A radial basis function network is also known as a radial basis network.

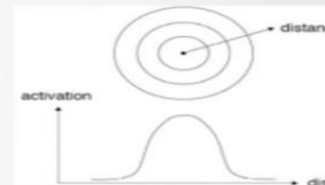
The radial basis function network uses radial basis functions as its activation functions. Like other kinds of neural networks, radial basis function networks have input layers, hidden layers and output layers. However, radial basis function networks often also include a nonlinear activation function of some kind. Output weights can be trained using gradient descent.

Structure of RBF Networks



How RBF networks work 2/2

- An RBF network positions one or more RBF neurons in the space described by the predictor variables (x_1, x_2 for example)
- This space has as many dimensions as there are predictor variables
- The Euclidean distance is computed from the point being evaluated (e.g., the green circle) to the center of each neuron, and a radial basis function (RBF) (also called a kernel function) is applied to the distance to compute the weight (influence) for each neuron
- The radial basis function is so named because the radius distance is the argument to the function
Weight = RBF(distance)
- The further a neuron is from the point being evaluated, the less influence it has



Source:
<http://www.dtreng.com>

- When the RBF centers have been established, the width of each RBF unit can be calculated using the **K-nearest neighbors** algorithm
- A number **K** is chosen, and for each center, the **K** nearest centers are found
- The root-mean squared distance between the current cluster center and its K nearest neighbors is calculated, and this is the value chosen for **the unit width (r)**
- So, if the current cluster center is **c_j** , the **r** value is:

$$r_j = \sqrt{\frac{\sum_{t=1}^k (c_j - c_t)^2}{k}}$$

3.Case-based reasoning (CBR)

Case-based reasoning (CBR) is an experience-based approach to solving new problems by adapting previously successful solutions to similar problems. Addressing memory, learning, planning and problem solving, CBR provides a foundation for a new technology of intelligent computer systems that can solve problems and adapt to new situations. In CBR, the “intelligent” reuse of knowledge from already-solved problems

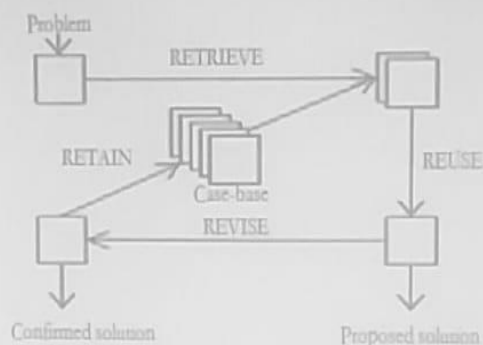
Four step process for CBR

In general, the case-based reasoning process entails:

1. Retrieve- Gathering from memory an experience closest to the current problem.
2. Reuse- Suggesting a solution based on the experience and adapting it to meet the demands of the new situation.
3. Revise- Evaluating the use of the solution in the new context.
4. Retain- Storing this new problem-solving method in the memory system.

Case-based reasoning

- Instance-based methods can also use more complex, symbolic representations
- In case-based learning, instances are represented in this fashion and the process for identifying neighbouring instances is elaborated accordingly



Advantages and disadvantages of CBR

On the plus side, remembering past experiences helps learners avoid repeating previous mistakes, and the reasoned can discern what features of a problem are significant and focus on them.

On the negative side, critics claim that the main premise of CBR is based on anecdotal evidence and that adapting the elements of one case to another may be complex and potentially lead to inaccuracies

REMARKS ON LAZY AND EAGER LEARNING

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU, Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956, ISO 9001:2015 Certified



Lazy learner:

1. Just store Data set **without** learning from it
2. Start classifying data when it receive **Test data**
3. So it takes less time learning and more time classifying data

Eager learner:

1. When it receive data set it starts classifying (learning)
2. Then it does not wait for test data to learn
3. So it takes long time learning and less time classifying data

Lazy : K - Nearest Neighbour, Case - Based Reasoning

Eager : Decision Tree, Naive Bayes, Artificial Neural Networks

Unit 4

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.

Five phases are considered in a genetic algorithm.

- Initial population
- Fitness function
- Selection
- Crossover
- Mutation
- The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve.
- An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution).
- In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome
- The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness score** to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection

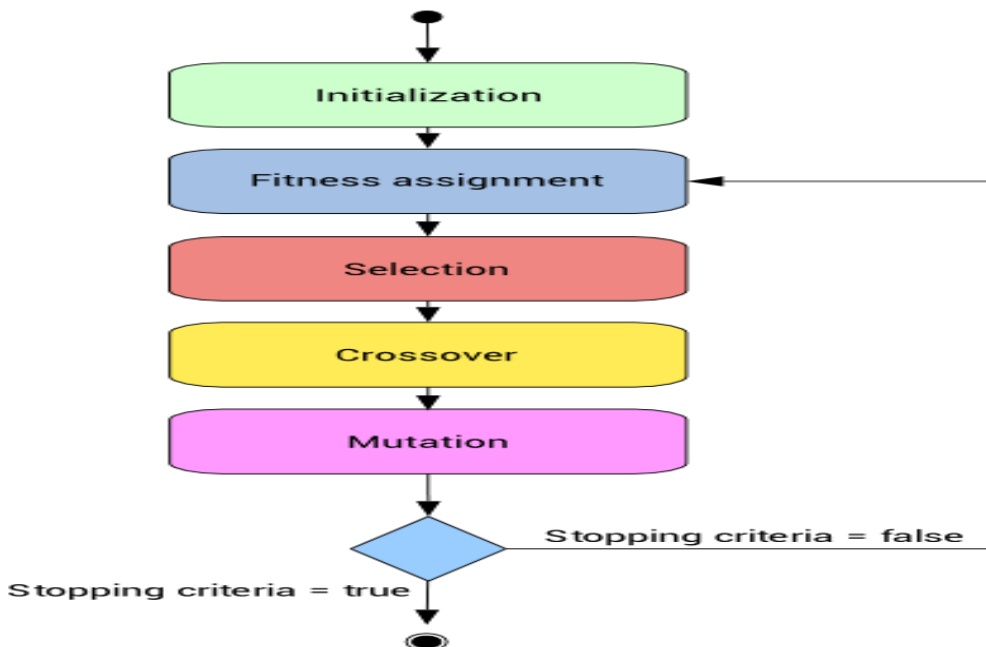
- The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.
- Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

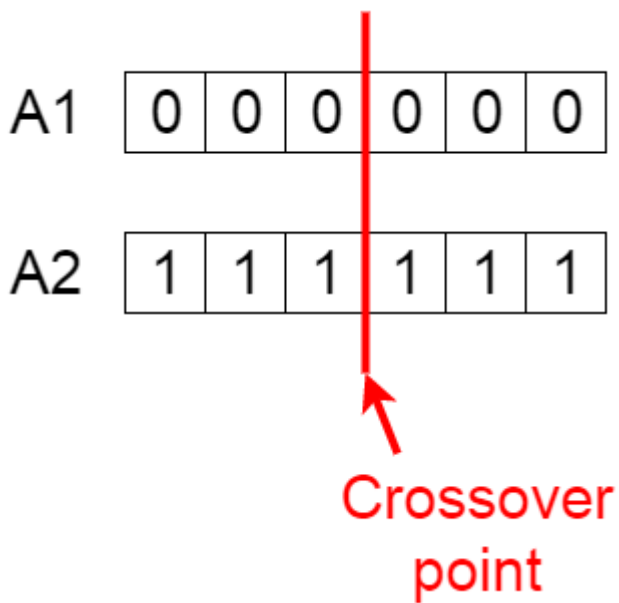
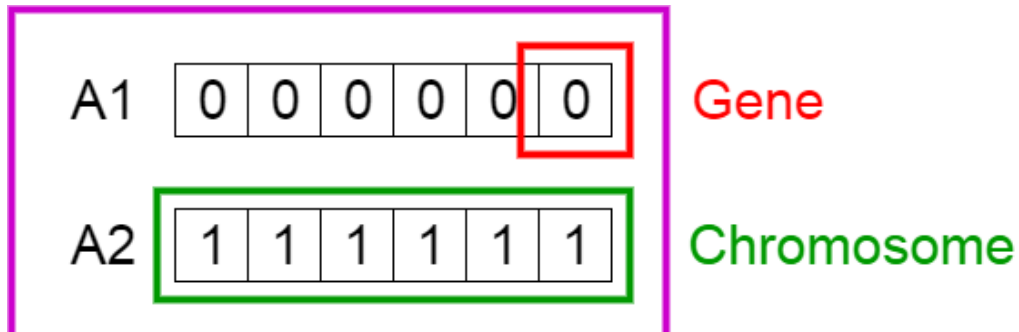
Crossover

- **Crossover** is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes.

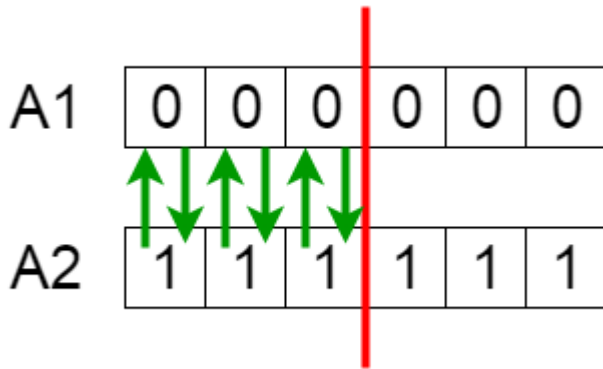
Mutation

- In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.

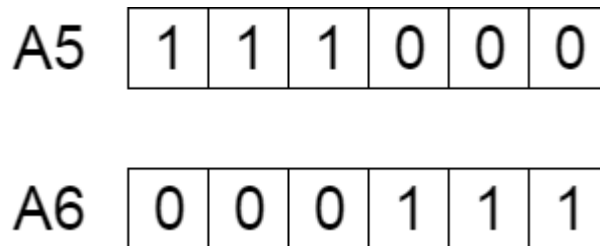




Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.

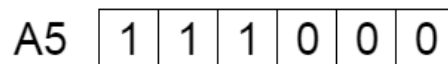


The new offspring are added to the population.

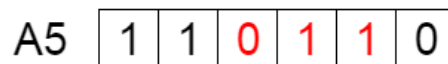


In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.

Before Mutation



After Mutation



Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

Application areas

Genetic algorithms are applied in the following fields:

- **Transport:** Genetic algorithms are used in the traveling salesman problem to develop transport plans that reduce the cost of travel and the time taken. They are also used to develop an efficient way of delivering products.
- **DNA Analysis:** They are used in DNA analysis to establish the DNA structure using spectrometric information.
- **Multimodal Optimization:** They are used to provide multiple optimum solutions in multimodal optimization problems.
- **Aircraft Design:** They are used to develop parametric aircraft designs. The parameters of the aircraft are modified and upgraded to provide better designs.
- **Economics:** They are used in economics to describe various models such as the game theory, asset pricing, and schedule optimization.

Advantages of Genetic Algorithms

- Parallelism
- A larger set of solution space
- Requires less information
- Provides multiple optimal solutions
- Probabilistic in nature
- Genetic representations using chromosomes

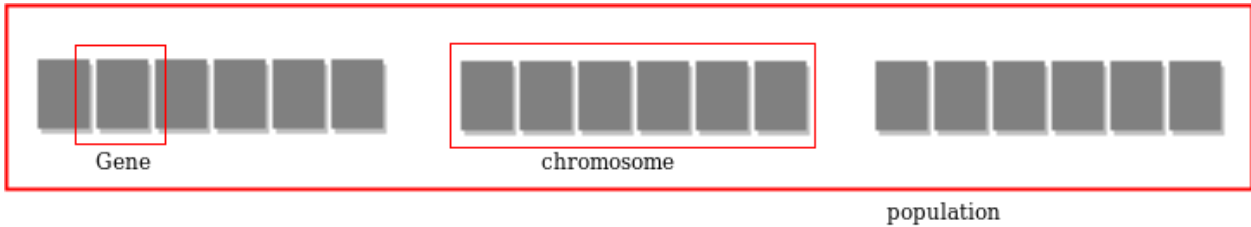
Disadvantages of Genetic Algorithms

- Computational complexity

Hypothesis Search space

- The population of individuals are maintained within search space. Each individual represents 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).



Genetic Programming Example

Genetic Algorithm

Example Problem
 Max ONE Methodology
 TO maximize the result according to the desired object to achieve by using GA.

Tossing coin max 60 times $l=10, n=6$.

$S_1 - 1111010101$
 $S_2 - 0111000101$
 $S_3 - 1110110101$
 $S_4 - 0100010011$
 $S_5 - 1110111101$
 $S_6 - 0100110000$

Flowchart:
 Initialization → Selection → Crossover → Mutation → Evaluate/Generate (with fitness improvement) → End. A feedback loop labeled 'High level para' connects the Evaluate/Generate step back to Selection.

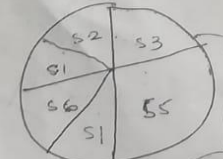
Individual	Fitness
S_1	7
S_2	5
S_3	7
S_4	4
S_5	8
S_6	3
Total fitness	34

selection → Roulette wheel, $n=6$.

Selection:

$$P_i = \frac{f(i)}{\sum f(i)}$$
 Maximum fitness is 8.

$f(S_5) = 8/34 = 23\%$
 $f(S_3) = 7/34 = 20\%$
 $f(S_1) = 7/34 = 20\%$



- Next method cross over maximum w
 other fitness values.

$$\text{Max} \left| \begin{array}{l} S_1 \\ S_3 \\ S_5 \end{array} \right.$$

$$\text{min} \left| \begin{array}{l} S_2 \\ S_4 \\ S_6 \end{array} \right.$$

- crossover

$$S_1 - 1111010101$$

$$S_2 - 0111000101$$

$$S_1' = 0111000101$$

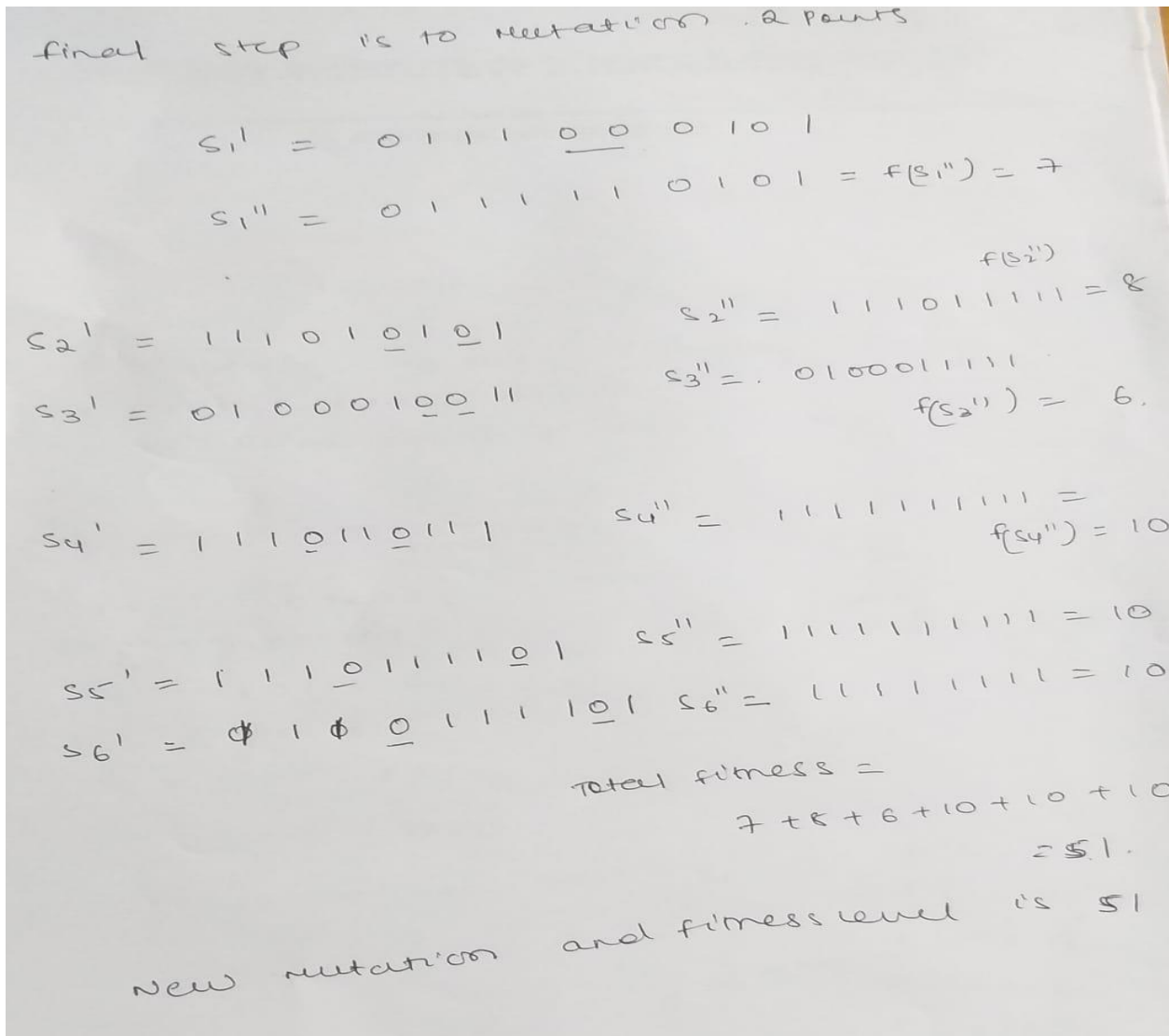
$$S_2' = 1111010101$$

$$S_3 = 1110110101$$

$$S_4 = 0100010011$$

$$S_3' = 0100010011$$

$$S_4' = 1110110111$$



Models of evolution and learning proposed by 2 authors

Lamarckian Evolution Theory:

The Lamarckian theory states the characteristic individual acquire during their lifetime pass them to their children. This theory is named after French biologist Jean Baptiste Lamarck. According to Lamarck's theory, learning is an important part of the evolution of species(or for our purpose in the Evolutionary algorithm). This theory is discredited in a biological context but can be used in genetic algorithms in machine learning.

Baldwin Effect:

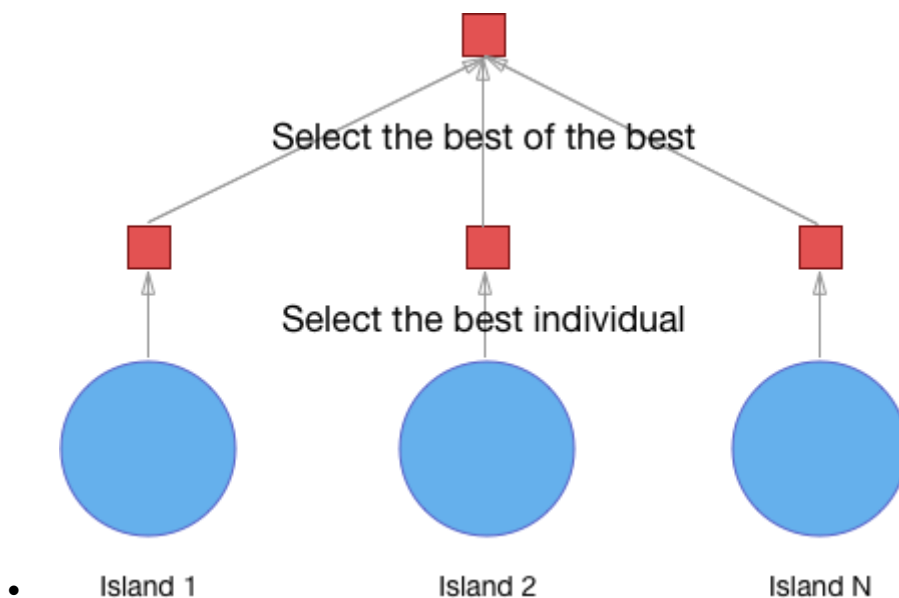
Baldwin proposed that individual learning can explain evolutionary phenomena that appear to require Lamarckian inheritance of acquired characteristics. The ability of individuals to learn can guide the evolutionary process. In effect, learning smooths the fitness landscape, thus facilitating evolution.

Baldwin Effect is first demonstrated by Hinton and Nolan in the context of machine learning in 1987. They take simple Neural Networks (NNs). In one experiment they take NNs of fixed weights while other NNs set to trainable. They concluded that:

- When there is no individual learning, the population(collection of NNs) failed to improve over time.
- When learning is applied in early stages, the population contains many individuals with many trainable weights, but in later stages, it achieved high fitness with the number of trainable weights decreases in individuals

- **Parallelizing Genetic Algorithms**

- parallel genetic algorithm is such an algorithm that uses multiple genetic algorithms to solve a single task . All these algorithms try to solve the same task and after they've completed their job, the best individual of every algorithm is selected, then the best of them is selected, and this is the solution to a problem. This is one of the most popular approach to parallel genetic algorithms, even though there are others. This approach is often called 'island model' because populations are isolated from each other, like real-life creature populations may be isolated living on different islands. Image 1 illustrates that.



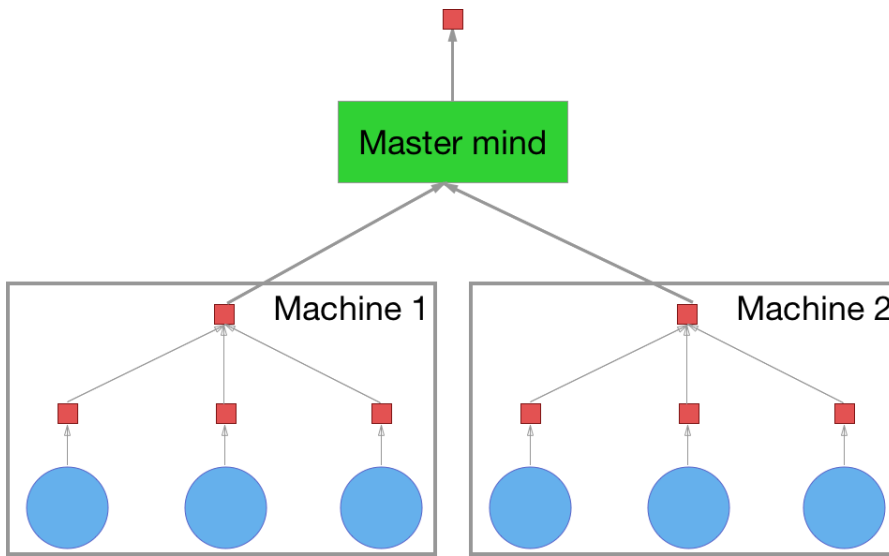
These genetic algorithms do not depend on each other, as a result, they can run in parallel, taking advantage of a multicore CPU. Each algorithm has its own set of individual, as a result these individuals may differ from individuals of another algorithm, because they have different mutation/crossover history.

Let's illustrate this with an example. Say we have three independent genetic algorithms and we want to crossover them in pairs. We take the first algorithm and randomly select the second element of a pair. So, we create as much pairs as many algorithms we have, in each pair the first element is chosen sequentially, and the second one is random. Then we perform crossover on populations of these two algorithms taking individuals from both of them. We pick individuals for crossover in such a way, that individuals from different algorithms are crossed over together. We use crossover mechanisms that are used by the first algorithm of a pair and this algorithm receives all of the individuals that were created as a result of a crossover; the second algorithm of a pair is simply a donor that provides its individuals. Therefore, each algorithm receives new individuals when it is the first element of a pair. If a crossover algorithm requires more than two individuals, additional individuals may be taken from any of the algorithms of a pair, it is only recommended that no algorithm dominate here

A parallel genetic algorithm may take a little more time than a non-parallel one, that is because it uses several computation threads which, in turn, cause the Operation System to perform context switching more frequently

Distributed genetic algorithm

Distributed genetic algorithm is actually a parallel genetic algorithm that has its independent algorithms running on separate machines. Moreover, in this case each of these algorithms may be in turn a parallel genetic algorithm! Distributed genetic algorithm also implements the 'island model' and each 'island' is even more isolated from others. If each machine runs a parallel genetic algorithm we may call this as 'archipelago model', because we have groups of islands. It actually does not matter what a single genetic algorithm is, because distributed genetic algorithm is about having multiple machines running independent genetic algorithms in order to solve the same task. Image 2 illustrates this.



Distributed genetic algorithm may also help when we have to create many individuals in order to observe the entire domain, but it is not possible to store all of them in memory of a single machine.

When we were discussing parallel genetic algorithm we introduced the ‘crossover between algorithms’ term. Distributed genetic algorithm enables us to perform crossover between separate machines

In case of distributed genetic algorithm, we have a kind of ‘master mind’ that controls the overall progress and coordinates these machines. It also controls crossover between machines, selecting how machines will be paired together to perform crossover. In general, process is the same as in case of parallel genetic algorithm, except that individuals are moved over the network from one machine to another.

Sequential Covering Algorithm

Sequential Covering is a popular algorithm based on Rule-Based Classification used for learning a disjunctive set of rules. The basic idea here is to learn one rule, remove the data that it covers, then repeat the same process. In this process, In this way, it covers all the rules involved with it in a sequential manner during the training phase.

The Sequential Learning algorithm takes care of to some extent, the low coverage problem in the Learn-One-Rule algorithm covering all the rules in a sequential manner.

Working on the Algorithm:

Faculty Name : Mrs Swapna

Subject Name :ML

The algorithm involves a set of ‘ordered rules’ or ‘list of decisions’ to be made.

Step 1 – create an empty decision list, ‘R’.

Step 2 – ‘Learn-One-Rule’ Algorithm

*Step 2.a – if all training examples \in class ‘y’, then it’s classified as **positive example**.*

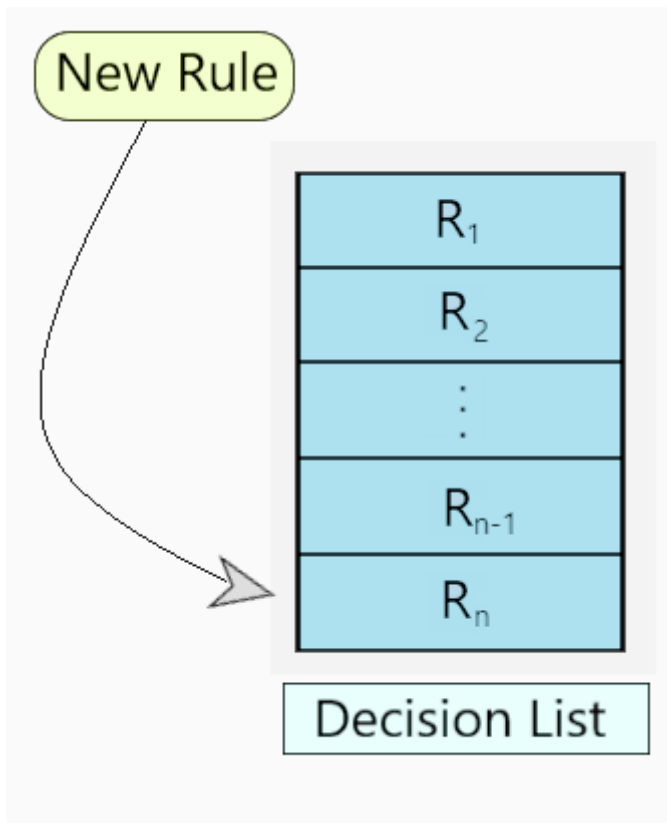
*Step 2.b – else if all training examples \notin class ‘y’, then it’s classified as **negative example**.*

*Step 3 – The rule becomes ‘**desirable**’ when it covers a majority of the positive examples.*

Step 4 – When this rule is obtained, delete all the training data associated with that rule.

(i.e. when the rule is applied to the dataset, it covers most of the training data, and has to be removed)

Step 5 – The new rule is added to the bottom of decision list, ‘R’. (Fig.3)



Below, is a visual representation describing the working of the algorithm.

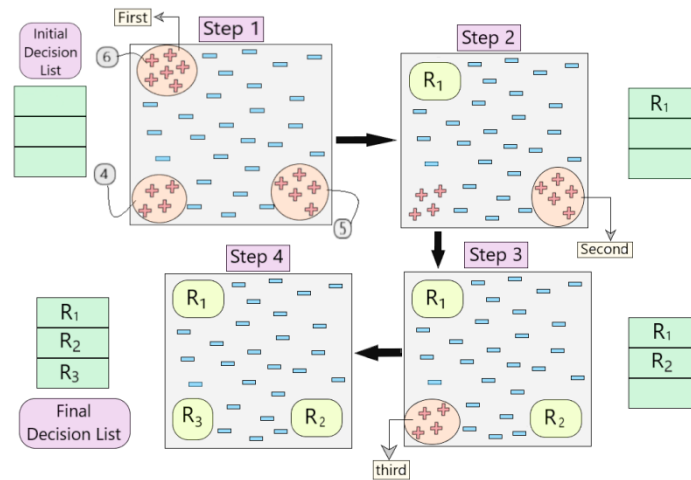


fig 4: Visual Representation of working of the algorithm

- Let us understand step by step how the algorithm is working in the example shown in Fig.4.
- First, we created an empty decision list. During Step 1, we see that there are three sets of positive examples present in the dataset. So, as per the algorithm, we consider the one with maximum no of positive example. (6, as shown in Step 1 of Fig 4)
- Once we cover these 6 positive examples, we get our first rule R_1 , which is then pushed into the decision list and those positive examples are removed from the dataset. (as shown in Step 2 of Fig 4)
- Now, we take the next majority of positive examples (5, as shown in Step 2 of Fig 4) and follow the same process until we get rule R_2 . (Same for R_3)
- In the end, we obtain our final decision list with all the desirable rules.

Propositional logic:

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.

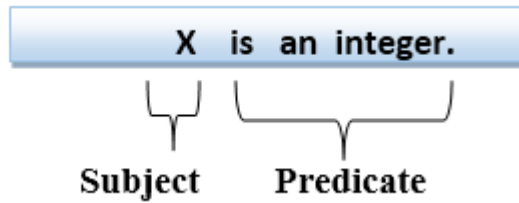
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.
- technique of knowledge representation in logical and mathematical form.
- Syntax of propositional logic:
- The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:
- **Atomic Propositions**
- **Compound propositions**
- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.
- **Example:**
- a) $2+2$ is 4, it is an atomic proposition as it is a **true** fact.
- b) "The Sun is cold" is also a proposition as it is a **false** fact.
- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.
- **Example:**
- a) "It is raining today, and street is wet."
- b) "Ankit is a doctor, and his clinic is in Mumbai."
- Limitations of Propositional logic:
- We cannot represent relations like ALL, some, or none with propositional logic. Example:
 - **All the girls are intelligent.**
 - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

First-Order Logic

- In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.
- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**
- To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.
- First-Order logic:
- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
- **Relations:** It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of,
- As a natural language, first-order logic also has two main parts:
- **Syntax**

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.
- **Consider the statement:** "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.

These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:

Universal Quantifier, (for all, everyone, everything)

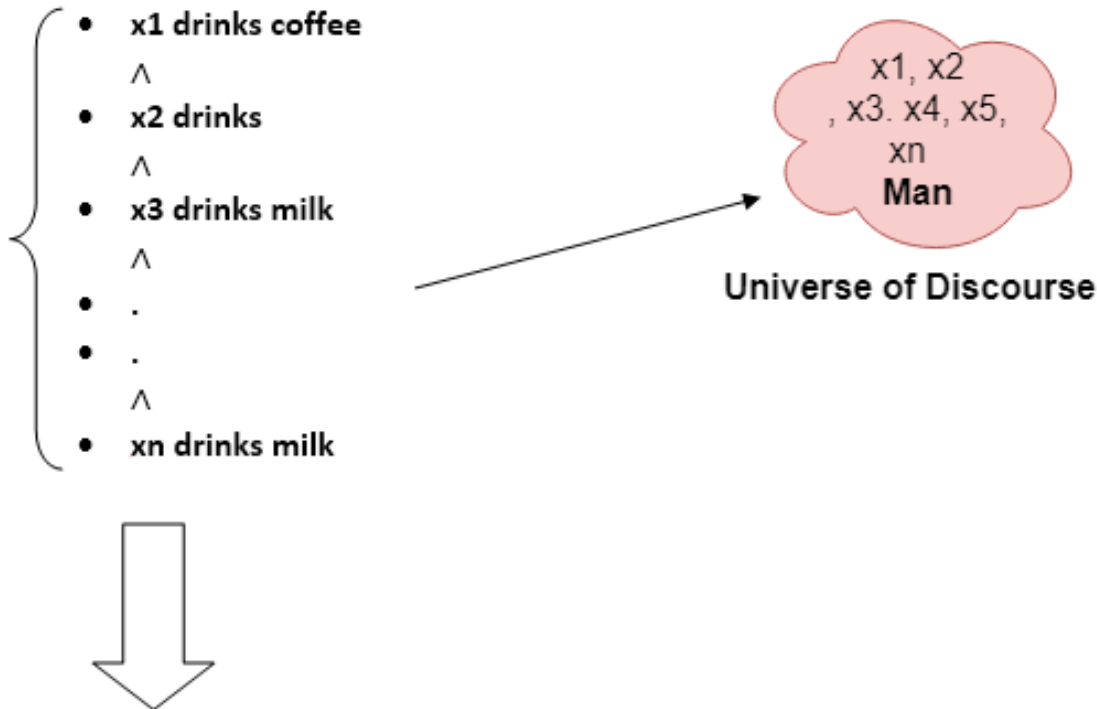
Existential quantifier, (for some, at least one).

Universal Quantifier:

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol \forall , which resembles an inverted A.

- If x is a variable, then $\forall x$ is read as:
 - **For all x**
 - **For each x**
 - **For every x .**
- Example:
 - **All man drink coffee.**
 - **$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee})$.**



So in shorthand notation, we can write it as :

it will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator \exists , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be $\exists x$ or $\exists(x)$. And it will be read as:

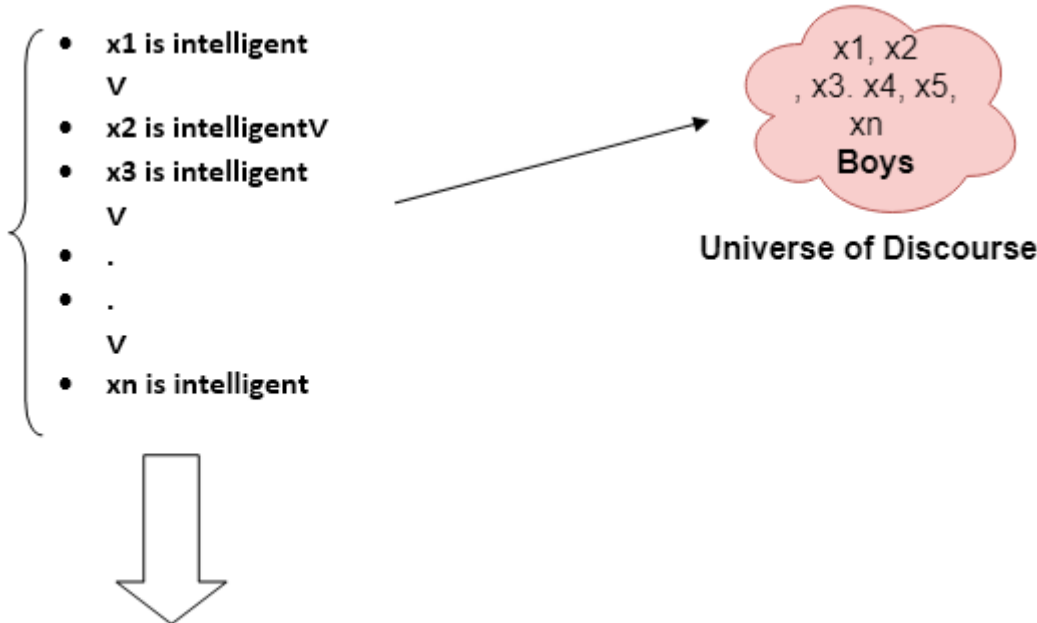
There exists a 'x.'

For some 'x.'

For at least one 'x.'

Example:

Some boys are intelligent



So in short-hand notation, we can write it as:

$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$

It will be read as: There are some x where x is a boy who is intelligent.

- Some Examples of FOL using quantifier:

- **1. All birds fly.**

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

- **2. Every man respects his parent.**

In this question, the predicate is "**respect(x, y)**," where **x=man, and y= parent**.

Since there is every man so will use \forall , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

- **3. Some boys play cricket.**

In this question, the predicate is "**play(x, y)**," where **x= boys, and y= game**. Since there are some boys so we will use \exists , and it will be represented as:

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

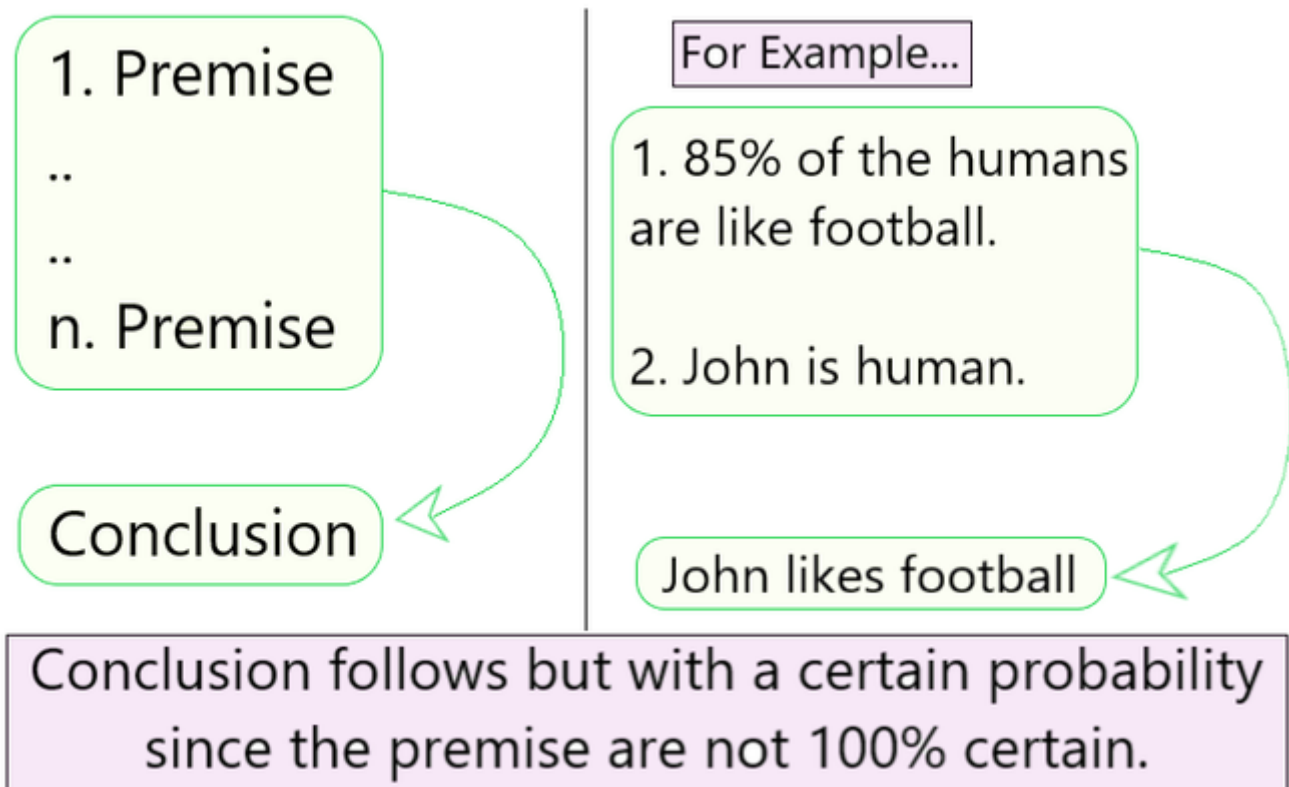
- **4. Not all students like both Mathematics and Science.**

In this question, the predicate is "like(x, y)," where x= student, and y= subject.

Since there are not all students, so we will use \forall with negation, so following representation for this:

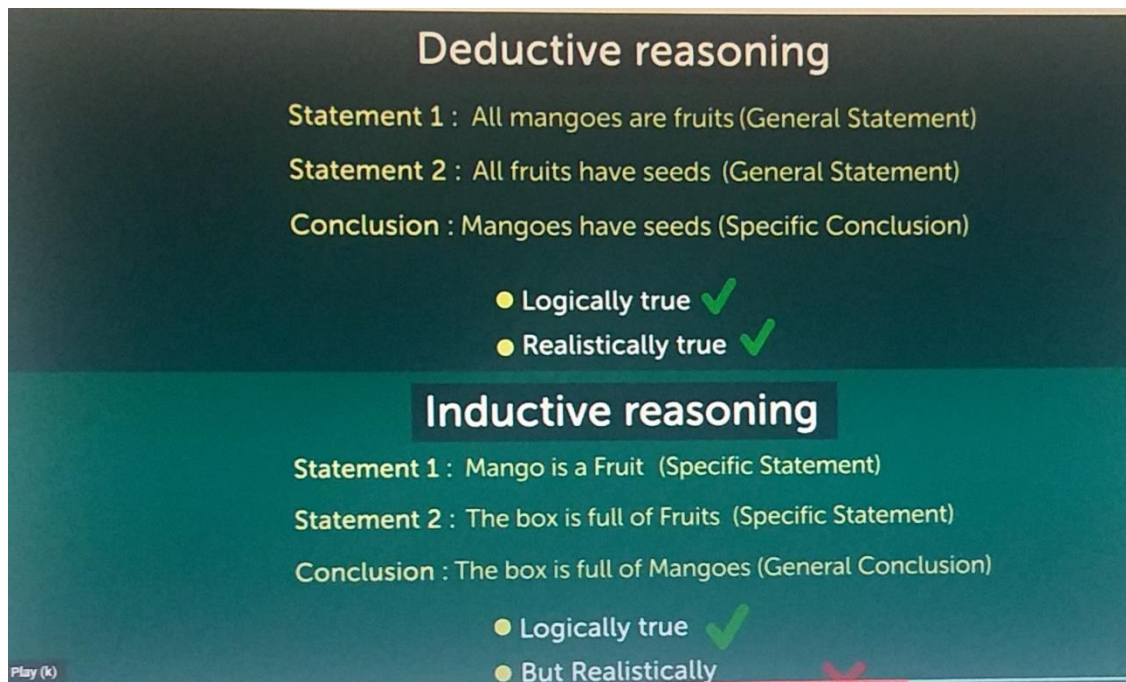
$$\neg \forall (x) [\text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})].$$

- **First Order Inductive Learner (FOIL)**
- In machine learning, first-order inductive learner (FOIL) is a rule-based learning algorithm. It is a natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms.
- **Inductive Learning:**
- Inductive learning analyzing and understanding the evidence and then using it to determine the outcome. It is based on Inductive Logic.



Induction as inverted deduction:

A different approach to inductive logic programming is based on the simple observation that induction is just the inverse of deduction.




Deductive reasoning

Statement 1 : All mangoes are fruits (General Statement)
Statement 2 : All fruits have seeds (General Statement)
Conclusion : Mangoes have seeds (Specific Conclusion)

- Logically true ✓
- Realistically true ✓

Inductive reasoning

Statement 1 : Mango is a Fruit (Specific Statement)
Statement 2 : The box is full of Fruits (Specific Statement)
Conclusion : The box is full of Mangoes (General Conclusion)

- Logically true ✓
- But Realistically 

Play (k)

INVERTING RESOLUTION :A general method for automated deduction is the resolution rule introduced by Robinson (1965). The resolution rule is a sound and complete rule for deductive inference in first-order logic.

It is easiest to introduce the resolution rule in propositional form, though it is readily extended to first-order representations. Let L be an arbitrary propositional literal, and let P and R be arbitrary propositional clauses. The resolution rule is

$$\begin{array}{ccc} P & \vee & L \\ \neg L & \vee & R \\ \hline P & \vee & R \end{array}$$

which should be read as follows: Given the two clauses above the line, conclude the clause below the line. Intuitively, the resolution rule is quite sensible. Given the two assertions $P \vee L$ and $\neg L \vee R$, it is obvious that either L or $\neg L$ must be false. Therefore, either P or R must be true. Thus, the conclusion $P \vee R$ of the resolution rule is intuitively satisfying.

Resolution

- Its an inference rule used in both Propositional as well as First-order Predicate Logic in different ways.
- It is also called **Proof by Refutation**
- It is basically used for proving the satisfiability of a sentence.
- Proof by Refutation technique is used to prove the given statement. It's a iterative process :
 1. Select two clauses that contain conflicting terms.
 2. Combine those two clauses and cancel out the conflicting terms, yielding a new clause that has been inferred from them.
- The key idea is to use the knowledge base and negated goal to obtain null clause(which indicates contradiction).
- Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

Steps in Resolution

1. Convert facts into First order logic (FOL)
2. Convert FOL into CNF (Conjunctive Normal Form)
3. Negate the statement to be proved , and add the result to the knowledge base.
4. Draw Resolution graph .
5. If empty clause (NIL) is produced , stop and report that original theorem is true .

Step 1 : Conversion to first order logic

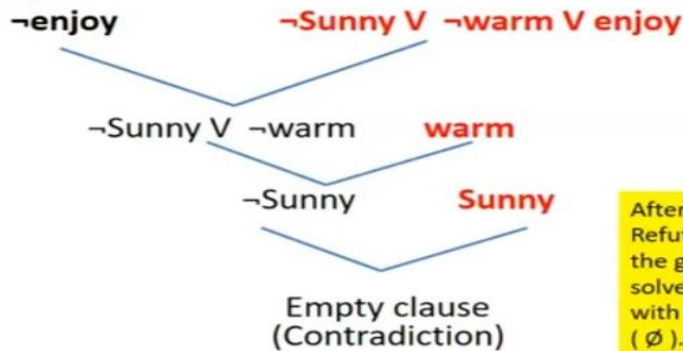
- If It is sunny and warm day you will enjoy
Sunny \wedge warm \rightarrow enjoy
- If it is raining you will get wet
raining \rightarrow wet
- It is warm day
warm
- It is raining
raining
- It is sunny
Sunny

Step 2 : conversion to CNF

- **Sunny \wedge Warm \rightarrow enjoy**
Eliminate implication:
 $\neg(\text{Sunny} \wedge \text{warm}) \vee \text{enjoy}$
Moving negation inside
 $\neg\text{Sunny} \vee \neg\text{warm} \vee \text{enjoy}$
- **raining \rightarrow wet**
Eliminate implication:
 $\neg\text{raining} \vee \text{wet}$
- **warm**
- **raining**
- **Sunny**

Step: 3 & 4 Resolution graph

- Negate the statement to be proved: $\neg \text{enjoy}$
- Now take the statements one by one and create resolution graph.



After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a Null clause (\emptyset). Hence, the goal is achieved.

Example #2

- Consider the following Knowledge Base:
 1. The humidity is high or the sky is cloudy.
 2. If the sky is cloudy, then it will rain.
 3. If the humidity is high, then it is hot.
 4. It is not hot.

Goal: It will rain.

Step 1 : Conversion to first order logic

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

- Let P : The humidity is high

- Let Q: sky is cloudy

The humidity is high or the sky is cloudy

$P \vee Q$

Conversion to first order logic...

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

- Let Q: sky is cloudy

- Let R: it will rain

If the sky is cloudy, then it will rain

$Q \rightarrow R$

Goal: It will rain.

Conversion to first order logic...

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

- Let P : The humidity is high

- Let S: it is hot

If the humidity is high, then it is hot

$P \rightarrow S$

- Goal: It will rain.

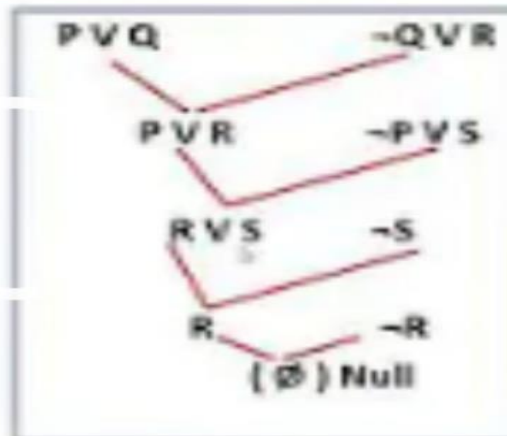
Statements in first order logic

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

- $P \vee Q$
- $Q \rightarrow R$
- $P \rightarrow S$
- $\neg S$

- Goal: It will rain. (R)

- $P \vee Q$
- $\neg Q \vee R$
- $\neg P \vee S$
- $\neg S$
- $\neg R$ (Goal)



REINFORCEMENT LEARNING

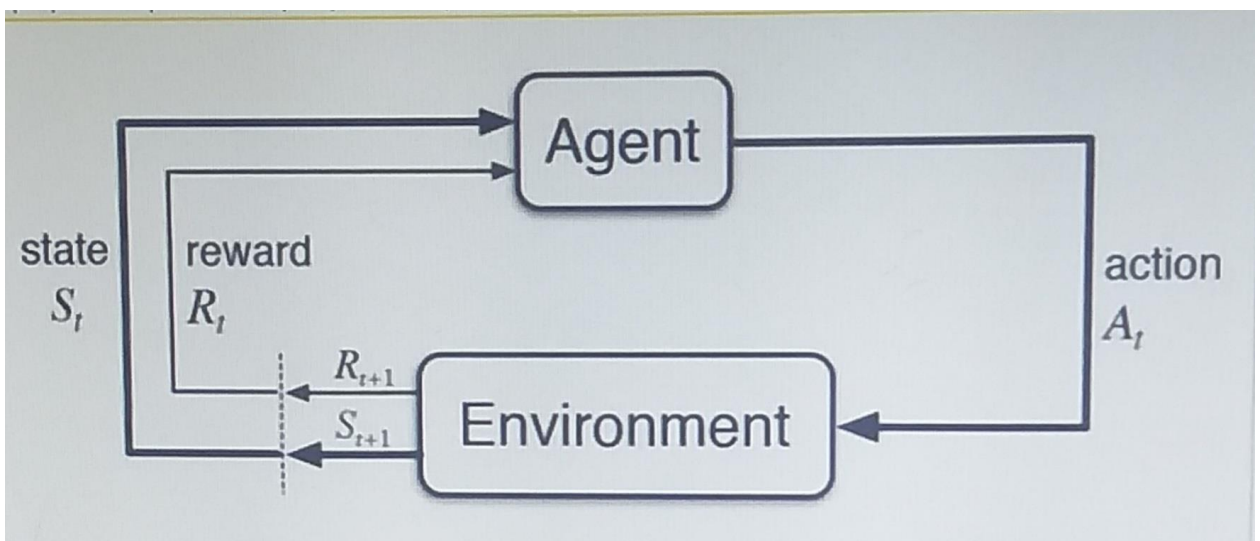
Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals. In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error.

INTRODUCTION

- Consider building a **learning robot**. The robot, or **agent**, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.
- Its task is to learn a control strategy, or **policy**, for choosing actions that achieve its goals.
- The goals of the agent can be defined by a **reward function** that assigns a numerical value to each distinct action the agent may take from each distinct state.
- This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot.
- The **task** of the robot is to perform sequences of actions, observe their consequences, and learn a control policy.
- The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Example:

- A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
- The robot may have a goal of docking onto its battery charger whenever its battery level is low.
- The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger



the difference table between RL and Supervised learning is given below:

Reinforcement Learning	Supervised Learning(ACTIVE LEARNING)
RL works by interacting with the environment.	Supervised learning works on the existing dataset.
The RL algorithm works like the human brain works when making some decisions.	Supervised Learning works as when a human learns t the supervision of a guide.
There is no labeled dataset is present	The labeled dataset is present.
No previous training is provided to the learning agent.	Training is provided to the algorithm so that it can pre output.
RL helps to take decisions sequentially.	In Supervised learning, decisions are made when given.

Types of Reinforcement: There are two types of Reinforcement:

1. **Positive –**

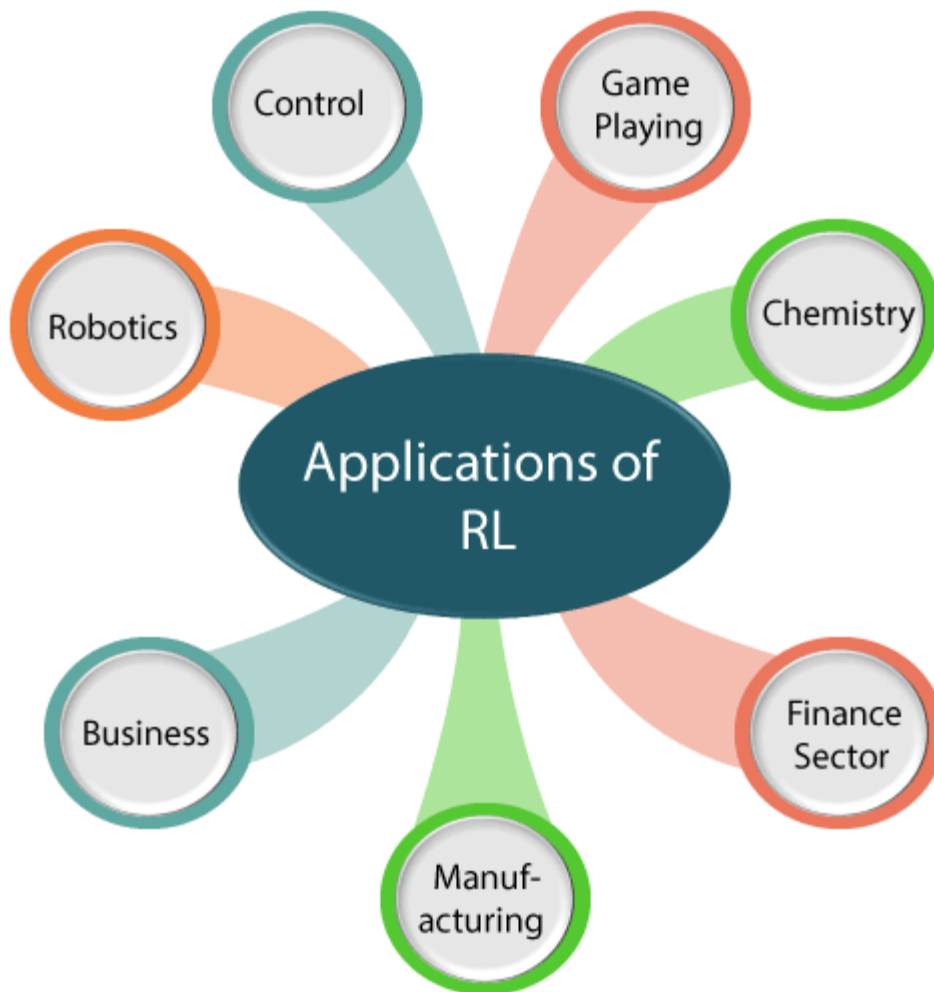
Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.

2. **Negative –**

Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.

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.



1. Robotics:

- a. RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.

Control:

- . RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.

Game Playing:

- . RL can be used in **Game playing** such as tic-tac-toe, chess, etc.

Chemistry:

- . RL can be used for optimizing the chemical reactions.

Business:

- . RL is now used for business strategy planning.

Manufacturing:

- . In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.

Finance Sector:

- . The RL is currently used in the finance sector for evaluating trading strategies.

Terms used in Reinforcement Learning

- o **Agent():** An entity that can perceive/explore the environment and act upon it.
- o **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- o **Action():** Actions are the moves taken by an agent within the environment.
- o **State():** State is a situation returned by the environment after each action taken by the agent.
- o **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- o **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- o **Value():** It is expected long-term return with the discount factor and opposite to the short-term reward.
- o **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

From the above discussion, we can say that Reinforcement Learning is one of the most interesting and useful parts of Machine learning. In RL, the agent explores the environment by exploring it without any human intervention. It is the main learning algorithm that is used in Artificial Intelligence. But there are some cases where it should not be used, such as if you have enough data to solve the problem, then other ML algorithms can be used more efficiently. The main issue with the RL algorithm is that some of the parameters may affect the speed of the learning, such as delayed feedback.

Q-learning OR Quality learning:

is a model-free **reinforcement learning** algorithm to **learn** the **value** of an action in a particular state. ... "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state

What is q-learning?

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

What's 'Q'?

The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

Create a q-table

When q-learning is performed we create what's called a *q-table* or matrix that follows the shape of $[state, action]$ and we initialize our values to zero. We then update and store our *q-values* after an episode.

Q-learning and making updates

The next step is simply for the agent to interact with the environment and make updates to the state action pairs in our q-table $Q[state, action]$.

Taking Action: Explore or Exploit

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as **exploiting** since we use the information we have available to us to make a decision.

The second way to take action is to act randomly. This is called **exploring**. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process.

Q-Values or Action-Values: Q-values are defined for states and actions. $Q(S,A)$ is an estimation of how good is it to take the action A at the state S. This estimation of $Q(S,A)$ will be iteratively computed using the **TD- Update rule**

Rewards and Episodes: An agent over the course of its lifetime starts from a start state, makes a number of transitions from its current state to a next state based on its choice of action and also the environment the agent is interacting in. At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point of time the agent ends up in one of the terminating states that means there are no further transition possible. This is said to be the completion of an episode.

Temporal Difference or TD-Update:

The Temporal Difference or TD-Update rule can be represented as follows

$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

This update rule to estimate the value of Q is applied at every time step of the agents interaction with the environment. The terms used are explained below. :

S: Current State of the agent.

A: Current Action Picked according to some policy.

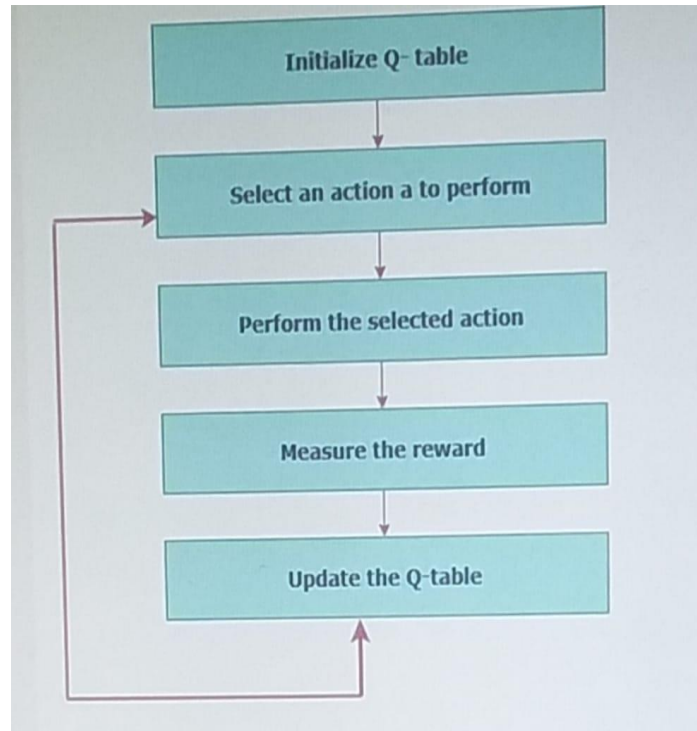
S' : Next State where the agent ends up.

A' : Next best action to be picked using current Q-value estimation, i.e. pick the action with the maximum Q-value in the next state.

R : Current Reward observed from the environment in Response of current action.

GAMMA=(>0 and ≤ 1) : Discounting Factor for Future Rewards. Future rewards are less valuable than current rewards so they must be discounted. Since Q-value is an estimation of expected rewards from a state, discounting rule applies here as well.

ALPHA : Step length taken to update the estimation of $Q(S, A)$.



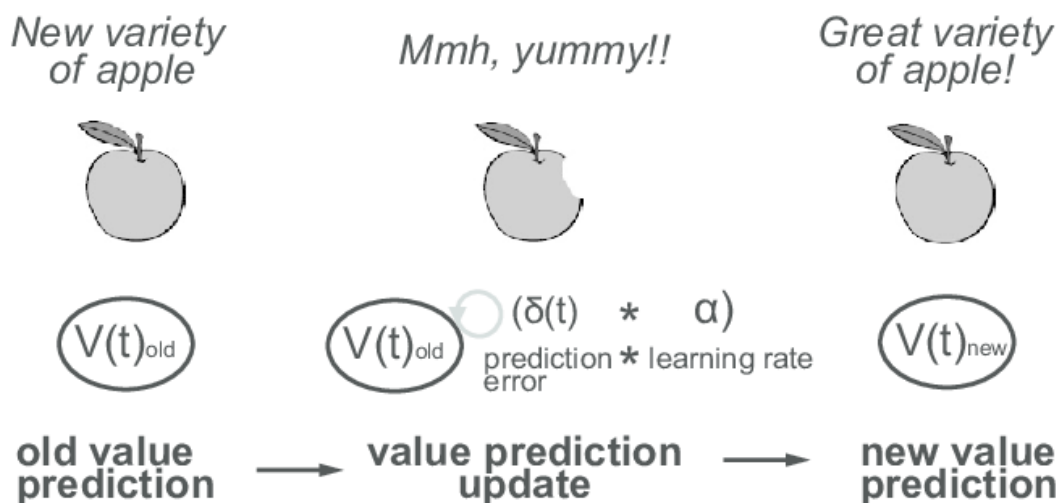
This is one of the form of Learning method in which the agent moves to one state and other state ...and predicts the reward expected

It is a supervised **learning** process **in** which the **training** signal **for** a prediction is a future prediction. **TD** algorithms are often used **in reinforcement learning** to predict a measure **of** the total amount **of** reward expected over the future.

TD Learning focuses on predicting a variable's future value in a sequence of states. Temporal difference learning was a major breakthrough in solving the problem of reward prediction. You could say that it employs a mathematical trick that allows it to replace complicated reasoning with a simple learning procedure that can be used to generate the very same results.

The trick is that rather than attempting to calculate the total future reward, temporal difference learning just attempts to predict the combination of immediate reward and its own reward prediction at the next moment in time. Now when the next moment comes and brings fresh information with it, the new prediction is compared with the expected prediction. If these two predictions are different from each other, the TD algorithm will calculate how different the predictions are from each other and make use of this temporal difference to adjust the old prediction toward the new prediction.

Temporal difference learning



The temporal difference algorithm always aims to bring the expected prediction and the new prediction together, thus matching expectations with reality and gradually increasing the accuracy of the entire chain of prediction.

Temporal Difference Learning aims to predict a combination of the immediate reward and its own reward prediction at the next moment in time.

In TD Learning, the training signal for a prediction is a future prediction. temporal difference methods tend to adjust predictions to match later, more accurate, predictions for the future, much before the final outcome is clear and know. This is essentially a type of bootstrapping.

For a **action to take given a particular state**. The distribution $\pi(a|s)$ is used for a stochastic policy and a mapping function $\pi:S \rightarrow A$ is used for a deterministic policy (fixed policy or rule) and non deterministic (customized policy with time bound) where S is the set of possible states and A is the set of possible actions.

Dynamic Programming

Dynamic programming algorithms solve a category of problems called planning problems. Herein given the complete model and specifications of the environment , we can successfully find an optimal policy for the agent to follow. It contains two main steps:

1. Break the problem into subproblems and solve it
2. Solutions to subproblems are cached or stored for reuse to find overall optimal solution to the problem at hand

- **Dynamic programming** is a method for solving complex problems by breaking them down into sub-problems. The solutions to the sub-problems are combined to solve overall problem.

MDP is used in Dynamic programming to solve complex task

Markov Decision Process or MDP, is used to formalize the reinforcement learning problems. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

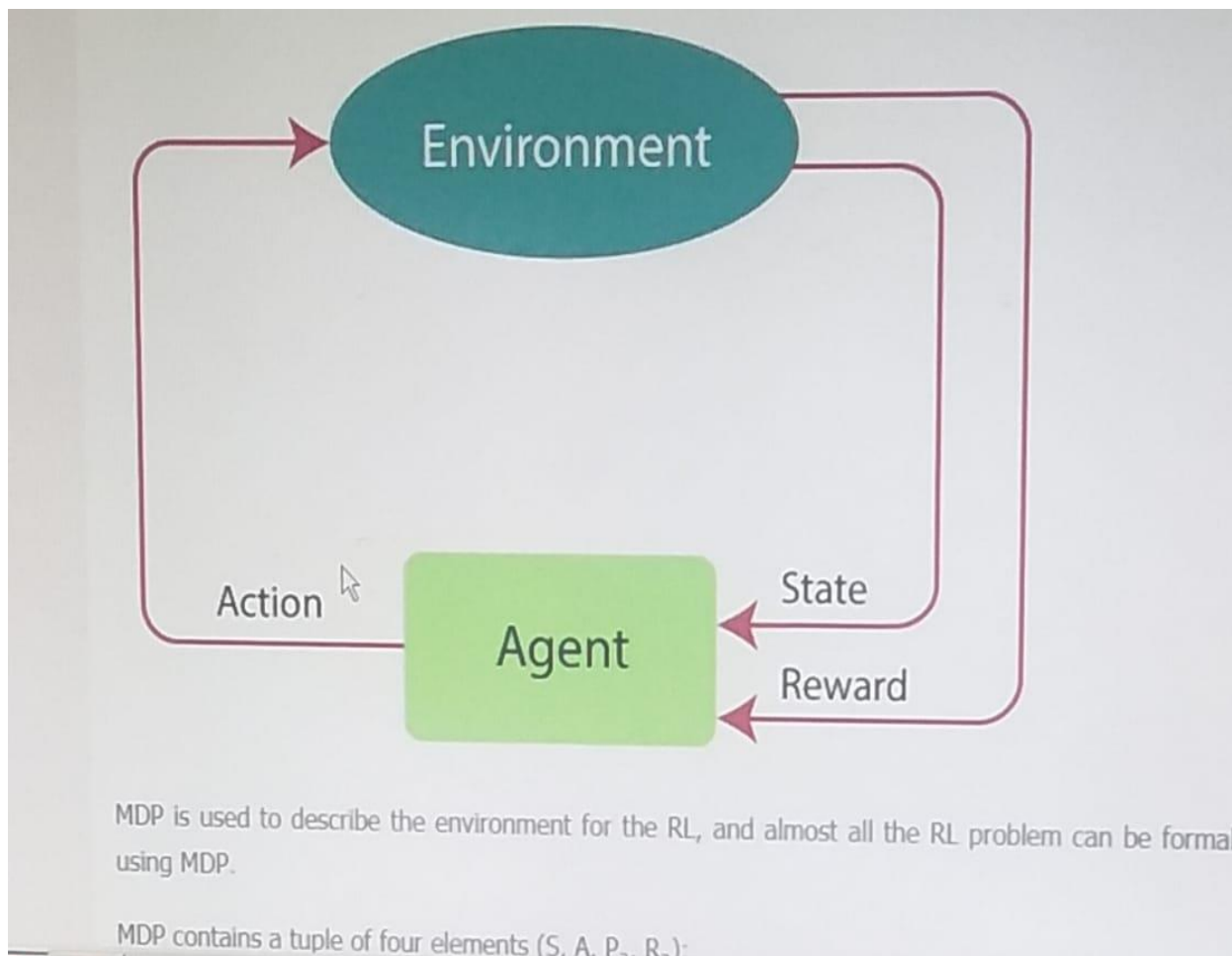
MDP contains a tuple of four elements (S, A, P_a, R_a) :

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S' , due to action a .
- Probability P_a .

MDP uses **Markov property**, and to better understand the MDP, we need to learn about it.

Markov Property:

It says that *"If the agent is present in the current state S_1 , performs an action a_1 and move to the state s_2 , then the state transition from s_1 to s_2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."*



UNIT 5

Analytical Learning-1

Inductive learning methods such as neural network and decision tree learning require a certain number of training examples to achieve a given level of accuracy.

Analytical learning uses prior knowledge and deductive reasoning to augment the information provided by the training examples. This chapter considers an analytical learning method called **explanation-based learning (EBL)**

In explanation-based learning, prior knowledge is used to analyze, or explain, how each observed training example satisfies the target concept.

This explanation is then used to distinguish the relevant features of the training example from the irrelevant features. Explanation-based learning has been successfully applied to learning search control rules for a variety of planning and scheduling tasks.

Inductive Learning methods: that is, methods that generalize from observed training examples by identifying features that empirically distinguish positive from negative training examples.

Decision tree learning, neural network learning, inductive logic programming, and genetic algorithms are all examples of inductive methods that operate in this fashion. The key practical limit on these inductive learners is that they perform poorly when insufficient data is available.

Explanation-based learning is one such approach. It uses prior knowledge to analyze, or explain, each training example in order to infer which example features are relevant to the target function and which are irrelevant. These explanations enable it to generalize more accurately than inductive systems that rely on the data.

Explanation based learning uses prior knowledge to reduce the complexity of the hypothesis space to be searched, thereby reducing sample complexity and improving generalization accuracy of the learner it is supported with evidence called domain theory(it is refers to the evidence that supports the prior data) .

EXAMPLE OF EXPLANATION: Chess Game

two moves.” Figure 11.1 shows a positive training example of this target concept. Inductive learning methods could, of course, be employed to learn this target concept. However, because the chessboard is fairly complex (there are 32 pieces that may be on any of 64 squares), and because the particular patterns that capture this concept are fairly subtle (involving the relative positions of various pieces on the board), we would have to provide thousands of training examples similar to the one in Figure 11.1 to expect an inductively learned hypothesis to generalize correctly to new situations.

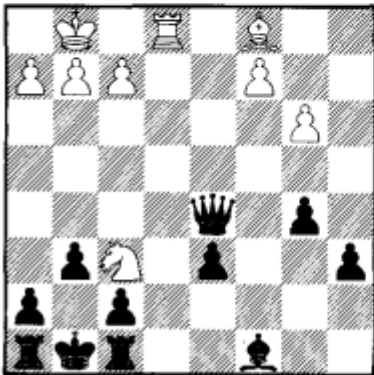


FIGURE 11.1

A positive example of the target concept “chess positions in which black will lose its queen within two moves.” Note the white knight is simultaneously attacking both the black king and queen. Black must therefore move its king, enabling white to capture its queen.

Inductive and Analytical Learning Problems

The essential difference between analytical and inductive learning methods is that they assume two different formulations of the learning problem:

- In inductive learning, the learner is given a hypothesis space H from which it must select an output hypothesis, and a set of training examples $D = \{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$ where $f(x_i)$ is the target value for the instance x_i . The desired output of the learner is a hypothesis h from H that is consistent with these training examples.
- In analytical learning, the input to the learner includes the same hypothesis space H and training examples D as for inductive learning. In addition, the learner is provided an additional input: A *domain theory* B consisting of background knowledge that can be used to explain observed training examples. The desired output of the learner is a hypothesis h from H that is consistent with both the training examples D and the domain theory B .

Let us introduce in detail a second example of an analytical learning problem--one that we will use for illustration throughout this chapter. Consider an instance space X in which each instance is a pair of physical objects. Each of the two physical objects in the instance is described by the predicates Color, Volume, Owner, Material, Type, and Density, and the relationship between the two objects is described by the predicate On.

Given this instance space, the task is to learn the target concept "pairs of physical objects, such that one can be stacked safely on the other," denoted by the predicate SafeToStack(x,y). Learning this target concept might be useful, for example, to a robot system that has the task of storing various physical objects within a limited workspace. The full definition of this analytical learning task is given in

Table 11.1.

Given:

- Instance space X : Each instance describes a pair of objects represented by the predicates *Type*, *Color*, *Volume*, *Owner*, *Material*, *Density*, and *On*.
- Hypothesis space H : Each hypothesis is a set of Horn clause rules. The head of each Horn clause is a literal containing the target predicate *SafeToStack*. The body of each Horn clause is a conjunction of literals based on the same predicates used to describe the instances, as well as the predicates *LessThan*, *Equal*, *GreaterThan*, and the functions *plus*, *minus*, and *times*. For example, the following Horn clause is in the hypothesis space:

$$\text{SafeToStack}(x, y) \leftarrow \text{Volume}(x, vx) \wedge \text{Volume}(y, vy) \wedge \text{LessThan}(vx, vy)$$

- Target concept: *SafeToStack*(x, y)
- Training Examples: A typical positive example, *SafeToStack*(*Obj1*, *Obj2*), is shown below:

<i>On</i> (<i>Obj1</i> , <i>Obj2</i>)	<i>Owner</i> (<i>Obj1</i> , <i>Fred</i>)
<i>Type</i> (<i>Obj1</i> , <i>Box</i>)	<i>Owner</i> (<i>Obj2</i> , <i>Louise</i>)
<i>Type</i> (<i>Obj2</i> , <i>Endtable</i>)	<i>Density</i> (<i>Obj1</i> , 0.3)
<i>Color</i> (<i>Obj1</i> , <i>Red</i>)	<i>Material</i> (<i>Obj1</i> , <i>Cardboard</i>)
<i>Color</i> (<i>Obj2</i> , <i>Blue</i>)	<i>Material</i> (<i>Obj2</i> , <i>Wood</i>)
<i>Volume</i> (<i>Obj1</i> , 2)	

- Domain Theory B :

SafeToStack(x, y) \leftarrow \neg *Fragile*(y)
SafeToStack(x, y) \leftarrow *Lighter*(x, y)
Lighter(x, y) \leftarrow *Weight*(x, wx) \wedge *Weight*(y, wy) \wedge *LessThan*(wx, wy)
Weight(x, w) \leftarrow *Volume*(x, v) \wedge *Density*(x, d) \wedge *Equal*($w, \text{times}(v, d)$)
Weight($x, 5$) \leftarrow *Type*($x, \text{Endtable}$)
Fragile(x) \leftarrow *Material*(x, Glass)
 ...

Determine:

- A hypothesis from H consistent with the training examples and domain theory.

TABLE 11.1

An analytical learning problem: *SafeToStack*(x, y).

Horn clauses express a subset of statements of first-order logic.

Horn clause supports atleast one positive and negative literal representation

A Horn Clause is a clause with **at most one positive literal**, it is thus either: A single positive literal, which is regarded as a fact, One or more negative literals, with no positive literal.

Learning With Perfect Domain Theories: PROLOG-EBG-PROGRAM LOGIC

This section presents an algorithm called PROLOG-EBG (Kedar-Cabelli and McCarty 1987) that is representative of several explanation-based learning algorithms.

PROLOG-EBG is a **sequential covering algorithm**. Prolog stands for programming in logic. it is a **logic programming language for artificial intelligence**. An artificial intelligence developed in Prolog will examine the link between a fact, a true statement, and a rule, a conditional statement, in order to come up with a question, or end objective.

PROLOG-EBG is guaranteed to output a hypothesis (set of rules) that is itself correct and that covers the observed positive training examples. For any set of training examples, the hypothesis output by PROLOG-EBG constitutes a set of logically sufficient conditions for the target concept, according to the domain theory.

A domain theory is said to be **correct** if each of its assertions is a truthful statement about the world.

A domain theory is said to be **complete** with respect to a given target concept and instance space, if the domain theory covers every positive example in the instance space.

PROLOG-EBG(*TargetConcept*, *TrainingExamples*, *DomainTheory*)

- *LearnedRules* ← {}
- *Pos* ← the positive examples from *TrainingExamples*
- for each *PositiveExample* in *Pos* that is not covered by *LearnedRules*, do
 1. *Explain*:
 - *Explanation* ← an explanation (proof) in terms of the *DomainTheory* that *PositiveExample* satisfies the *TargetConcept*
 2. *Analyze*:
 - *SufficientConditions* ← the most general set of features of *PositiveExample* sufficient to satisfy the *TargetConcept* according to the *Explanation*.
 3. *Refine*:
 - *LearnedRules* ← *LearnedRules* + *NewHornClause*, where *NewHornClause* is of the form
$$\textit{TargetConcept} \leftarrow \textit{SufficientConditions}$$
- Return *LearnedRules*

TABLE 11.2

The explanation-based learning algorithm PROLOG-EBG. For each positive example that is not yet covered by the set of learned Horn clauses (*LearnedRules*), a new Horn clause is created. This new Horn clause is created by (1) explaining the training example in terms of the domain theory, (2) analyzing this explanation to determine the relevant features of the example, then (3) constructing a new Horn clause that concludes the target concept when this set of features is satisfied.

PROLOGEBG computes the most general rule that can be justified by the explanation, by computing the weakest preimage of the explanation

PROLOG-EBG computes the weakest preimage of the target concept with respect to the explanation, using a general procedure called regression (Waldinger 1977). The regression procedure operates on a domain theory represented by an arbitrary set of Horn clauses.

It works iteratively backward through the explanation, first computing the weakest preimage of the target concept with respect to the final proof step in the explanation, then computing the weakest preimage of the resulting expressions with respect to the preceding step, and so on.

The procedure terminates when it has iterated over all steps :in the explanation, yielding the weakest precondition of the target concept with respect to the literals at the leaf nodes of the explanation

Definition: The **weakest preimage** of a conclusion C with respect to a proof P is the most general set of initial assertions A , such that A entails C according to P .

For example, the weakest preimage of the target concept $SafeToStack(x,y)$, with respect to the explanation from Table 11.1, is given by the body of the following rule. This is the most general rule that can be justified by the explanation of Figure 11.2:

$$SafeToStack(x, y) \leftarrow Volume(x, vx) \wedge Density(x, dx) \wedge \\ Equal(wx, times(vx, dx)) \wedge LessThan(wx, 5) \wedge \\ Type(y, Endtable)$$

Notice this more general rule does not require the specific values for $Volume$ and $Density$ that were required by the first rule. Instead, it states a more general constraint on the values of these attributes.

PROLOG-EBG computes the weakest preimage of the target concept with respect to the explanation, using a general procedure called *regression* (Waldinger 1977). The regression procedure operates on a domain theory represented by an arbitrary set of Horn clauses. It works iteratively backward through the explanation, first computing the weakest preimage of the target concept with respect to the final proof step in the explanation, then computing the weakest preimage of the resulting expressions with respect to the preceding step, and so on. The procedure terminates when it has iterated over all steps in the explanation, yielding the weakest precondition of the target concept with respect to the literals at the leaf nodes of the explanation.

A trace of this regression process is illustrated in Figure 11.3. In this figure, the explanation from Figure 11.2 is redrawn in standard (nonitalic) font. The frontier of regressed expressions created at each step by the regression procedure is shown underlined in italics. The process begins at the root of the tree, with the frontier initialized to the general target concept *SafeToStack(x,y)*. The first step is to compute the weakest preimage of this frontier expression with respect to the final (top-most) inference rule in the explanation. The rule in this case is $SafeToStack(x, y) \leftarrow Lighter(x, y)$, so the resulting weakest preimage is *Lighter(x, y)*. The process now continues by regressing the new frontier, *Lighter(x, y)*, through the next Horn clause in the explanation, resulting in the regressed expressions *Weight(x, wx), LessThan(wx, wy), Weight(y, wy)*. This indicates that the explanation will hold for any *x* and *y* such that the weight *wx* of *x* is less than the weight *wy* of *y*. The regression of this frontier back to the leaf nodes of the explanation continues in this step-by-step fashion, finally

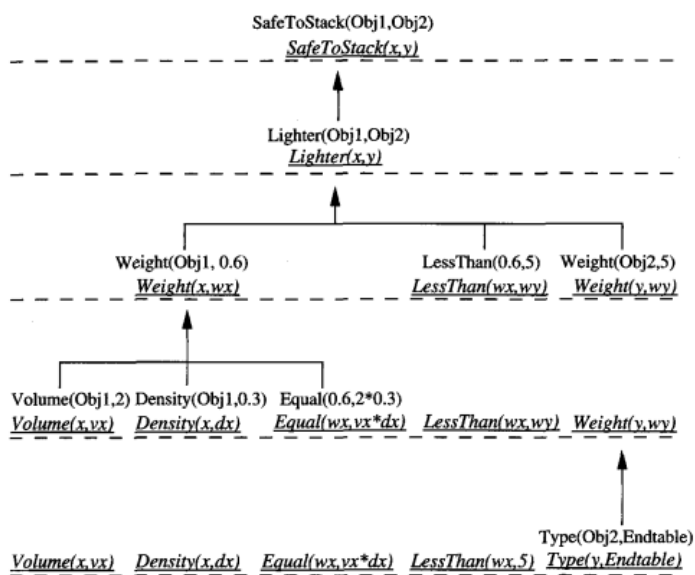


FIGURE 11.3

Computing the weakest preimage of *SafeToStack(Obj1, Obj2)* with respect to the explanation. The target concept is regressed from the root (conclusion) of the explanation, down to the leaves. At each step (indicated by the dashed lines) the current frontier set of literals (underlined in italics) is regressed backward over one rule in the explanation. When this process is completed, the conjunction of resulting literals constitutes the weakest preimage of the target concept with respect to the explanation. This weakest preimage is shown by the italicized literals at the bottom of the figure.

In prolog, We declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (,). Their relation is expressed at the start and outside the parenthesis. Every fact/rule ends with a dot (.). So, a typical prolog fact goes as follows :

Format : relation(entity1, entity2,k'th entity).

Example :

friends(raju, mahesh).
 singer(sonu).
 odd_number(5).

Explanation :

These facts can be interpreted as :

raju and mahesh are friends.

sonu is a singer.

5 is an odd number.

Key Features of Prolog used in AI:

- 1. Unification** : The basic idea is, can the given terms be made to represent the same structure.
- 2. Backtracking** : When a task fails, prolog traces backwards and tries to satisfy previous task.
- 3. Recursion** : Recursion is the basis for any search in program.

Running queries :

A typical prolog query can be asked as :

Query 1 : ?- singer(sonu).

Output : Yes.

Explanation : As our knowledge base contains the above fact, so output was 'Yes', otherwise it would have been 'No'.

Query 2 : ?- odd_number(7).

Output : No.

Explanation : As our knowledge base does not contain the above fact, so output was 'No'.

11.3 REMARKS ON EXPLANATION-BASED LEARNING

As we saw in the above example, PROLOG-EBG conducts a detailed analysis of individual training examples to determine how best to generalize from the specific example to a general Horn clause hypothesis. The following are the key properties of this algorithm.

- Unlike inductive methods, PROLOG-EBG produces *justified* general hypotheses by using prior knowledge to analyze individual examples.
- The explanation of how the example satisfies the target concept determines which example attributes are relevant: those mentioned by the explanation.
- The further analysis of the explanation, regressing the target concept to determine its weakest preimage with respect to the explanation, allows deriving more general constraints on the values of the relevant features.
- Each learned Horn clause corresponds to a sufficient condition for satisfying the target concept. The set of learned Horn clauses covers the positive training examples encountered by the learner, as well as other instances that share the same explanations.
- The generality of the learned Horn clauses will depend on the formulation of the domain theory and on the sequence in which training examples are considered.
- PROLOG-EBG implicitly assumes that the domain theory is correct and complete. If the domain theory is incorrect or incomplete, the resulting learned concept may also be incorrect.

There are several related perspectives on explanation-based learning that help to understand its capabilities and limitations.

- *EBL as theory-guided generalization of examples.* EBL uses its given domain theory to generalize *rationaly* from examples, distinguishing the relevant example attributes from the irrelevant, thereby allowing it to avoid the bounds on sample complexity that apply to purely inductive learning. This is the perspective implicit in the above description of the PROLOG-EBG algorithm.
- *EBL as example-guided reformulation of theories.* The PROLOG-EBG algorithm can be viewed as a method for reformulating the domain theory into a more operational form. In particular, the original domain theory is reformulated by creating rules that (a) follow deductively from the domain theory,

Thus, in its pure form EBL involves reformulating the domain theory to produce general rules that classify examples in a single inference step.

This kind of knowledge reformulation is sometimes referred to as knowledge compilation, indicating that the transformation is an efficiency improving one

EXPLANATION-BASED LEARNING OF SEARCH CONTROL KNOWLEDGE

PRODIGY and SOAR demonstrate that explanation-based learning methods can be successfully applied to acquire search control knowledge in a variety of problem domains.

Prodigy means-ends planning strategy.

The PRODIGY architecture was initially conceived by Jaime Carbonell and Steven Minton, as an Artificial Intelligence (AI) system to test and develop ideas on the role of machine learning in planning and problem solving.

In general, learning in problem solving seemed meaningless without measurable performance improvements.

Thus, PRODIGY was created to be a testbed for the systematic investigation of the loop between learning and performance in planning systems.

As a result, PRODIGY consists of a core general- purpose planner and several learning modules that refine both the planning domain knowledge and the control knowledge to guide the search process effectively

A *planning problem* is defined by

- (1) a set of available objects of each type,
- (2) an *initial state* , and

(3) a goal statement .

The Algorithm

Table 4 shows the basic procedure to learn quality-enhancing control knowledge, in the case that a human expert provides a better plan. Steps 2, 3 and 4 correspond to the interactive plan checking module, that asks the expert for a better solution and checks for its correctness. Step 6 constructs a problem solving trace from the expert solution and obtains decision points where control knowledge is needed, which in turn become learning opportunities. Step 8 corresponds to the actual learning phase. It compares the plan trees obtained from the problem solving traces in Step 7, explains why one solution was better than the other, and builds new control knowledge. These steps are described now in detail.

-
1. Run PRODIGY with the current set of control rules and obtain a solution .
 2. Show to the expert.
Expert provides new solution possibly using as a guide.
 3. Test . If it solves the problem, continue. Else go back to step 2.
 4. Apply the plan quality evaluation function to .
If it is better than , continue. Else go back to step 2.
 5. Compute the partial order for identifying the goal dependencies between plan steps.
 6. Construct a problem solving trace corresponding to a solution that satisfies .
This determines the set of decision points in the problem solving trace where control knowledge is missing.
 7. Build the plan trees and, corresponding respectively to the search trees for and .
 8. Compare and explaining why is better than , and build control rules.
-

Table 4: Top level procedure to learn quality-enhancing control knowledge.

A second example of a general problem-solving architecture that incorporates a form of explanation-based learning is the SOAR system (Laird et al. 1986; Newell 1990). SOAR supports a broad variety of problem-solving strategies that subsumes PRODIGY'S means-ends planning strategy.

Like PRODIGY, however, SOAR learns by explaining situations in which its current search strategy leads to inefficiencies.

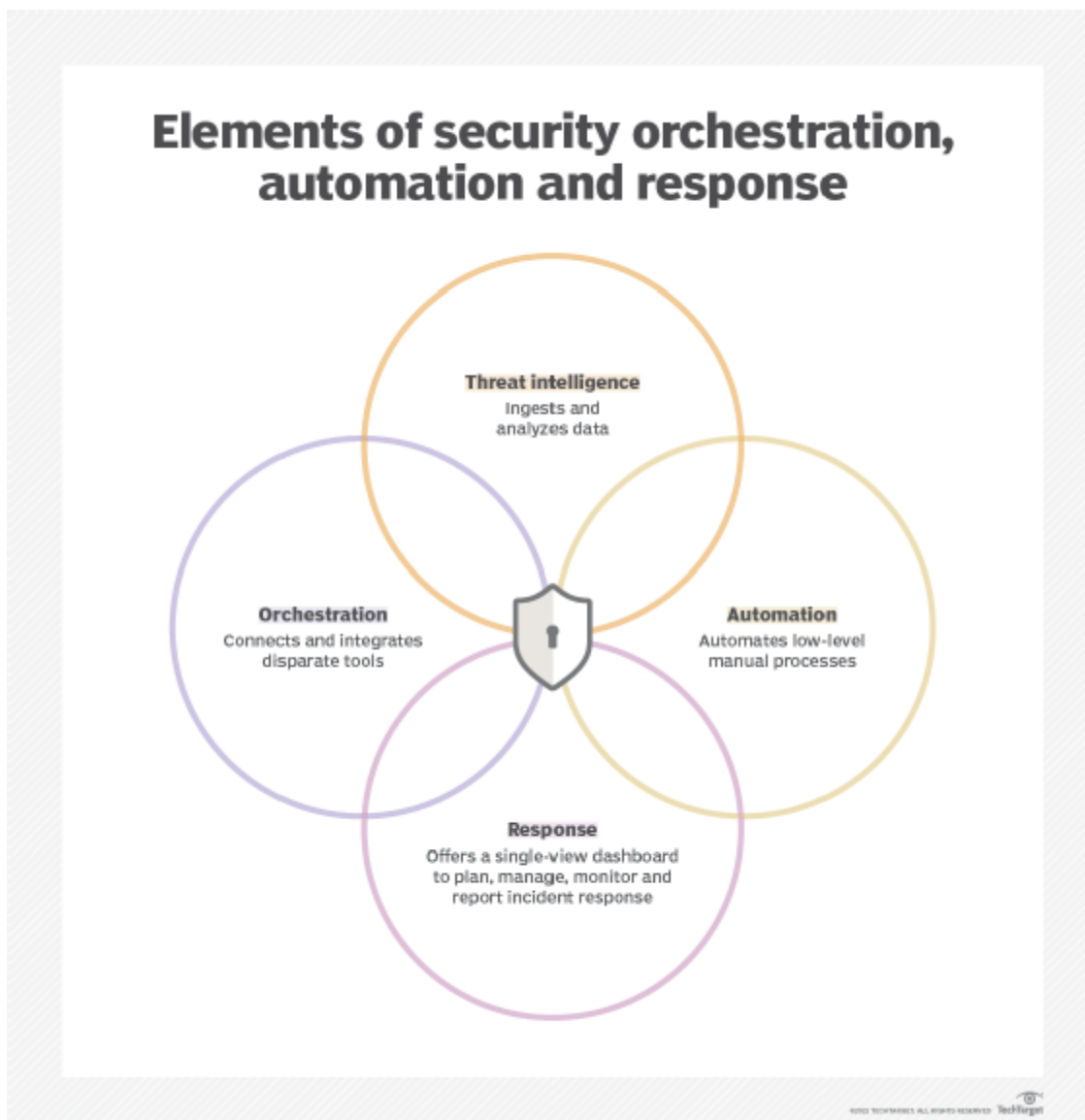
SOAR has been applied in a great number of problem domains and has also been proposed as a psychologically plausible model of human learning processes (Newell 1990).

SOAR (security orchestration, automation and response) is a stack of compatible software programs that enables an organization to collect data about security threats and respond

to [security events](#) without human assistance. The goal of using a SOAR platform is to improve the efficiency of physical and digital security operations.

What is SOAR?

SOAR platforms have three main components: security orchestration, security automation and security response.



Benefits of SOAR

SOAR platforms offer many benefits for enterprise security operations ([SecOps](#)) teams, including the following:

- **Faster incident detection and reaction times.** The volume and velocity of security threats and events are constantly increasing. SOAR's improved data context, combined with automation, can bring lower mean time to detect ([MTTD](#)) and mean time to respond (MTTR). By detecting and responding to threats more quickly, their impact can be lessened.
- **Better threat context.** By integrating more data from a wider array of tools and systems, SOAR platforms can offer more context, better analysis and up-to-date threat information.
- **Simplified management.** SOAR platforms consolidate various security systems' dashboards into a single interface. This helps SecOps and other teams by centralizing information and data handling, simplifying management and saving time.
- **Scalability.** Scaling time-consuming manual processes can be a drain on employees and even impossible to keep up with as security event volume grows. SOAR's orchestration, automation and workflows can meet scalability demands more easily.
- **Boosting analysts' productivity.** Automating lower-level threats augments SecOps and security operations center ([SOC](#)) teams' responsibilities, enabling them to prioritize tasks more effectively and respond to threats that require human intervention more quickly.

Vyasapuri, Bandlaguda, Post:Keshavgiri
Hyderabad-500005,Telangana, India
Tel:040-29880079, 86,8978380692, 9642703342
9652216001, 9550544411, Website:www.mist.ac.in
Email:principal@mist.ac.in
principal.mahaveer@gmail.com
Counseling code:MHVR, University Code:E3

MAHAVEER
INSTITUTE OF SCIENCE & TECHNOLOGY
(AN UGC AUTONOMOUS INSTITUTION)
Approved by AICTE, Affiliated to JNTU, Hyderabad
Accredited by NAAC with 'A' Grade
Recognized Under 2(f) of UGC Act 1956, ISO 9001:2015 Certified



Analytical Learning-2-

Inductive Learning	Analytical Learning
1) Model is made by implementing hypotheses	1) Model is learned by implementing domain theory.
2) Hypothesis gets the data	2) Hypothesis gets domain theory
3) Requires cost Refers to hypothesis and lear Requires a little prior knowledge	3) Refers to prior knowledge and then the domain theory to make correct decision.
4) It is a <u>slowly learning</u> it is a process where the learner discovers <u>slowly</u> by observing training data.	It is called as <u>Explanatory Based learning</u> which require a domain theory to learn proper data.
5) It decides decision based on <u>statistical data</u> or calculations.	5) It decides and can also <u>help to detect</u> reasoning <u>memory</u>
6) Ex: decision tree, neural network, genetic alg etc	6) Ex: Bayes theorem, BBN (Bayesian belief conditional prob w/w) Artificial intelligence etc
7) can fail when there is <u>incorrect data</u>	7) can fail when there is <u>incorrect domain theory</u>
8) <u>low computational cost</u>	8) <u>high computational cost</u>

Motivation for combining Inductive and Analytical approaches which address following specific properties

1. Given no domain theory, it should learn at least as effectively as purely inductive methods.
2. Given a perfect domain theory, it should learn at least as effectively as purely analytical methods.
3. Given an imperfect domain theory and imperfect training data, it should combine the two to out perform either purely inductive or purely analytical methods.
4. It should accommodate an unknown level of error in the training data.

5. It should accommodate an unknown level of error in the domain theory
INDUCTIVE-ANALYTICAL APPROACHES TO LEARNING

The Learning Problem specified as

To summarize, the learning problem considered in this chapter is

Given:

A set of training examples D , possibly containing errors

A domain theory B , possibly containing errors

A space of candidate hypotheses H

Determine:

A hypothesis that best fits the training examples and domain theory

To address this learning problem we develop a hypothesis space search combining both inductive and analytical approaches

we explore three different methods for using **prior knowledge to alter the search performed by purely inductive methods.**

1. USE PRIOR KNOWLEDGE TO DERIVE AN INITIAL HYPOTHESIS FROM WHICH TO BEGIN THE SEARCH

2. USE PRIOR KNOWLEDGE TO ALTER THE OBJECTIVE OF THE HYPOTHESIS SPACE SEARCH.

3. USING PRIOR KNOWLEDGE TO AUGMENT SEARCH STEPS

1. Use prior knowledge to derive an initial hypothesis from which to begin the search

In this approach the domain theory B is used to construct an initial hypothesis h_0 that is consistent with B . A standard inductive method is then applied, starting with the initial hypothesis h_0 .

This approach is used by the **KBANN (Knowledge-Based Artificial Neural Network)** algorithm to learn artificial neural networks.

In KBANN an initial network is first constructed so that for every possible instance, the classification assigned by the network is identical to that assigned by the domain theory.

The BACKPROPAGATION algorithm is then employed to adjust the weights of this initial network as needed to fit the training examples. It is easy to see the motivation for this technique: if the domain theory is correct, the initial hypothesis will correctly classify all the training examples and there will be no need to revise it.

However, if the initial hypothesis is found to imperfectly classify the training examples, then it will be refined inductively to improve its fit to the training examples.

The KBANN Algorithm

The KBANN algorithm exemplifies the initialize-the-hypothesis approach to using domain theories.

The input and output of KBANN are as follows:

KBANN(*Domain_Theory, Training_Examples*)

Domain_Theory: Set of propositional, nonrecursive Horn clauses.

Training_Examples: Set of (input output) pairs of the target function.

Analytical step: Create an initial network equivalent to the domain theory.

1. For each instance attribute create a network input.
2. For each Horn clause in the *Domain_Theory*, create a network unit as follows:
 - Connect the inputs of this unit to the attributes tested by the clause antecedents.
 - For each non-negated antecedent of the clause, assign a weight of W to the corresponding sigmoid unit input.
 - For each negated antecedent of the clause, assign a weight of $-W$ to the corresponding sigmoid unit input.
 - Set the threshold weight w_0 for this unit to $-(n - .5)W$, where n is the number of non-negated antecedents of the clause.
3. Add additional connections among the network units, connecting each network unit at depth i from the input layer to all network units at depth $i + 1$. Assign random near-zero weights to these additional connections.

Inductive step: Refine the initial network.

4. Apply the BACKPROPAGATION algorithm to adjust the initial network weights to fit the *Training_Examples*.
-

An Illustrative Example

Domain theory:

Cup ← *Stable, Lifiable, OpenVessel*
Stable ← *BottomIsFlat*
Lifiable ← *Graspable, Light*
Graspable ← *HasHandle*
OpenVessel ← *HasConcavity, ConcavityPointsUp*

Training examples:

	<i>Cups</i>				<i>Non-Cups</i>				
<i>BottomIsFlat</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>ConcavityPointsUp</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>Expensive</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>Fragile</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>HandleOnTop</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>HandleOnSide</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>HasConcavity</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>HasHandle</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>Light</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>MadeOfCeramic</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>MadeOfPaper</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓
<i>MadeOfStyrofoam</i>	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE 12.3

The *Cup* learning task. An approximate domain theory and a set of training examples for the target concept *Cup*.

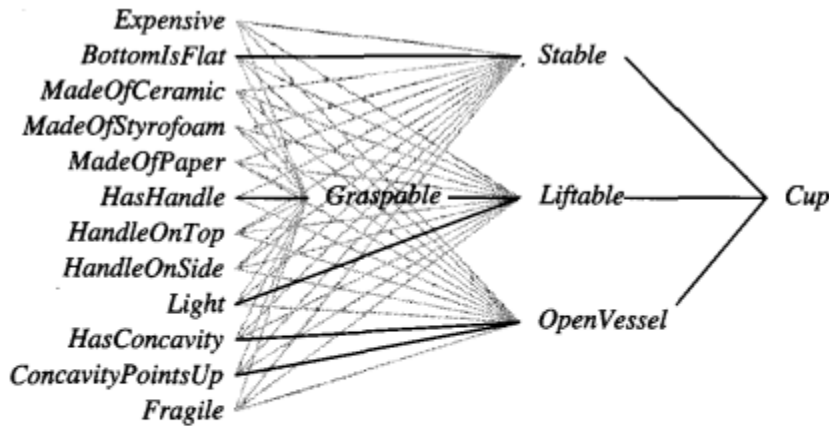


FIGURE 12.2

A neural network equivalent to the domain theory. This network, created in the first stage of the KBANN algorithm, produces output classifications identical to those of the given domain theory clauses. Dark lines indicate connections with weight W and correspond to antecedents of clauses from the domain theory. Light lines indicate connections with weights of approximately zero.

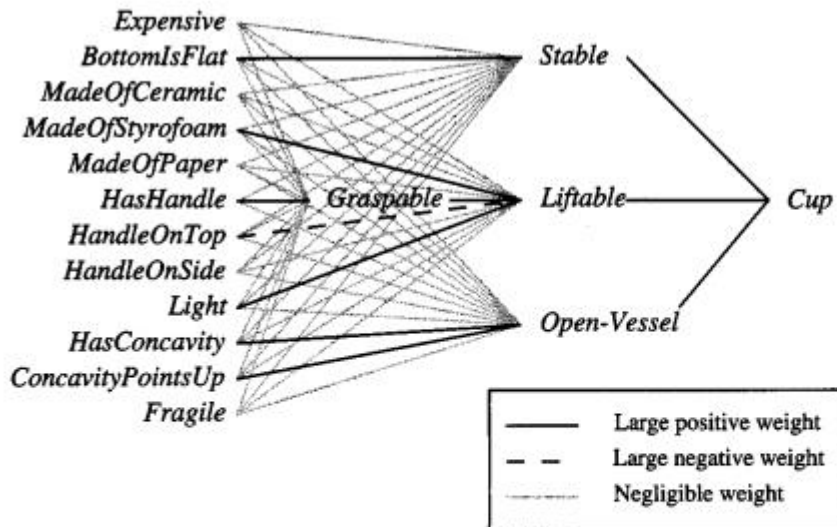


FIGURE 12.3

Result of inductively refining the initial network. KBANN uses the training examples to modify the network weights derived from the domain theory. Notice the new dependency of *Liftable* on *MadeOfStyrofoam* and *HandleOnTop*.

Remarks To summarize, KBANN analytically creates a network equivalent to the given domain theory, then inductively refines this initial hypothesis to better fit the training data.

In doing so, it modifies the network weights using backpropagation as needed to overcome inconsistencies between the domain theory and observed data.

Limitations of KBANN include the fact that it can accommodate only propositional domain theories

It is also possible for KBANN to be misled when given highly inaccurate domain theories, so that its generalization accuracy can deteriorate below the level of BACKPROPAGATION

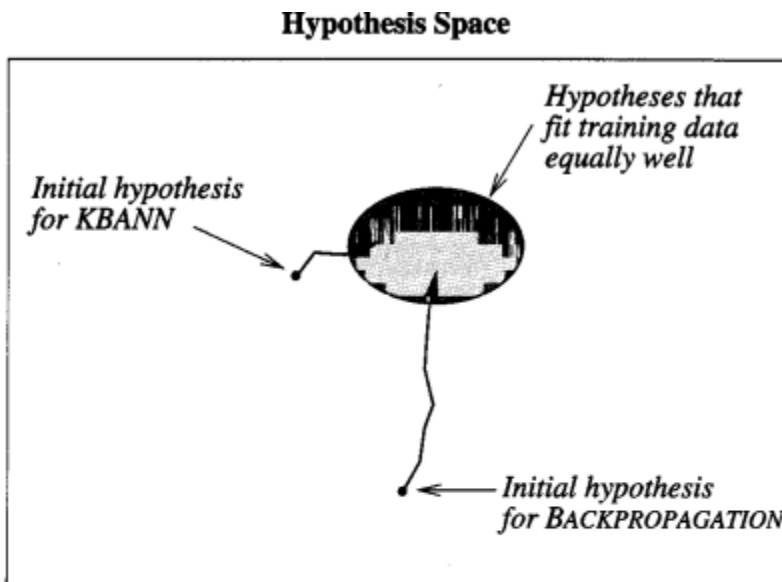


FIGURE 12.4

Hypothesis space search in KBANN. KBANN initializes the network to fit the domain theory, whereas BACKPROPAGATION initializes the network to small random weights. Both then refine the weights iteratively using the same gradient descent rule. When multiple hypotheses can be found that fit the training data (shaded region), KBANN and BACKPROPAGATION are likely to find different hypotheses due to their different starting points.

2.USING PRIOR KNOWLEDGE TO ALTER THE SEARCH OBJECTIVE

In this approach, the goal criterion G is modified to require that the output hypothesis fits the domain theory as well as the training examples.

For example, the **EBNN(EXPLANATION BASED NEURAL NETWORK)** system described below learns neural networks in this way. Whereas inductive learning of neural networks performs gradient descent search to minimize the squared error of the network over the training data, EBNN performs gradient descent to optimize a different criterion. This modified criterion includes an additional term that measures the error of the learned network relative to the domain theory.

The TANGENTPROP Algorithm TANGENTPROP (Simard et al. 1992) accommodates domain knowledge expressed as derivatives of the target function with respect to transformations of its inputs. Consider a learning task involving an instance space X and target function f .

The TANGENTPROP algorithm assumes various training derivatives of the target function are also provided. For example, if each instance x_i is described by a single real value, then each training example may be of the form $(x_i, f(x_i), q_i)$. Here q_i denotes the derivative of the target function f with respect to x , evaluated at the point $x = x_i$.

To develop an intuition for the benefits of providing training derivatives as well as training values during learning, consider the simple learning task depicted in Figure

The task is to learn the target function f shown in the leftmost plot of the figure, based on the three training examples shown: $(x_1, f(x_1))$, $(x_2, f(x_2))$, and $(x_g, f(x_g))$.

Given these three training examples, the BACKPROPAGATION algorithm can be expected to hypothesize a smooth function, such as the function g depicted in the middle plot of the figure. The rightmost plot shows the effect of

providing training derivatives, or slopes, as additional information for each training example (e.g., $(x_i, f(x_i), q_i)$). By fitting both the training values $f(x_i)$ and these training derivatives q_i , the learner has a better chance to correctly generalize from the sparse training data.

To summarize, the impact of including the training derivatives is to override the usual syntactic inductive bias of BACKPROPAGATION that favors a smooth interpolation between points, replacing it by explicit input information about required derivatives. The resulting hypothesis h shown in the rightmost plot of the figure provides a much more accurate estimate of the true target function f .

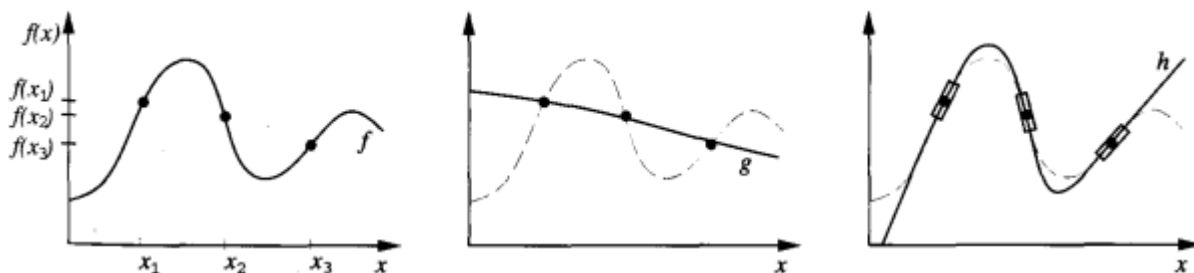


FIGURE 12.5

Fitting values and derivatives with TANGENTPROP. Let f be the target function for which three examples $(x_1, f(x_1))$, $(x_2, f(x_2))$, and $(x_3, f(x_3))$ are known. Based on these points the learner might generate the hypothesis g . If the derivatives are also known, the learner can generalize more accurately h .

Each transformation must be of the form $s_j(a, x)$ where a_j is a continuous parameter, where s_j is differentiable, and where $s_j(0, x) = x$ (e.g., for rotation of zero degrees the transformation is the identity function). For each such transformation, $s_j(a, x)$,

In the Figure one $f(x)$ are the hypothesis and x_1, x_2, x_3 are the instances and these instances fit to proper hypothesis shown in first figure and in second fig we can see the instances classified and machine learns to fit to proper hypothesis by doing necessary modification by using

TANGENTPROP considers the squared error between the specified training derivative and the actual derivative of the learned neural network. The modified error function is

$$E = \sum_i \left[(f(x_i) - \hat{f}(x_i))^2 + \mu \sum_j \left(\frac{\partial f(s_j(\alpha, x_i))}{\partial \alpha} - \frac{\partial \hat{f}(s_j(\alpha, x_i))}{\partial \alpha} \right)_{\alpha=0}^2 \right]$$

where μ is a constant provided by the user to determine the relative importance of fitting training values versus fitting training derivatives.

Notice the first term in this definition of E is the original squared error of the network versus training values, and the second term is the squared error in the network versus training derivatives.

In the third figure we can see the instances are classified properly and maintaining accuracy.

An Illustrative Example

Simard et al. (1992) present results comparing the generalization accuracy of TANGENTPROP and purely inductive BACKPROPAGATION for the problem of recognizing handwritten characters. More specifically, the task in this case is to label images containing a single digit between 0 and 9. In one experiment, both TANGENTPROP and BACKPROPAGATION were trained using training sets of varying size, then evaluated based on their performance over a separate test set of 160 examples. The prior knowledge given to TANGENTPROP was the fact that the classification of the digit is invariant of vertical and horizontal translation of the image (i.e., that the derivative of the target function was 0 with respect to these transformations). The results, shown in Table 12.4, demonstrate the ability of TANGENTPROP using this prior knowledge to generalize more accurately than purely inductive BACKPROPAGATION.

Training set size	Percent error on test set	
	TANGENTPROP	BACKPROPAGATION
10	34	48
20	17	33
40	7	18
80	4	10
160	0	3
320	0	0

TABLE 12.4
 Generalization accuracy for TANGENTPROP and BACKPROPAGATION, for handwritten digit recognition. TANGENTPROP generalizes more accurately due to its prior knowledge that the identity of the digit is invariant of translation. These results are from Simard et al. (1992).

Remarks To summarize, TANGENTPROP uses prior knowledge in the form of desired derivatives of the target function with respect to transformations of its inputs.

It combines this prior knowledge with observed training data, by minimizing an objective function that measures both the network's error with respect to the training example values (fitting the data) and its error with respect to the desired derivatives (fitting the prior knowledge).

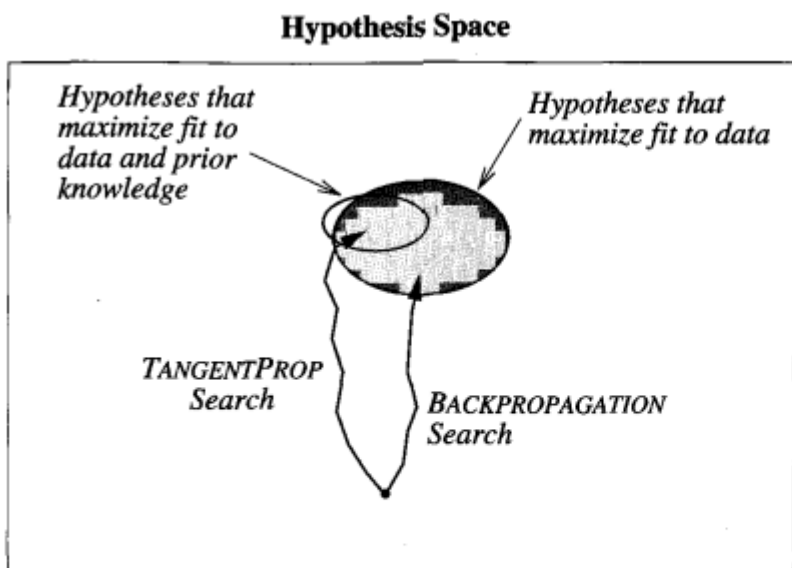


FIGURE 12.6

Hypothesis space search in TANGENTPROP. TANGENTPROP initializes the network to small random weights, just as in BACKPROPAGATION. However, it uses a different error function to drive the gradient descent search. The error used by TANGENTPROP includes both the error in predicting training *values* and in predicting the training *derivatives* provided as prior knowledge.

It is interesting to compare the search through hypothesis space (weight space) performed by TANGENTPROP, KBANN, and BACKPROPAGATION.

TANGENTPROP incorporates prior knowledge to influence the hypothesis search by altering the objective function to be minimized by gradient descent

TANGENTPROP objective will be a subset of those satisfying the weaker BACKPROPAGATION objective. The difference between these two sets of final hypotheses is the set of incorrect hypotheses that will be considered by BACKPROPAGATION, but ruled out by TANGENTPROP due to its prior knowledge.

The EBNN Algorithm

The **EBNN (Explanation-Based Neural Network learning)** algorithm (Mitchell and Thrun 1993a; Thrun 1996) builds on the TANGENTPROP algorithm in two significant ways.

First, instead of relying on the user to provide training derivatives, EBNN computes training derivatives itself for each observed training example. These training derivatives are calculated by explaining each training example in terms of a given domain theory, then extracting training derivatives from this explanation.

Second, EBNN addresses the issue of how to weight the relative importance of the inductive and analytical components of learning .

The inputs to EBNN include (1) a set of training examples of the form $(x_i, f(x_i))$ with no training derivatives provided, and (2) a domain theory analogous to that used in explanation-based learning and in KBANN, but represented by a set of previously trained neural networks The output of EBNN is a new neural network that approximates the target function f . This learned network is trained to fit both the training examples $(x_i, f(x_i))$ and training derivatives of f extracted from the domain theory. Fitting the training examples $(x_i, f(x_i))$ constitutes the inductive component of learning, whereas fitting the training derivatives extracted from the domain theory provides the analytical component.

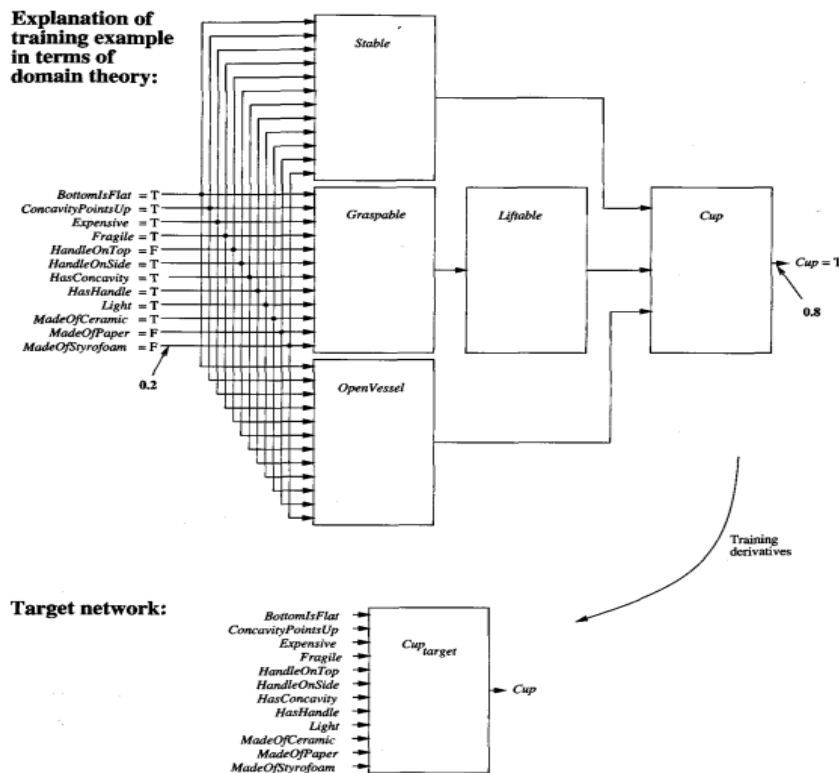


FIGURE 12.7
 Explanation of a training example in EBNN. The explanation consists of a prediction of the target function value by the domain theory networks (top). Training derivatives are extracted from this explanation in order to train the separate target network (bottom). Each rectangular block represents a distinct multilayer neural network.

(i.e., *False*), and the domain theory prediction is that $Cup = 0.8$ (i.e., *True*). EBNN calculates the partial derivative of this prediction with respect to each instance feature, yielding the set of derivatives

$$\left[\frac{\partial Cup}{\partial BottomIsFlat}, \frac{\partial Cup}{\partial ConcavityPointsUp}, \dots, \frac{\partial Cup}{\partial MadeOfStyrofoam} \right]_{x=x_i}$$

Remarks To summarize, the EBNN algorithm uses a domain theory expressed as a set of previously learned neural networks, together with a set of training examples, to train its output hypothesis (the target network).

For each training example EBNN uses its domain theory to explain the example, then extracts training derivatives from this explanation.

For each attribute of the instance, a training derivative is computed that describes how the target function value is influenced by a small change to this attribute value, according to the domain theory.

USING PRIOR KNOWLEDGE TO AUGMENT SEARCH STEPS

The two previous sections examined two different roles for prior knowledge in learning: initializing the learner's hypothesis and altering the objective function that guides search through the hypothesis space.

In this section we consider a third way of using prior knowledge to alter the hypothesis space search: using it to alter the set of operators that define legal steps in the search through the hypothesis space. This approach is followed by systems such as FOCL

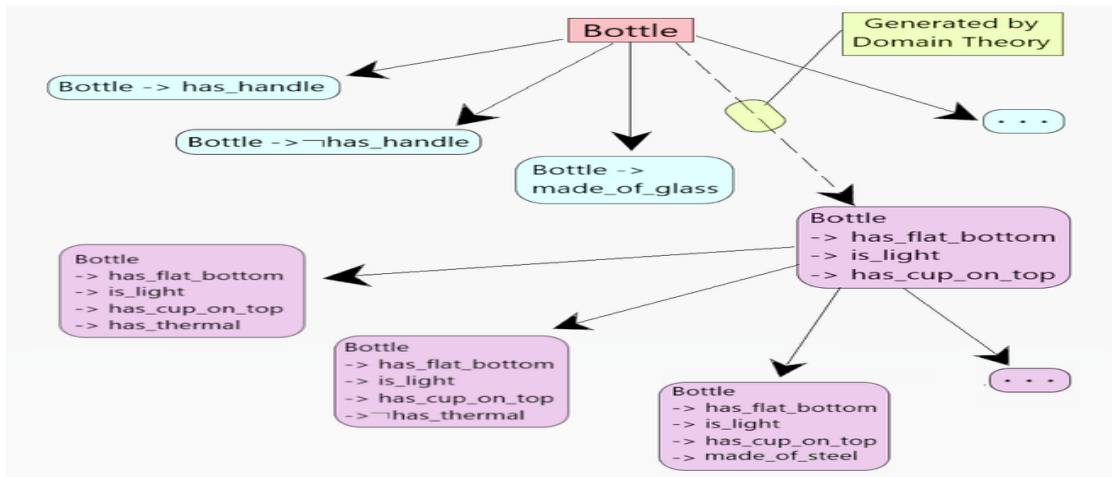
The First Order Combined Learner (FOCL) Algorithm is an extension of the purely inductive, FOIL Algorithm. It uses domain theory to further improve the search for the best-rule and greatly improves accuracy.

First Order Inductive Learner (FOIL)

In machine learning, (FOIL) is a rule-based learning algorithm. It is a natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms

FOCL also tends to perform an iterative process of learning a set of best-rules to cover the training examples and then remove all the training examples covered by that best rule. (using a sequential covering algorithm)

However, what makes the FOCL algorithm more powerful is the approach that it adapts while searching for that best-rule.



358 MACHINE LEARNING

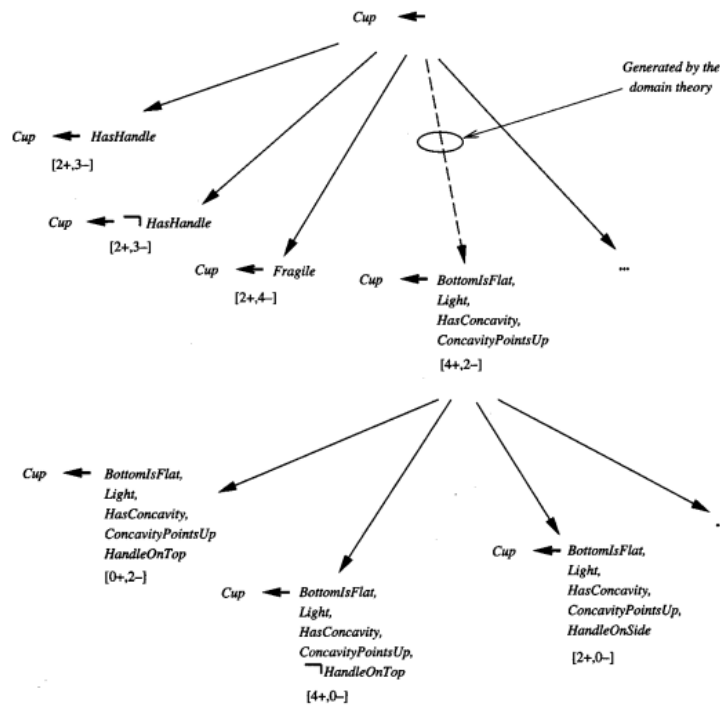


FIGURE 12.8

Hypothesis space search in FOCL. To learn a single rule, FOCL searches from general to increasingly specific hypotheses. Two kinds of operators generate specializations of the current hypothesis. One kind adds a single new literal (solid lines in the figure). A second kind of operator specializes the rule by adding a set of literals that constitute logically sufficient conditions for the target concept, according to the domain theory (dashed lines in the figure). FOCL selects among all these candidate specializations, based on their performance over the data. Therefore, imperfect domain theories will impact the hypothesis only if the evidence supports the theory. This example is based on the same training data and domain theory as the earlier KBANN example.

At each point in it moves from general-to-specific search, FOCL expands its current hypothesis h using the following two operators

1. For each operational literal that is not part of h , create a specialization of h by adding this single literal to the precondition s . This is also the method used by FOIL to generate candidate successors. The solid arrows in Figure 12.8 denote this type of specialization
2. Create an operational, logically sufficient condition for the target concept according to the domain theory. Add this set of literals to the current preconditions of h .

Finally, prune the preconditions of h by removing any literals that are unnecessary according to the training data.

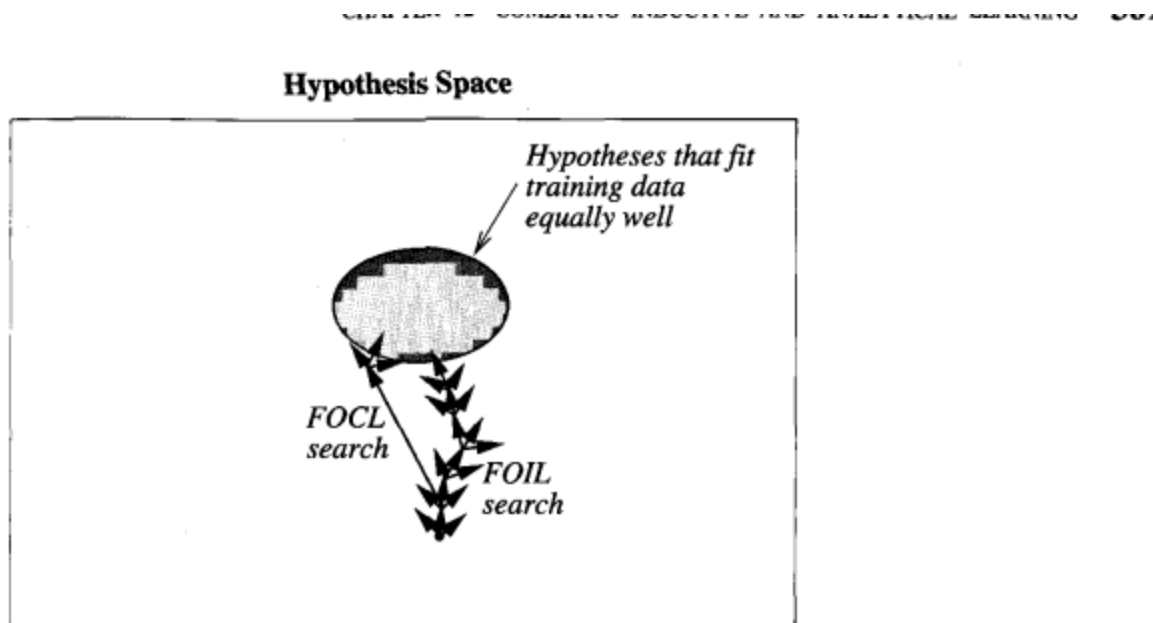


FIGURE 12.9

Hypothesis space search in FOCL. FOCL augments the set of search operators used by FOIL. Whereas FOIL considers adding a single new literal at each step, FOIL also considers adding multiple literals derived from the domain theory.