ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

by **B.SUNDARESAN, M.E.,**
R.NITHIN KUMAR, M.TECH.

First Edition: **January 2023**

Price: Rs. 200/-

ISBN 978-81-960453-0-2

Published by and copies can be had from:

Head Office:
LAKSHMI PUBLICATIONS
Plot No.73, VGP Gokul Nagar,
2nd Main Road (40 Feet Road),
Perumbakkam, Chennai-600 100,
Tamil Nadu, INDIA.
Phone: 044-49523977
Mobile: 98945 98598
E-mail: lakshmipublication@gmail.com
suchitrapublications@gmail.com
Website: www.lakshmipublications.com

Branch Office:
LAKSHMI PUBLICATIONS
No.88, Pidari South Street,
(Govt. Hospital Road),
Sirkali – 609 110. (TK)
Nagapattinam (Dt)
Phone: 044 - 4952 3977

PREFACE

The goal of writing the textbook "**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**" was to satisfy the needs of Tamilnadu engineering students enrolled at Anna University. This book is appropriate for Fourth-semester Engineering College students in the **COMPUTER SCIENCE AND ENGINEERING** branch in accordance with Regulations 2021.

FEATURE OF THE BOOK

The main concept is to give the pupils a textbook, which is not very common at the moment. More effort is made to ensure that the subject's contents are as accurate as possible. In the book, clarity and correctness are scrupulously adhered to.

The audience's goal and purpose are more tightly focused. More effort has been put into getting more accurate results. Readers will feel more at ease and comprehend the book's depth thanks to its contents.

A brief introduction and concise captions for the figures and tables are provided for the content in the book. More emphasis is placed on avoiding grammar mistakes and using the proper format.

AUTHORS

B. Sundaresan, M.E.,
R. Nithin Kumar, M.TECH.

ACKNOWLEDGEMENT

We first and foremost express our gratitude to **GOD ALMIGHTY** for giving us the mental fortitude required to complete the book work preparation.

We would like to extend our sincere appreciation to everyone who helped out, discussed things, supplied feedback, enabled us to use their comments, and helped with the editing, proofreading, and design.

We would like to express our gratitude to **Management, Principal Dr. N. Balaji, Vice Principal Dr. S. Soundararajan, HOD Dr. V.P.Gladispushparathi.**

My **(B.Sundaresan)** Wholehearted thanks to my father **Mr. K.S. Balasubramanian,** my mother **Mrs. B. Vijaya** & my spouse **Mrs. R. Lakshmi & my daughters** for their constant support.

Also my **(R.Nithin Kumar)** heartfelt thanks to my Brothers **Mr.R.Ahilan Vinoth Kumar, Mr.R.Deepak Dinesh Kumar** and My Sister-In-Law **Mrs.P.M.Rajamanohar Vinoth kumar** for their continuous support.

With great concern for the welfare of engineering students, we would like to extend a very special thanks to **Mrs.Nirmala Durai,** Proprietrix of **Lakshmi Publications, Mr.A.Durai, B.E., Founder and Managing Director of Lakshmi Publications, Suchitra Publications** and **Mrs. P.S. Malathi** for providing professional support in all of the activities throughout the development and production of this book.

Suggestions and comments for further improvements of this book are most welcome. Please mail me at dixsonnithinkumar1993@gmail.com

SYLLABUS

ANNA UNIVERSITY, CHENNAI

For B.E., COMPUTER SCIENCE AND ENGINEERING BRANCH

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIT I: PROBLEM SOLVING 9

Introduction to AI - AI Applications - Problem solving agents – search algorithms – uninformed search strategies – Heuristic search strategies – Local search and optimization problems – adversarial search – constraint satisfaction problems (CSP)

UNIT II: PROBABILISTIC REASONING 9

Acting under uncertainty – Bayesian inference – naïve bayes models. Probabilistic reasoning – Bayesian networks – exact inference in BN – approximate inference in BN – causal networks.

UNIT III: SUPERVISED LEARNING 9

Introduction to machine learning – Linear Regression Models: Least squares, single & multiple variables, Bayesian linear regression, gradient descent, Linear Classification Models: Discriminant function – Probabilistic discriminative model - Logistic regression, Probabilistic generative model – Naive Bayes, Maximum margin classifier – Support vector machine, Decision Tree, Random forests

UNIT IV: ENSEMBLE TECHNIQUES AND UNSUPERVISED LEARNING 9

Combining multiple learners: Model combination schemes, Voting, Ensemble Learning - bagging, boosting, stacking, Unsupervised learning: K-means, Instance Based Learning: KNN, Gaussian mixture models and Expectation maximization

UNIT V: NEURAL NETWORKS 9

Perceptron - Multilayer perceptron, activation functions, network training – gradient descent optimization – stochastic gradient descent, error back propagation, from shallow networks to deep networks – Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout.

CONTENTS

UNIT I

UNIT II

PROBABILISTIC REASONING 2.1 - 2.28

UNIT III

SUPERVISED LEARNING 3.1 - 3.67

UNIT IV

ENSEMBLE TECHNIQUES AND UNSUPERVISED LEARNING 4.1 - 4.30

UNIT V

NEURAL NETWORKS 5.1 - 5.44

1943 – Artificial Neuron
1950 – Turing Test – whether the machine has also the intelligence like a human
1956 – John Mccarthy Coined name AI father of AI American Scientist
1972 –

UNIT I

PROBLEM SOLVING

Introduction to AI - AI Applications - Problem solving agents – search algorithms – uninformed search strategies – Heuristic search strategies – Local search and optimization problems – adversarial search – constraint satisfaction problems (CSP)

1.1. INTRODUCTION TO ARTIFICIAL INTELLIGENCE

"Artificial Intelligence is a branch of computer science that deals with developing intelligent machines which can behave like human, think like human, and has ability to take decisions by their own."

Artificial Intelligence is a combination of two words Artificial and Intelligence, which refers to man-made intelligence. Therefore, when machines are equipped with man-made intelligence to perform intelligent tasks similar to humans, it is known as Artificial Intelligence. It is all about developing intelligent machines that can simulate the human brain and work & behave like human beings

1.1.1. HISTORY OF ARTIFICIAL INTELLIGENCE

Artificial intelligence is assumed a new technology, but in reality, it is not new. The researchers in the field of AI are much older. It is said that the concept of intelligent machines was found in Greek Mythology. Below are some keystones in the development of AI:

- In the year 1943, Warren McCulloch and Walter pits proposed a model of Artificial neurons.
- In the year 1950, Alan Turing published a "Computer Machinery and Intelligence" paper in which he introduced a test, known as a Turing Test. This test is used to determine intelligence in machines by checking if the machine is capable of thinking or not.
- In the year 1956, for the first time, the term Artificial Intelligence was coined by the American Computer scientist John Mc Carthy at the Dartmouth Conference. John McCarthy is also known as the Father of AI.

- In the year 1972, the first full-scale intelligent humanoid robot, WABOT1, was created in Japan.
- In the year 1980, AI came with the evolution of Expert Systems. These systems are computer programs, which are designed to solve complex problems.
- In the year 1997, IBM Deep Blue beat world chess champion Gary Kasparov and became the first computer to defeat a world chess champion.
- In the year 2006, AI came into the business world. World's top companies like Facebook, Twitter, and Netflix also started using AI in their applications.

1.1.2. TYPES OF ARTIFICIAL INTELLIGENCE

Artificial Intelligence can be divided in various types, there are mainly two types of main categorization which are based on capabilities and based on functionally of AI. Following is flow diagram which explain the types of AI.

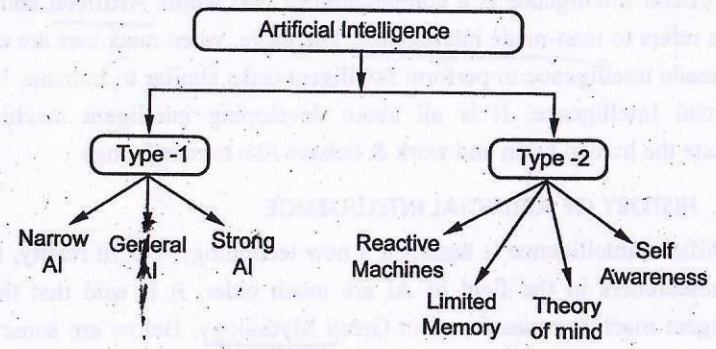


Fig. 1.1. AI type-1: Based on Capabilities

1. Weak AI or Narrow AI:

- Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence.
- Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.

- Apple Siri is a good example of Narrow AI, but it operates with a limited pre-defined range of functions.
- IBM's Watson supercomputer also comes under Narrow AI, as it uses an Expert system approach combined with Machine learning and natural language processing.
- Some Examples of Narrow AI are playing chess, purchasing suggestions on e-commerce site, self-driving cars, speech recognition, and image recognition.

2. General AI:

- General AI is a type of intelligence which could perform any intellectual task with efficiency like a human.
- The idea behind the general AI to make such a system which could be smarter and think like a human by its own.
- Currently, there is no such system exist which could come under general AI and can perform any task as perfect as a human.
- The worldwide researchers are now focused on developing machines with General AI.
- As systems with general AI are still under research, and it will take lots of efforts and time to develop such systems.

3. Super AI:

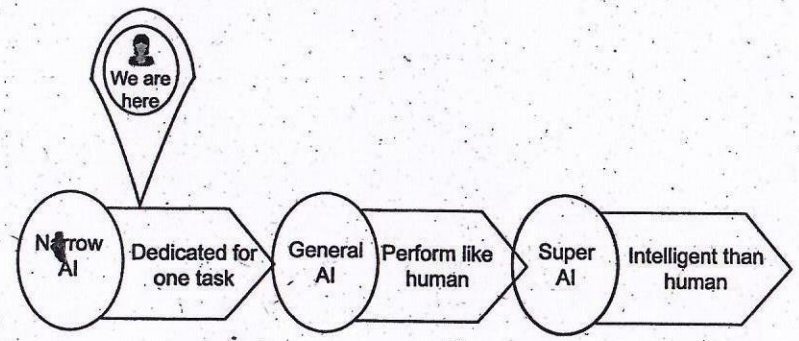


Fig. 1.2.

- Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI.
- Some key characteristics of strong AI include capability include the ability to think, to reason, solve the puzzle, make judgments, plan, learn, and communicate by its own.
- Super AI is still a hypothetical concept of Artificial Intelligence. Development of such systems in real is still world changing task.

1.1.3. ARTIFICIAL INTELLIGENCE TYPE-2: BASED ON FUNCTIONALITY

1. Reactive Machines

- Purely reactive machines are the most basic types of Artificial Intelligence.
- Such AI systems do not store memories or past experiences for future actions.
- These machines only focus on current scenarios and react on it as per possible best action.
- IBM's Deep Blue system is an example of reactive machines.
- Google's AlphaGo is also an example of reactive machines.

2. Limited Memory

- Limited memory machines can store past experiences or some data for a short period of time.
- These machines can use stored data for a limited time period only.
- Self-driving cars are one of the best examples of Limited Memory systems. These cars can store recent speed of nearby cars, the distance of other cars, speed limit, and other information to navigate the road.

3. Theory of Mind

- Theory of Mind AI should understand the human emotions, people, beliefs, and be able to interact socially like humans.
- This type of AI machines is still not developed, but researchers are making lots of efforts and improvement for developing such AI machines.

4. Self-Awareness

- Self-awareness AI is the future of Artificial Intelligence. These machines will be super intelligent, and will have their own consciousness, sentiments, and self-awareness.
- These machines will be smarter than human mind.
- Self-Awareness AI does not exist in reality still and it is a hypothetical concept.

A.I Approaches

The definitions of AI according to some text books are categorized into four approaches and are summarized in the table below:

- Systems that think like human
- Systems that act like human
- Systems that think rationally
- Systems that act rationally

Thinking Humanly	Thinking Rationally
"The exciting new effort to make computers think … machines with minds, in the full and literal sense". (Haugeland, 1985) "(The automation of) activities that we associate with human thinking, activities such as decision-making, problem solving, learning …" (Bell man, 1978)	"The study of mental faculties through the use of computational models". (Chamiak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act". ("Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people". (Kurzweil, 1990)	"Computational Intelligence is the study of the design of intelligent agents" (Poole et al., 1998)

"The study of how to make computers do things at which, at the moment, people are better". (Rich and Knight, 1991)	"AI... is concerned with intelligent behavior in artifacts". (Nilsson, 1998)

Table 1.1. Some definitions of artificial intelligence, organized into four categories

Thinking humanly: The cognitive modeling approach

To say that a program thinks like a human, consider the human thinking which can be expressed in three ways:

introspection - trying to catch our own thoughts as they go by;

psychological experiments - observing a person in action;

brain imaging - observing the brain in action.

Once there is sufficient precise theory of the mind, it becomes possible to express the theory as a computer program

Cognitive Study of Human Mind:

It is a highly interdisciplinary field which combines ideas and methods from psychology, computer science, philosophy, linguistics and neuroscience.

The goal of cognitive science is to characterize the nature of human knowledge and how that knowledge is used, processed and acquired

Acting humanly: The Turing Test approach

The Turing Test, proposed by Alan Turing(1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.

The computer would need to possess the following capabilities:

- ❖ natural language processing to enable it to communicate successfully in English
- ❖ knowledge representation to store what it knows or hears;
- ❖ automated reasoning to use the stored information to answer questions and to draw new conclusions;
- ❖ machine learning to adapt to new circumstances and to detect and to discover new patterns

- ❖ computer vision to perceive object and
- ❖ robotics to manipulate objects and to move about

Thinking rationally: The "laws of thought" approach

- ❖ Aristotle The concept of "right thinking" was proposed by Aristotle. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises.
- ❖ The canonical example starts with Socrates is a man and all men are mortal and concludes that Socrates is mortal.
- ❖ These laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic.
- ❖ Logicians developed a precise notation for statements about objects in the world and the relations among them .Logics are needed to create intelligent systems.

Acting rationally: The rational agent approach

An agent is just something that acts operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, create and pursue goals. A rational agent is one that acts so as to achieve the best outcome

One way to act rationally is to deduce that a given action is best and then to act on that conclusion. On the other hand, there are ways of acting rationally that cannot be said to involve inference.

The rational-agent approach to AI has two advantages over the other approaches. First, it is more general than the "laws of thought" approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is suitable for scientific development.

1.2. APPLICATIONS OF AI

1. Game Playing:

AI is widely used in Gaming. Different strategic games such as Chess, where the machine needs to think logically, and video games to provide real-time experiences use Artificial Intelligence.

2. Robotics:

Artificial Intelligence is commonly used in the field of Robotics to develop intelligent robots. AI implemented robots use real-time updates to sense any obstacle in their path and can change the path instantly. AI robots can be used for carrying goods in hospitals and industries and can also be used for other different purposes.

3. Healthcare:

In the healthcare sector, AI has diverse uses. In this field, AI can be used to detect diseases and cancer cells. It also helps in finding new drugs with the use of historical data and medical intelligence.

4. Computer Vision:

Computer vision enables the computer system to understand and derive meaningful information from digital images, video, and other visual input with the help of AI.

5. Agriculture:

AI is now widely used in Agriculture; for example, with the help of AI, we can easily identify defects and nutrient absences in the soil. To identify these defects, AI robots can be utilized. AI bots can also be used in crop harvesting at a higher speed than human workers.

6. E-commerce

AI is one of the widely used and demanding technologies in the E-commerce industry. With AI, e-commerce businesses are gaining more profit and grow in business by recommending products as per the user requirement.

7. Social Media

Different social media websites such as Facebook, Instagram, Twitter, etc., use AI to make the user experiences much better by providing different features. For example, Twitter uses AI to recommend tweets as per the user interest and search history.

1.3. PROBLEM SOLVING AGENTS

When the correct action to take is not immediately obvious, an agent may need to to plan ahead: to consider a sequence of actions that form a path to a goal state. Such an agent is called a problem- solving agent, and the computational process it undertakes is called search.

The agent can follow this four-phase problem-solving process:

- **Goal Formulation:** Goals organize behavior by limiting the objectives and hence the actions to be considered.
- **Problem Formulation:** The agent devises a description of the states and actions necessary to reach the goal - an abstract model of the relevant part of the world.
- **Search:** Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a solution. The agent might have to simulate multiple sequences that do not reach the goal, but eventually it will find a solution (such as going from Arad to Sibiu to Fagaras to Bucharest), or it will find that no solution is possible.
- **Execution:** The agent can now execute the actions in the solution, one at a time. It is an important property that in a fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions. If the model is correct, then once the agent has found a solution, it can ignore its percepts while it is executing the actions—because the solution is guaranteed to lead to the goal. Control theorists call this an open-loop system: ignoring the percepts breaks the loop between agent and environment. If there is a chance that the model is incorrect, or the environment is nondeterministic, then the agent would be safer using a closed-loop approach that monitors the precepts.

1.3.1. COMPONENTS TO FORMULATE THE ASSOCIATED PROBLEM

- **Initial State:** This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- **Action:** This stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.
- **Transition:** This stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.
- **Goal test:** This stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determine the cost to achieve the goal.
- **Path costing:** This component of problem-solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

1.4. SEARCH ALGORITHMS

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.

Problem-solving agents:

In Artificial Intelligence, Search techniques are universal problem-solving methods. Rational agents or Problem-solving agents in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

1.4.1. SEARCH ALGORITHM TERMINOLOGIES

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- **Search Space:** Search space represents a set of possible solutions, which a system may have.
- **Start State:** It is a state from where agent begins the search.
- **Goal Test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

Actions: It gives the description of all the available actions to the agent.

Transition model: A description of what each action do, can be represented as a transition model.

Path Cost: It is a function which assigns a numeric cost to each path.

Solution: It is an action sequence which leads from the start node to the goal node.

Optimal Solution: If a solution has the lowest cost among all solutions.

1.4.2. PROPERTIES OF SEARCH ALGORITHMS

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

1.4.3. TYPES OF SEARCH ALGORITHMS

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.

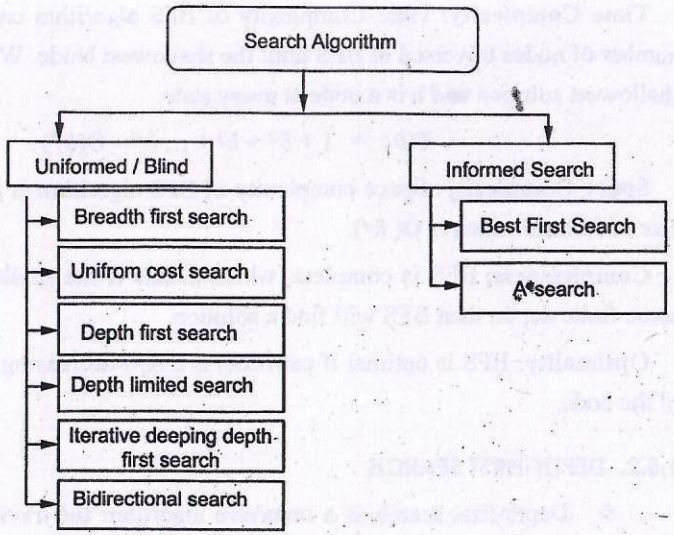


Fig. 1.3.

1.4.4. UNINFORMED/BLIND SEARCH:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- ❖ Breadth-first search
- ❖ Uniform cost search
- ❖ Depth-first search
- ❖ Iterative deepening depth-first search
- ❖ Bidirectional Search

1.4.5. INFORMED SEARCH

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search

strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

1.5. UNINFORMED SEARCH ALGORITHMS

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

1.5.1. BREADTH-FIRST SEARCH:

- ❖ Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- ❖ BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S → A → B → C → D → G → H → E → F → I → K

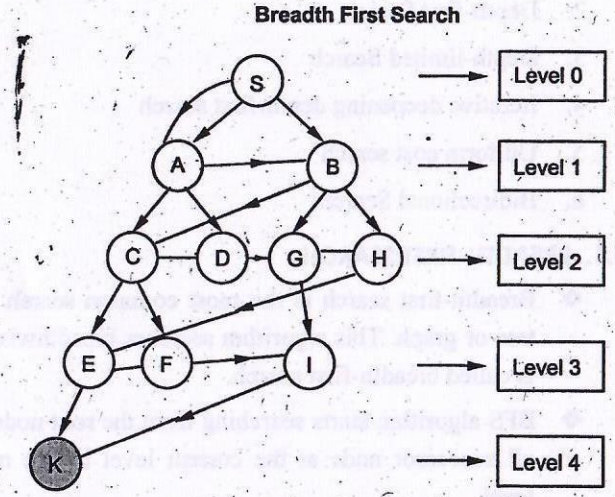


Fig. 1.4. Breadth First Search

Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d = depth of shallowest solution and b is a node at every state.

$$T(b) = 1 + b^2 + b^3 + \dots b^d - O(b^d)$$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

1.5.2. DEPTH-FIRST SEARCH

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node → Left node → right node

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

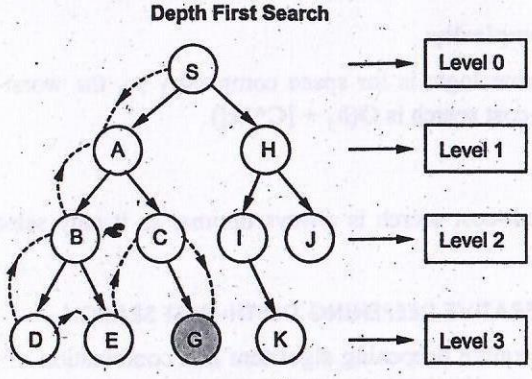


Fig. 1.5. Depth First Search

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n) = 1 + n^2 + n^3 + \ldots + n^m = O(n^m)$$

Where, m = maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

1.5.3. DEPTH-LIMITED SEARCH ALGORITHM:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:

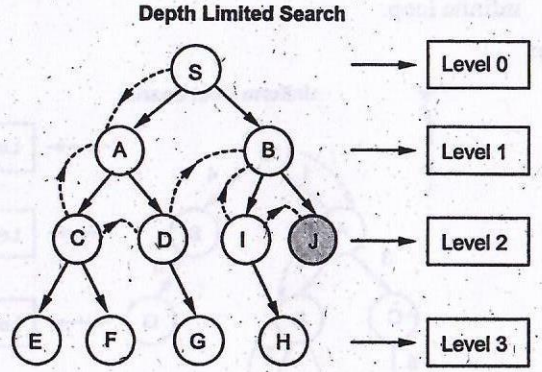


Fig. 1.6. Depth Limited Search

Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b^{ℓ})**.

Space Complexity: Space complexity of DLS algorithm is **O($b \times \ell$)**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

1.5.4. UNIFORM-COST SEARCH ALGORITHM

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:

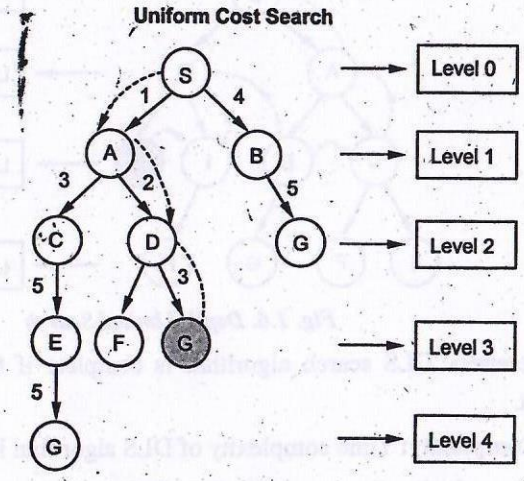


Fig. 1.7. Uniform Cost Search

Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and ε is each step to get closer to the goal node. Then the number of steps is = C*/ ε + 1. Here we have taken +1, as we start from state 0 and end to C*/ ε.

Hence, the worst-case time complexity of Uniform-cost search is $\mathbf{O(b_1 + [C^*/ \varepsilon])}$.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b_1 + [C^*/ \varepsilon])}$.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

1.5.5. ITERATIVE DEEPENING DEPTH-FIRST SEARCH

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:

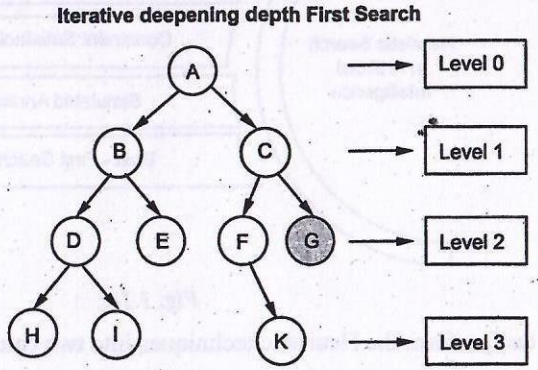


Fig. 1.8. Iterative deepening depth first search

1st Iteration → A

2nd Iteration → A, B, C

3rd Iteration →A, B, D, E, C, F, G

4th Iteration → A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is if the branching factor is finite.

Time Complexity:

Let's suppose *b* is the branching factor and depth is *d* then the worst-case time complexity is **O(bd)**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.

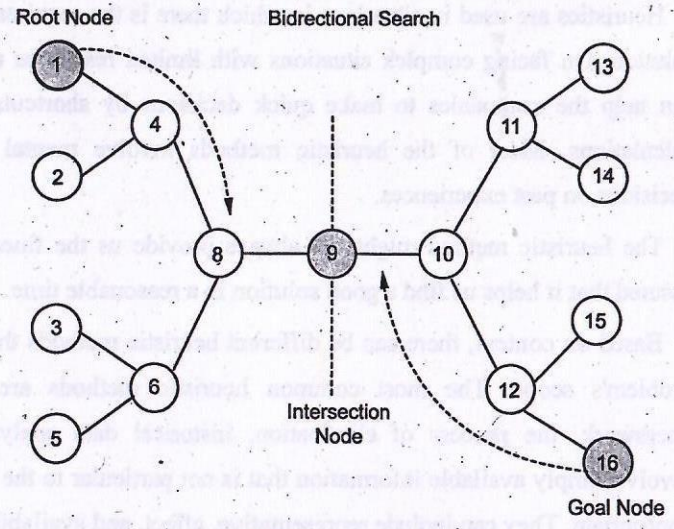


Fig. 1.9. Bidirectional Search

Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **O(bd)**.

Space Complexity: Space complexity of bidirectional search is **O(bd)**.

Optimal: Bidirectional search is Optimal.

1.6. HEURISTIC SEARCH STRATEGIES

A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not. Heuristics are said to be the problem-solving techniques that result in practical and quick solutions.

Heuristics are strategies that are derived from past experience with similar problems. Heuristics use practical methods and shortcuts used to produce the solutions that may or may not be optimal, but those solutions are sufficient in a given limited timeframe.

Why do we need heuristics?

Heuristics are used in situations in which there is the requirement of a short-term solution. On facing complex situations with limited resources and time, Heuristics can help the companies to make quick decisions by shortcuts and approximated calculations. Most of the heuristic methods involve mental shortcuts to make decisions on past experiences.

The heuristic method might not always provide us the finest solution, but it is assured that it helps us find a good solution in a reasonable time.

Based on context, there can be different heuristic methods that correlate with the problem's scope. The most common heuristic methods are - trial and error, guesswork, the process of elimination, historical data analysis. These methods involve simply available information that is not particular to the problem but is most appropriate. They can include representative, affect, and availability heuristics.

Heuristic search techniques in AI (Artificial Intelligence)

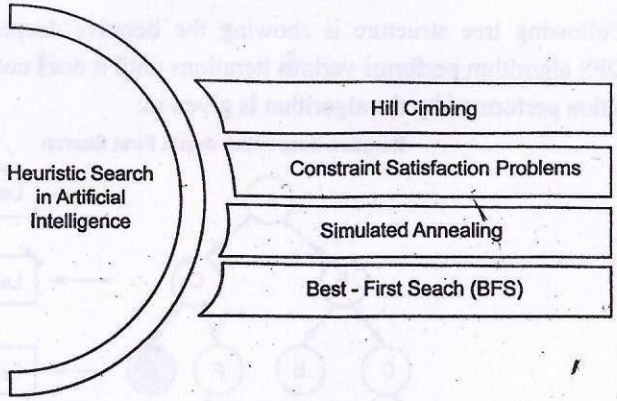


Fig. 1.10.

We can perform the Heuristic techniques into two categories:

Direct Heuristic Search techniques in AI

It includes Blind Search, Uninformed Search, and Blind control strategy. These search techniques are not always possible as they require much memory and time. These techniques search the complete space for a solution and use the arbitrary ordering of operations.

The examples of Direct Heuristic search techniques include Breadth-First Search (BFS) and Depth First Search (DFS).

Weak Heuristic Search techniques in AI

It includes Informed Search, Heuristic Search, and Heuristic control strategy. These techniques are helpful when they are applied properly to the right types of tasks. They usually require domain-specific information.

The examples of Weak Heuristic search techniques include Best First Search (BFS) and A*.

Before describing certain heuristic techniques, let's see some of the techniques listed below:

- Bidirectional Search
- A* search
- Simulated Annealing

- Hill Climbing
- Best First search
- Beam search

First, let's talk about the Hill climbing in Artificial intelligence.

Hill Climbing Algorithm

It is a technique for optimizing the mathematical problems. Hill Climbing is widely used when a good heuristic is available.

It is a local search algorithm that continuously moves in the direction of increasing elevation/value to find the mountain's peak or the best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. Traveling-salesman Problem is one of the widely discussed examples of the Hill climbing algorithm, in which we need to minimize the distance traveled by the salesman.

It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. The steps of a simple hill-climbing algorithm are listed below:

Step 1: Evaluate the initial state. If it is the goal state, then return success and Stop.

Step 2: Loop Until a solution is found or there is no new operator left to apply.

Step 3: Select and apply an operator to the current state.

Step 4: Check new state:

If it is a goal state, then return to success and quit.

Else if it is better than the current state, then assign a new state as a current state.

Else if not better than the current state, then return to step 2.

Step 5: Exit.

Best first search (BFS)

This algorithm always chooses the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It lets us to take the benefit of both algorithms. It uses the heuristic function and search. With the help of the best-first search, at each step, we can choose the most promising node.

Best first search algorithm:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n from the OPEN list, which has the lowest value of $h(n)$, and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is the goal node, then return success and stop the search, else continue to next step.

Step 6: For each successor node, the algorithm checks for evaluation function $f(n)$ and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list.

Step 7: Return to Step 2.

A* Search Algorithm

A* search is the most commonly known form of best-first search. It uses the heuristic function $h(n)$ and cost to reach the node n from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.

It finds the shortest path through the search space using the heuristic function. This search algorithm expands fewer search tree and gives optimal results faster.

Algorithm of A* search:

Step 1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not. If the list is empty, then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of the evaluation function ($g + h$). If node n is the goal node, then return success and stop, otherwise.

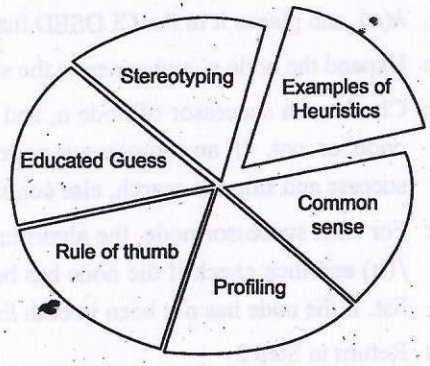
Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the

OPEN or CLOSED list. If not, then compute the evaluation function for n' and place it into the Open list.

Step 5: Else, if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to Step 2.

Examples of heuristics in everyday life



Some of the real-life examples of heuristics that people use as a way to solve a problem:

- **Common sense:** It is a heuristic that is used to solve a problem based on the observation of an individual.
- **Rule of thumb:** In heuristics, we also use a term rule of thumb. This heuristic allows an individual to make an approximation without doing an exhaustive search.
- **Working backward:** It lets an individual solve a problem by assuming that the problem is already being solved by them and working backward in their minds to see how much a solution has been reached.
- **Availability heuristic:** It allows a person to judge a situation based on the examples of similar situations that come to mind.
- **Familiarity heuristic:** It allows a person to approach a problem on the fact that an individual is familiar with the same situation, so one should act similarly as he/she acted in the same situation before.
- **Educated guess:** It allows a person to reach a conclusion without doing an exhaustive search. Using it, a person considers what they have observed in the past and applies that history to the situation where there is not any definite answer has decided yet.

Types of heuristics

There are various types of heuristics, including the availability heuristic, affect heuristic and representative heuristic. Each heuristic type plays a role in decision-making. Let's discuss about the Availability heuristic, affect heuristic, and Representative heuristic.

Availability heuristic

Availability heuristic is said to be the judgment that people make regarding the likelihood of an event based on information that quickly comes into mind. On making decisions, people typically rely on the past knowledge or experience of an event. It allows a person to judge a situation based on the examples of similar situations that come to mind.

Representative heuristic

It occurs when we evaluate an event's probability on the basis of its similarity with another event.

Example: We can understand the representative heuristic by the example of product packaging, as consumers tend to associate the products quality with the external packaging of a product. If a company packages its products that remind you of a high quality and well-known product, then consumers will relate that product as having the same quality as the branded product.

So, instead of evaluating the product based on its quality, customers correlate the products quality based on the similarity in packaging.

Affect heuristic

It is based on the negative and positive feelings that are linked with a certain stimulus. It includes quick feelings that are based on past beliefs. Its theory is one's emotional response to a stimulus that can affect the decisions taken by an individual.

When people take a little time to evaluate a situation carefully, they might base their decisions based on their emotional response.

Example: The affect heuristic can be understood by the example of advertisements. Advertisements can influence the emotions of consumers, so it affects the purchasing decision of a consumer. The most common examples of

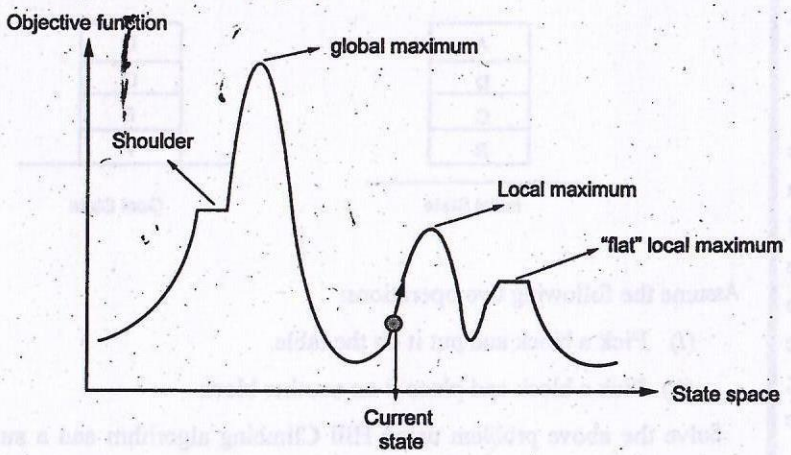
advertisements are the ads of fast food. When fast-food companies run the advertisement, they hope to obtain a positive emotional response that pushes you to positively view their products.

If someone carefully analyzes the benefits and risks of consuming fast food, they might decide that fast food is unhealthy. But people rarely take time to evaluate everything they see and generally make decisions based on their automatic emotional response. So, Fast food companies present advertisements that rely on such type of Affect heuristic for generating a positive emotional response which results in sales.

1.7. LOCAL SEARCH AND OPTIMIZATION PROBLEMS

Local search algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, nor the set of states that have been reached. That means they are not systematic - they might never explore a portion of the search space where a solution actually resides. However, they have two key advantages: (1) they use very little memory and (2) they can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable. Local search algorithms can also solve optimization problems, in which the aim is to find the best state according to an objective function.

To understand local search, consider the states of a problem laid out in a state-space landscape, as shown in Figure.



Each point (state) in the landscape has an "elevation," defined by the value of the objective function. If elevation corresponds to an objective function, then the aim is to find the highest peak - a global maximum - this is known as hill climbing. If elevation corresponds to cost, then the aim is to find the lowest valley - a global minimum - this is known as gradient descent.

Hill-climbing Search

The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor. The hill-climbing search algorithm keeps track of one current state and on each iteration moves to the neighboring state with highest value - that is, it heads in the direction that provides the steepest ascent. It terminates when it reaches a "peak" where no neighbor has a higher value. Hill climbing does not look ahead beyond the immediate neighbors of the current state.

Algorithm: Hill Climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
```

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.

Unfortunately, hill climbing can get stuck for any of the following reasons:

Local Maxima: A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.

Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move. Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

Plateaus: A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible.

Many variants of hill climbing have been invented. Stochastic hill climbing chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions. First-choice hill climbing implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.

Another variant is random-restart hill climbing, which adopts the quote "If at first you don't succeed, try, try again." It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found. It is complete with probability 1, because it will eventually generate a goal state as the initial state. If each hill-climbing search has a probability of success, then the expected number of restarts required is $1 / p$. The expected number of steps is the cost of one successful iteration plus $(1 - p) / p$ times the cost of failure. For 8-queens, random-restart hill climbing is very effective indeed. Even for three million queens, the approach can find solutions in seconds

The success of hill climbing depends very much on the shape of the state-space landscape: if there are few local maxima and plateaus, random-restart hill climbing will find a good solution very quickly.

Example:

To illustrate hill climbing, Consider the 8-queens problem. The complete-state formulation is used here, which means that every state has all the components of a solution, but they might not all be in the right place. In this case every state has 8 queens on the board, one per column. The initial state is chosen at random, and the successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has $8 \times 7 = 56$ successors). The heuristic cost function is the number of pairs of queens that are attacking each other; this will be zero only for solutions. (It counts as an attack if two pieces are in the same line, even if there is an intervening piece between them.)

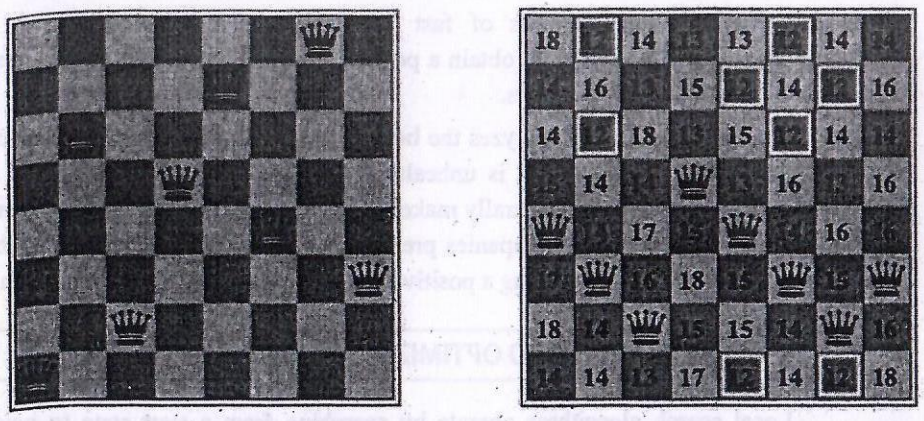
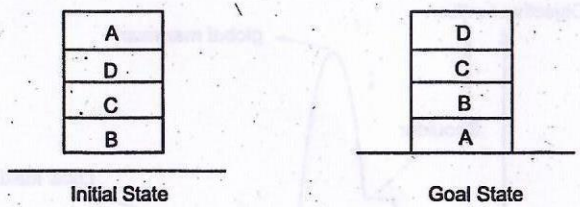


Figure (a) : The 8-queens problem: place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.) The figure (b) shows the h values of all its successors.

Blocks world problem

Consider the blocks world problem with the four blocks A, B, C, D with the start and goal states given below.



Assume the following two operations:

(*i*) Pick a block and put it on the table.

(*ii*) Pick a block and place it on another block

Solve the above problem using Hill Climbing algorithm and a suitable heuristic function. Show the intermediate decisions and states

Solution:

Define the heuristic function

$h(x) = +1$ for all the blocks in the structure if the block is correctly positioned or $h(x) = -1$ for all uncorrectly placed blocks in the structure.

Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI:

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games.

  Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply.
- Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.

- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do

Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, – 1, and 0.

Game tree:

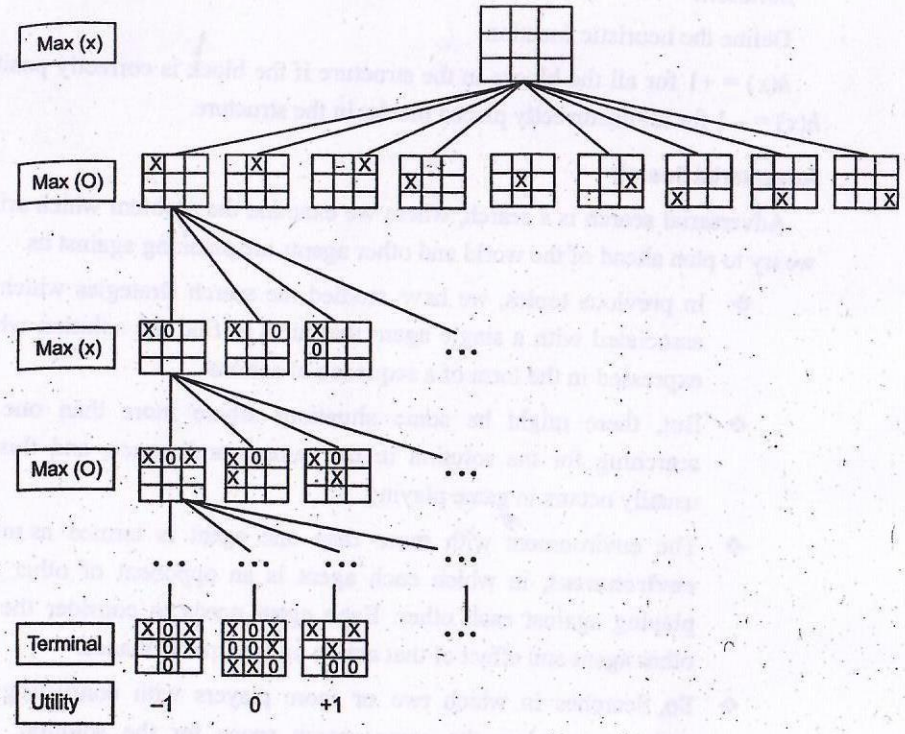
A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree

- MIN minimizes the result.



Example Explanation:

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.

- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Hence adversarial Search for the minimax procedure works as follows:

- It aims to find the optimal strategy for MAX to win the game.
- It follows the approach of Depth-first search.
- In the game tree, optimal leaf node could appear at any depth of the tree.
- Propagate the minimax values up to the tree until the terminal node discovered.

In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

For a state S MINIMAX(s)

$$= \begin{cases} \text{UTILITY(s)} & \text{If TERMINAL-TEST(s)} \\ \max_{a \in \text{Actions(s)}} \text{MINIMAX(RESULT(s,al))} & \text{If PLAYER(s) = MAX} \\ \min_{a \in \text{Actions(s)}} \text{MINIMAX(RESULT(s, a))} & \text{If PLAYER(s) = MIN} \end{cases}$$

1.8. CONSTRAINT SATISFACTION PROBLEMS

A problem is solved when each variable has a value that satisfies all the constraints on the variable. Such a problem is called a constraint satisfaction problem, or CSP.

Defining Constraint Satisfaction Problems

A constraint satisfaction problem consists of three components, X, D and C : X is a set of variables, $\{X_1,\ldots,X_n\}$.

D is a set of domains, one for each variable $\{D_1,\ldots,D_n\}$.

C is a set of constraints that specify allowable combinations of values.

A domain, D_i , consists of a set of allowable values, $\{v_1,\ldots,v_k\}$, for variable X_i . For example, a Boolean variable would have the domain {true, false}. Different variables can have different domains of different sizes. Each constraint consists of a pair C_j (scope,rel) , where scope is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on.

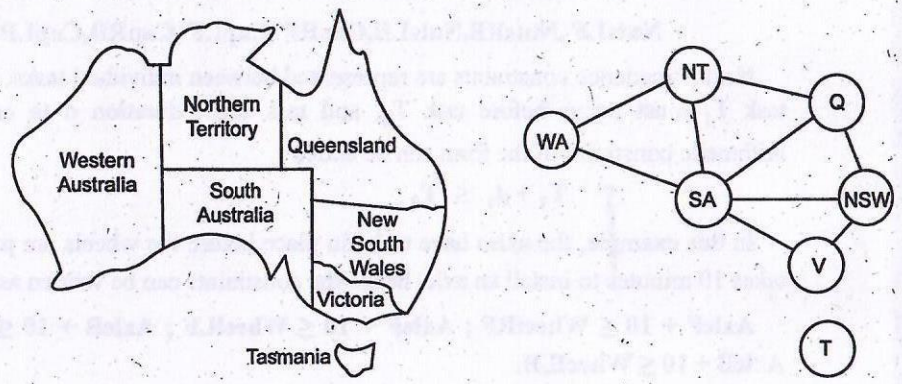
For example, if X_1 and X_2 both have the domain, {1, 2, 3}then the constraint saying that X_1 must be greater than X2 can be written as $((X_1,X_2),\{(3,1),(3,2),(2,1)\})$ $((X_1, X_2),X_1 > X_2)$

CSPs deal with assignments of values to variables, $\{X_i = v_i, X_j = v_j,\ldots\}$. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is assigned a value, and a solution to a CSP is a consistent, complete assignment. A partial assignment is one that leaves some variables unassigned, and a partial solution is a partial assignment that is consistent.

Example problem: Map coloring

Consider a map of Australia showing each of its states and territories .The task is to color each region with either red, green, or blue in such a way that no two neighboring regions have the same color. To formulate this as a CSP, the variables are the regions defined as follows:

$$X = \{WA, NT, Q, NSW, V, SA, T\}$$



(a) (b)

The domain of every variable is the set D_i = {red, green, blue} The constraints require neighboring regions to have distinct colors.

C = {SA ≠ WA,SA ≠ NT,SA ≠ Q,SA ≠ NSW,SA ≠ V,WA ≠ NT,NT ≠ Q,Q ≠ NSW, NSW ≠ V }.

There are many possible solutions to this problem, such as

{WA = red,NT = green,Q = red,NSW = green,V = red,SA = blue,T = red }.

It can be helpful to visualize a CSP as a constraint graph. The nodes of the graph correspond to variables of the problem, and an edge connects any two variables that participate in a constraint.

Example problem: Job-shop scheduling

Factories have the problem of scheduling a day's worth of jobs, subject to various constraints. In practice, many of these problems are solved with CSP techniques. Consider the problem of scheduling the assembly of a car. The whole job is composed of tasks. Constraints can assert that one task must occur before another - for example, a wheel must be installed before the hubcap is put on. Constraints can also specify that a task takes a certain amount of time to complete.

Consider a small part of the car assembly, consisting of 15 tasks: install axles (front and back), affix all four wheels (right and left, front and back), tighten nuts for each wheel, affix hubcaps, and inspect the final assembly. The tasks can be represented with 15 variables:

X = {AxleF, AxleB, WheelRF ,WheelLF ,WheelRB, WheelLB,NutsRF NutsLF ,NutsRB,NutsLB,CapRF ,CapLF ,CapRB,CapLB,Inspect}.

Next precedence constraints are represented between individual tasks. Whenever a task T_1 must occur before task T_2, and task takes duration d to complete, an arithmetic constraint of the form can be added

$$T_1 + d_1 \leq T_2$$

In this example, the axles have to be in place before the wheels are put on, and it takes 10 minutes to install an axle, hence the constraints can be written as

AxleF + 10 ≤ WheelRF ; AxleF + 10 ≤ WheelLF ; AxleB + 10 ≤ WheelRB; AxleB + 10 ≤ WheelLB.

Cryptarithmetic Problems : Examples

1.

	SEND	+	MORE	=	MONEY

	5	4	3	2	1
		S	E	N	D
+		M	O	R	E
		c_3	c_2	c_1	
	M	O	N	E	Y

Cryptarithmetic Problems: It is an arithmetic problem represented using letters. It involves the decoding of digits represented by a character.

Constraints:

- Assign a decimal digit to each of the letters in such a way that the answer is correct
- Assign decimal digit to letters
- Cannot assign different digits to same letters
- No two letters have the same digit
- Unique digit assigned to each letter

Rules:

- From Column 5, M = 1, since it is only carry-over possible from sum of 2 single digit number in column 4.

	5	4	3	2	1
		S	E	N	D
+		1	O	R	E
		c_3	c_2	c_1	
	1	O	N	E	Y

- To produce a carry from column 4 to column 5 'S + M' is at least 9 so 'S = 8 or 9' so 'S + M = 9 or 10' & so 'O = 0 or 1' . But 'M = 1', so 'O = 0'.

	5	4	3	2	1
		9	E	N	D
+		1	O	R	E
		c_3	c_2	c_1	
	1	0	N	E	Y

- If there is carry from column 3 to 4 then 'E = 9' & so 'N = 0'. But 'O = 0' so there is no carry & 'S = 9' & 'C_3 = 0'.
- If there is no carry from column 2 to 3 then 'E = N' which is impossible, therefore there is carry & 'N = E + 1' & 'C_2 = 1'.

- If there is carry from column 1 to 2 then 'N + R = E mod 10' & 'N = E + 1' so 'E + 1 + R = E mod 10', so 'R = 9' but 'S = 9', so there must be carry from column 1 to 2. Therefore 'c1=1' & 'R = 8'.
- To produce carry 'c1=1' from column 1 to 2, there must be 'D + E = 10 + Y' as Y cannot be 0/1 so D + E is at least 12. As D is at most 7 & E is at least 5(D cannot be 8 or 9 as it is already assigned). N is atmost 7 & 'N = E + 1' so 'E = 5 or 6'.
- If E were 6 & D + E atleast 12 then D would be 7, but 'N = E + 1' & N would also be 7 which is impossible. Therefore 'E = 5' & 'N = 6'.
- D + E is atleast 12 hence 'D = 7' & 'Y = 2'

Solution:

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

Letter	Digit Value
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Constraint Propagation : Inference in CSP

In CSPs there is a choice: an algorithm can search (choose a new variable assignment from several possibilities) or do a specific type of inference called constraint propagation: using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.

The key idea is local consistency. If each variable is treated as a node in a graph and each binary constraint as an arc, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph. There are different types of local consistency, which are as follows.

Node consistency

A single variable (corresponding to a node in the CSP network) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints. For example, in the variant of the Australia map-coloring problem where South Australians dislike green, the variable SA starts with domain {red , green, blue}, and this can be made node consistent by eliminating green, leaving SA with the reduced domain {red , blue}. Thus a network is node-consistent if every variable in the network is node-consistent. It is always possible to eliminate all the unary constraints in a CSP by running node consistency.

Arc consistency

A variable in a CSP is arc-consistent if every value in its domain satisfies the variable's binary constraints. More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain Di there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j). A network is arc-consistent if every variable is arc consistent with every other variable. For example, consider the constraint $Y = X_2$ where the domain of both X and Y is the set of digits. This constraint can be explicitly written as {(X, Y), {(0, 0), (1, 1), (2, 4), (3, 9)) })

To make X arc-consistent with respect to Y, reduce X's domain to {0, 1, 2, 3}. and also to make Y arc-consistent with respect to X, then Y 's domain becomes {0, 1, 4, 9} and the whole CSP is arc-consistent.

The most popular algorithm for arc consistency is called AC-3 .

function AC-3(csp) returns false if an inconsistency is found and true otherwise

queue ← a queue of arcs, initially all the arcs in csp

while queue is not empty do

(X_i, X_j) ←Pop(queue)

if REVISE(csp, X_i, X_j) then

if size of $D_i = 0$ then return false

for each X_k in X_i, NEIGHBORS – $\{X_j\}$ do

add (X_k, X_i) to queue

return true

function REVISE(csp, X_i, X_j)returns true iff we revise the domain of X_i

revised ← false

for each x in D_i do

if no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j then

delete x from D_i

revised ← true

return revised

To make every variable arc-consistent, the AC-3 algorithm maintains a queue of arcs to consider. Initially, the queue contains all the arcs in the CSP. AC-3 then pops off an arbitrary arc (X_i, X_j) from the queue and makes X_i arc-consistent with respect to X_j. If this leaves D_i unchanged, the algorithm just moves on to the next arc. But if this revises D_i (makes the domain smaller), then add all arcs (X_k, X_i) to the queue where X_k is a neighbor of X_i. This has to be done because the change in D_i might enable further reductions in the domains of D_k, even if X_k is considered previously. If D_i is revised down to nothing, then the whole CSP has no consistent solution, and AC-3 can immediately return failure. Otherwise, it keeps checking, trying to remove values from the domains of variables until no more arcs are in the queue. At that point, a CSP that is equivalent to the original CSP is left - they both have the same solutions - but the arc-consistent CSP will in most cases be faster to search because its variables have smaller domains

Path Consistency

Path consistency tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.

A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$. This is called path consistency because one can think of it as looking at a path from X_i to X_j with X_m in the middle.

Let's consider the path consistency fares in coloring the Australia map with two colors. Consider the set {WA, SA}which is path consistent with respect to NT. By enumerating the consistent assignments to the set. there are only two assignments: {WA = red ,SA = blue}and {WA = blue, SA = red}.In both of these assignments NT can be neither red nor blue (because it would conflict with either WA or SA). Because there is no valid choice for NT, both assignments can be eliminated there is no valid assignments for {WA, SA}. Therefore, there can be no solution to this problem.

K-consistency

Stronger forms of propagation can be defined with the notion of k -consistency. A CSP is k- consistent if, for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any k^{th} variable

A CSP is strongly k -consistent if it is k-consistent and is also$(k - 1)$ -consistent, $(k - 2)$ - consistent, all the way down to 1-consistent.

Global constraints

A global constraint is one involving an arbitrary number of variables (but not necessarily all variables). Global constraints occur frequently in real problems and can be handled by special- purpose algorithms. For example, the All diff constraint says that all the variables involved must have distinct values (as in the cryptarithmetic problem and Sudoku puzzles).

One simple form of inconsistency detection for Alldiff constraints works as follows: if m variables are involved in the constraint, and if they n have possible distinct values altogether, and m>n , then the constraint cannot be satisfied.

This leads to the following simple algorithm:

- First, remove any variable in the constraint that has a singleton domain, and delete that variable's value from the domains of the remaining variables.
- Repeat as long as there are singleton variables.
- If at any point an empty domain is produced or there are more variables than domain values left, then an inconsistency has been detected.

Sudoku

- The popular Sudoku puzzle has introduced millions of people to constraint satisfaction problems, although they may not realize it.
- A Sudoku board consists of 81 squares, some of which are initially filled with digits from 1 to 9. The puzzle is to fill in all the remaining squares such that no digit appears twice in any row, column, or 3 × 3 box
- A row, column, or box is called a unit.

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Fig. 1.11.

A Sudoku puzzle can be considered a CSP with 81 variables, one for each square. The variable names A1 through A9 is used for the top row (left to right), down to I1 through I9 for the bottom row. The empty squares have the domain{1, 2, 3, 4, 5, 6, 7, 8, 9} and the pre-filled squares have a domain consisting of a single value. In addition, there are 27 different Alldiff constraints, one for each unit (row, column, and box of 9 squares):

Alldi ff(A1,A2,A3,A4,A5,A6,A7,A8,A9)

Alldi ff(B1,B2,B3,B4,B5,B6,B7,B8,B9)

...

Alldi ff(A1,B1,C1,D1,E1,F1,G1,H1,I1)

Alldi ff(A2,B2,C2,D2,E2,F2,G2,H2,I2)

...

Alldi ff(A1,A2,A3,B1,B2,B3,C1,C2,C3)

Alldi ff(A4,A5,A6,B4,B5,B6,C4,C5,C6)

...

Backtracking Search for CSP

The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The algorithm is shown in Figure.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({},csp)
function BACKTRACK(assignment.csp) returns a solution, or failure
    if assignment is complete then return assignment
    var←SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment,csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INTERFENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ←BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
    remove {var = value} and inferences from assignment
return failure
```

It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

Part of the search tree for the Australia problem is shown in Figure ,

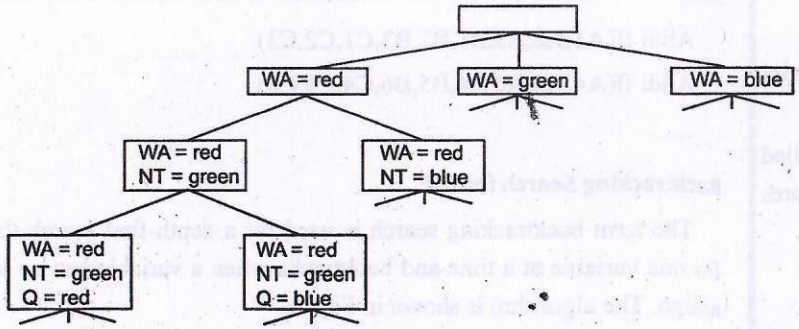


Fig. 1.12.

where variables are assigned in the order WA,NT,Q, ... Because the representation of CSPs is standardized, there is no need to supply BACKTRACKING-SEARCH with a domain-specific initial state, action function, transition model, or goal test.

TWO MARKS QUESTION AND ANSWERS (PART - A)

1. ***When did first A.I idea was Proposed?***

In the year 1943, Warren Mc Culloch and Walter pits proposed a model of Artificial neurons.

2. ***What is A.I?***

Artificial Intelligence is a branch of computer science that deals with developing intelligent machines which can behave like human, think like human, and has ability to take decisions by their own.

Artificial Intelligence is a combination of two words Artificial and Intelligence, which refers to man-made intelligence. Therefore, when machines are equipped with man-made intelligence to perform intelligent tasks similar to humans, it is known as Artificial Intelligence.

3. ***List Some Applications of A.I?***

- **Game Playing:**

  AI is widely used in Gaming. Different strategic games such as Chess, where the machine needs to think logically, and video games to provide real-time experiences use Artificial Intelligence.

- **Robotics:**

  Artificial Intelligence is commonly used in the field of Robotics to develop intelligent robots. AI implemented robots use real-time updates to sense any obstacle in their path and can change the path instantly. AI robots can be used for carrying goods in hospitals and industries and can also be used for other different purposes.

- **Healthcare:**

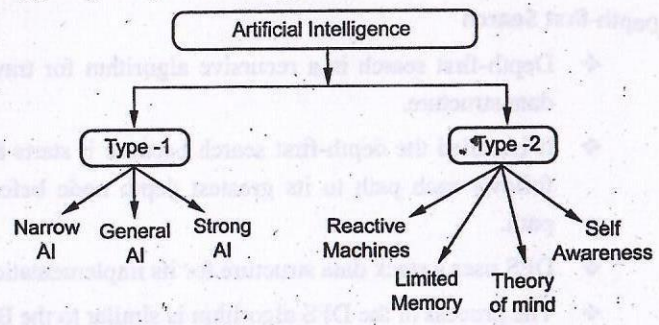
  In the healthcare sector, AI has diverse uses. In this field, AI can be used to detect diseases and cancer cells. It also helps in finding new drugs with the use of historical data and medical intelligence.

4. ***What are the Four Phases of Problem solving Agents?***

- Goal Formulation
- Problem Formulation

- ❖ Search
- ❖ Execution

5. *List the Types of Artificial Intelligence?*



6. *Different Between Super A.I & Weak A.I?*

S.No	Weak AI	Super AI
1	Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence.	Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI.
2	Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.	Some key characteristics of strong AI include capability include the ability to think, to reason, solve the puzzle, make judgments, plan, learn, and communicate by its own.

7. *What are the Search algorithm Terminologies?*

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- ❖ **Search Space:** Search space represents a set of possible solutions, which a system may have.

- ❖ **Start State:** It is a state from where agent begins the search.
- ❖ **Goal Test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

Actions: It gives the description of all the available actions to the agent.

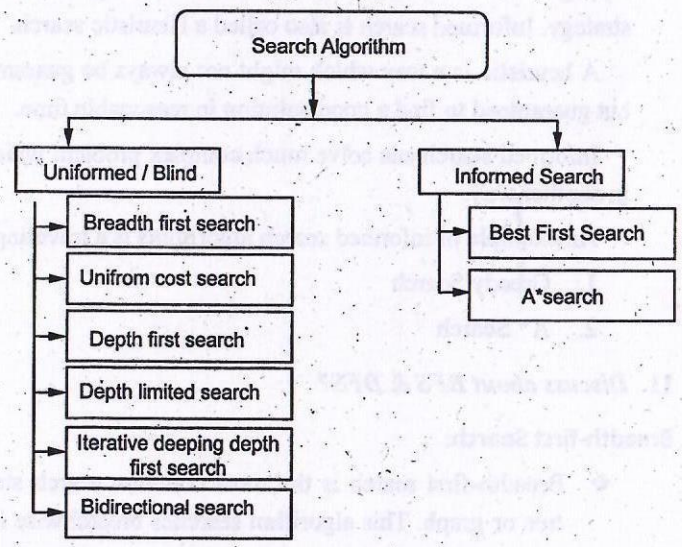
Transition model: A description of what each action do, can be represented as a transition model.

Path Cost: It is a function which assigns a numeric cost to each path.

Solution: It is an action sequence which leads from the start node to the goal node.

Optimal Solution: If a solution has the lowest cost among all solutions.

8. *What are the types of Search Algorithm?*



9. *What is Blind Search and it types?*

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and

goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

10. *Write about Informed search?*

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

11. *Discuss about BFS & DFS?*

Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Depth-first Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

12. *Explain about heuristic search strategies?*

A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not. Heuristics are said to be the problem-solving techniques that result in practical and quick solutions.

Heuristics are strategies that are derived from past experience with similar problems. Heuristics use practical methods and shortcuts used to produce the solutions that may or may not be optimal, but those solutions are sufficient in a given limited timeframe.

13. *Why do we need heuristics?*

Heuristics are used in situations in which there is the requirement of a short-term solution. On facing complex situations with limited resources and time, Heuristics can help the companies to make quick decisions by shortcuts and approximated calculations. Most of the heuristic methods involve mental shortcuts to make decisions on past experiences.

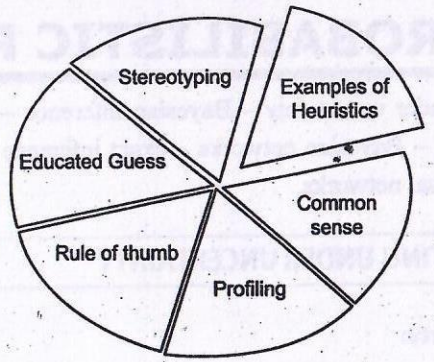
14. *What is Local maxima and Ridges in Hill Climbing?*

Local Maxima: A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.

Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached

in a single move. Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

15. *Give an real life example of Heuristics search?*



16. *Types of games in A.I?*

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.

17. *Define CSP.*

A problem is solved when each variable has a value that satisfies all the constraints on the variable. Such a problem is called a constraint satisfaction problem, or CSP

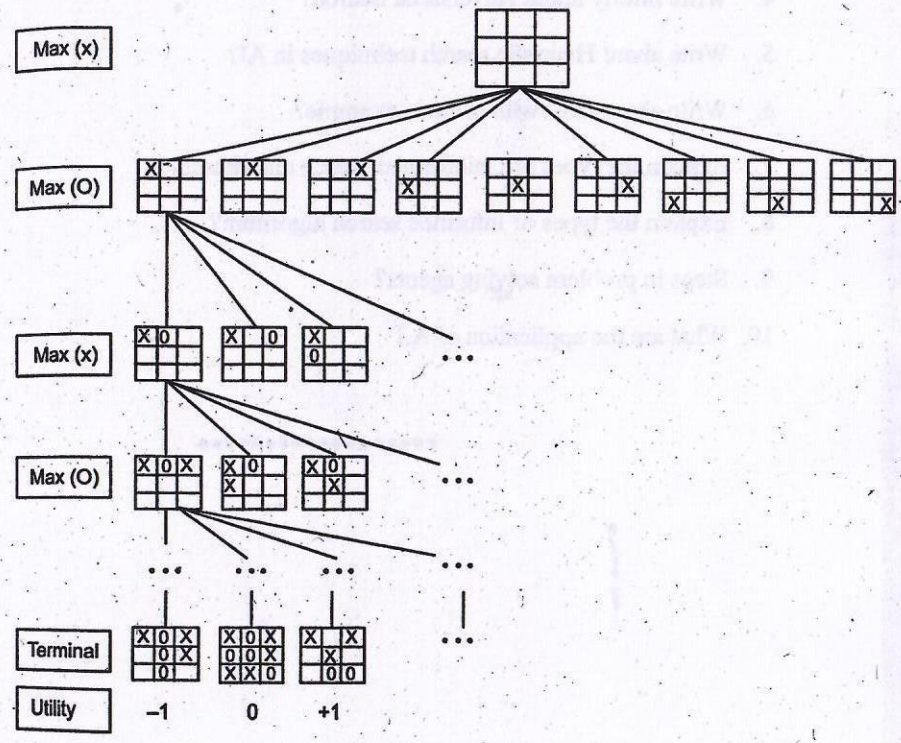
18. *What is K-Consistency?*

Stronger forms of propagation can be defined with the notion of k consistency. A CSP is k - consistent if, for any set of $k-1$ variables and for any

consistent assignment to those variables, a consistent value can always be assigned to any k^{th} variable

A CSP is strongly k - consistent if it is k - consistent and is also$(k-1)$ - consistent,$(k-2)$ - consistent, all the way down to 1-consistent.

19. *Draw a Game Tree?*



PART - B & C

1. Give your detail views on A.I
2. Solve the Cryptarithmetic problem CSP – SEND + MORE = MONEY
3. Explain Hill Climbing Search algorithm
4. Write briefly about Adversarial Search?
5. Write about Heuristic search techniques in AI?
6. Write about CSP with suitable example?
7. Explain the types of Uninformed search algorithms?
8. Explain the types of Informed search algorithm?
9. Steps in problem solving agents?
10. What are the application of A.I

****

UNIT II

PROBABILISTIC REASONING

Acting under uncertainty – Bayesian inference – naïve bayes models. Probabilistic reasoning – Bayesian networks – exact inference in BN – approximate inference in BN – causal networks.

2.1. ACTING UNDER UNCERTAINTY

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of Uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic Reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of Probabilistic Reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- Bayes' rule
- Bayesian Statistics

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.

$P(A) = 0$, indicates total uncertainty in an event A.

$P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\text{Probability of Occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional Probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

Where $P(A \wedge B)$ = Joint probability of A and B

$P(B)$ = Marginal probability of B.

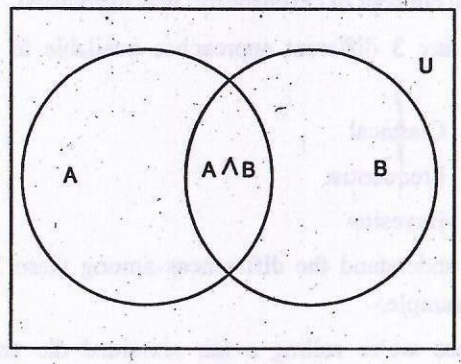


Fig. 2.1.

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B \mid A) = \frac{P(A \wedge B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of $P(A \wedge B)$ by $P(B)$.

Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics

2.2. BAYESIAN INFERENCE

Statistics is the study to help us quantify the way to measure uncertainty and hence, the concept of 'Probability' was introduced.

There are 3 different approaches available to determine the probability of an event.

- Classical
- Frequentist
- Bayesian

Let's understand the differences among these 3 approaches with the help of a simple example.

Suppose we're rolling a fair six-sided die and we want to ask what is the probability that the die shows a four? Under the Classical framework, all the possible outcomes are equally likely i.e., they have equal probabilities or chances. Hence, answering the above question, there are six possible outcomes and they are all equally likely. So, the probability of a four on a fair six-sided die is just 1/6. This Classical approach works well when we have well-defined equally likely outcomes. But when things get a little subjective then it may become a little complex.

On the other hand, Frequentist definition requires us to have a hypothetical infinite sequence of a particular event and then to look at the relevant frequency in that hypothetical infinite sequence. In the case of rolling a fair six-sided die, if we roll it for the infinite number of times then 1/6th of the time, we will get a four and hence, the probability of rolling four in a six-sided die will be 1/6 under frequentist definition as well.

Now if we proceed a little further and ask if our die is fair or not. Under frequentist paradigm, the probability is either zero when it's not a fair die and one if it is a fair die because under frequentist approach everything is measured from a physical perspective and hence, the die can be either fair or not. We cannot assign a probability to the fairness of the die. Frequentists are very objective in how they define probabilities but their approach cannot give intuitive answers for some of the deeper subjective issues.

Here comes the advantage of the Bayesian approach

Bayesian perspective allows us to incorporate personal belief/opinion into the decision-making process. It takes into account what we already know about a particular problem even before any empirical evidence. Here we also have to acknowledge the fact my personal belief about a certain event may be different than others and hence, the outcome that we will get using the Bayesian approach may also be different.

For example, I may say that there is a 90% probability that it will rain tomorrow whereas my friend may say I think there is a 60% chance that it will rain tomorrow. So inherently Bayesian perspective is a subjective approach to probability, but it gives more intuitive results in a mathematically rigorous framework than the Frequentist approach. Let's discuss this in detail in the following sections.

2.2.1. BAYES' THEOREM

Simplistically, Bayes' theorem can be expressed through the following mathematical equation

$$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)}$$

where A is an event and B is evidence. So, P(A) is the prior probability of event A and P(B) is evidence of event B. Hence, P(B|A) is the likelihood. The denominator is

a normalizing constant. So, Bayes' Theorem gives us the probability of an event based on our prior knowledge of the conditions that might be related to the event and updates that conditional probability when some new information or evidence comes up.

$$\text{Posterior Probability} = \frac{\text{Prior} \times \text{Likelihood Function}}{\text{Evidence}}$$

Now let's focus on the 3 components of the Bayes' theorem

- Prior
- Likelihood
- Posterior

Prior Distribution – This is the key factor in Bayesian inference which allows us to incorporate our personal beliefs or own judgements into the decision-making process through a mathematical representation. Mathematically speaking, to express our beliefs about an unknown parameter θ we choose a distribution function called the prior distribution. This distribution is chosen before we see any data or run any experiment.

How do we choose a prior? Theoretically, we define a cumulative distribution function for the unknown parameter θ. In basic context, events with the prior probability of zero will have the posterior probability of zero and events with the prior probability of one, will have the posterior probability of one. Hence, a good Bayesian framework will not assign a point estimate like 0 or 1 to any event that has already occurred or already known not to occur. A very handy widely used technique of choosing priors is using a family of distribution functions that is sufficiently flexible such that a member of the family will represent our beliefs. Now let's understand this concept a little better.

(*i*) **Conjugate Priors** – Conjugacy occurs when the final posterior distribution belongs to the family of similar probability density functions as the prior belief but with new parameter values which have been updated to reflect new evidence / information. Examples Beta-Binomial, Gamma - Poisson or Normal-Normal.

(*ii*) **Non-conjugate Priors** – Now, it is also quite possible that the personal belief cannot be expressed in terms of a suitable conjugate prior and for those cases simulation tools are applied to approximate the posterior distribution. An example can be Gibbs sampler.

(*iii*) **Un-informative prior** – Another approach is to minimize the amount of information that goes into the prior function to reduce the bias. This is an attempt to have the data have maximum influence on the posterior. These priors are known as uninformative Priors but for these cases, the results might be pretty similar to the frequentist approach.

Likelihood – Suppose θ is the unknown parameter that we are trying to estimate. Let's represent fairness of a coin with θ. Now to check the fairness, we are flipping a coin infinitely and each time it is either appearing as 'head' or 'tail' and we are assigning a 1 or 0 value accordingly. This is known as the Bernoulli Trials. Probability of all the outcomes or 'X's taking some value of x given a value of theta. We're viewing each of these outcomes as independent and hence, we can write this in product notation. This is the probability of observing the actual data that we collected (head or tail), conditioned on a value of the parameter theta (fairness of coin) and can be expressed as follows-

$$p(X \mid \theta) = (\theta)^x \times (1-\theta)^{1-x}$$

This is the concept of likelihood which is the density function thought of as a function of theta. To maximize the likelihood i.e., to make the event most likely to occur for the data we have, we will choose the theta that will give us the largest value of the likelihood. This is referred to as the maximum likelihood estimate or MLE. Additionally, a quick reminder is that the generalization of the Bernoulli when we have N repeated and independent trials is a binomial. We will see the application later in the article.

Posterior Distribution – This is the result or output of the Bayes' Theorem. A posterior probability is the revised or updated probability of an event occurring after taking into consideration new information. We calculate the posterior probability $p(\theta|X)$ i.e., how probable is our hypothesis about θ given the observed evidence.

Mechanism of Bayesian Inference:

The Bayesian approach treats probability as a degree of beliefs about certain event given the available evidence. In Bayesian Learning, Theta is assumed to be a random variable. Let's understand the Bayesian inference mechanism a little better with an example.

Inference example using Frequentist vs Bayesian approach: Suppose my friend challenged me to take part in a bet where I need to predict if a particular coin is fair or not. She told me "Well; this coin turned up 'Head' 70% of the time when I flipped it several times. Now I am giving you a chance to flip the coin 5 times and then you have to place your bet." Now I flipped the coin 5 times and Head came up twice and tail came up thrice. At first, I thought like a frequentist.

So, θ is an unknown parameter which is a representation of fairness of the coin and can be defined as

$$\theta = \{\text{fair, loaded}\}$$

Additionally, I assumed that the outcome variable X (whether head or tail) follows Binomial distribution with the following functional representation

$$(nCx)\, p^x \times (1-p)^{n-x}$$

$$nCx = \frac{n!}{x!(n-x)!}$$

Now in our case $n = 5$.

Now my likelihood function will be

$$f(X \mid \theta) = \frac{5!}{X! \times (5-X)!} \times \left(\frac{1}{2}\right)^5, \text{ if } \theta = \text{fair}$$

$$f(X \mid \theta) = \frac{5!}{X! \times (5-X)!} \times (0.7)^X \times (0.3)^{5-X}, \text{ if } \theta = \text{loaded}$$

Now, I saw that head came up twice, so my $X = 2$.

When $X = 2$, $f(\theta \mid X = 2) = 0.31$ if θ = fair

= 0.13 if θ != loaded

Therefore, using the frequentist approach I can conclude that maximum likelihood i.e., MLE (theta hat) = fair.

$$\text{MLE}(\hat{\theta}) = \text{fair}$$

Now comes the tricky part. If the question comes how sure am I about my prediction? I will not be able to answer that question perfectly or correctly as in a frequentist world, a coin is a physical object and hence, my probability can be either 0 or 1 i.e., the coin is either fair or not.

Here comes the role of Bayesian inference which will tell us the uncertainty in my prediction i.e., P(θ | X = 2). The Bayesian inference allows us to incorporate our

knowledge/information about the unknown parameter θ even before looking at any data. Here, suppose I know my friend pretty well and I can say with 90 % probability that she has given me a loaded coin.

Therefore, my prior P(loaded) = 0.9. I can now update my prior belief with data and get the posterior probability using Bayes' Theorem.

$$f(X \mid \theta) = \frac{f(X \mid \theta) f(\theta)}{\sum f(X \mid \theta) f(\theta)}$$

My numerator calculation will be as follows-

$$f(X \mid \theta) \times f(\theta) = \frac{5!}{X! \times (5-X)!} \times \left(\frac{1}{2}\right)^5 \times (0.1), \text{ when } \theta = \text{fair}$$

$$f(X \mid \theta) \times f(\theta) = \frac{5!}{X! \times (5-X)!} \times (0.7)^X \times (0.3)^{5-X} \times (0.9), \text{when } \theta = \text{loaded}$$

The denominator is a constant and can be calculated as the expression below. Please note that we are here basically summing up the expression over all possible values of θ which is only 2 in this case i.e., fair or loaded.

$$\sum f(X \mid \theta) f(\theta) = \frac{5!}{X! \times (5-X)!} \times \left(\frac{1}{2}\right)^5 + \frac{5!}{X! \times (5-X)!} \times (0.7)^X \times (0.3)^{5-X}$$

Hence, after replacing X with 2 we can calculate the Bayesian probability of the coin being loaded or fair. Do it yourself and let me know your answer! However, you will realize that this conclusion contains more information to make a bet than the frequentist approach.

2.2.2. APPLICATION OF BAYESIAN INFERENCE IN FINANCIAL RISK MODELING

Bayesian inference has found its application in various widely used algorithms e.g., regression, Random Forest, neural networks, etc. Apart from that, it also gained popularity in several Bank's Operational Risk Modelling. Bank's operation loss data typically shows some loss events with low frequency but high severity. For these typical low-frequency cases, Bayesian inference turns out to be useful as it does not require a lot of data.

Earlier, Frequentist methods were used for operational risk models but due to its inability to infer about the parameter uncertainty, Bayesian inference was considered to be more informative as it has the capacity of combining expert opinion with actual data to derive the posterior distributions of the severity and frequency distribution parameters. Generally, for this type of statistical modeling, the bank's internal loss

data is divided into several buckets and the frequencies of each bucket loss are determined by expert judgment and then fitted into probability distributions.

2.3. NAÏVE BAYES MODELS

Naïve Bayes Algorithm

- 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

Naïve Bayes

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

- The formula for Bayes' theorem is given as:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Where,

P(A|B) is Posterior Probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood Probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

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	5

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(Yes|Sunny) = P(Sunny|Yes)*P(Yes) / P(Sunny)

P(Sunny|Yes) = 3 / 10 = 0.3

P(Sunny) = 0.35

P(Yes) = 0.71

So P(Yes|Sunny) = 0.3*0.71 / 0.35 = **0.60**

P(No|Sunny) = P(Sunny|No) * P(No) / P(Sunny)

P(Sunny|NO) = 2 / 4 = 0.5

P(No) = 0.29

P(Sunny) = 0.35

So P(No|Sunny) = 0.5*0.29 / 0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny) > P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems.**

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for Credit Scoring.
- It is used in medical data classification.
- It can be used in real-time predictions because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as Spam filtering and Sentiment analysis.

2.3.1. BAYESIAN NETWORK

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.**

Bayesian networks are probabilistic, because these networks are built from a **probability distribution,** and also use probability theory for prediction and anomaly detection.

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:

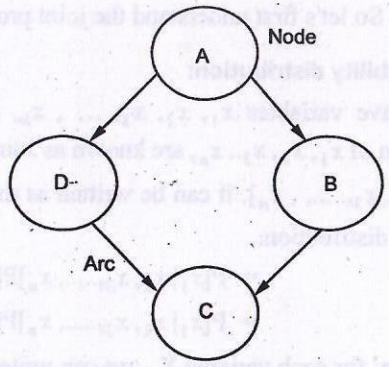


Fig. 2.2.

- 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.

Note The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG.

The Bayesian network has mainly two components:

- Causal Component
- Actual numbers

Each node in the Bayesian network has condition probability distribution $P(X_i \mid Parent(X_i))$, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables $x_1, x_2, x_3, \ldots, x_n$, then the probabilities of a different combination of $x_1, x_2, x_3 .. x_n$, are known as Joint probability distribution.

$P[x_1, x_2, x_3, \ldots, x_n]$, it can be written as the following way in terms of the joint probability distribution.

$$= P[x_1 | x_2, x_3, \ldots, x_n] P[x_2, x_3, \ldots, x_n]$$

$$= P[x_1 | x_2, x_3, \ldots, x_n] P[x_2 | x_3, \ldots, x_n] \ldots P[x_n - 1 | x_n] P[x_n].$$

In general for each variable X_i, we can write the equation as:

$$P(X_i | X_i - 1, \ldots, X_1) = P(X_i | \text{Parents}(X_i))$$

2.3.2. 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 the Harry.

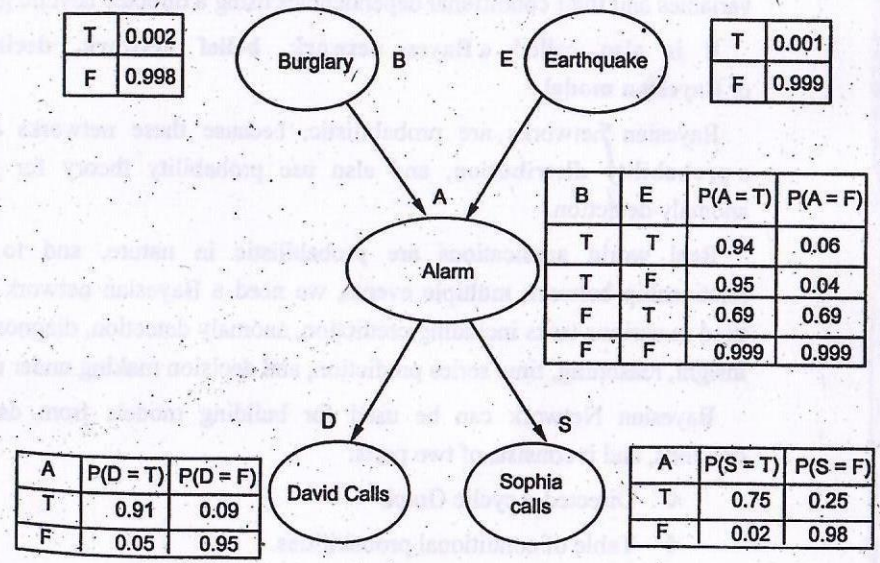
Solution:

- 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 or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- Burglary (B)
- Earthquake(E)
- Alarm(A)
- David Calls(D)
- Sophia calls(S)



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A = T)	P(A = F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P(D = T)	P(D = F)
T	0.91	0.09
F	0.05	0.95

A	P(S = T)	P(S = F)
T	0.75	0.25
F	0.02	0.98

Fig. 2.3.

We can write the events of problem statement in the form of probability: P[D, S, A, B, E], can rewrite the above probability statement using joint probability distribution:

$$\begin{aligned} P[D, S, A, B, E] &= P[D \mid S, A, B, E].\ P[S, A, B, E] \\ &= P[D \mid S, A, B, E].\ P[S \mid A, B, E].\ P[A, B, E] \\ &= P[D \mid A].\ P[S \mid A, B, E].\ P[A, B, E] \\ &= P[D \mid A].\ P[S \mid A].\ P[A \mid B, E].\ P[B, E] \\ &= P[D \mid A].\ P[S \mid A].\ P[A \mid B, E].\ P[B \mid E].\ P[E] \end{aligned}$$

Let's take the observed probability for the Burglary and earthquake component:

P(B = True) = 0.002, which is the probability of burglary.

P(B = False) = 0.998, which is the probability of no burglary.

P(E = True) = 0.001, which is the probability of a minor earthquake

P(E = 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:

$$\begin{aligned} P(S, D, A, \neg B, \neg E) &= P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E). \\ &= 0.75 * 0.91 * 0.001 * 0.998 * 0.999 \\ &= 0.00068045. \end{aligned}$$

2.3.3. EXACT INFERENCE IN BAYESIAN NETWORK

Inference by enumeration

Any conditional probability can be computed by summing terms from the full joint distribution. More specifically, a query P (X | *e*) can be answered using Equation (13.6), which we repeat here for convenience:

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_{y} P(X, e, y)$$

Now, a Bayesian network gives a complete representation of the full joint distribution. More specifically, Equation shows that the terms $P(x, e, y)$ in the joint distribution can be written as products of conditional probabilities from the network. Therefore, a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.

In Figure, an algorithm, ENUMERATE-JOINT-ASK, was given for inference by enumeration from the full joint distribution.

The algorithm takes as input a full joint distribution P and looks up values therein. It is a simple matter to modify the algorithm so that it takes as input a Bayesian network b_n and "looks up" joint entries by multiplying the corresponding CPT entries from bn.

Consider the query P(Burglary | John Calls = true, Mary Calls = true).

The hidden variables for this query are Earth quake and Alarm. Equation using initial letters for the variables in order to shorten the expressions, we have

$$P(B \mid j, m) = \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, e, a, j, m)$$

The semantics of Bayesian networks (Equation) then gives us an expression in terms of entries. For simplicity, we will do this just for Burglary = true:

$$P(b \mid j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a)$$

To compute this expression, we have to add four terms, eaclh computed by multiplying five numbers. In the worst case, where we have to sum out almost all the variables, the complexity of the algorithm for a network with n Boolean variables is O(n2n). An improvement can be obtained from the following simple observations:

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(j \mid a)P(m \mid a)$$

the P(b) term is a constant and can be moved outside the summaltions over a and e, and the 13(e) term can be moved outside the summation over a. Hence, we have

This expression can be evaluated by looping through the variables in order, multiplying CPT entries as we go. For each summation, we also need to loop over the variable's possible values. The structure of this computation is shown in Figure 14.8. Using the numbers from Figure, it obtain $P(b_1 \mid j, m) = a \times 0.00059224$. 'The corresponding computation for yields a × 0.0014919; hence

$$P(B \mid j, m) = \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle$$

That is, the chance of a burglary, given calls from both neighbors, is about 28%. The evaluation process for the expression in Equation is shown as an expression tree in Figure. The ENUMERATION-ASK algorithm evaluates such trees using depth-first recursion. Thus, the space complexity of ENUMERATION-ASK is only linear in the number of variables-effectively, the algorithm sums over the full joint distribution without ever constructing it explicitly. Unfortunately, its time complexity for a network with n Boolean variables is always 0(2n)-better than the O(n2n) for the simple approach described earlier, but still rather grim. One thing to note about the tree is that it makes explicit the repeated subexpressions that are evaluated lby the algorithm. The products P(jla) P(mla) and P(jl1a) P(ml1a) are computed twice, once for each value of e. The next section describes a general method that avoids such wasted computations.

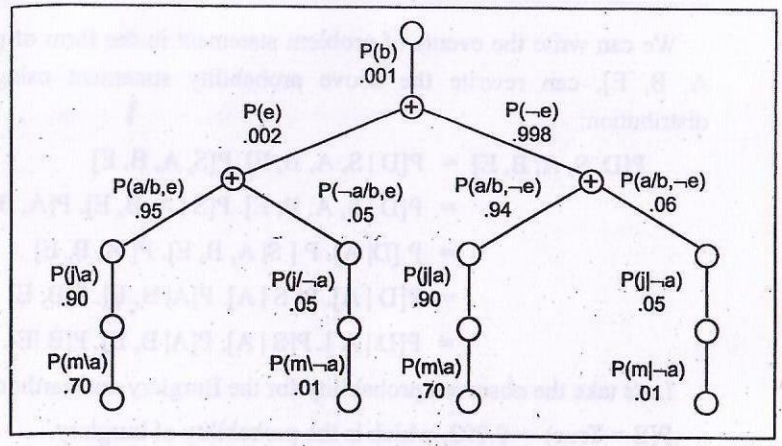


Fig. 2.4. The Structure of the expression shown in equation. The evaluation proceeds top-down, multiplying values along each path and summing at the '+' nodes. Notice the repetition of the paths for and m.

2.3.4. APPROXIMATE INFERENCE IN BAYESIAN NETWORK

- A method of estimating probabilities in Bayesian networks also called 'Monte Carlo' algorithms
- We will discuss two types of algorithms: direct sampling and Markov chain sampling
- Exact inference becomes intractable for large multiply-connected networks
- Variable elimination can have exponential time and space complexity
- Exact inference is strictly HARDER than NP-complete problems (#P-hard)

Direct Sampling

- Take samples of events
- We expect the frequency of the samples to converge on the probability of the event
- Used to compute conditional probabilities P(X|e)
- Generate samples as before
- Reject samples that do not match evidence
- Estimate by counting the how often event X is in the resulting samples

Likelihood Weighting

- Avoid inefficiency of rejection sampling
- Fix values for evidence variables and only sample the remaining variables
- Weight samples with regard to how likely they are

Markov Chain Sampling

- Generate events by making a random change to the preceding event
- This change is made using the Markov Blanket of the variable to be changed
- Markov Blanket = parents, children, children's parents
- Tally and normalize results

2.4. CAUSAL NETWORK

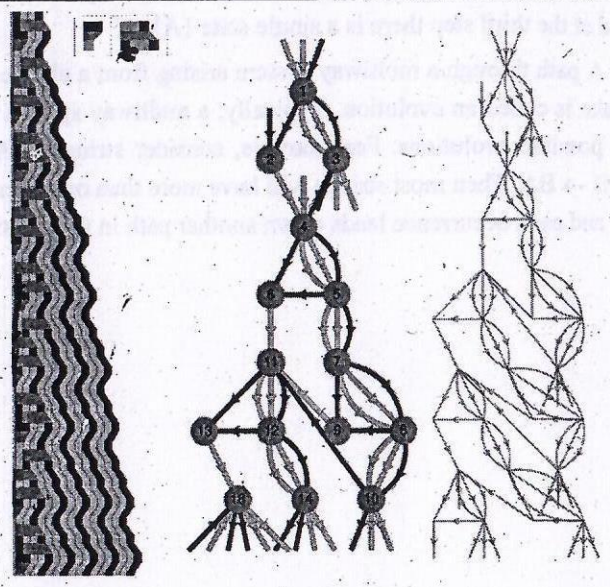


Fig. 2.5.

A causal network is an acyclic digraph arising from an evolution of a substitution system, (BB→A, AAB →BAAB) and representing its history. A B A A B The

illustration above shows a causal network corresponding to the rules (applied in a left-to-right scan) and initial condition.

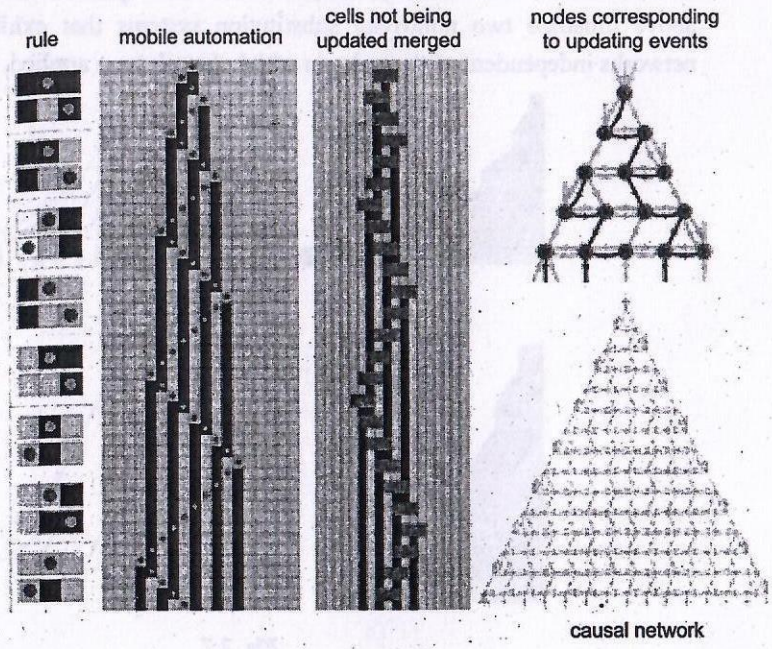


Fig. 2.6.

The figure above shows the procedure for diagrammatically creating a causal network from a mobile automaton

In an evolution of a multiway system, each substitution event is a vertex in a causal network. Two events which are related by causal dependence, meaning one occurs just before the other, have an edge between the corresponding vertices in the causal network. More precisely, the edge is a directed edge leading from the past event to the future event.

Some causal networks are independent of the choice of evolution, and these are called causally invariant.

2.4.1. CAUSAL INVARIANCE

A multiway system that generates causal networks which are all isomorphic as acyclic digraphs is said to exhibit causal invariance, and the causal network itself

is also said to be causally invariant. Essentially, causal invariance means that no matter which evolution is chosen for a system, the history is the same in the sense that the same events occur and they have the same causal relationships. The figures above illustrate two nontrivial substitution systems that exhibit the same causal networks independent of the order in which the rules are applied.

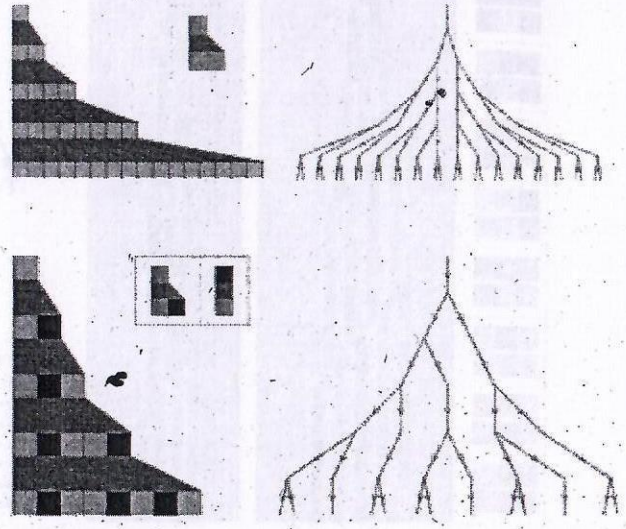


Fig. 2.7.

Whenever two rule hypotheses overlap in an evolution, the corresponding system is not causally invariant. Hence, the simplest way to search for causal invariance is to use rules whose hypotheses can never overlap except trivially. An overlap can involve one or two strings. For example, A B does not have any overlaps. However, A B A can overlap as A B A B A and the set of strings {ABB, AAB} can overlap as AABB

A mobile automaton simulated by a string substitution system is an example of a causally invariant network in which rule hypotheses overlap, as long as the initial condition contains only a single active cell.

Multiway System

A multiway system is a kind of substitution system in which multiple states are permitted at any stage. This accommodates rule systems in which there is more than one possible way to perform an update.

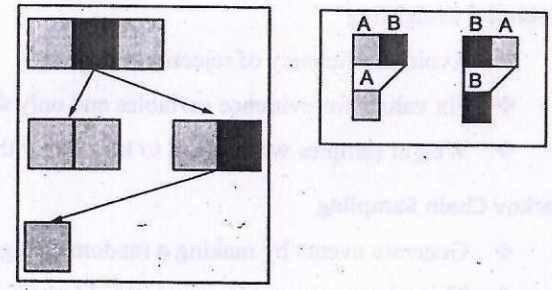


Fig. 2.8.

A simple example is a string substitution system. For instance, take the rules {AB→ A, BA → B} and the initial condition ABA. There are two choices for how to proceed. Applying the first rule yields the evolution ABA to AA, while applying the second rule yields the evolution ABA→AB → A. So at the first step, there is a single state ({ABA}), at the second step there are two states {AA, AB}, and at the third step there is a single state {A}.

A path through a multiway system arising from a choice of which substitutions to make is called an evolution. Typically, a multiway system will have a large number of possible evolutions. For example, consider strings of A s and Bs with the rule A B → BA. Then most strings will have more than one occurrence of the substring A B, and each occurrence leads down another path in the multiway system.

TWO MARKS QUESTION AND ANSWERS (PART - A)

1. ***What is Probabilistic reasoning?***
   - ❖ Probabilistic reasoning is a form of knowledge representation in which the concept of probability is used to indicate the degree of uncertainty in knowledge. In AI, probabilistic models are used to examine data using statistical codes.
   - ❖ Probabilistic reasoning is using logic and probability to handle uncertain situations. An example of probabilistic reasoning is using past situations and statistics to predict an outcome.

2. ***What is Markov's Decision process?***

   The solution for a reinforcement learning problem can be achieved using the Markov decision process or MDP. Hence, MDP is used to formalize the RL problem. It can be said as the mathematical approach to solve a reinforcement learning problem. The main aim of this process is to gain maximum positive rewards by choosing the optimum policy.

   MDP has four elements, which are:
   - ❖ A set of finite states S
   - ❖ A set of finite actions A
   - ❖ Rewards
   - ❖ Policy Pa

   In this process, the agent performs an action A to take a transition from state S_1 to S_2 or from the start state to the end state, and while doing these actions, the agent gets some rewards. The series of actions taken by the agent can be defined as the policy.

3. ***What is a Bayesian network, and why is it important in AI?***

   Bayesian networks are the graphical models that are used to show the probabilistic relationship between a set of variables. It is a directed cycle graph that contains multiple edges, and each edge represents a conditional dependency.

   Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection. It is important in AI as it is based on Bayes theorem and can be used to answer the probabilistic questions.

4. ***Define Bayesian Learning.***

   It calculates the probability of each hypotheses, given the data and makes predictions on that basis, (i.e.) predictions are made by using all the hypotheses, weighted by their probabilities rather than by using just single "best" hypotheses.

5. ***Define Naïve Bayes model.***

   In this model, the "class" variable C is the root and the "attribute" variable XI are the leaves. This model assumes that the attributes are conditionally independent of each other, given the class.

6. ***What is exact inference in Bayesian network?***

   Exact inference. In exact inference, we analytically compute the conditional probability distribution over the variables of interest.

7. ***What are the 3 different approaches available to determine the probability of an event?***
   - ❖ Classical
   - ❖ Frequentist
   - ❖ Bayesian

8. ***Write 3 types of Naïve Bayes Model.***
   - ❖ Gaussian
   - ❖ Multinomial
   - ❖ Bernoulli

9. ***What are the applications of Naïve Bayes Classifier?***
   - ❖ It is used for Credit Scoring.
   - ❖ It is used in medical data classification.
   - ❖ It can be used in real-time predictions because Naïve Bayes Classifier is an eager learner.
   - ❖ It is used in Text classification such as Spam filtering and Sentiment analysis.

10. ***What is approximate inferencing in Bayesian network?***

    Computing posterior and marginal probabilities constitutes the backbone of almost all inferences in Bayesian networks. These computations are known to be

intractable in general, both to compute exactly and to approximate (e.g., by sampling algorithms).

11. ***What is Markov Chain Sampling?***
    - ❖ Generate events by making a random change to the preceding event
    - ❖ This change is made using the Markov Blanket of the variable to be changed
    - ❖ Markov Blanket = parents, children, children's parents
    - ❖ Tally and normalize results
12. ***What is causal network?***

    A causal network is an acyclic digraph arising from an evolution of a substitution system, and representing its history. The illustration above shows a causal network {B B →A, A A B → B A A B} corresponding to the rules (applied in a left-to-right scan) and initial condition A B A A B

PART - B & C

1. Explain Bayes' theorem with example.
2. Explain Working of Naïve Bayes' Classifier with example.
3. Write in detail about exact inference in Bayesian network.
4. Explain in detail about approximate inference in Bayesian network.
5. Illustrate causal network with neat diagram.

****

UNIT III

SUPERVISED LEARNING

Introduction to machine learning – Linear Regression Models: Least squares, single & multiple variables, Bayesian linear regression, gradient descent, Linear Classification Models: Discriminant function – Perceptron algorithm, Probabilistic discriminative model - Logistic regression, Probabilistic generative model – Naive Bayes, Maximum margin classifier – Support vector machine, Decision Tree, Random Forests

3.1. LINEAR REGRESSION MODELING

Linear Regression has actually been around for a very long time (around 200 years). It is a linear model, i.e. it assumes a linear relationship between the input variables(x) and a single output variable(y). The y here is calculated by the linear combination of the input variables. Linear regression is a type of machine learning algorithm that is used to model the relation between scalar dependent and one or more independent variables. The case of having one independent variable is known as simple linear regression, while the case of having multiple linear regressions is known as multiple linear regressions.

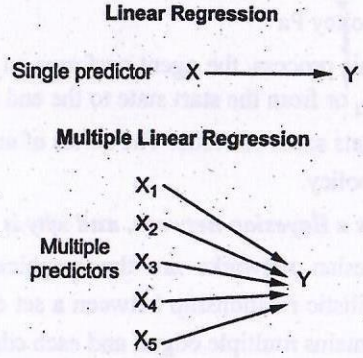


Fig. 3.1.

In both of these linear regressions, the model is constructed using a linear predictor function. The unknown data parameters are estimated using the available

dataset because this feature has various applications such as finance, economics, epidemiology, etc.

3.1.1. REGRESSION ALGORITHMS

Regression algorithms are used when you notice that the output is a continuous variable, whereas classification algorithms are used when the output is divided into sections such as Pass/Fail, Good / Average / Bad, etc. We have various algorithms for performing the regression or classification actions, with Linear Regression Algorithm being the basic algorithm in Regression.

Coming to this Regression, before getting into the algorithm, let me set the base for you. In schooling, I hope you remember the line equation concept. Let me give a brief about it. You were given two points on the XY plane, i.e. say (x_1, y_1) and (x_2, y_2), where y_1 is the output of x_1 and y_2 is the output of x_2, then the line equation that passes through the points is $(y - y_1) = m(x - x_1)$ where m is the slope of the line.

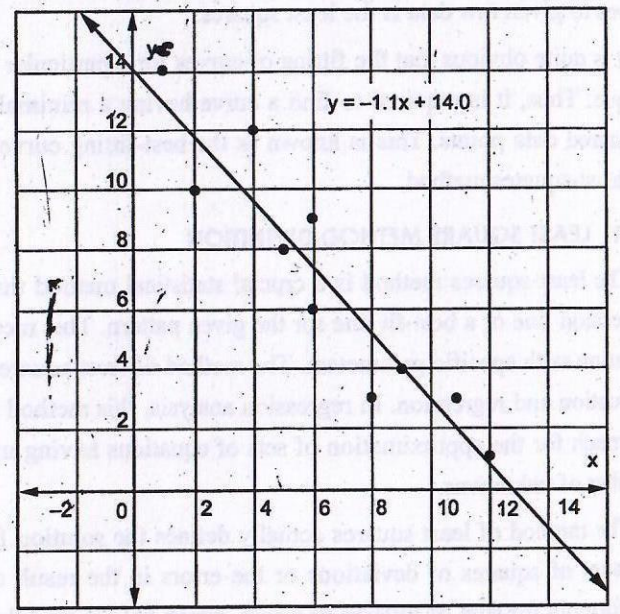


Fig. 3.2.

After finding the line equation, if you are given a point, say (x_3, y_3), you would be easily able to predict if the point lies on the line or the distance of the point from the line. This was the basic regression that I had done in schooling without even

realizing that this would have such great importance in Machine Learning. We generally do in this to identify the equation line or curve that could fit the input and output of the train data set properly and then use the same equation to predict the output value of the test data set. This would result in a continuous desired value.

Two Types of Linear Regression

Let's talk about two types of Linear Regression

- Simple Linear Regression
- Multiple Linear Regression

1. Simple Linear Regression

In Simple Linear Regression, we try to find the relationship between a single independent variable (input) and a corresponding dependent variable (output). This can be expressed in the form of a straight line.

The same equation of a line can be re-written as:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

1. **Y** represents the output or dependent variable.
2. β_0 **and** β_1 are two unknown constants that represent the intercept and coefficient (slope) respectively.
3. **ε** (Epsilon) is the error term.

The following is a sample graph of a Simple Linear Regression Model :

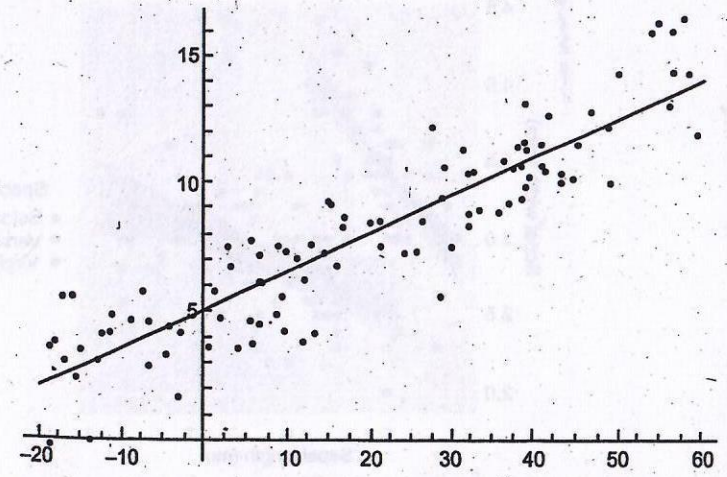


Fig. 3.3. Graph of Simple Linear Regression Model

Applications of Simple Linear Regression include :

1. **Predicting crop yields based on the amount of rainfall:** Yield is dependent variable while the amount of rainfall is independent variable.
2. **Marks scored by student based on number of hours studied (ideally) :** Here marks scored is dependent and number of hours studied is independent.
3. **Predicting the Salary of a person based on years of experience :** Thus Experience become the independent variable while Salary becomes the dependent variable

2. Multiple Linear Regression

In Multiple Linear Regression, we try to find the relationship between 2 or more independent variables (inputs) and the corresponding dependent variable (output). The independent variables can be continuous or categorical.

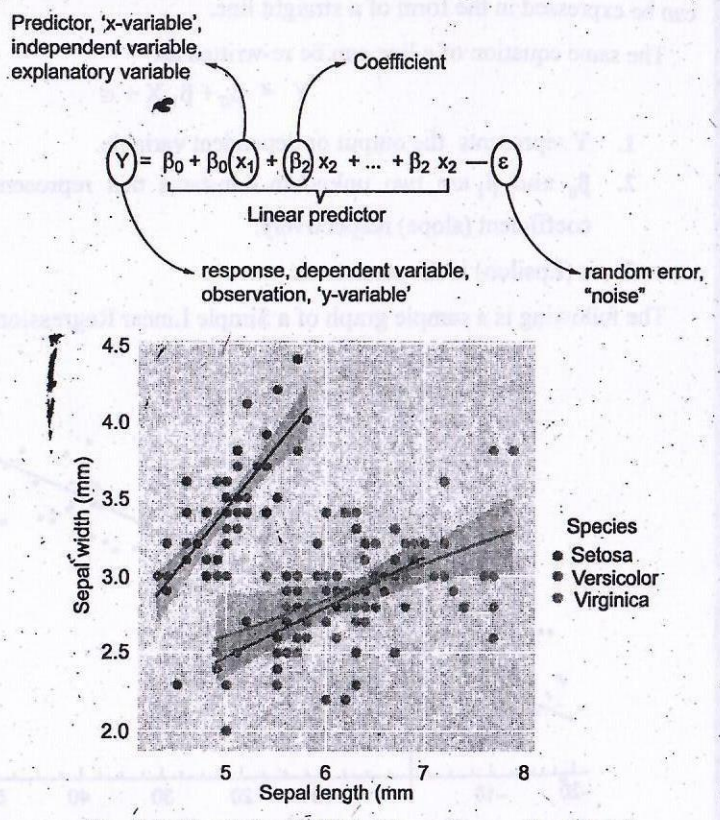


Fig. 3.4. Graph for Multiple Linear Regression Model

The equation that describes how the predicted values of y is related to p independent variables is called as Multiple Linear Regression equation

Above is the graph for Multiple Linear Regression Model, applied on the iris data set.

3.2. LEAST SQUARE METHOD

The **least square method** is the process of finding the best-fitting curve or line of best fit for a set of data points by reducing the sum of the squares of the offsets (residual part) of the points from the curve. During the process of finding the relation between two variables, the trend of outcomes are estimated quantitatively. This process is termed as **regression analysis**. The method of curve fitting is an approach to regression analysis. This method of fitting equations which approximates the curves to given raw data is the least squares.

It is quite obvious that the fitting of curves for a particular data set are not always unique. Thus, it is required to find a curve having a minimal deviation from all the measured data points. This is known as the best-fitting curve and is found by using the least-squares method.

3.2.1. LEAST SQUARE METHOD DEFINITION

The least-squares method is a crucial statistical method that is practised to find a regression line or a best-fit line for the given pattern. This method is described by an equation with specific parameters. The method of least squares is generously used in evaluation and regression. In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns.

The method of least squares actually defines the solution for the minimization of the sum of squares of deviations or the errors in the result of each equation. Find the formula for sum of squares of errors, which help to find the variation in observed data. The least-squares method is often applied in data fitting. The best fit result is assumed to reduce the sum of squared errors or residuals which are stated to be the differences between the observed or experimental value and corresponding fitted value given in the model.

There are two basic categories of least-squares problems:

- Ordinary or linear least squares
- Nonlinear least squares

These depend upon linearity or nonlinearity of the residuals. The linear problems are often seen in regression analysis in statistics. On the other hand, the non-linear problems are generally used in the iterative method of refinement in which the model is approximated to the linear one with each iteration.

3.2.2. LEAST SQUARE METHOD GRAPH

In linear regression, the line of best fit is a straight line as shown in the following diagram:

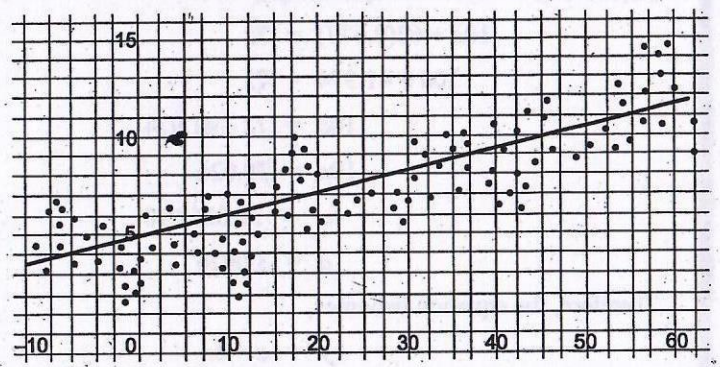
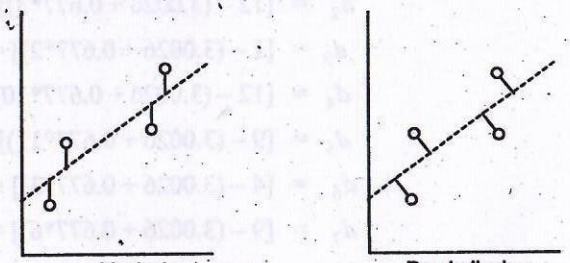


Fig. 3.5.

The given data points are to be minimized by the method of reducing residuals or offsets of each point from the line. The vertical offsets are generally used in surface, polynomial and hyper plane problems, while perpendicular offsets are utilized in common practice.



3.2.3. LEAST SQUARE METHOD FORMULA

The least-square method states that the curve that best fits a given set of observations, is said to be a curve having a minimum sum of the squared residuals (or deviations or errors) from the given data points. Let us assume that the given points of data are (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n) in which all x's are independent variables, while all y's are dependent ones. Also, suppose that $f(x)$ is the fitting curve and d represents error or deviation from each given point.

Now, we can write:

$$d_1 = y_1 - f(x_1)$$
$$d_2 = y_2 - f(x_2)$$
$$d_3 = y_3 - f(x_3)$$
$$\ldots\ldots$$
$$d_n = y_n - f(x_n)$$

The least-squares explain that the curve that best fits is represented by the property that the sum of squares of all the deviations from given values must be minimum, i.e:

$$S = \sum_{i=1}^{n} d_i^2$$

$$S = \sum_{i=1}^{n} [y_i - f_{x_i}]^2$$

$$S = d_1^2 + d_2^2 + d_3^2 + \ldots + d_n^2$$

Sum = Minimum Quantity

Suppose when we have to determine the equation of line of best fit for the given data, then we first use the following formula.

The equation of least square line is given by $Y = a + b\,X$

Normal equation for 'a':

$$\sum Y = na + b\sum X$$

Normal equation for 'b':

$$\sum XY = a\sum X + b\sum X^2$$

Solving these two normal equations we can get the required trend line equation.

Thus, we can get the line of best fit with formula $y = ax + b$

Solved Example

The Least Squares Model for a set of data (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n) passes through the point (x_a, y_a) where x_a is the average of the x_i's and y_a is the average of the y_i's. The below example explains how to find the equation of a straight line or a least square line using the least square method.

Question:

Consider the time series data given below:

x_i	8	3	2	10	11	3	6	5	6	8
y_i	4	12	1	12	9	4	9	6	1	14

Use the least square method to determine the equation of line of best fit for the data. Then plot the line.

Solution:

Mean of x_i values = $(8 + 3 + 2 + 10 + 11 + 3 + 6 + 5 + 6 + 8) / 10 = 62 / 10 = 6.2$

Mean of y_i values = $(4 + 12 + 1 + 12 + 9 + 4 + 9 + 6 + 1 + 14) / 10 = 72 / 10 = 7.2$

Straight line equation is $y = a + bx$.

The normal equations are

$$\sum y = an + b\sum x$$

$$\sum xy = a\sum x + b\sum x^2$$

X	y	x^2	xy
8	4	64	32
3	12	9	36
2	1	4	2
10	12	100	120
11	9	121	99
3	4	9	12
6	9	36	54
5	6	25	30
6	1	36	6
8	14	64	112
$\sum x = 62$	$\sum y = 72$	$\sum x^2 = 468$	$\sum xy = 503$

Substituting these values in the normal equations,

$$10a + 62b = 72 \quad \ldots(1)$$

$$62a + 468b = 503 \quad \ldots(2)$$

(1) × 62 – (2) × 10,

$$620a + 3844b - (620a + 4680b) = 4464 - 5030$$

$$-836b = -566$$

$$b = 566 / 836$$

$$b = 283/418$$

$$b = 0.677$$

Substituting $b = 0.677$ in equation (1),

$$10a + 62(0.677) = 72$$

$$10a + 41.974 = 72$$

$$10a = 72 - 41.974$$

$$10a = 30.026$$

$$a = 30.026 / 10$$

$$a = 3.0026$$

Therefore, the equation becomes,

$$y = a + bx$$

$$y = 3.0026 + 0.677x$$

This is the required trend line equation.

Now, we can find the sum of squares of deviations from the obtained values as:

$$d_1 = [4 - (3.0026 + 0.677*8)] = (-4.4186)$$

$$d_2 = [12 - (3.0026 + 0.677*3)] = (6.9664)$$

$$d_3 = [1 - (3.0026 + 0.677*2)] = (-3.3566)$$

$$d_4 = [12 - (3.0026 + 0.677*10)] = (2.2274)$$

$$d_5 = [9 - (3.0026 + 0.677*11)] = (-1.4496)$$

$$d_6 = [4 - (3.0026 + 0.677*3)] = (-1.0336)$$

$$d_7 = [9 - (3.0026 + 0.677*6)] = (1.9354)$$

$$d_8 = [6 - (3.0026 + 0.677*5)] = (-0.3876)$$

$$d_9 = [1 - (3.0026 + 0.677*6)] = (-6.0646)$$

$$d_{10} = [14 - (3.0026 + 0.677*8)] = (5.5814)$$

$$\sum d^2 = (-4.4186)^2 + (6.9664)^2 + (-3.3566)^2 + (2.2274)^2 + (-1.4496)^2 + (-1.0336)^2 + (1.9354)^2 + (-0.3876)^2 + (-6.0646)^2 + (5.5814)^2$$

$$= 159.27990$$

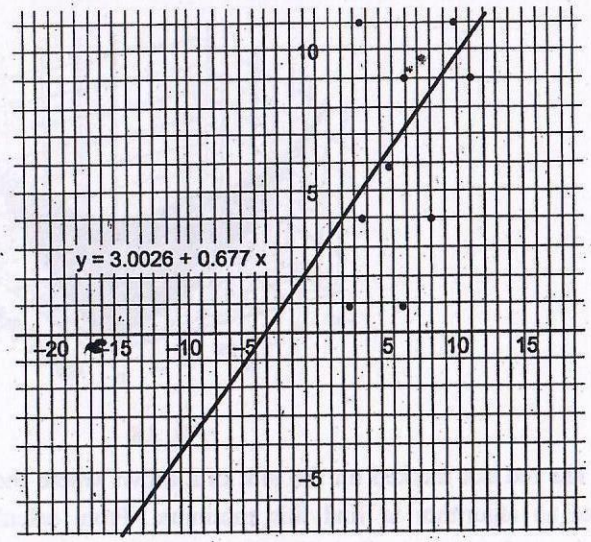


Fig. 3.6.

3.3. SINGLE VARIABLE REGRESSION

Single variable linear regression is one of the fundamental tools for an interpretation of data. Using linear regression, we can predict continuous variable outcomes given some data, if the data has a roughly linear shape, i.e. it generally ha the shape a line.

Visually, it appears that this data is approximated pretty well by a "line of be fit". This is certainly not the only way to approximate this data, but for now it's pretty good. Single variable linear regression is the tool to find this line of best fit. The lin of best fit can then be used to guess how many homicide deaths there would be fo ages we don't have data on.

By the end of this tutorial you can run linear regression on this homicide data, and in fact solve any single variable regression problem.

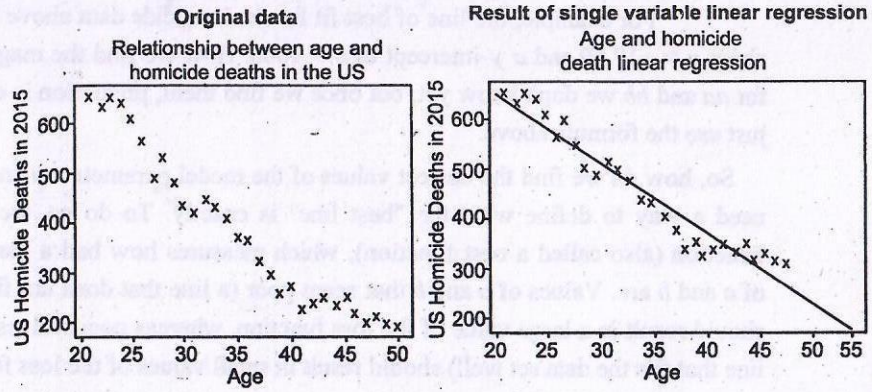


Fig. 3.7.

Data Set format

For regression problems, the goal is to predict a continuous variable output, given some input variables (also called **features**). For single variable regression, we only have one input variable, called x, and our *desired* output y. Our data set D then consists of many examples of x and y, so:

$$D = \{(x_1, y_1),(x_2, y_2),\ldots,(x_m, y_m)\}$$

where *mm* is the number of examples in the data set. For a concrete example, the homicide data set plotted above looks like:

$$D = \{(21, 652),(22, 633),\ldots,(50, 197)\}$$

We will write code to load data sets from files later.

Model Concept

So, how can we mathematically model single linear regression? Since the goal is to find the perfect line, let's start by defining the **model** (the mathematical description of how predictions will be created) as a line:

$$y'(x, a, b) = ax + b$$

where x is an input, y' is the prediction for the input x, and a and b are **model parameters**. Note that although this is an equation for a line with xx as the variable,

the values of a and b determine what specific line it is. To find the best line, we just need to find the best values for a (the slope) and b (the y-intercept).

For example, the line of best fit for the homicide data above has a slope of about $a \approx -17.69$ and a y-intercept of $b \approx 1000$. How we find the magic best values for aa and bb we don't know yet, but once we find them, prediction is easy, since we just use the formula above.

So, how do we find the correct values of the model parameters a and b? First, we need a way to define what the "best line" is exactly. To do so, we define a **loss function** (also called a cost function), which measures how bad a particular choice of a and b are. Values of a and b that seem poor (a line that does not fit the data set) should result in a large value of the loss function, whereas good values of a and b (a line that fits the data set well) should result in small values of the loss function.

In other words, the loss function should measure how far the predicted line is from each of the data points, and add this value up for all data points. We can write this as:

$$L(a, b) = \sum_{i=1}^{m} (y'(x_i, a, b) - y_i)^2$$

Recall that there are mm examples in the data set, x_i is the i^{th} input, and y_i is the i^{th} desired output. So, $(y'(x_i, a, b) - y_i)^2$ measures how far the i^{th} prediction is from the i^{th} desired output. For example, if the prediction y' is 7, and the correct output y is 10, then we would get $(7 - 10)^2 = 9$. Squaring it is important so that it is always positive.

Finally, we just add up all of these individual losses. Since the smallest possible values for the squared terms indicate that the line fits the data as closely as possible, the line of best fit (determined by the choice of a and b occurs exactly at the smallest value of $L(a, b)$. For this reason, the model is also called least squares regression.

Optimizing the Model

At this point, we have fully defined both our model:

$$y'(x, a, b) = ax + b$$

and our loss function, into which we can substitute the model:

$$L(a, b) = \sum_{i=1}^{m} (y'(x_i, a, b) - y_i)^2 = \sum_{i=1}^{m} (ax_i + b - y_i)^2$$

We crafted $L(a, b)$ so that it is smallest exactly when each predicted y' is as close as possible to actual data y. When this happens, since the distance between the data points and predicted line is as small as possible, using a and b produces the line of best fit. Therefore, our goal is to find the values of a and b that minimize the function $L(a, b)$. But what does L really look like? Well, it is essentially a 3D parabola which looks like:

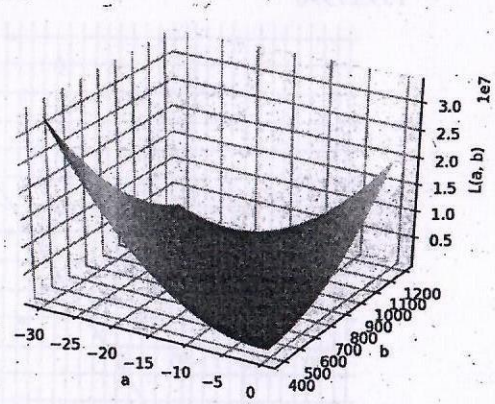


Fig. 3.8.

The red dot marked on the plot of L shows where the desired minimum is. We need an algorithm to find this minimum. From calculus, we know that at the minimum L must be entirely flat, that is the derivatives are both 0:

$$\frac{\partial L}{\partial a} = \sum_{i=1}^{m} 2(a x_i + b - y_i)\, x_i = 0$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} 2(a x_i + b - y_i) = 0$$

If you need to review this aspect of calculus, I would recommend Khan Academy videos. Now, for this problem it is possible to solve for a and b using the equations above, like we would in a typical calculus course. But for more advanced machine learning this is impossible, so instead we will learn to use an algorithm called gradient descent to find the minimum.

The idea is intuitive: place a ball at an arbitrary location on the surface of LL, and it will naturally roll downhill towards the flat valley of L and thus find the minimum. We know the direction of "downhill" at any location since we know the derivatives

of L: the derivatives are the direction of greatest upward slope (this is known as the gradient), so the opposite (negative) derivatives are the most downhill direction. Therefore, if the ball is currently at location (a, b), we can see where it would go by moving it to location (a', b') like so:

$$a' = a - \alpha \frac{\partial L}{\partial a}$$

$$b' = b - \alpha \frac{\partial L}{\partial b}$$

where αα is a constant called the **learning rate**, which we will talk about more later. If we repeat this process then the ball will continue to roll downhill into the minimum. An animation of this process looks like:

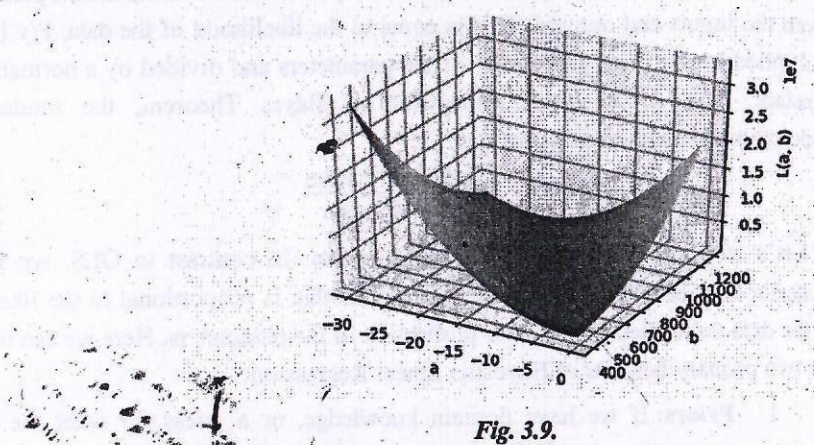


Fig. 3.9.

When we run the gradient descent algorithm for long enough, then it will find the optimal location for (a, b). Once we have the optimal values of a and b, then that's it, we can just use them to predict a rate of homicide deaths given any age, using the model:

$$y'(x) = ax + b$$

3.4. MULTIPLE VARIABLE REGRESSION

Multiple Variable regression is used to estimate the relationship between two or more independent variables and one dependent variable. You can use multiple Variable Regression when you want to know:

1. How strong the relationship is between two or more independent variables and one dependent variable (e.g. how rainfall, temperature, and amount of fertilizer added affect crop growth).
2. The value of the dependent variable at a certain value of the independent variables (e.g. the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

Multiple linear regression example

You are a public health researcher interested in social factors that influence heart disease. You survey 500 towns and gather data on the percentage of people in each town who smoke, the percentage of people in each town who bike to work, and the percentage of people in each town who have heart disease. Because you have two independent variables and one dependent variable, and all your variables are quantitative, you can use multiple linear regression to analyze the relationship between them.

Assumptions of multiple linear regression

Multiple linear regression makes all of the same assumptions as simple linear regression:

1. **Homogeneity of variance (homoscedasticity):** the size of the error in our prediction doesn't change significantly across the values of the independent variable.
2. **Independence of observations:** the observations in the dataset were collected using statistically valid sampling methods, and there are no hidden relationships among variables.

   In multiple linear regression, it is possible that some of the independent variables are actually correlated with one another, so it is important to check these before developing the regression model. If two independent variables are too highly correlated (r2 > ~ 0.6), then only one of them should be used in the regression model.
3. **Normality:** The data follows a normal distribution.
4. **Linearity:** the line of best fit through the data points is a straight line, rather than a curve or some sort of grouping factor.

Multiple linear regression formula

The formula for a multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \epsilon$$

- y = the predicted value of the dependent variable
- B_0 = the y-intercept (value of y when all other parameters are set to 0)
- $B_1 X_1$ = the regression coefficient (B_1) of the first independent variable (X_1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)
- ... = do the same for however many independent variables you are testing
- $B_n X_n$ = the regression coefficient of the last independent variable
- ϵ = model error (a.k.a. how much variation there is in our estimate of y)

To find the best-fit line for each independent variable, multiple linear regression calculates three things:

- The regression coefficients that lead to the smallest overall model error.
- The t statistic of the overall model.
- The associated p value (how likely it is that the t statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true).

It then calculates the t statistic and p value for each regression coefficient in the model.

3.5. BAYESIAN LINEAR REGRESSION

In the Bayesian viewpoint, we formulate linear regression using probability distributions rather than point estimates. The response, y, is not estimated as a single value, but is assumed to be drawn from a probability distribution. The model for Bayesian Linear Regression with the response sampled from a normal distribution is:

$$y \sim N(\beta^T X, \sigma^2 I)$$

The output, y is generated from a normal (Gaussian) Distribution characterized by a mean and variance. The mean for linear regression is the transpose of the weight matrix multiplied by the predictor matrix. The variance is the square of the standard

deviation σ (multiplied by the Identity matrix because this is a multi-dimensional formulation of the model).

The aim of Bayesian Linear Regression is not to find the single "best" value of the model parameters, but rather to determine the posterior distribution for the model parameters. Not only is the response generated from a probability distribution, but the model parameters are assumed to come from a distribution as well. The posterior probability of the model parameters is conditional upon the training inputs and outputs:

$$P(\beta | y, X) = \frac{P(y | \beta, X) * P(\beta | X)}{P(y | X)}$$

Here, $P(\beta | y, X)$ is the posterior probability distribution of the model parameters given the inputs and outputs. This is equal to the likelihood of the data, $P(y | \beta, X)$, multiplied by the prior probability of the parameters and divided by a normalization constant. This is a simple expression of Bayes Theorem, the fundamental underpinning of Bayesian Inference:

$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Normalization}}$$

Let's stop and think about what this means. In contrast to OLS, we have a posterior distribution for the model parameters that is proportional to the likelihood of the data multiplied by the prior probability of the parameters. Here we can observe the two primary benefits of Bayesian Linear Regression.

1. **Priors:** If we have domain knowledge, or a guess for what the model parameters should be, we can include them in our model, unlike in the frequentist approach which assumes everything there is to know about the parameters comes from the data. If we don't have any estimates ahead of time, we can use non-informative priors for the parameters such as a normal distribution.
2. **Posterior:** The result of performing Bayesian Linear Regression is a distribution of possible model parameters based on the data and the prior. This allows us to quantify our uncertainty about the model: if we have fewer data points, the posterior distribution will be more spread out.

As the amount of data points increases, the likelihood washes out the prior, and in the case of infinite data, the outputs for the parameters converge to the values

obtained from OLS. The formulation of model parameters as distributions encapsulates the Bayesian worldview: we start out with an initial estimate, our prior, and as we gather more evidence, our model becomes less wrong. Bayesian reasoning is a natural extension of our intuition. Often, we have an initial hypothesis, and as we collect data that either supports or disproves our ideas, we change our model of the world (ideally this is how we would reason).

3.5.1. IMPLEMENTING BAYESIAN LINEAR REGRESSION

In practice, evaluating the posterior distribution for the model parameters is intractable for continuous variables, so we use sampling methods to draw samples from the posterior in order to approximate the posterior. The technique of drawing random samples from a distribution to approximate the distribution is one application of Monte Carlo methods. There are a number of algorithms for Monte Carlo sampling, with the most common being variants of Markov Chain Monte Carlo (see this post for an application in Python).

3.5.2. BAYESIAN LINEAR MODELING APPLICATION

The basic procedure for implementing Bayesian Linear Regression is: specify priors for the model parameters (I used normal distributions in this example), creating a model mapping the training inputs to the training outputs, and then have Markov Chain Monte Carlo (MCMC) algorithm draw samples from the posterior distribution for the model parameters. The end result will be posterior distributions for the parameters. We can inspect these distributions to get a sense of what is occurring. The first plots show the approximations of the posterior distributions of model parameters. These are the result of 1000 steps of MCMC, meaning the algorithm drew 1000 steps from the posterior distribution.

If we compare the mean values for the slope and intercept to those obtained from OLS (the intercept from OLS was -21.83 and the slope was 7.17), we see that they are very similar. However, while we can use the mean as a single point estimate, we also have a range of possible values for the model parameters. As the number of data points increases, this range will shrink and converge one a single value representing greater confidence in the model parameters. (In Bayesian inference a range for a variable is called a credible interval and which has a slightly different interpretation from a confidence interval in frequentist inference).

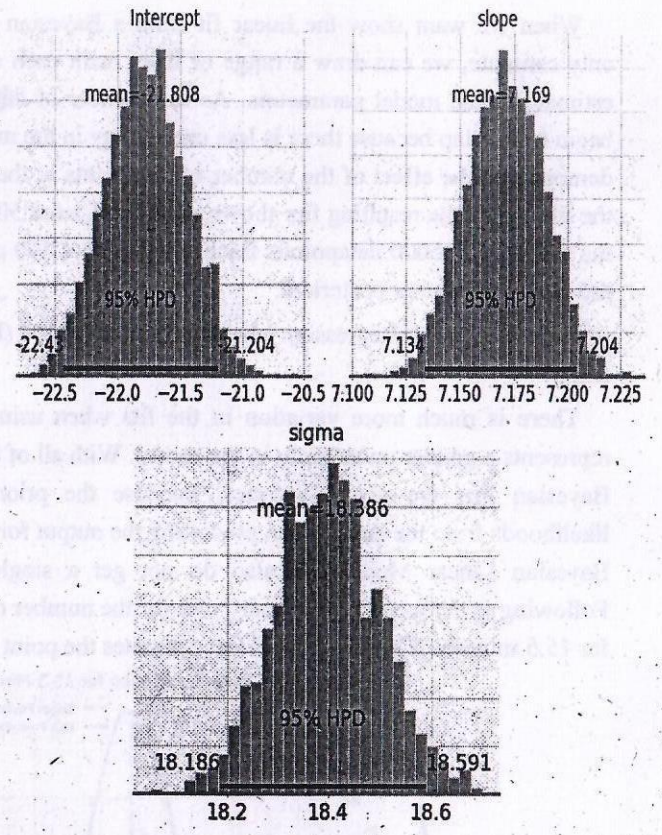
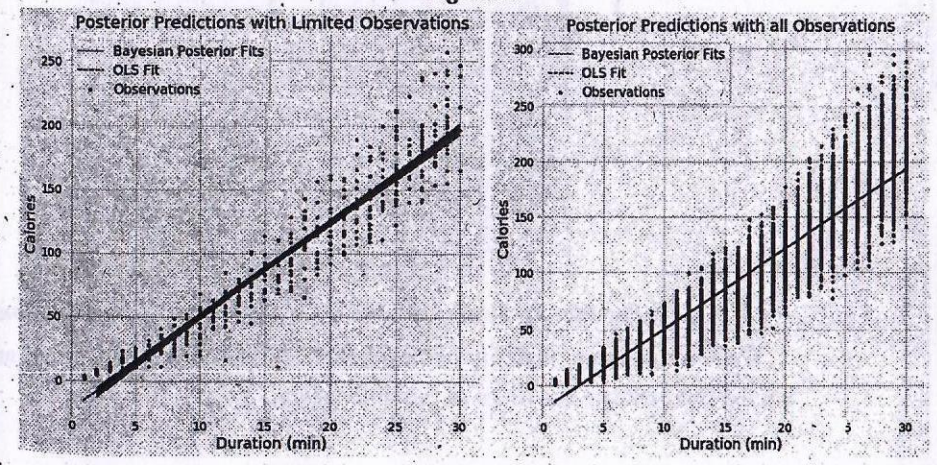


Fig. 3.10.



When we want show the linear fit from a Bayesian model, instead of showing only estimate, we can draw a range of lines, with each one representing a different estimate of the model parameters. As the number of datapoints increases, the lines begin to overlap because there is less uncertainty in the model parameters. In order to demonstrate the effect of the number of datapoints in the model, I used two models, the first, with the resulting fits shown on the left, used 500 datapoints and the one on the right used 15000 datapoints. Each graph shows 100 possible models drawn from the model parameter posteriors.

Bayesian Linear Regression Model Results with 500 (left) and 15000 observations (right)

There is much more variation in the fits when using fewer data points, which represents a greater uncertainty in the model. With all of the data points, the OLS and Bayesian Fits are nearly identical because the priors are washed out by the likelihoods from the data. When predicting the output for a single datapoint using our Bayesian Linear Model, we also do not get a single value but a distribution. Following is the probability density plot for the number of calories burned exercising for 15.5 minutes. The red vertical line indicates the point estimate from OLS.

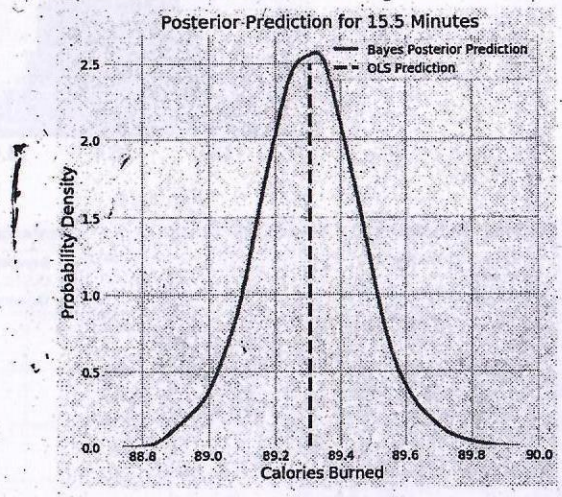


Fig. 3.11. Posterior Probability Density of Calories Burned from Bayesian Model

We see that the probability of the number of calories burned peaks around 89.3, but the full estimate is a range of possible values.

3.6. GRADIENT DESCENT

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks. In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function $f(x)$ parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters.

The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications. The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.

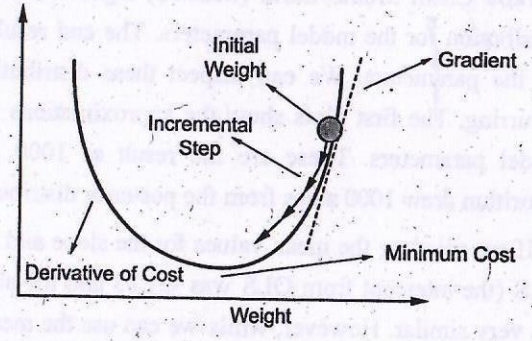


Fig. 3.12.

This entire procedure is known as Gradient Ascent, which is also known as steepest descent. ***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

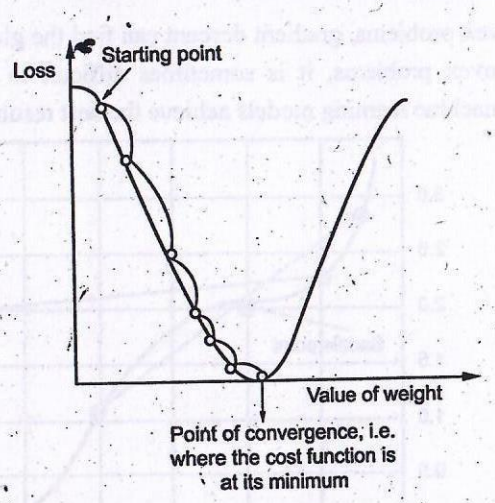
- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

How does Gradient Descent work?

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

$$Y = mX + c$$

Where '*m*' represents the slope of the line, and '*c*' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called a point of convergence.

The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required: **Direction & Learning Rate**

These two factors are used to determine the partial derivative calculation of future iteration and allow it to the point of convergence or local minimum or global minimum. Let's discuss learning rate factors in brief;

Learning Rate:

It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function. If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum. At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.

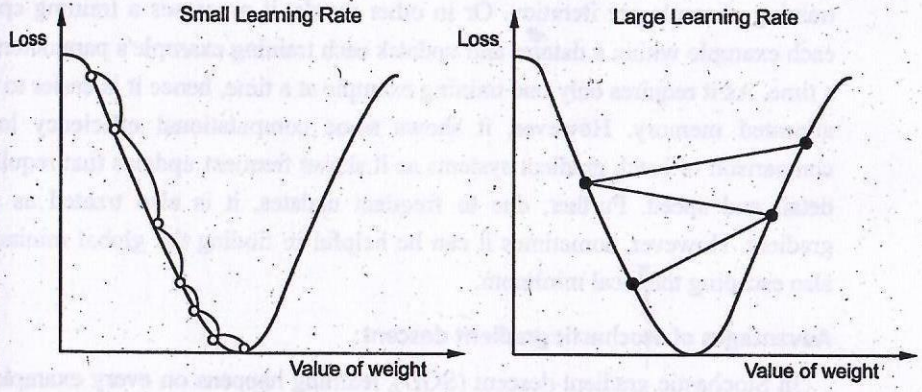


Fig. 3.13.

3.6.1. TYPES OF GRADIENT DESCENT

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into

- Batch gradient descent,
- Stochastic gradient descent, and
- Mini-batch gradient descent.

Let's understand these different types of gradient descent:

1. Batch Gradient Descent:

Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

Advantages of Batch gradient descent:

- ❖ It produces less noise in comparison to other gradient descent.
- ❖ It produces stable gradient descent convergence.
- ❖ It is Computationally efficient as all resources are used for all training samples.

2. Stochastic gradient descent

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed. Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

Advantages of Stochastic gradient descent:

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

- ❖ It is easier to allocate in desired memory.
- ❖ It is relatively fast to compute than batch gradient descent.
- ❖ It is more efficient for large datasets.

3. Mini Batch Gradient Descent:

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch

gradient descent and speed of stochastic gradient descent. Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

Advantages of Mini Batch Gradient Descent:

- ❖ It is easier to fit in allocated memory.
- ❖ It is computationally efficient.
- ❖ It produces stable gradient descent convergence.

3.6.2. CHALLENGES WITH THE GRADIENT DESCENT

Although we know Gradient Descent is one of the most popular methods for optimization problems, it still also has some challenges. There are a few challenges as follows:

1. Local Minima and Saddle Point:

For convex problems, gradient descent can find the global minimum easily, while for non-convex problems, it is sometimes difficult to find the global minimum, where the machine learning models achieve the best results.

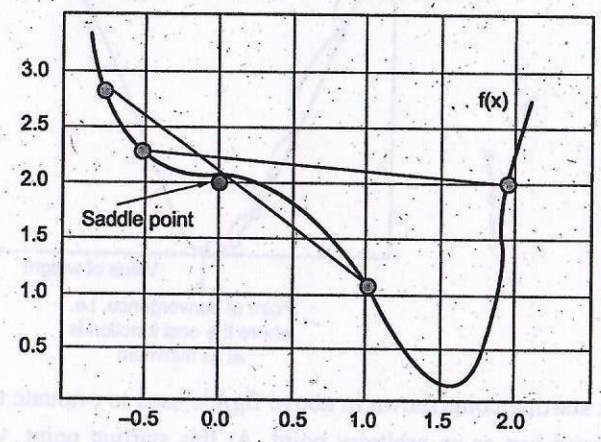


Fig. 3.14.

Whenever the slope of the cost function is at zero or just close to zero, this model stops learning further. Apart from the global minimum, there occur some scenarios that can show this slop, which is saddle point and local minimum. Local minima

generate the shape similar to the global minimum, where the slope of the cost function increases on both sides of the current points.

In contrast, with saddle points, the negative gradient only occurs on one side of the point, which reaches a local maximum on one side and a local minimum on the other side. The name of a saddle point is taken by that of a horse's saddle. The name of local minima is because the value of the loss function is minimum at that point in a local region. In contrast, the name of the global minima is given so because the value of the loss function is minimum there, globally across the entire domain the loss function.

2. Vanishing and Exploding Gradient

In a deep neural network, if the model is trained with gradient descent and backpropagation, there can occur two more issues other than local minima and saddle point.

Vanishing Gradients:

Vanishing Gradient occurs when the gradient is smaller than expected. During back propagation, this gradient becomes smaller that causing the decrease in the learning rate of earlier layers than the later layer of the network. Once this happens, the weight parameters update until they become insignificant.

Exploding Gradient:

Exploding gradient is just opposite to the vanishing gradient as it occurs when the Gradient is too large and creates a stable model. Further, in this scenario, model weight increases, and they will be represented as NaN. This problem can be solved using the dimensionality reduction technique, which helps to minimize complexity within the model.

3.7. LINEAR CLASSIFICATION

A linear classifier does classification decision based on the value of a linear combination of the characteristics. Imagine that the linear classifier will merge into it's weights all the characteristics that define a particular class. (Like merge all samples of the class cars together)

This type of classifier works better when the problem is linear separable.

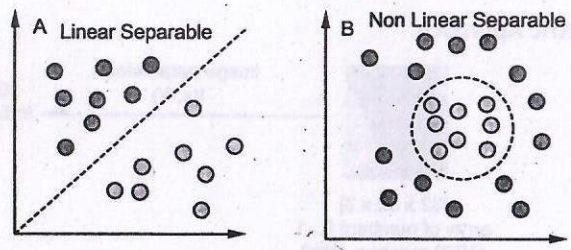


Fig. 3.15.

$$f(\vec{x}, \vec{W}, \vec{b}) = \sum j\,(W_j x_j) + b$$

x : input vector

W : Weight matrix

b : Bias vector

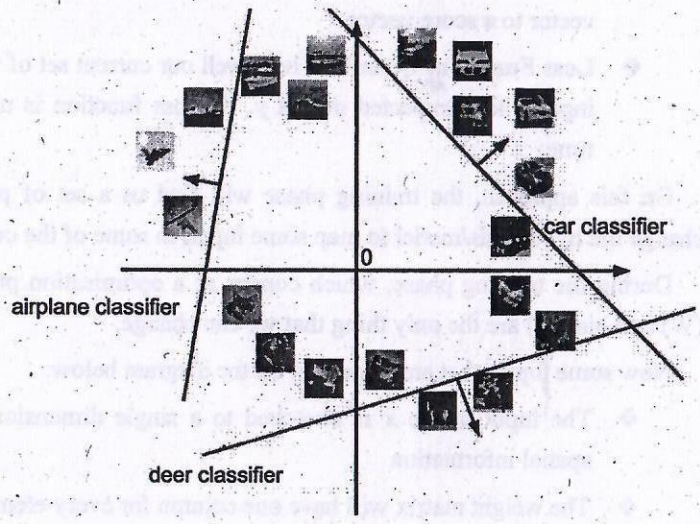


Fig. 3.16.

The weight matrix will have one row for every class that needs to be classified and one column for ever element (feature) of x. On the picture above each line will be represented by a row in our weight matrix.

Weight and Bias Effect

The effect of changing the weight will change the line angle, while changing the bias, will move the line left/right

Parametric Approach

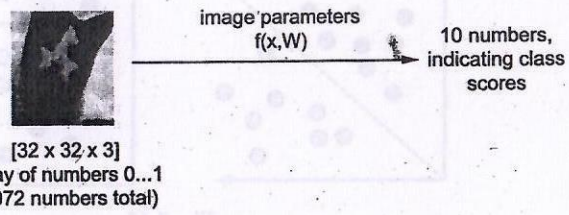


Fig. 3.17.

The idea is that out hypothesis/model has parameters, that will aid the mapping between the input vector to a specific class score. The parametric model has two important components:

- **Score Function:** Is a function $f(x, W, b)$ that will map our raw input vector to a score vector
- **Loss Function:** Quantifies how well our current set of weights maps some input x to a expected output y, the loss function is used during training time.

On this approach, the training phase will find us a set of parameters that will change the hypothesis/model to map some input, to some of the output class.

During the training phase, which consist as a optimisation problem, the weights (W) and bias (b) are the only thing that we can change.

Now some topics that are important on the diagram below:

- The input image x is stretched to a single dimension vector, this loose spatial information
- The weight matrix will have one column for every element on the input
- The weight matrix will have one row for every element of the output (on this case 3 labels)
- The bias will have one row for every element of the output (on this case 3 labels)
- The loss will receive the current scores and the expected output for its current input X

Consider each row of W a kind of pattern match for a specified class. The score for each class is calculated by doing a inner product between the input vector X and the specific row for that class.

Ex: $score_{cat} = [0.2(56) - 0.5(231) + 0.1(24) + 2(2)] + 1.1 = -96.8$

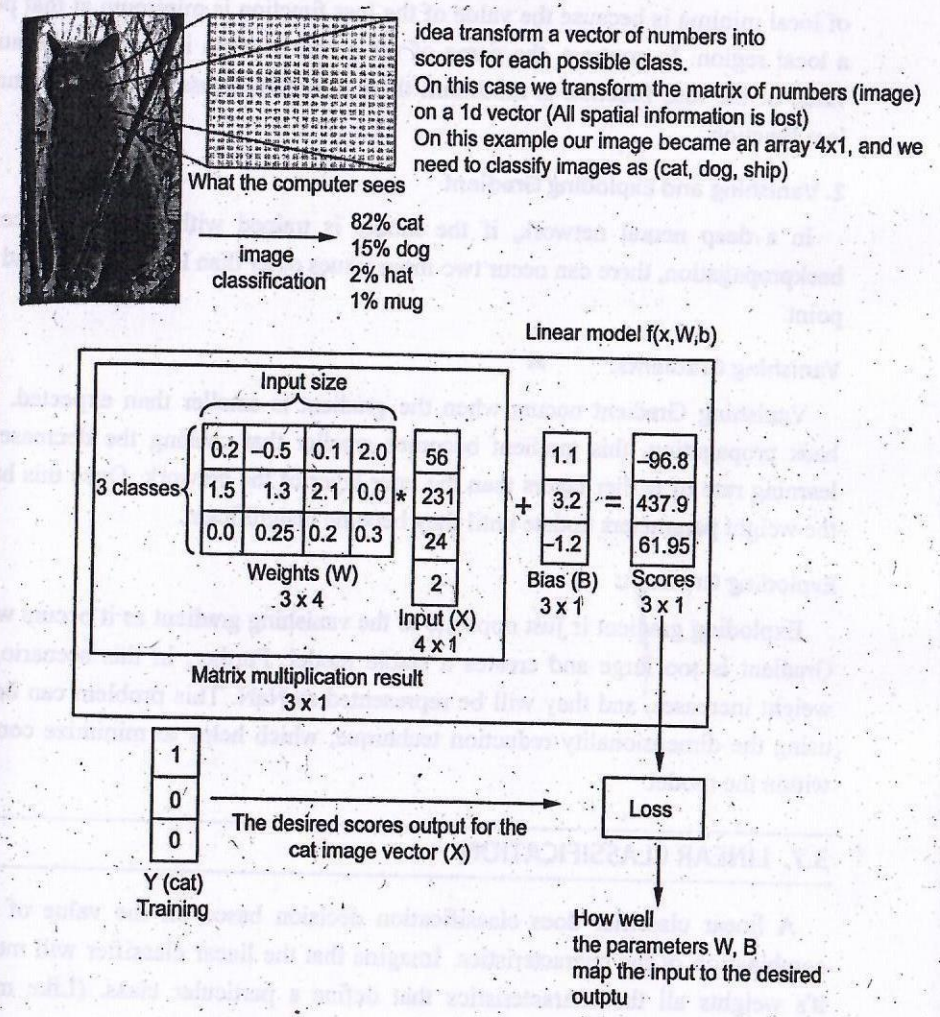


Fig. 3.18.

Bias trick

Some learning libraries implementations, does a trick to consider the bias as part of the weight matrix, the advantage of this approach is that we can solve the linear classification with a single matrix multiplication.

$$f(x, W) = W \cdot x$$

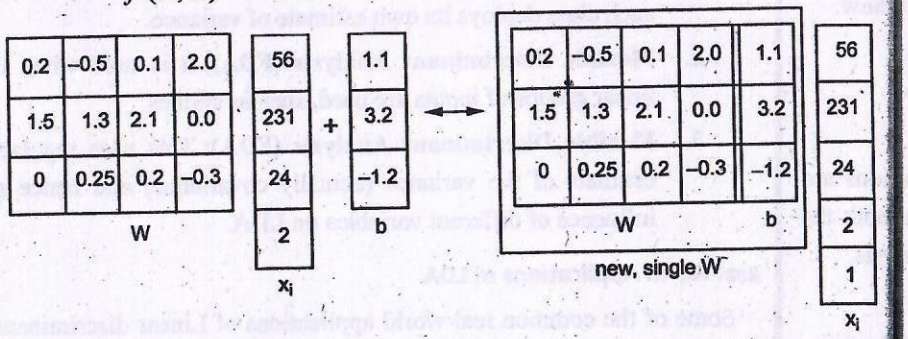


Fig. 3.19.

Basically you add an extra row at the end of the input vector, and concatenate a column on the W matrix

3.8. LINEAR DISCRIMINANT FUNCTION ANALYSIS

Linear Discriminant Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.

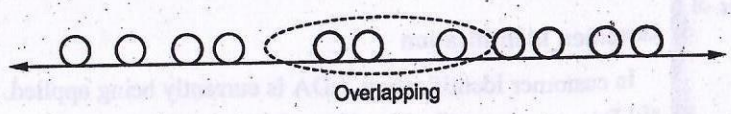


Fig. 3.20.

Example:

Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

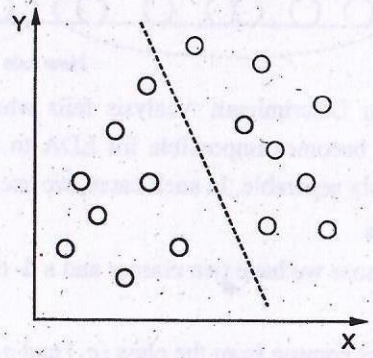


Fig. 3.21.

Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

1. Maximize the distance between means of the two classes.
2. Minimize the variation within each class

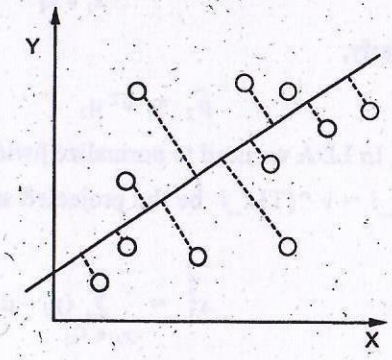
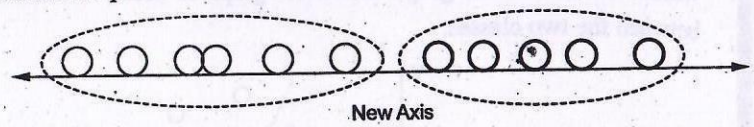


Fig. 3.22.

In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



But Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both the classes linearly separable. In such cases, we use non-linear discriminant analysis.

Mathematics

Let's suppose we have two classes and a d- dimensional samples such as $x_1, x_2 \ldots x_n$, where:

n_1 samples coming from the class (c_1) and n_2 coming from the class (c_2).

If x_i is the data point, then its projection on the line represented by unit vector v can be written as $v^T x_i$

Let's consider u_1 and u_2 be the means of samples class c_1 and c_2 respectively before projection and u_1 that denotes the mean of the samples of class after projection and it can be calculated by:

$$\tilde{\mu_1} = \frac{1}{n_1} \sum_{x_i \in c_1}^{n_1} v^T x_i = v^T \mu_1$$

Similarly,

$$\tilde{\mu_2} = v^T \mu_2$$

Now, In LDA we need to normalize |\widetilde{\mu_1} -\widetilde{\mu_2} |.

Let $y\_i = v\ ^{T}x\_i$ be the projected samples, then scatter for the samples of c_1 is:

$$\tilde{s_1^2} = \sum_{y_i \in c_1} (y_i - \mu_1)^2$$

3.8.1. EXTENSION TO LINEAR DISCRIMINANT ANALYSIS (LDA)

Linear Discriminant analysis is one of the most simple and effective methods to solve classification problems in machine learning. It has so many extensions and variations as follows:

1. **Quadratic Discriminant Analysis (QDA):** For multiple input variables, each class deploys its own estimate of variance.
2. **Flexible Discriminant Analysis (FDA):** it is used when there are non-linear groups of inputs are used, such as splines.
3. **Flexible Discriminant Analysis (FDA):** This uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

Real-world Applications of LDA

Some of the common real-world applications of Linear discriminant Analysis are given below:

Face_Recognition

Face recognition is the popular application of computer vision, where each face is represented as the combination of a number of pixel values. In this case, LDA is used to minimize the number of features to a manageable number before going through the classification process. It generates a new template in which each dimension consists of a linear combination of pixel values. If a linear combination is generated using Fisher's linear discriminant, then it is called Fisher's face.

Medical

In the medical field, LDA has a great application in classifying the patient disease on the basis of various parameters of patient health and the medical treatment which is going on. On such parameters, it classifies disease as mild, moderate, or severe. This classification helps the doctors in either increasing or decreasing the pace of the treatment.

Customer_Identification

In customer identification, LDA is currently being applied. It means with the help of LDA; we can easily identify and select the features that can specify the group of customers who are likely to purchase a specific product in a shopping mall. This can

be helpful when we want to identify a group of customers who mostly purchase a product in a shopping mall.

For_Predictions

LDA can also be used for making predictions and so in decision making. For example, "will you buy this product" will give a predicted result of either one or two possible classes as a buying or not.

In_Learning

Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

3.9. PERCEPTRON IN MACHINE LEARNING

Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers. This algorithm enables neurons to learn elements and processes them one by one during preparation.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e. input values, weights and Bias, net sum, and an activation function.

Binary classifier

Binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class. It can be considered as linear classifiers. In simple words, we can understand it as a classification algorithm that can predict linear predictor function in terms of weight and feature vectors.

Basic Components of Perception

The perception model as a binary classifier which contains three main components. These are as follows:

Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

Wight and Bias:

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

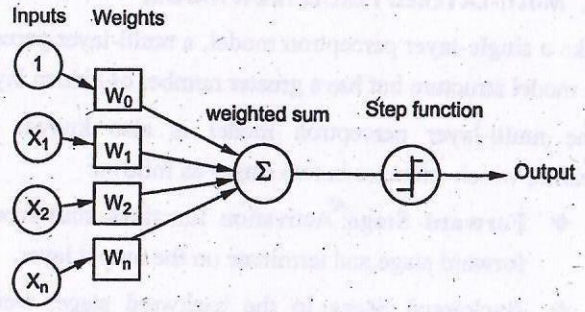


Fig. 3.23.

3.9.1. TYPES OF PERCEPTRON MODELS

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up

all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

"Single-layer perceptron can learn only linearly separable patterns."

3.9.2. MULTI-LAYERED PERCEPTRON MODEL

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

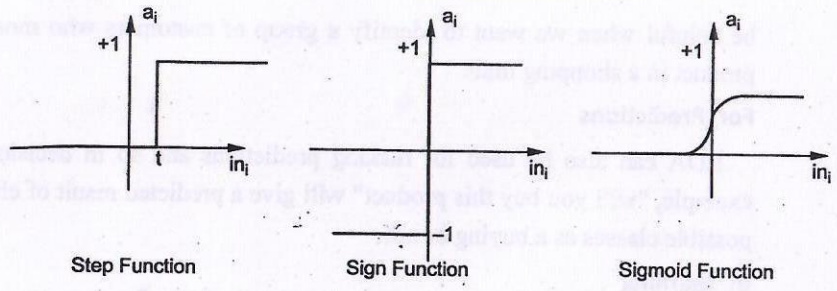


Fig. 3.24.

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

Perceptron Working Model

Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function f to obtain the desired output. This activation function is also known as the step function and is represented by f.

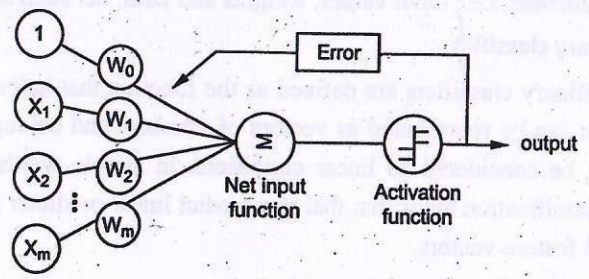


Fig. 3.25. Perceptron rule

This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (–1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + \ldots w_n * x_n$$

Add a special term called bias 'b' to this weighted sum to improve the model's performance.

$$\sum w_i * x_i + b$$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f(\sum w_i * x_i + b)$$

5. Generative and Discriminative Models

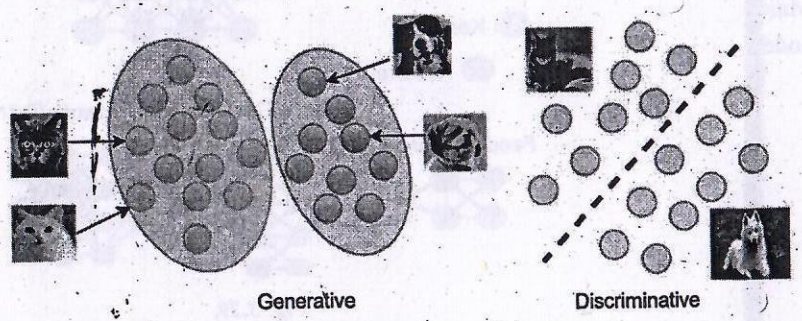


Fig. 3.26.

Most of the Machine Learning and Deep Learning problems that you solve are conceptualized from the Generative and Discriminative Models. In Machine Learning, one can clearly distinguish between the two modelling types:

- Classifying an image as a dog or a cat falls under Discriminative Modelling

- Producing a realistic dog or a cat image is a Generative Modelling problem

The more the neural networks got adopted, the more the generative and discriminative domains grew. To understand the algorithms based on these models, you need to study the theory and all the modeling concepts.

3.10. DISCRIMINATIVE MODELS

The four most common and interesting problems in the Image domain.

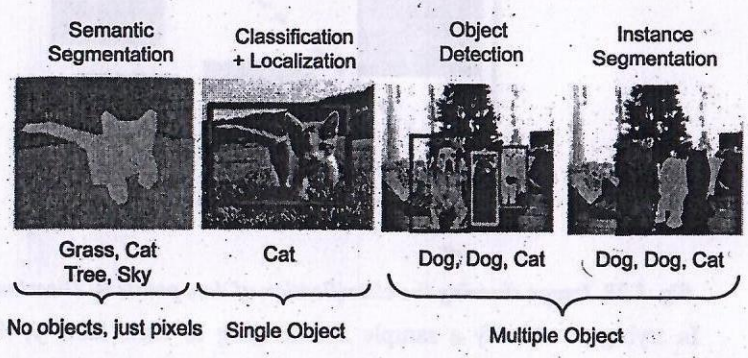


Fig. 3.27.

Many of you may have already used Discriminative Modelling to solve a classification problem in Machine Learning or Deep Learning. Being a superset of Machine Learning and Deep Learning algorithms, Discriminative Modelling is not limited to classification tasks. It is also widely used in object detection, semantic segmentation, panoptic segmentation, keypoint detection, regression problems, and language modelling.

The discriminative model falls under the **supervised learning** branch. In a classification task, given that the data is labelled, it tries to distinguish among classes, for example, a car, traffic light and a truck. Also known as classifiers, these models correspond image samples X to class labels Y, and discover the probability of image sample $x \in X$ belonging to class label $y \in Y$.

They learn to model the decision boundaries among classes (such as cats, dogs and tigers). The decision boundary could be linear or non-linear. The data points that are far away from the decision boundary (i.e. the outliers) are not very important. The discriminative model tries to learn a boundary that separates the positive from

the negative class, and comes up with the decision boundary. Only those closest to this boundary are considered.

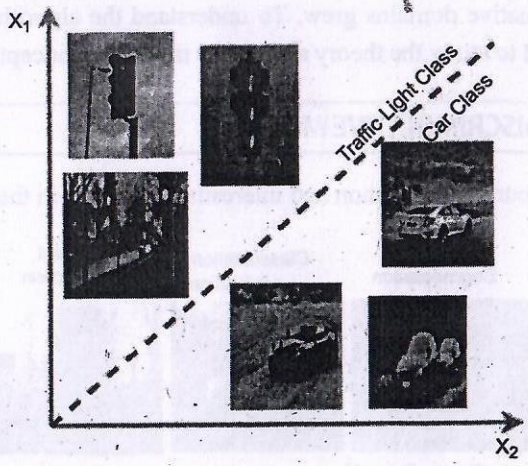


Fig. 3.28. Image showing the classification of data points by Discriminative models.

In trying to classify a sample x belonging to class label y, the discriminative model indirectly learns certain features of the dataset that make its task easier. For example, a car has four wheels of a circular shape and more length than width, while the traffic light is vertical with three circular rings. These features help the model distinguish between the two classes.

The discriminative models could be:

Probabilistic

- ❖ logistic regression
- ❖ a deep neural network, which models P(Y|X)

Non-probabilistic

- ❖ Support Vector Machine (SVM), which tries to learn the mappings directly from the data points to the classes with a hyperplane.

Discriminative modelling learns to model the conditional probability of class label y given set of features x as P(Y|X).

Some of the discriminative models are:

Support Vector Machine

- ❖ Logistic Regression
- ❖ k-Nearest Neighbour (kNN)
- ❖ Random Forest
- ❖ Deep Neural Network (such as AlexNet, VGGNet, and ResNet)

Let's study a few discriminative models and discover their prominent features.

Deep Neural Network

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships. The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification. We restrict ourselves to feed forward neural networks.

We have an input, an output, and a flow of sequential data in a deep network.

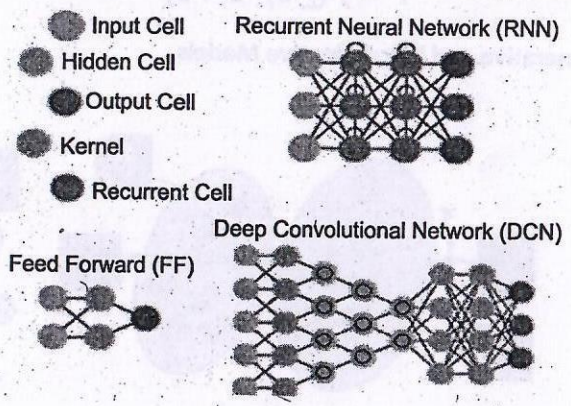
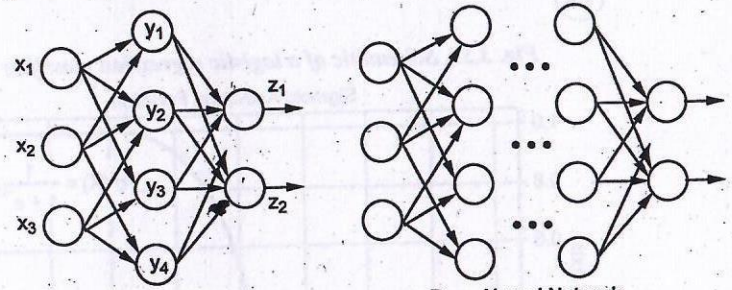


Fig. 3.29.

Neural networks are widely used in supervised learning and reinforcement learning problems. These networks are based on a set of layers connected to each other. In deep learning, the number of hidden layers, mostly non-linear, can be large; say about 1000 layers. DL models produce much better results than normal ML networks. We mostly use the gradient descent method for optimizing the network and minimising the loss function.

We can use the Imagenet, a repository of millions of digital images to classify a dataset into categories like cats and dogs. DL nets are increasingly used for dynamic images apart from static ones and for time series and text analysis. Training the data sets forms an important part of Deep Learning models. In addition, Backpropagation is the main algorithm in training DL models. DL deals with training large neural networks with complex input output transformations.

One example of DL is the mapping of a photo to the name of the person(s) in photo as they do on social networks and describing a picture with a phrase is another recent application of DL.



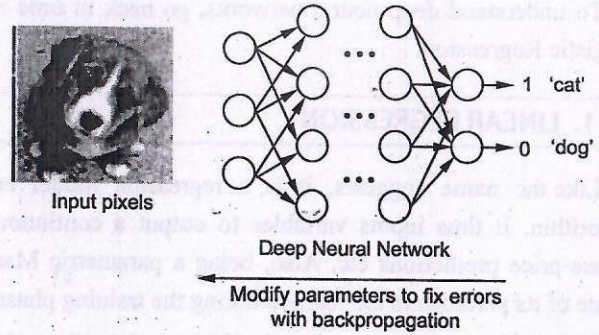
$$y_1 = \sigma(w_{1,1}x_1 + w_{1,2}x_2 + w_{1,3}w_3)$$

$$z_1 = \sigma(v_{1,1}y_1 + v_{1,3}x_3 + v_{1,3}y_3 + v_{1,4}y_4)$$

Neural networks are functions that have inputs like x_1 , x_2 , x_3... that are transformed to outputs like z_1 , z_2 , z_3 and so on in two (shallow networks) or several intermediate operations also called layers (deep networks).The weights and biases change from layer to layer. 'w' and 'v' are the weights or synapses of layers of the neural networks. The best use case of deep learning is the supervised learning problem. Here, we have large set of data inputs with a desired set of outputs.

Here we apply back propagation algorithm to get correct output prediction.

The most basic data set of deep learning is the MNIST, a dataset of handwritten digits. We can train deep a Convolutional Neural Network with Keras to classify images of handwritten digits from this dataset. The firing or activation of a neural net classifier produces a score. For example, to classify patients as sick and healthy, we consider parameters such as height, weight and body temperature, blood pressure etc. A high score means patient is sick and a low score means he is healthy.



Each node in output and hidden layers has its own classifiers. The input layer takes inputs and passes on its scores to the next hidden layer for further activation and this goes on till the output is reached. This progress from input to output from left to right in the forward direction is called forward propagation.

Credit assignment path (CAP) in a neural network is the series of transformations starting from the input to the output. CAPs elaborate probable causal connections between the input and the output.CAP depth for a given feed forward neural network or the CAP depth is the number of hidden layers plus one as the output layer is included. For recurrent neural networks, where a signal may propagate through a layer several times, the CAP depth can be potentially limitless.

Architecture of Deep Neural Network

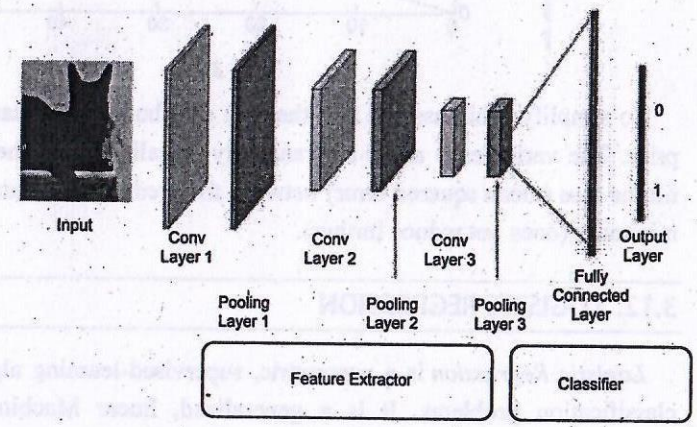


Fig. 3.30.

To understand deep neural networks, go back in time and first know Linear and Logistic Regression.

3.11. LINEAR REGRESSION

Like the name suggests, it is a regression model and a supervised learning algorithm. It thus inputs variables to output a continuous target like stock price, house-price predictions etc. Also, being a parametric Machine Learning algorithm, some of its parameters are learned during the training phase.

Linear Regression tries to find a linear decision boundary. The algorithm is quite simple and intuitive, and based on the equation of a line from two points: $y = mx + b$, where y is the dependent variable (predicted output y-axis), m is the slope of the line, x is the independent input variable (x-axis), and b is the y-intercept.

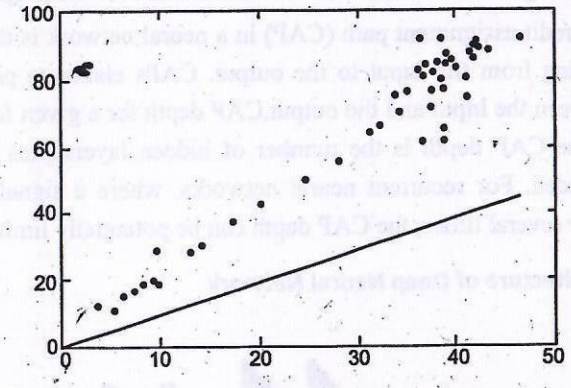


Fig. 3.31.

To simplify a bit, assume x is the area of a house (in square feet), while y is its price. The variables m and b are randomly initialized, and their values are updated till the loss (mean squared error) between the prediction ( ) and the true value (y) is minimum (does not reduce further).

3.12. LOGISTIC REGRESSION

Logistic Regression is a parametric, supervised-learning algorithm used to solve classification problems. It is a generalized, linear Machine Learning-algorithm because the outcome always depends on the sum of the inputs and parameters. Don't

get confused by its name though. Addition of a *Logistic Function* to Linear Regression built Logistic Regression. This was done to convert the target variable from continuous to deterministic, & bounded between 0 – 1.

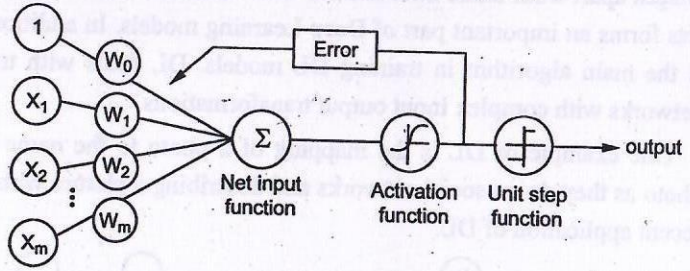


Fig. 3.32. Schematic of a logistic regression classifier

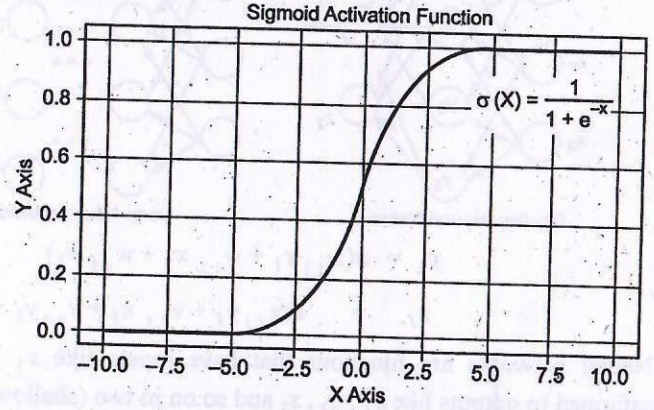


Fig. 3.33. Sigmoid Activation Function

A graph showing the curve fitting of Sigmoid Activation function

The above figure shows the Logistic Function, also commonly known as the *sigmoid activation function*, which takes a real-valued number (x) and squashes it into a range between 0 and 1. It converts large negative numbers to 0 and large positive numbers to 1.

The modified linear regression equation can be given as $y = \sigma(m\,x + b)$ where σ is the sigmoid activation function. Like Linear Regression, here too, the parameters are updated by calculating the loss between the predicted output and true label. A binary cross-entropy function is used as the loss function.

3.13. GENERATIVE MODELLING

Generative modelling defines how a dataset is generated. It tries to understand the distribution of data points, providing a model of how the data is actually generated in terms of a probabilistic model. (e.g., support vector machines or the perceptron algorithm gives a separating decision boundary, but no model of generating synthetic data points). The aim is to generate new samples from what has already been distributed in the training data.

Assume you have an autonomous driving dataset with an urban-scene setting. You now want to generate images from it that are semantically and spatially similar . This is a perfect example of a generative modelling problem. To do this, the generative model must understand the data's underlying structure and learn the realistic, generalized representation of the dataset, such as the sky is blue, buildings are usually tall, and pedestrians use sidewalks.

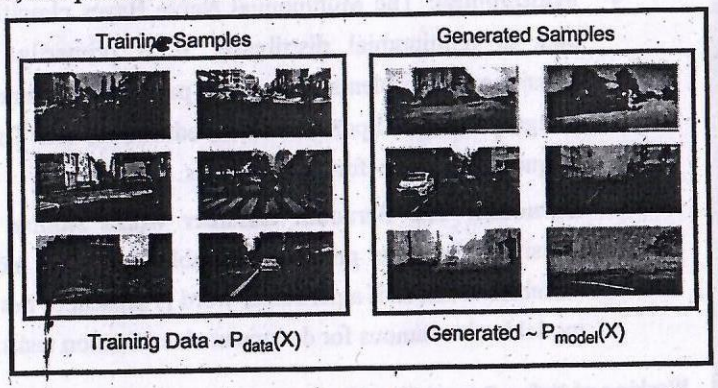


Fig. 3.34.

Probability of x sampled from training data or true distribution (Left). Probabili of x sampled from modelled distribution (Right).

The above image is an example of sample generation, using generative modellin The goal is to input training samples with probability distribution *Pdata,* and learn distribution such that: When you sample from the distribution *Pmodel,,* it generat realistic observations that represent the true distribution.

To generate such training samples, you need a training dataset, which consists unlabelled data points. Each data point has its own features, such as individual pi values (image-domain) and a set of vocabulary (text-domain). This whole process

generation is stochastic and influences the individual samples generated by the model.

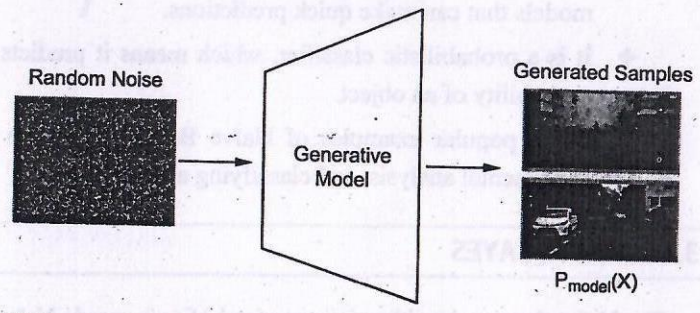


Fig. 3.35. Image visualizing the concept of 'noise adds randomness, so the images generated are diverse

Generative models, you know by now, try to learn the probabilistic distribution of the training data. This helps them represent the data more realistically. In the above figure, the generative model learns to generate urban-scene images, taking a random noise as a matrix or vector. Its task is to generate realistic samples X, with probability distribution similar to Pdata (original data from the training set). The noise adds randomness to the model and ensures that the images generated are diverse.

Types of Generative Models

- Naive Bayes
- Hidden Markov Models
- Autoencoder
- Boltzmann Machines
- Variational Autoencoder
- Generative Adversarial Networks

Naïve Bayes Classifier Algorithm

- 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

3.14. NAÏVE BAYES

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes:** It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian:** The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial:** The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli:** The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Working of Naïve Bayes' Classifier:

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	5

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:

$$P(Yes|Sunny) = P(Sunny|Yes)*P(Yes) / P(Sunny)$$

$$P(Sunny|Yes) = 3 / 10 = 0.3$$

$$P(Sunny) = 0.35$$

$$P(Yes) = 0.71$$

So $P(Yes|Sunny) = 0.3*0.71 / 0.35 = \mathbf{0.60}$

$$\mathbf{P(No|Sunny) = P(Sunny|No) * P(No)/P(Sunny)}$$

$$P(Sunny|NO) = 2 / 4 = 0.5$$

$$P(No) = 0.29$$

$$P(Sunny) = 0.35$$

So $P(No|Sunny) = 0.5*0.29 / 0.35 = \mathbf{0.41}$

So as we can see from the above calculation that **P(Yes|Sunny) > P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems.**

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring.**
- It is used in **medical data classification.**
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.

- It is used in Text classification such as **Spam filtering** and **Sentiment analysis.**

3.15. MAXIMUM MARGIN CLASSIFIER

The maximal margin classifier is the optimal hyperplane defined in the (rare) case where two classes are *linearly separable*. Given an $n \times p$ data matrix X with a binary response variable defined as $y \in [-1,1]]$ it *might* be possible to define a p-dimensional hyperplane.

$$h(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \cdots + \beta_p X_p = x\, T_i \beta + \beta_0 = 0$$

such that all observations of each class fall on opposite sides of the hyperplane. This *separating hyperplane* has the property that if β is constrained to be a unit vector, $\|\beta\| = \sum\beta^2 = 1$, then the product of the hyperplane and response variables are positive perpendicular distances from the hyperplane, the smallest of which may be termed the hyperplane *margin*, M,

$$y_i (x_i' \beta + \beta_0) \geq M$$

The maximal margin classifier is the hyperplane with the maximum margin, max{M} subject to $\|\beta\| = 1$. A separating hyperplane rarely exists. In fact, even if a separating hyperplane does exist, its resulting margin is probably undesirably narrow. Here is the maximal margin classifier.

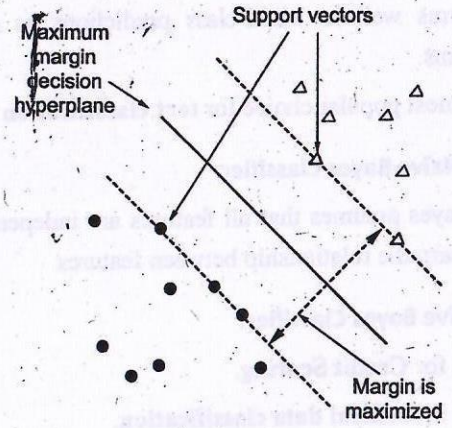


Fig. 3.36.

The data set has two linearly separable classes, $y \in [-1,1]$ described by two features, X_1 and X_2 . The code is unimportant - just trying to produce the visualization.

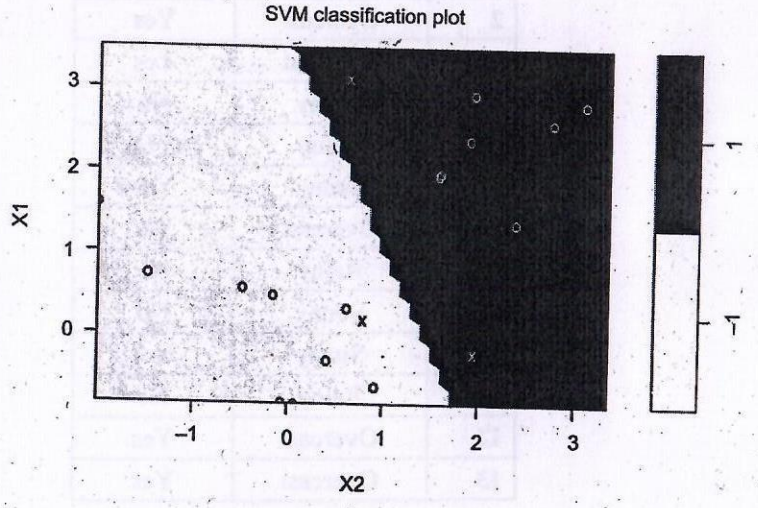


Fig. 3.37.

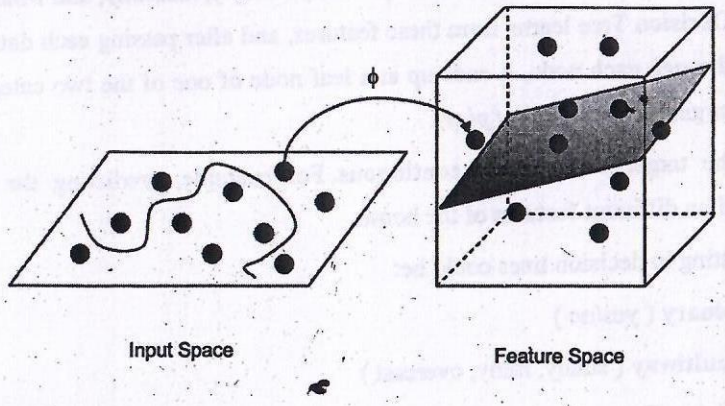
3.16. SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) is a non-parametric, supervised learning technique very popular with engineers for it produces excellent results with significantly less compute. A Machine Learning algorithm, it can be applied to both classification (output is deterministic) and regression (output is continuous) problems. It is largely used in text classification, image classification, protein and gene classification.

Image showing the Kernel Trick technique, the dimensional space, using which SVM can be easily implemented.

SVM can separate both linear and non-linear data points. A **kernel-trick** helps separate non-linear points. Having no kernel, the **Linear SVM** finds the hyperplane with the maximum margin-linear solution to the problem. The boundary points in the feature space are called support vectors (as shown in the figure above). Based on

their relative position, the maximum margin is derived and an optimal hyperplane drawn at the midpoint. The hyperplane is N – 1 dimensional, where N is the number of features present in the given dataset. For example, A line will indicate the decision boundary if a dataset has two features (2d input space).



Why do we need a hyperplane with maximum margin?

The decision boundary with the maximum margin works best, increases the chance of generalization. Enough freedom to the boundary points reduces the chance of misclassification. On the other hand, a decision boundary with smaller margins usually leads to overfitting.

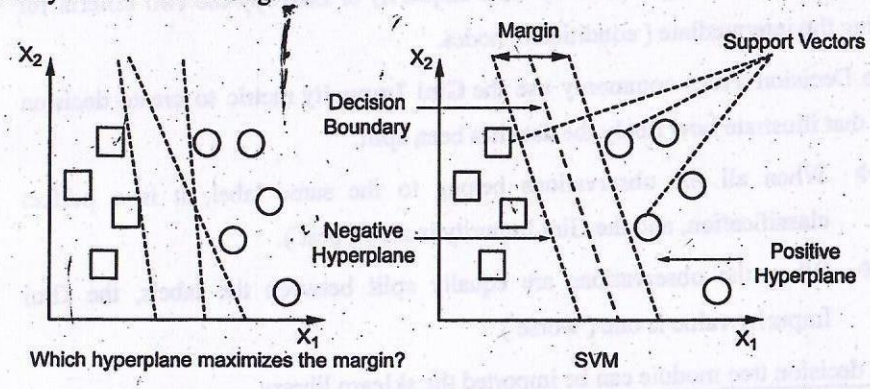


Fig. 3.38.

Image depicting the selection of the hyperplane that maximises the margin between the data points.

When the data points are not separated in a linear fashion, the Non-Linear SVM is used. A kernel function or a kernel trick helps obtain a new hyperplane for all the training data. As shown in the figure above, the input space is projected to a higher-dimensional feature space, such that the distribution of data points in the new hyperplane will be linear. Various kernel tricks, such as polynomial and radial basis function are used to solve nonlinear classification problems with SVM.

3.17. RANDOM FOREST

Like SVM, Random Forest also falls in the class of discriminative modelling. It is one of the most popular and powerful Machine Learning algorithms to perform classification and regression. Random Forest became a hit in the Kaggle Community as it helped win many competitions.

It is an ensemble of decision-tree models. Hence, to understand the Random Forest algorithm, you need to know Decision Trees. Microsoft trained a deep, randomized decision-forest classifier to predict 3D positions of body joints, from a single-depth image. There was no temporal information, and hundreds of thousands of training images were used.

What are Decision Trees?

A *Decision Tree* is a non-parametric, supervised-learning algorithm, used in both classification and regression problems. However, it is predominantly used for classification. The decision tree progressively splits the data into smaller groups, based on certain attributes, until they reach an end, where the data can be termed a label. Once it learns to model the data using labels, it tries to label the test set accordingly. It is a tree-structured classifier, consisting of a root node, an intermediate or decision node and a leaf node.

The data is represented in a tree structure, where each:

- ❖ internal node denotes a test on an attribute (basically a condition)
- ❖ branch represents an outcome of the test
- ❖ leaf node holds a class label

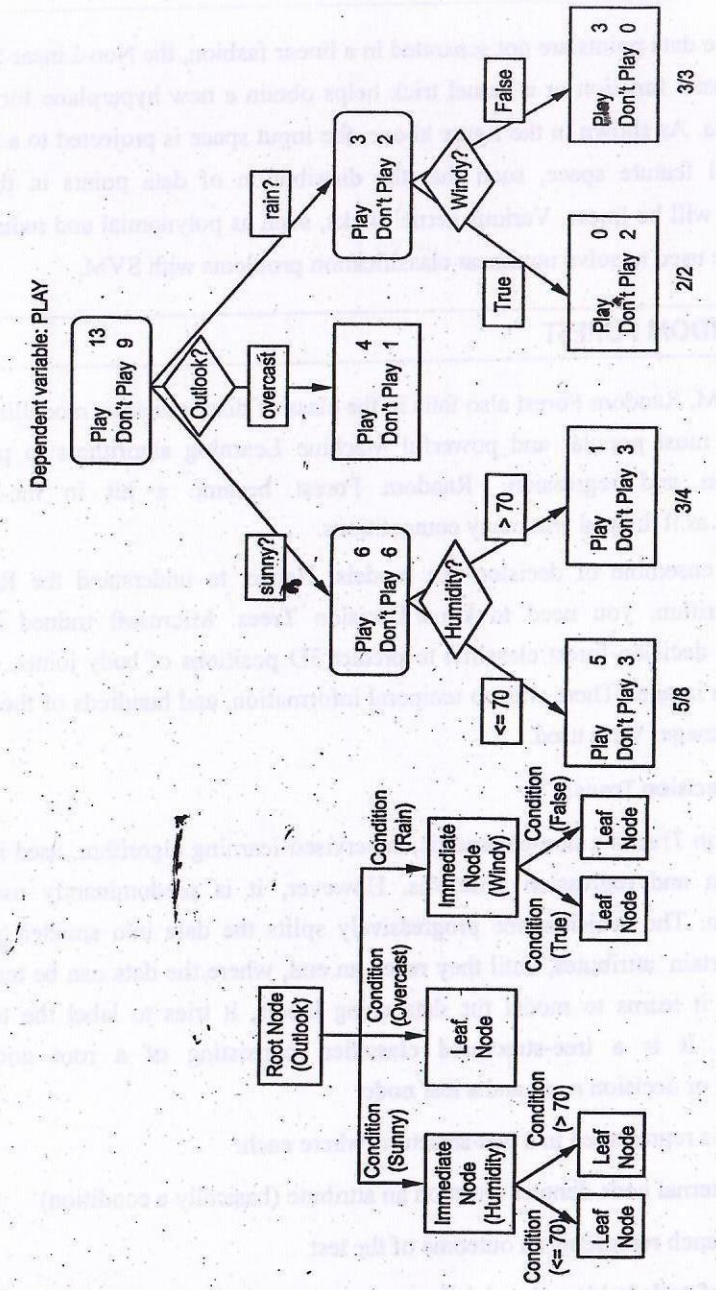


Fig. 3.39. Image showing the architecture and working of a decision tree.

Decision Trees can broadly be categorised into two types:

When the target variable is **categorical** (as shown in the figure above),

- It predicts one of the two categories: *play* and *do not play*.
- Features include: *outlook* (sunny, windy, rainy), *humidity*, and *windy*. The Decision Tree learns from these features, and after passing each data point through each node, it ends up at a leaf node of one of the two categorical targets (*play* or *do not play)*.

When the target variables are **continuous.** For example, predicting the house price, based on different features of the house.

The splitting in decision trees could be:

- **binary** (yes/no)
- **multiway** (sunny, rainy, overcast)

The Decision Trees analyse all the data features to find the ones that split the training data into subsets to give the best classification results. The training phase also determines which data attributes will be the root node, branches and intermediate. Data is split recursively till the tree reaches the leaf node. How the Decision Tree splits is governed by **Gini Impurity** or **Entropy**-the two criteria for selecting the intermediate (conditions) nodes.

The Decision Trees commonly use the **Gini Impurity** metric to create decision points that illustrate how finely the data has been split.

- When all the observations belong to the same label, it is a perfect classification, and the Gini Impurity is zero (best).
- When the observations are equally split between the labels, the Gini Impurity value is one (worse).

The decision tree module can be imported the sklearn library.

```
1  from sklearn.tree import DecisionTreeClassifier
```

Now apply all you learnt about the Decision Trees to know **Random Forest.**

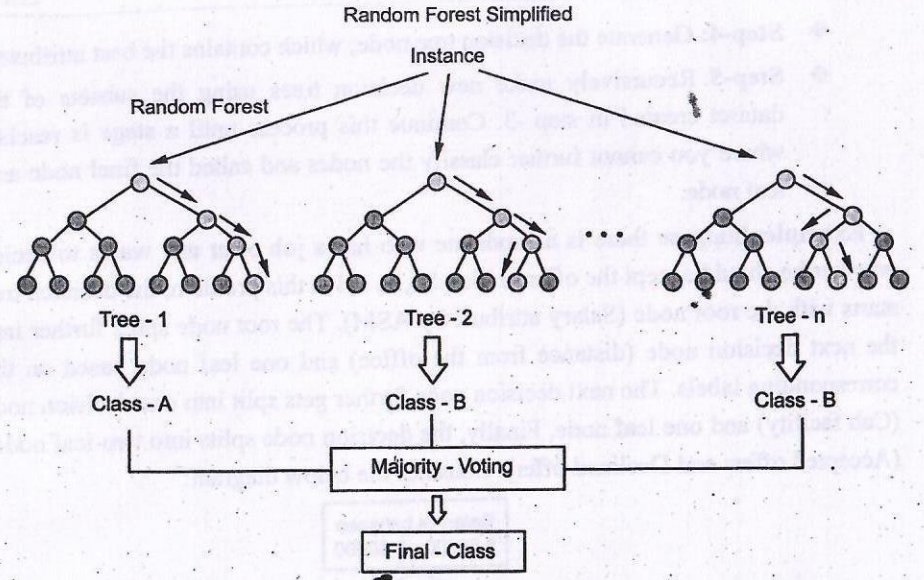


Fig. 3.40. Image showing the architecture of a Random Forest classifier.

You can build the Random Forest classifier by combining multiple Decision Trees (as shown in the figure above). This brings the weak classifiers together and you end up with a more robust classifier. The derived model is not just stronger, but also more accurate and generalizes better (improves variance).

Import the Random Forest algorithm directly from the sklearn library.

```
1 from sklearn.ensemble import RandomForestClassifier
```

Because *bootstrap* samples train the Random Forest algorithm, random samples or data points can be drawn from the dataset, with replacement.

To ensemble the Decision Trees, Random Forest uses the *bagging method*. This one differs a bit though from conventional bagging. Instead of using all the features in a Random Forest, draw its random subsets to train each tree. The random feature selection allows the trees more independence. Each tree can then capture unique information from the dataset, which in turn improves the model's accuracy and training time. After ensembling all the trees, it calculates the final output, using majority voting. The label with the maximum number of votes is termed the final prediction.

Decision Tree Classification Algorithm

- 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.

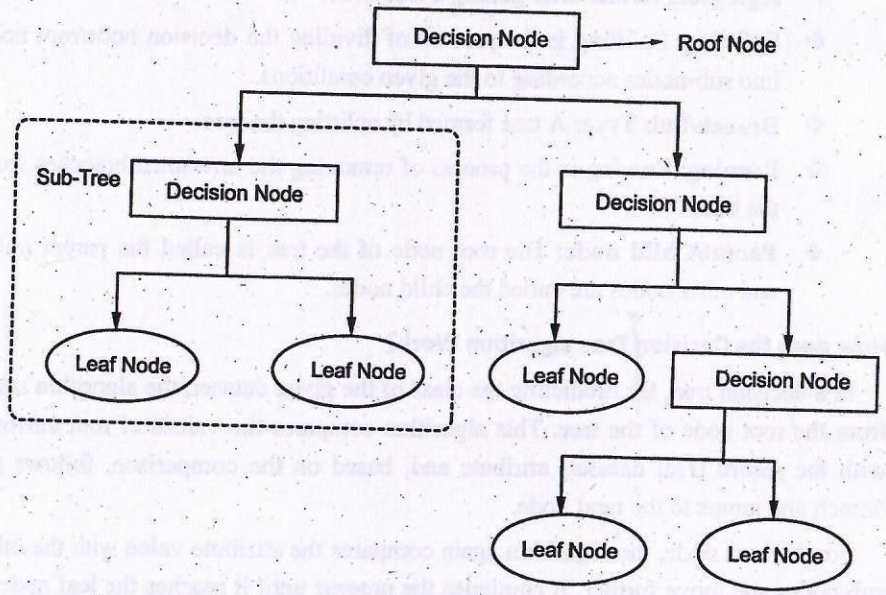


Fig. 3.41.

- 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.

- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Above diagram explains the general structure of a decision tree:

3.18. 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.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-2:** Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.

- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

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:

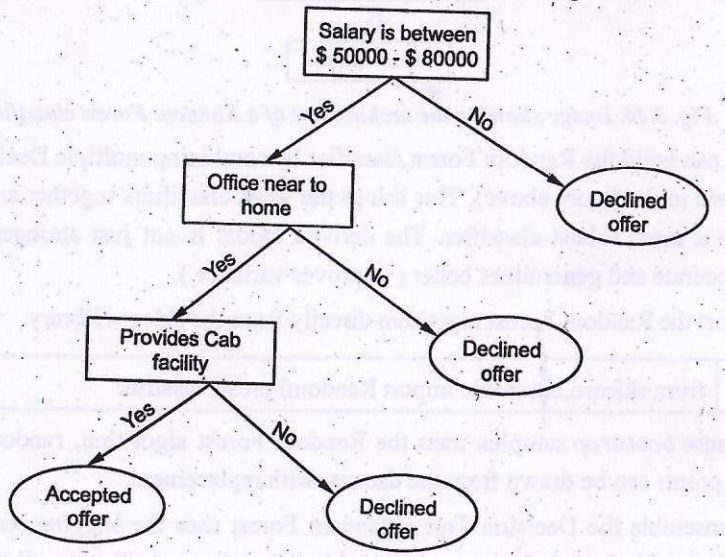


Fig. 3.42.

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. There are two popular techniques for ASM, which are:

- Information Gain
- Gini Index

1. 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)

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

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum j P_j^2$$

TWO MARKS QUESTIONS & ANSWERS (PART - A)

1. ***Define Regression algorithms?***

Regression algorithms are used when you notice that the output is a continuous variable, whereas classification algorithms are used when the output is divided into sections such as Pass/Fail, Good/Average/Bad, etc. We have various algorithms for performing the regression or classification actions, with Linear Regression Algorithm being the basic algorithm in Regression.

2. ***What are the two types of Linear Regression?***

- Simple Linear Regression
- Multiple Linear Regression

3. ***Write simple linear regression?***

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- Y represents the output or dependent variable.
- β_0 and β_1 are two unknown constants that represent the intercept and coefficient (slope) respectively.
- ϵ (Epsilon) is the error term.

4. ***Write Applications of Simple Linear Regression?***

- **Predicting crop yields based on the amount of rainfall:** Yield is dependent variable while the amount of rainfall is independent variable.
- **Marks scored by student based on number of hours studied (ideally) :** Here marks scored is dependent and number of hours studied is independent.
- **Predicting the Salary of a person based on years of experience :** Thus Experience become the independent variable while Salary becomes the dependent variable

5. ***What is mean by Homogeneity of variance?***

The size of the error in our prediction doesn't change significantly across the values of the independent variable.

6. ***Define Linearity?***

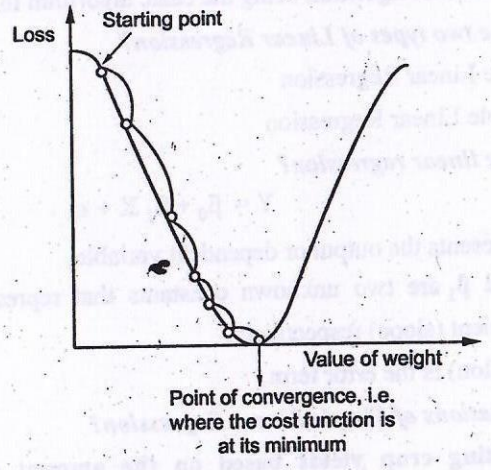
The line of best fit through the data points is a straight line, rather than a curve or some sort of grouping factor.

7. *How does Gradient Descent work?*

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

$$Y = mX + c$$

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

8. *Write the Advantages of Batch gradient descent?*

- ❖ It produces less noise in comparison to other gradient descent.
- ❖ It produces stable gradient descent convergence.
- ❖ It is computationally efficient as all resources are used for all training samples.

9. *What is SGD?*

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory

10. *Write about Mini Batch Gradient Descent?*

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent. Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

11. *What is QDA?*

Quadratic Discriminate Analysis (QDA) For multiple input variables, each class deploys its own estimate of variance.

12. *What is FDA?*

FDA uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

13. *What is Binary classifier?*

Binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class. It can be considered as linear classifiers. In simple words, we can understand it as a classification algorithm that can predict linear predictor function in terms of weight and feature vectors.

14. *Some of the discriminative models?*

- ❖ Support Vector Machine
- ❖ Logistic Regression
- ❖ k-Nearest Neighbour (kNN)
- ❖ Random Forest
- ❖ Deep Neural Network (such as AlexNet, VGGNet, and ResNet)

15. *Type of Generative Models?*

- ❖ Naive Bayes

- Hidden Markov Models
- Autoencoder
- Boltzmann Machines
- Variational Autoencoder
- Generative Adversarial Networks

16. *Write applications of Naïve Bayes Classifier?*

- It is used for **Credit Scoring.**
- It is used in **medical data classification.**
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis.**

17. *Write about SVM?*

Support Vector Machine (SVM) is a non-parametric, supervised learning technique very popular with engineers for it produces excellent results with significantly less compute. A Machine Learning algorithm, it can be applied to both classification (output is deterministic) and regression (output is continuous) problems. It is largely used in text classification, image classification, protein and gene classification.

18. *Define Random Forest?*

Like SVM, Random Forest also falls in the class of discriminative modelling. It is one of the most popular and powerful Machine Learning algorithms to perform classification and regression. Random Forest became a hit in the *Kaggle Community* as it helped win many competitions.

PART - B & C

1. Explain the types of Linear Regression?
2. Write in detail about Least Square Method?
3. Write in detail about Bayesian Linear Regression?

4. What are the Types of Gradient Descent?
5. Explain about Linear Classification?
6. Discuss about Real-world Applications of LDA?
7. Write Briefly about Perceptron working Model?
8. Compare Generative and Discriminative Models?
9. What is Deep Neural Network?
10. Compare Linear Regression and Logistic Regression?
11. Write in detail about Generative Modelling?
12. Explain in detail about Naïve Bayes Classifier Algorithm?
13. Write in detail about Bayes' Theorem?
14. Explain in detail about Maximum Margin Classifier?
15. Write in detail about Decision tree?

****

UNIT IV

ENSEMBLE TECHNIQUES AND UNSUPERVISED LEARNING

Combining multiple learners: Model combination schemes, Voting, Ensemble Learning - bagging, boosting, stacking, Unsupervised learning: K-means, Instance Based Learning: KNN, Gaussian mixture models and Expectation maximization

4.1. COMBINING MULTIPLE LEARNERS

Model composed of multiple learners that complement each other which are combined to attain higher accuracy .Individual algorithms are called base learners.

- Each base learner should be simple & reasonably accurate.
- Together producing the required accuracy.

Ensemble Technique

It is a machine learning technique that combines several base models in order to produce one optimal predictive model.

Decision Trees is best to outline the definition and practicality of Ensemble Methods (however it is important to note that Ensemble Methods do not only pertain to Decision Trees).

A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account, and given each factor either makes a decision or asks another question. In this example, every time it is overcast. However, if it is raining, it is needed to ask if it is windy or not? If windy, do not play. But given no wind, so ready to go outside to play.

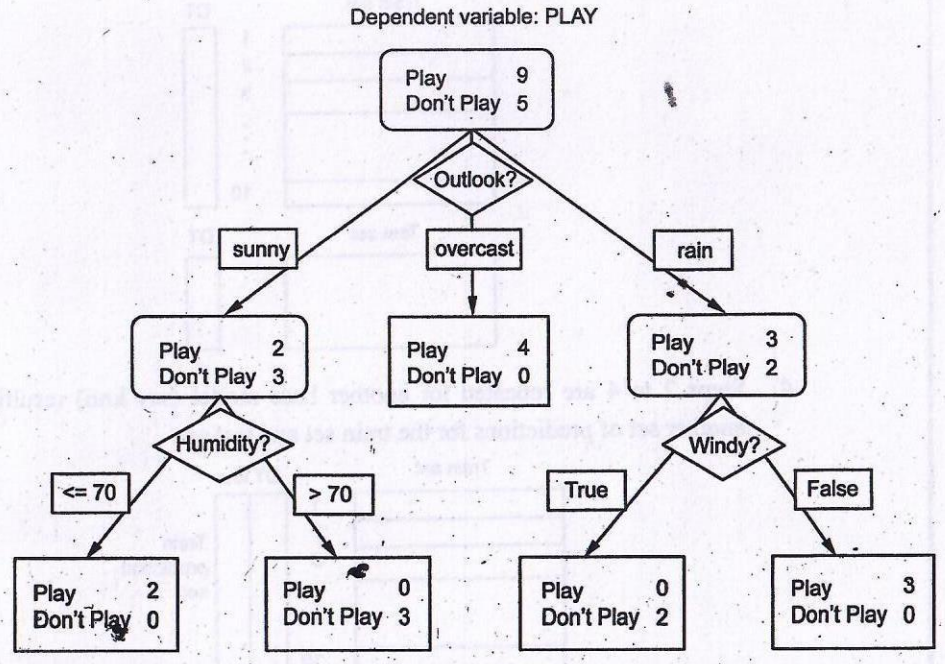


Fig. 4.1.

4.1.1. SIMPLE ENSEMBLE TECHNIQUES

A few simple but powerful techniques, namely:

1. Max Voting
2. Averaging
3. Weighted Averaging

1. Max Voting

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a 'vote'. The predictions which we get from the majority of the models are used as the final prediction.

For example, when you asked 5 of your colleagues to rate your movie (out of 5); we'll assume three of them rated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken as 4. **You can consider this as taking the mode of all the predictions.**

CSE 0811

The result of max voting would be something like this:

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4

2. Averaging

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

For example, in the below case, the averaging method would take the average of all the values.

i.e. $(5+4+5+4+4)/5 = 4.4$

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4.4

3. Weighted Average

This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.

The result is calculated as $[(5 \times 0.23) + (4 \times 0.23) + (5 \times 0.18) + (4 \times 0.18) + (4 \times 0.18)] = 4.41$.

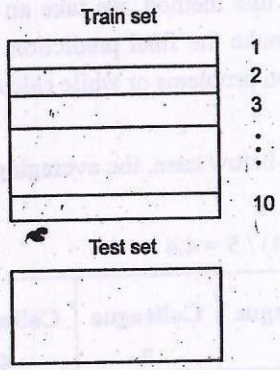
	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
Weight	0.23	0.23	0.18	0.18	0.18	
Rating	5	4	5	4	4	4.41

4.2. ADVANCED ENSEMBLE TECHNIQUES

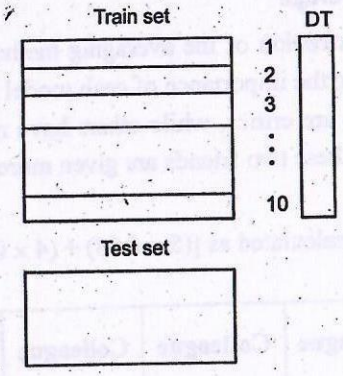
4.2.1. STACKING

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set. Below is a step-wise explanation for a simple stacked ensemble:

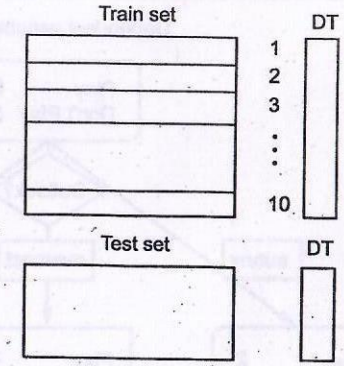
1. The train set is split into 10 parts.



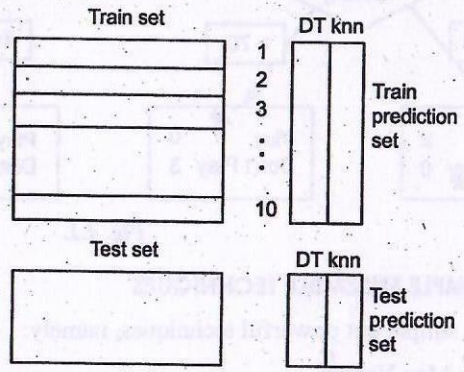
2. A base model (suppose a decision tree) is fitted on 9 parts and predictions are made for the 10th part. This is done for each part of the train set.



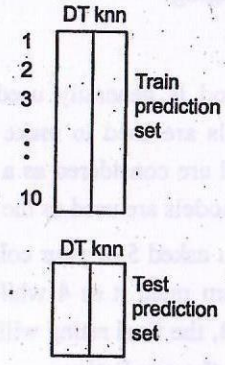
3. The base model (in this case, decision tree) is then fitted on the whole train dataset. Using this model, predictions are made on the test set.



4. Steps 2 to 4 are repeated for another base model (say knn) resulting in another set of predictions for the train set and test set.



5. The predictions from the train set are used as features to build a new model.

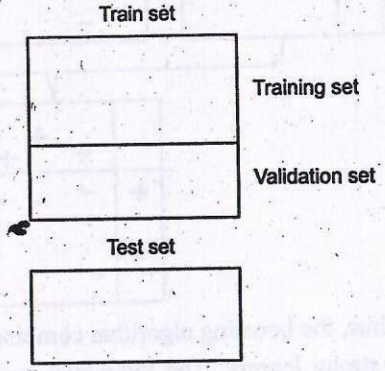


6. This model is used to make final predictions on the test prediction set.

4.2.2. BLENDING

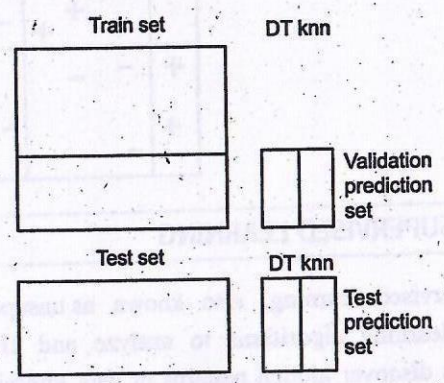
Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions. In other words, unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model which is run on the test set. Here is a detailed explanation of the blending process:

1. The train set is split into training and validation sets.



Model(s) are fitted on the training set.

The predictions are made on the validation set and the test set.



The validation set and its predictions are used as features to build a new model.

This model is used to make final predictions on the test and meta-features.

4.2.3. BAGGING

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here's a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem?

One of the techniques is bootstrapping.

Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

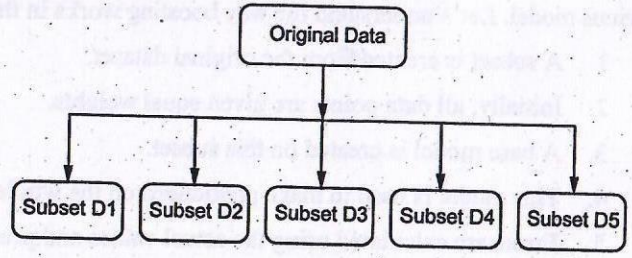


Fig. 4.2.

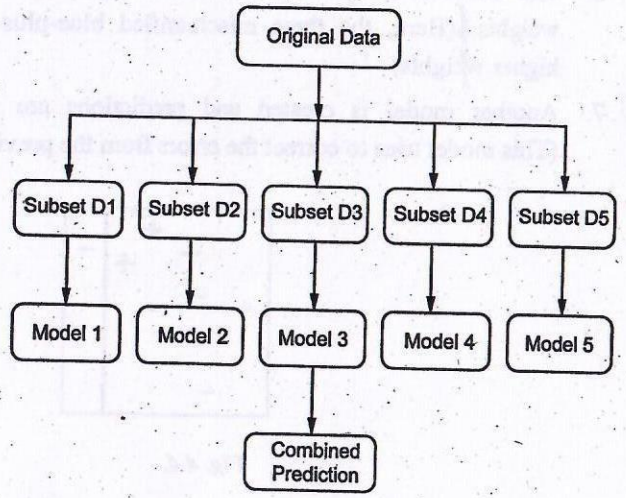


Fig. 4.3.

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

4.2.4. BOOSTING

Before we go further, here's another question for you: If a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let's understand the way boosting works in the below steps.

1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole dataset.
5. Errors are calculated using the actual values and predicted values.
6. The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
7. Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)

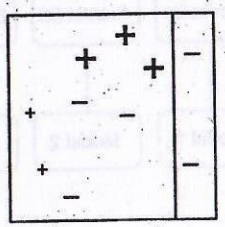
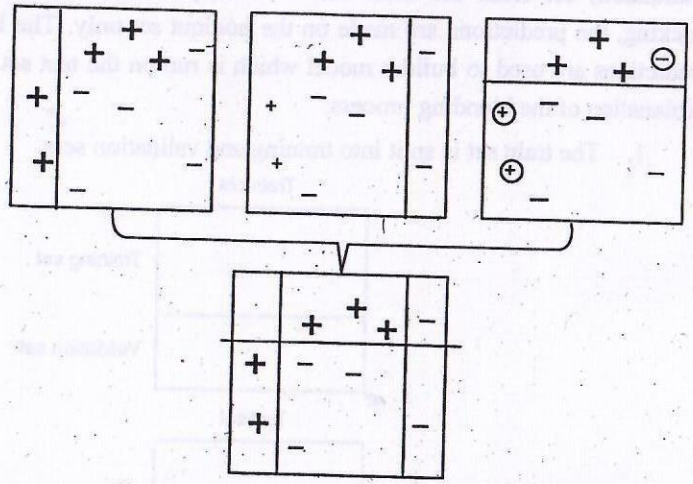
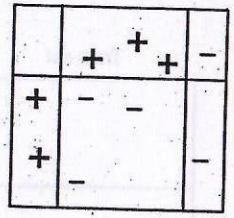


Fig. 4.4.

8. Similarly, multiple models are created, each correcting the errors of the previous model.
9. The final model (strong learner) is the weighted mean of all the models (weak learners).



10. Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.



4.3. UNSUPERVISED LEARNING

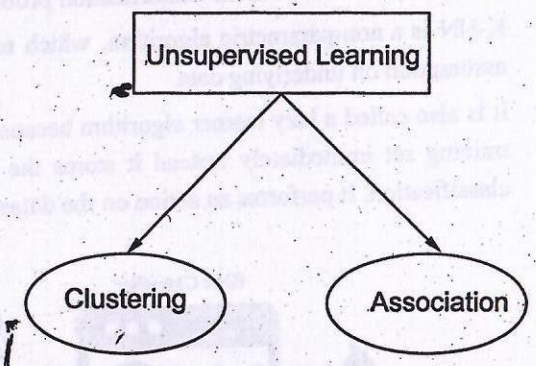
Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it

the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

Otherwise, Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

4.3.1. TYPES OF UNSUPERVISED LEARNING ALGORITHM



- **Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association:** An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

4.3.2. UNSUPERVISED LEARNING ALGORITHMS

Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

4.3.3. K-MEANS CLUSTERING

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

The term 'K' is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

For a better understanding of k-means, let's take an example from cricket. Imagine you received data on a lot of cricket players from all over the world, which gives information on the runs scored by the player and the wickets taken by them in the last ten matches. Based on this information, we need to group the data into two clusters, namely batsman and bowlers.

Steps to create these clusters

Solution:

Assign data points

Here, we have our data set plotted on 'x' and 'y' coordinates. The information on the y-axis is about the runs scored, and on the x-axis about the wickets taken by the players.

If we plot the data, this is how it would look:

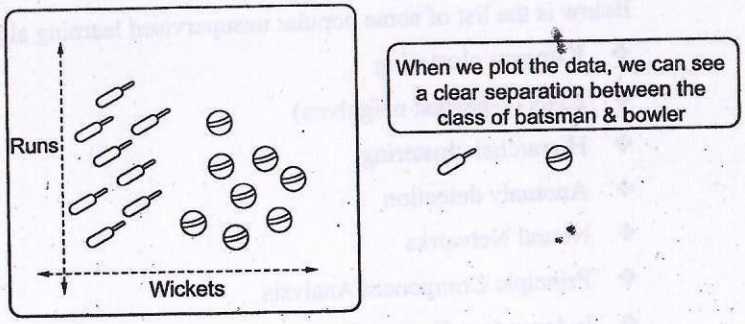


Fig. 4.5.

4.3.4. APPLICATIONS OF K-MEANS CLUSTERING

K-Means clustering is used in a variety of examples or business cases in real life, like:

- Academic performance
- Diagnostic systems
- Search engines
- Wireless sensor networks

The flowchart below shows how k-means clustering works:

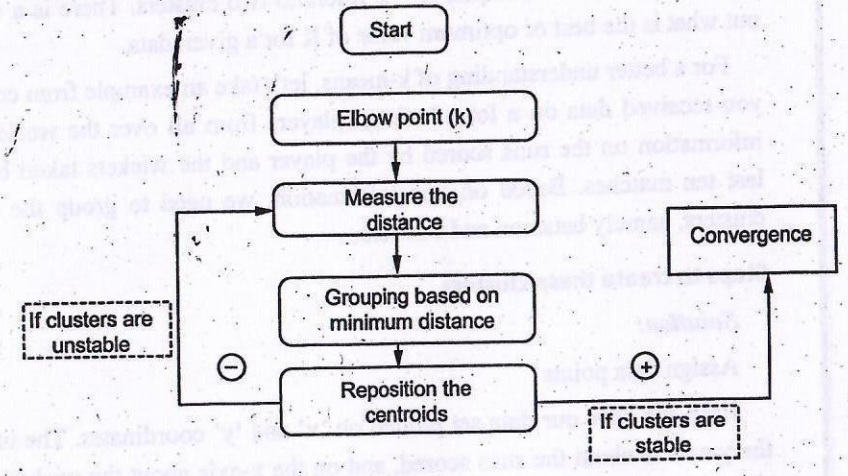


Fig. 4.6.

4.3.5. K-NEAREST NEIGHBOUR (KNN) ALGORITHM FOR MACHINE 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.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

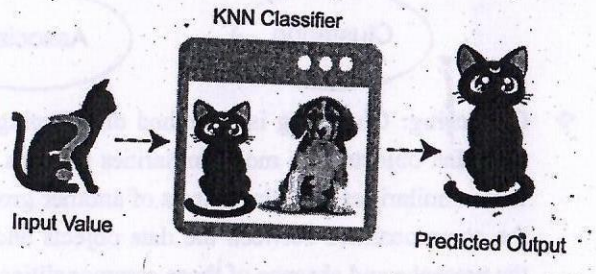


Fig. 4.7.

- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity

measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of K number of neighbors
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

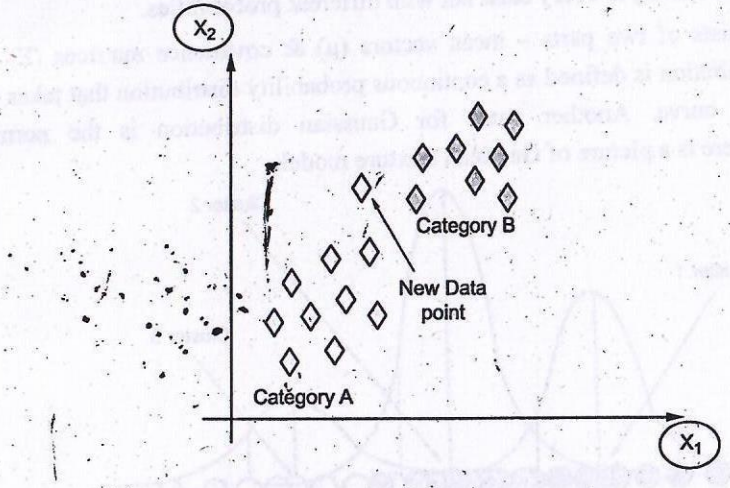
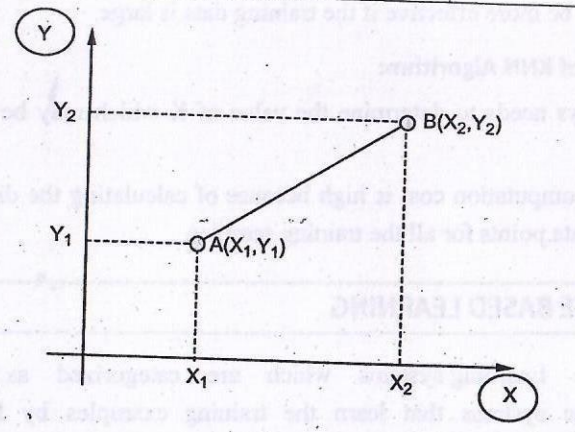


Fig. 4.8.

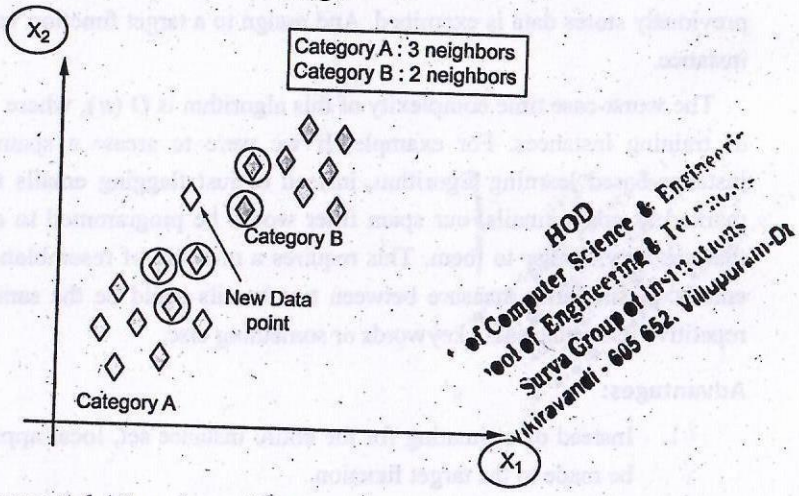
Firstly, we will choose the number of neighbors, so we will choose the $k = 5$.

- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and $B_2 = \sqrt{X_2 - X_1) + (Y_2 - Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

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.

4.4. INSTANCE BASED LEARNING

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** (because they delay processing until a new instance must be classified). The time complexity of this algorithm depends upon the size of training data. Each time whenever a new query is encountered, its previously stores data is examined. And assign to a target function value for the new instance.

The worst-case time complexity of this algorithm is **O (*n*)**, where n is the number of training instances. For example, If we were to create a spam filter with an instance-based learning algorithm, instead of just flagging emails that are already marked as spam emails, our spam filter would be programmed to also flag emails that are very similar to them. This requires a measure of resemblance between two emails. A similarity measure between two emails could be the same sender or the repetitive use of the same keywords or something else.

Advantages:

1. Instead of estimating for the entire instance set, local approximations can be made to the target function.
2. This algorithm can adapt to new data easily, one which is collected as we go .

Disadvantages:

1. Classification costs are high

2. Large amount of memory required to store the data, and each query involves starting the identification of a local model from scratch.

Some of the instance-based learning algorithms are :

1. K Nearest Neighbor (KNN)
2. Self-Organizing Map (SOM)
3. Learning Vector Quantization (LVQ)
4. Locally Weighted Learning (LWL)
5. Case-Based Reasoning

4.5. GAUSSIAN MIXTURE MODEL (GMM)

This model is a soft probabilistic clustering model that allows us to describe the membership of points to a set of clusters using a mixture of Gaussian densities. It is a soft classification (in contrast to a hard one) because it assigns probabilities of belonging to a specific class instead of a definitive choice. In essence, each observation will belong to every class but with different probabilities.

GMM consists of two parts – mean vectors (μ) & covariance matrices (Σ). A Gaussian distribution is defined as a continuous probability distribution that takes on a bell-shaped curve. Another name for Gaussian distribution is the normal distribution. Here is a picture of Gaussian mixture models:

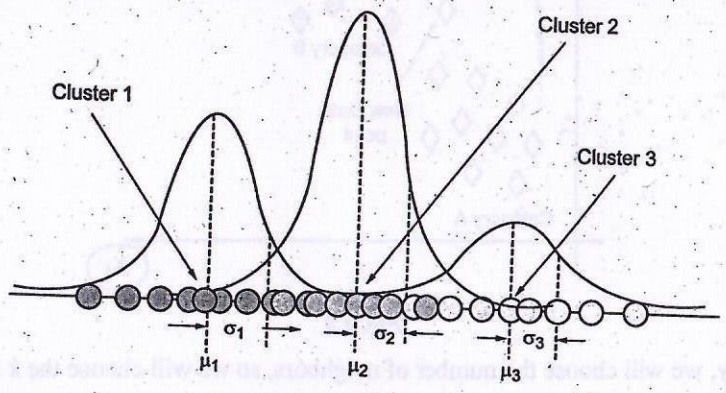


Fig. 4.9.

GMM has many applications, such as density estimation, clustering, and image segmentation. For density estimation, GMM can be used to estimate the probability

density function of a set of data points. For clustering, GMM can be used to group together data points that come from the same Gaussian distribution. And for image segmentation, GMM can be used to partition an image into different regions.

Gaussian mixture models can be used for a variety of use cases, including identifying customer segments, detecting fraudulent activity, and clustering images. In each of these examples, the Gaussian mixture model is able to identify clusters in the data that may not be immediately obvious. As a result, Gaussian mixture models are a powerful tool for data analysis and should be considered for any clustering task.

The following are three different steps to using gaussian mixture models:

- Determining a covariance matrix that defines how each Gaussian is related to one another. The more similar two Gaussians are, the closer their means will be and vice versa if they are far away from each other in terms of similarity. A gaussian mixture model can have a covariance matrix that is diagonal or symmetric.
- Determining the number of Gaussians in each group defines how many clusters there are.
- Selecting the hyperparameters which define how to optimally separate data using gaussian mixture models as well as deciding on whether or not each gaussian's covariance matrix is diagonal or symmetric.

The following are different scenarios when GMMs can be used:

- Gaussian mixture models can be used in a variety of scenarios, including when data is generated by a mix of Gaussian distributions when there is uncertainty about the correct number of clusters, and when clusters have different shapes. In each of these cases, the use of a Gaussian mixture model can help to improve the accuracy of results. For example, when data is generated by a mix of Gaussian distributions, using a Gaussian mixture model can help to better identify the underlying patterns in the data. In addition, when there is uncertainty about the correct number of clusters, the use of a Gaussian mixture model can help to reduce the error rate.
- Gaussian mixture models can be used for anomaly detection; by fitting a model to a dataset and then scoring new data points, it is possible to flag points that are significantly different from the rest of the data (i.e. outliers). This can be useful for identifying fraud or detecting errors in data collection.
- In the case of time series analysis, GMMs can be used to discover how volatility is related to trends and noise which can help predict future stock prices. One cluster could consist of a trend in the time series while another can have noise and volatility from other factors such as seasonality or external events which affect the stock price. In order to separate out these clusters, GMMs can be used because they provide a probability for each category instead of simply dividing the data into two parts such as that in the case of K-means.
- Gaussian mixture models can generate synthetic data points that are similar to the original data, they can also be used for data augmentation.

Here are some real-world problems which can be solved using Gaussian mixture models:

- **Finding patterns in medical datasets:** GMMs can be used for segmenting images into multiple categories based on their content or finding specific patterns in medical datasets. They can be used to find clusters of patients with similar symptoms, identify disease subtypes, and even predict outcomes. In one recent study, a Gaussian mixture model was used to analyze a dataset of over 700,000 patient records. The model was able to identify previously unknown patterns in the data, which could lead to better treatment for patients with cancer.
- **Modeling natural phenomena:** GMM can be used to model natural phenomena where it has been found that noise follows Gaussian distributions. This model of probabilistic modeling relies on the assumption that there exists some underlying continuum of unobserved entities or attributes and that each member is associated with measurements taken at equidistant points in multiple observation sessions.
- **Customer behavior analysis:** GMMs can be used for performing customer behavior analysis in marketing to make predictions about future purchases based on historical data.
- **Stock price prediction:** Another area Gaussian mixture models are used is in finance where they can be applied to a stock's price time series. GMMs

can be used to detect change points in time series data and help find turning points of stock prices or other market movements that are otherwise difficult to spot due to volatility and noise.

- **Gene expression data analysis:** Gaussian mixture models can be used for gene expression data analysis. In particular, GMMs can be used to detect differentially expressed genes between two conditions and identify which genes might contribute toward a certain phenotype or disease state.

4.6. EXPECTATION-MAXIMIZATION

What is an EM algorithm?

The Expectation-Maximization (EM) algorithm is defined as the combination of various unsupervised machine learning algorithms, which is used to determine the **local maximum likelihood estimates (MLE)** or **maximum a posteriori estimates (MAP)** for unobservable variables in statistical models. Further, it is a technique to find maximum likelihood estimation when the latent variables are present. It is also referred to as the **latent variable model.** A latent variable model consists of both observable and unobservable variables where observable can be predicted while unobserved are inferred from the observed variable. These unobservable variables are known as latent variables.

Key Points:

- It is known as the latent variable model to determine MLE and MAP parameters for latent variables.
- It is used to predict values of parameters in instances where data is missing or unobservable for learning, and this is done until convergence of the values occurs.

4.6.1. EM ALGORITHM

The EM algorithm is the combination of various unsupervised ML algorithms, such as the **k-means clustering algorithm.** Being an iterative approach, it consists of two modes. In the first mode, we estimate the missing or latent variables. Hence it is referred to as the **Expectation/estimation step (E-step).** Further, the other mode is

used to optimize the parameters of the models so that it can explain the data more clearly. The second mode is known as the **maximization-step or M-step.**

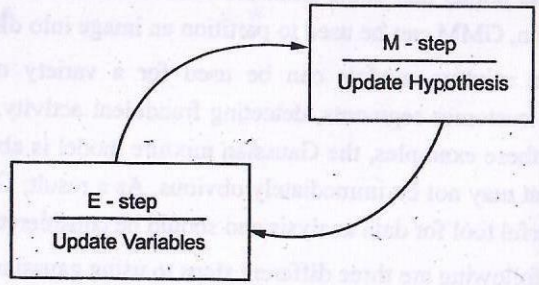


Fig. 4.10.

- **Expectation step (E - step):** It involves the estimation (guess) of all missing values in the dataset so that after completing this step, there should not be any missing value.
- **Maximization step (M - step):** This step involves the use of estimated data in the E-step and updating the parameters.
- **Repeat E-step** and M-step until the convergence of the values occurs.

The primary goal of the EM algorithm is to use the available observed data of the dataset to estimate the missing data of the latent variables and then use that data to update the values of the parameters in the M-step.

What is Convergence in the EM algorithm?

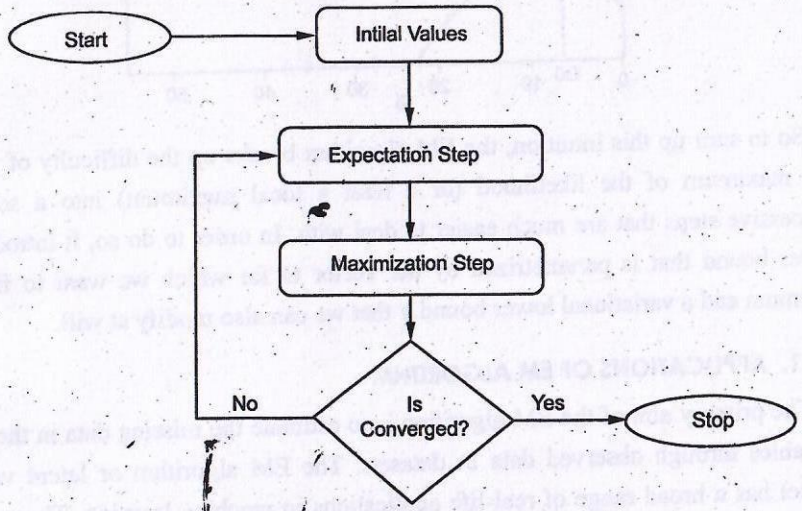
Convergence is defined as the specific situation in probability based on intuition, e.g., if there are two random variables that have very less difference in their probability, then they are known as converged. In other words, whenever the values of given variables are matched with each other, it is called convergence.

4.6.2. STEPS IN EM ALGORITHM

The EM algorithm is completed mainly in 4 steps, which include Initialization Step, Expectation Step, Maximization Step, and convergence Step. These steps are explained as follows:

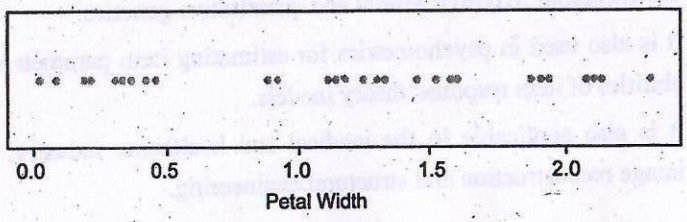
- **1st Step:** The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.

- **2nd Step:** This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- **3rd Step:** This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
- **4th step:** The last step is to check if the values of latent variables are converging or not. If it gets "yes", then stop the process; else, repeat the process from step 2 until the convergence occurs.

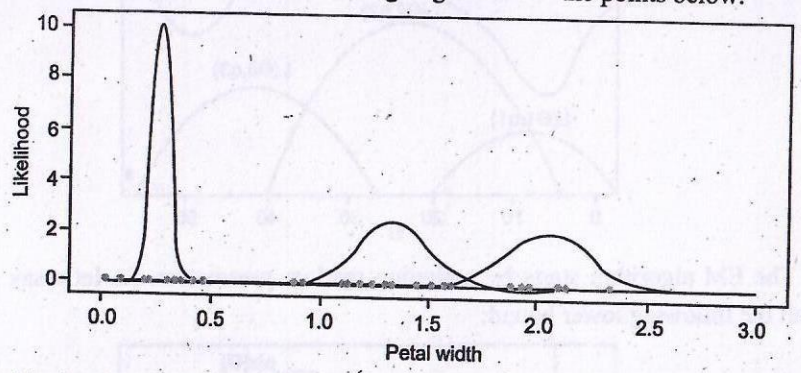


4.6.3. GMM TRAINING INTUITION

First, we are going to visually describe what happens during the training of a GMM model because it will really help to build the necessary intuition for EM. So let's say we are back into the one-dimensional example but without labels this time. Try to imagine how we could assign cluster labels to the below observations:



Well, if we already knew where the Gaussians are in the above plot, for each observation, we could compute the cluster probabilities. Let's draw this picture so you have it in mind. So we would be assigning a color to the points below:



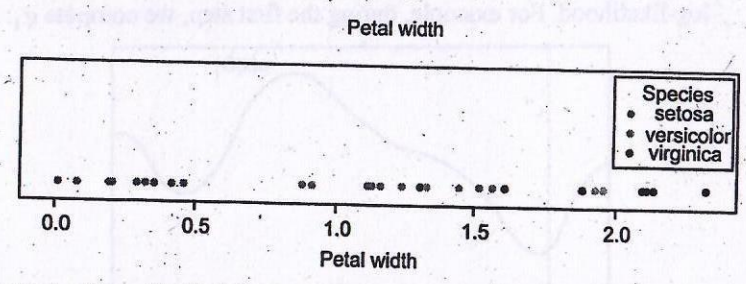
Intuitively, for one selected observation and one selected Gaussian, the probability of the observation belonging to the cluster would be the ratio between the Gaussian value and the sum of all the Gaussians. Something like:

$$P(x_i \text{ belongs to Cluster 1}) = \frac{\alpha_1 N(x_i \mid \mu_1, \Sigma_1)}{Z}$$

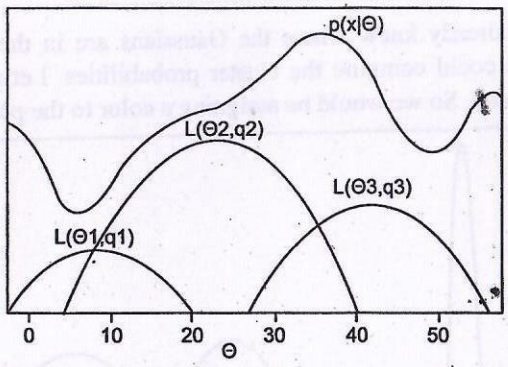
$$P(x_i \text{ belongs to Cluster 2}) = \frac{\alpha_2 N(x_i \mid \mu_2, \Sigma_2)}{Z} \quad \text{with } Z = \sum_{j=1}^{K} \alpha_j N(x_i \mid \mu_j, \Sigma_j)$$

$$P(x_i \text{ belongs to Cluster 3}) = \frac{\alpha_3 N(x_i \mid \mu_3, \Sigma_3)}{Z}$$

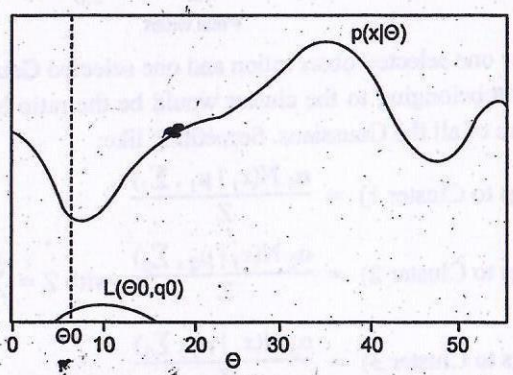
To find, where the Gaussians are to be located in the above plot, already in possession of the observation's labels, like so:



Now we could easily find the parameter values and draw the Gaussians. It suffices to consider the points independently, say the red points, and find the maximum likelihood estimate. For a Gaussian distribution, one can demonstrate the following results:

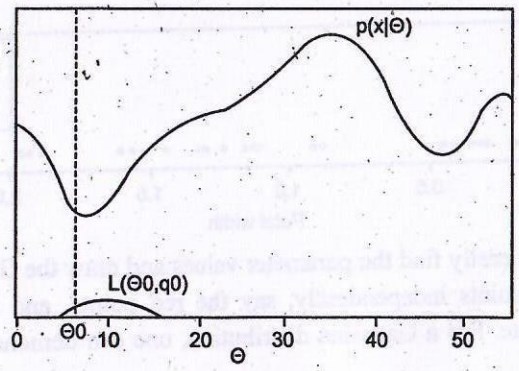


The EM algorithm starts by assigning random parameters. So let's say we start with the following lower bound:

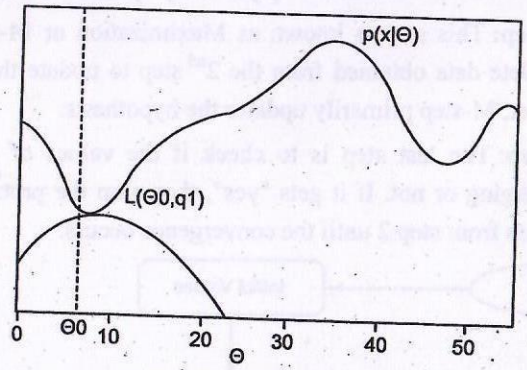


The algorithm will now perform two successive steps:

1. Fix Θ and adjust q so that the lower-bound gets as close as possible to the log-likelihood. For example, during the first step, we compute q_1:



2. Fix q and adjust Θ so that the lower-bound gets maximized. For example, during the second step, we compute Θ1:



So to sum up this intuition, the EM algorithm breaks up the difficulty of finding the maximum of the likelihood (or a least a local maximum) into a series of successive steps that are much easier to deal with. In order to do so, it introduces a lower bound that is parametrized by the vector Θ for which we want to find the optimum and a variational lower bound q that we can also modify at will.

4.7.1. APPLICATIONS OF EM ALGORITHM

The primary aim of the EM algorithm is to estimate the missing data in the latent variables through observed data in datasets. The EM algorithm or latent variable model has a broad range of real-life applications in machine learning. These are as follows:

- The EM algorithm is applicable in data clustering in machine learning.
- It is often used in computer vision and NLP (Natural language processing).
- It is used to estimate the value of the parameter in mixed models such as the **Gaussian Mixture Model** and quantitative genetics.
- It is also used in psychometrics for estimating item parameters and latent abilities of item response theory models.
- It is also applicable in the medical and healthcare industry, such as in image reconstruction and structural engineering.

Advantages of EM algorithm

- It is very easy to implement the first two basic steps of the EM algorithm in various machine learning problems, which are E-step and M- step.
- It is mostly guaranteed that likelihood will enhance after each iteration.
- It often generates a solution for the M-step in the closed form.

Disadvantages of EM algorithm

- The convergence of the EM algorithm is very slow.
- It can make convergence for the local optima only.
- It takes both forward and backward probability into consideration. It is opposite to that of numerical optimization, which takes only forward probabilities.

TWO MARKS QUESTIONS AND ANSWERS (PART - A)

1. ***Define Ensemble methods***

It is a machine learning technique that combines several base models in order to produce one optimal predictive model. Decision Trees is best to outline the definition and practicality of Ensemble Methods (however it is important to note that Ensemble Methods do not only pertain to Decision Trees).

2. ***List the Ensemble Techniques***

A few simple but powerful techniques, namely:

- Max Voting
- Averaging
- Weighted Averaging

3. ***What are the Advanced Ensemble techniques***

- Stacking
- Blending
- Bagging
- Boosting

4. ***What do you mean by Unsupervised learning***

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition

5. ***List some of the popular unsupervised learning algorithms:***

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

6. ***State the Applications of K-Means Clustering.***

K-Means clustering is used in a variety of examples or business cases in real life, like:

- Academic performance
- Diagnostic systems
- Search engines
- Wireless sensor networks

7. ***State the Gaussian Mixture Model (GMM).***

This model is a soft probabilistic clustering model that allows us to describe the membership of points to a set of clusters using a mixture of Gaussian densities. It is a soft classification (in contrast to a hard one) because it assigns probabilities of belonging to a specific class instead of a definitive choice. In essence, each observation will belong to every class but with different probabilities.

8. ***Mention the pros and cons of KNN Algorithm.***

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.

9. ***What is meant by Expectation Maximization (EM) intuition?***

The Expectation-Maximization algorithm is performed exactly the same way. In fact, the optimization procedure we describe above for GMMs is a specific implementation of the EM algorithm. The EM algorithm is just more generally and formally defined (as it can be applied to many other optimization problems).

So the general idea is that we are trying to maximize a likelihood (and more frequently a log-likelihood), that is, we are trying to solve the following optimization problem:

$$\max_{\Theta} \log P(X \mid \Theta) = \max_{\Theta} \log \left[\prod_{i} P(x_i \mid \Theta) \right] = \max_{\Theta} \sum_{i} \log (P(x_i \mid \Theta))$$

PART - B & C

1. Elaborate Ensemble Techniques in detail
2. Compare the Advanced Ensemble techniques
3. Explain the Working of Unsupervised Learning
4. Clarify on Unsupervised Learning Algorithm
5. Detail the Steps to create k-means clusters
6. Demonstrate Gaussian Mixture Model (GMM)
7. Discuss on Expectation Maximization (EM) intuition

****

UNIT V

NEURAL NETWORKS

Perceptron - Multilayer perceptron, activation functions, network training – gradient descent optimization – stochastic gradient descent, error backpropagation, from shallow networks to deep networks – Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout.

What is the Perceptron Model in Machine Learning?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:

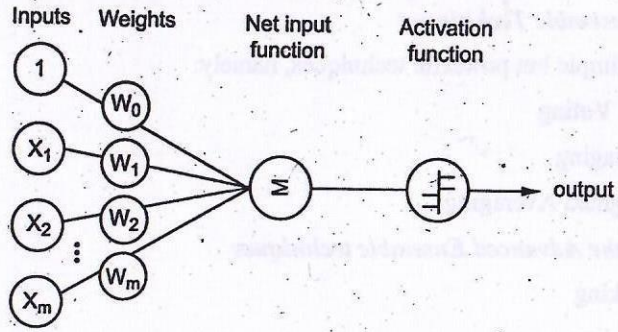


Fig. 5.1.

Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

Wight and Bias:

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- ❖ Sign function
- ❖ Step function, and
- ❖ Sigmoid function

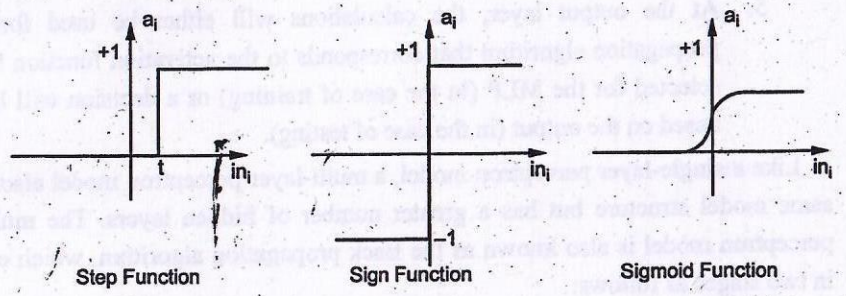


Fig. 5.2.

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and

Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function '*f*' to obtain the desired output. This activation function is also known as the **step function** and is represented by '*f*'.

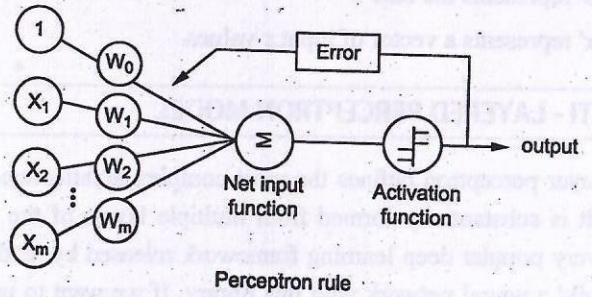


Fig. 5.3.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + \ldots w_n * x_n$$

Add a special term called **bias** '*b*' to this weighted sum to improve the model's performance.

$$\sum w_i * x_i + b$$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f(\sum w_i * x_i + b)$$

Perceptron Function

Perceptron function "$f(x)$" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

$$f(x) = 1; \text{ if } w \cdot x + b > 0$$

$$\text{otherwise}, f(x) = 0$$

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

5.1. MULTI - LAYERED PERCEPTRON MODEL

Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron. Tensor Flow 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, we have to develop a multi-layer perceptron from scratch using Numpy.

The pictorial representation of multi-layer perceptron learning is as shown below-

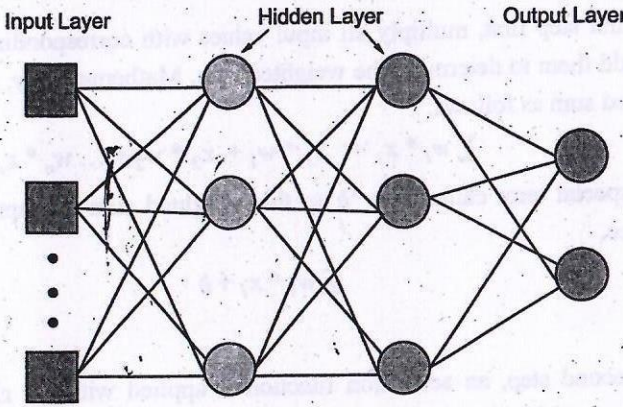


Fig. 5.4.

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 backpropagation for training the network. MLP is a deep learning method.

The algorithm for the MLP is as follows:

1. Just as with the perceptron, the inputs are pushed forward through the MLP by taking the dot product of the input with the weights that exist between the input layer and the hidden layer (WH). This dot product yields a value at the hidden layer. We do not push this value forward as we would with a perceptron though.
2. MLPs utilize activation functions at each of their calculated layers. There are many activation functions to discuss: rectified linear units (ReLU), sigmoid function, tanh. Push the calculated output at the current layer through any of these activation functions.
3. Once the calculated output at the hidden layer has been pushed through the activation function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.
4. Repeat steps two and three until the output layer is reached.
5. At the output layer, the calculations will either be used for a back propagation algorithm that corresponds to the activation function that was selected for the MLP (in the case of training) or a decision will be made based on the output (in the case of testing).

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers. The multi-layer perceptron model is also known as the Back propagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain

linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

5.2. ACTIVATION FUNCTIONS

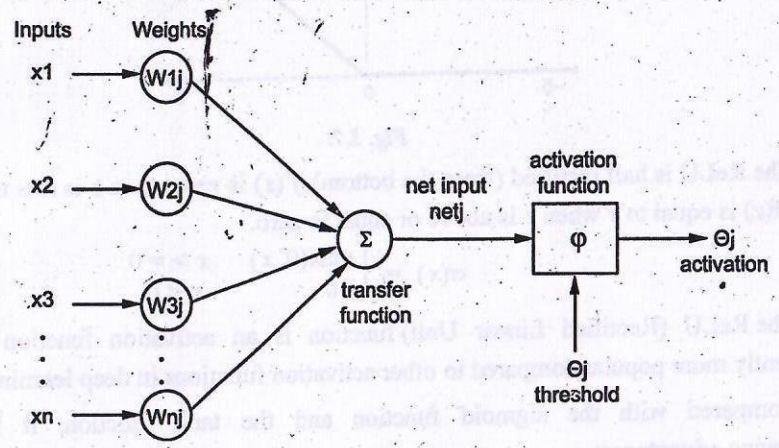


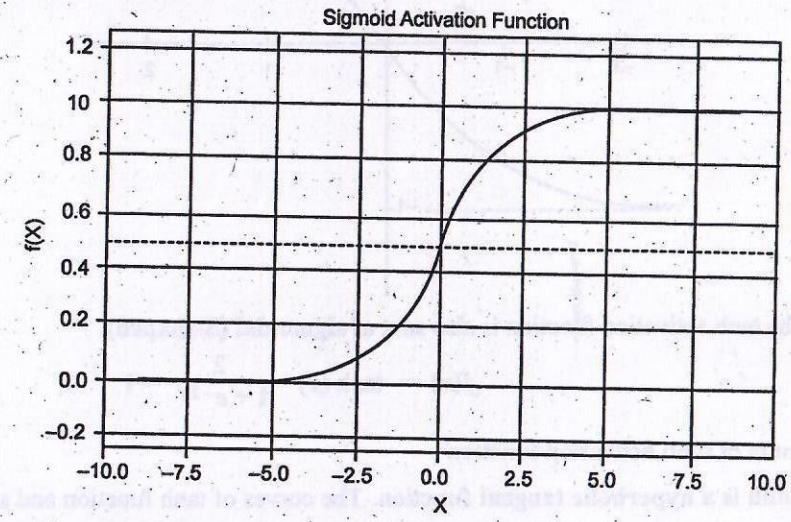
Fig. 5.5.

An activation function is a function that is added to an artificial neural network in order to help the network learn complex patterns in the data. When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding what is to be fired to the next neuron. The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern. Thus the activation function is an important part of an artificial neural network. They basically decide whether a neuron should be activated or not. Thus it bounds the value of the net input. The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.

5.2.1. TYPES OF ACTIVATION FUNCTIONS

Several different types of activation functions are used in Deep Learning. Some of them are explained below:

1. Sigmoid Activation Function -



The **Sigmoid Function** looks like an **S-shaped** curve.

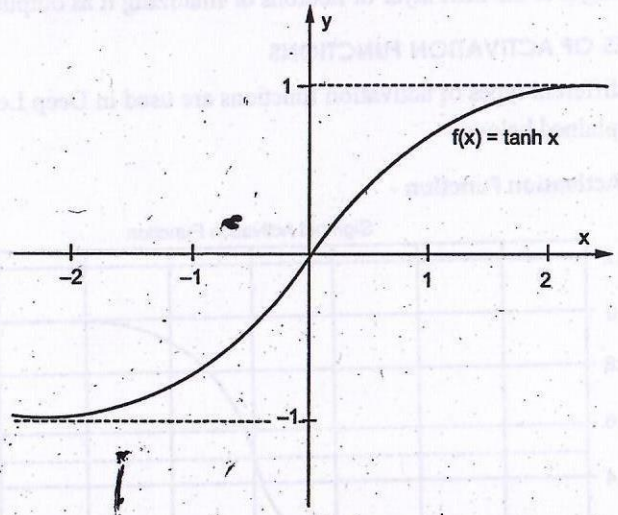
Formula : $f(z) = 1 / (1 + e^{-z})$

Why and when do we use the Sigmoid Activation Function?

1. The output of a sigmoid function ranges between 0 and 1. Since, output values bound between 0 and 1, it normalizes the output of each neuron.

2. Specially used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the perfect choice.
3. Smooth gradient, preventing "jumps" in output values.
4. The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points.
5. Clear predictions, i.e very close to 1 or 0.

2. Tanh or Hyperbolic Tangent Activation Function -



The tanh activation function is also **sort of sigmoidal (S-shaped).**

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Formula of tanh activation function

Tanh is a hyperbolic tangent function. The curves of tanh function and sigmoid function are relatively similar. But it has some advantage over the sigmoid function. Let's look at what it is.

Why is tanh **better compared** to sigmoid activation function?

1. First of all, when the input is large or small, the output is almost smooth and the gradient is small, which is not conducive to weight update. The

difference is the output interval. The output interval of tanh is 1, and the whole function is 0-centric, which is better than sigmoid.
2. The major advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

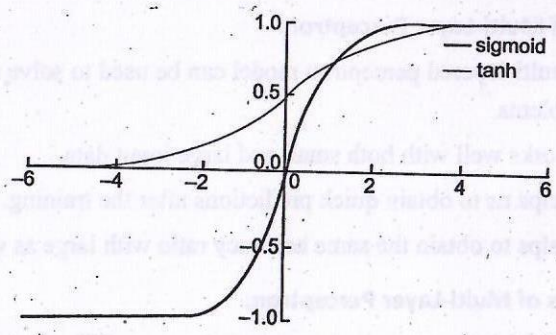


Fig. 5.6.

3. ReLU (Rectified Linear Unit) Activation Function-

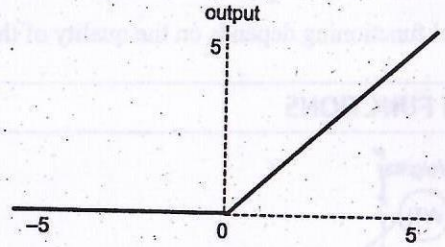


Fig. 5.7.

The ReLU is half rectified (from the bottom). $f(z)$ is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

$$\sigma(x) = \begin{cases} \max(0, x) & x >= 0 \\ 0 & x < 0 \end{cases}$$

The ReLU (Rectified Linear Unit) function is an activation function that is currently more popular compared to other activation functions in deep learning.

Compared with the sigmoid function and the tanh function, it has the following advantages:

1. When the input is positive, there is no gradient saturation problem.

2. The calculation speed is much faster. The ReLU function has only a linear relationship. Whether it is forward or backward, it is much faster than sigmoid and tanh. (Sigmoid and tanh need to calculate the exponent, which will be slower.)

4. Leaky ReLU Activation Function

An activation function specifically designed to compensate for the dying ReLU problem.

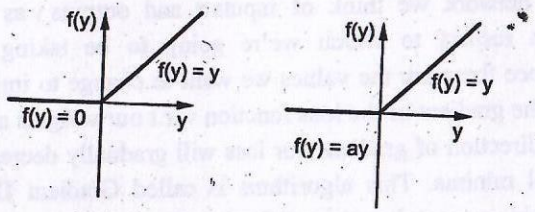


Fig. 5.8. ReLU vs Leaky ReLU

Why Leaky ReLU is better than ReLU?

$$f(y_i) = \begin{cases} y_i & \text{if } y_i > 0 \\ a_i y_i & \text{if } y_i \leq 0 \end{cases}$$

1. The leaky ReLU adjusts the problem of zero gradients for negative value, by giving a very small linear component of x to negative inputs(0.01 x).
2. The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.
3. Range of the Leaky ReLU is (-infinity to infinity).

5. ELU (Exponential Linear Units) Function-

ELU is also proposed to solve the problems of ReLU. In contrast to ReLUs, ELUs have negative values which pushes the mean of the activations closer to zero. Mean activations that are closer to zero enable faster learning as they bring the gradient closer to the natural gradient.

$$g(x) = \text{ELU}(x) = \begin{cases} x & x > 0 \\ a(e^x - 1) & x \leq 0 \end{cases}$$

Obviously, ELU has all the advantages of ReLU, and:

- No Dead ReLU issues, the mean of the output is close to 0, zero-centered.

- In contrast to ReLUs, ELUs have negative values which allows them to push mean unit activations closer to zero like batch normalization but with lower computational complexity. Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect.
- ELUs saturate to a negative value with smaller inputs and thereby decrease the forward propagated variation and information.

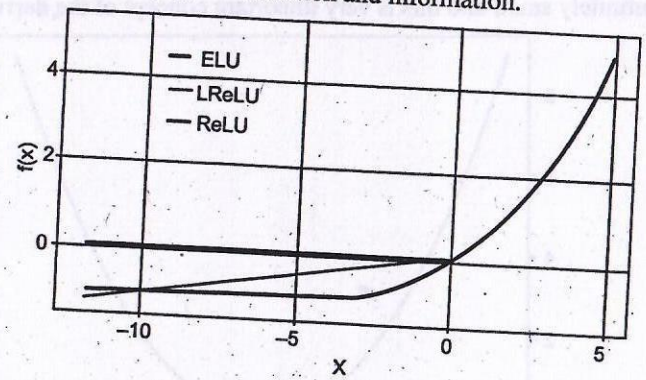


Fig. 5.9. ELU vs Leaky ReLU vs ReLU

5.3. NETWORK TRAINING

In the process of training, we want to start with a bad performing neural network and wind up with network with high accuracy. In terms of loss function, we want our loss function to much lower in the end of training. Improving the network is possible, because we can change its function by adjusting weights. We want to find another function that performs better than the initial one. The problem of training is equivalent to the problem of minimizing the loss function. Why minimize loss instead of maximizing? Turns out loss is much easier function to optimize. There are a lot of algorithms that optimize functions. These algorithms can gradient-based or not, in sense that they are not only using the information provided by the function, but also by its gradient.

First, we need to remember what a derivative is with respect to some variable. Let's take some easy function $f(x) = x$. If we remember the rules of calculus from high school we know, that the derivative of that is one at every value of x. The

derivative is the rate of how fast our function is changing when we take infinitely small step in the positive direction. Mathematically it can be written as the following:

$$\nabla L \approx \partial \frac{L}{\partial x_i} \nabla x_i$$

Which means: how much our function changes(left term) approximately equals to derivative of that function with respect to some variable x multiplied with how much we changed that variable. That approximation is going to be exact when we step we take is infinitely small and this is very important concept of the derivative.

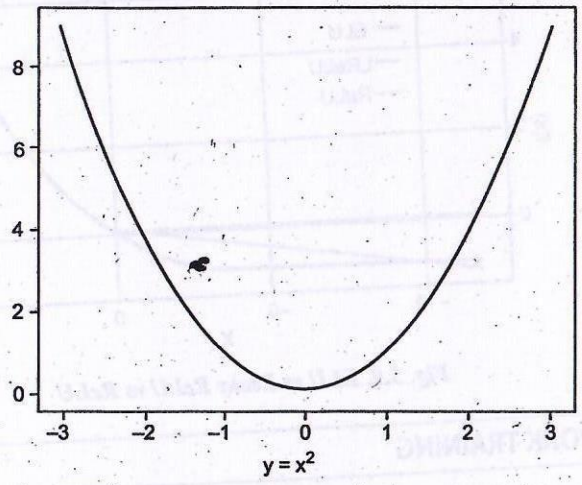


Fig. 5.10.

Let's go back to our function $f(x) = x^2$. Obviously, the minimum of that function is at point x = 0, but how would a computer know it ? Suppose, we start off with some random value of x and this value is 2. The derivative of the function in that in $x = 2$ equals 4. Which means that is we take a step in positive direction our function will change proportionally to 4.

Our derivative only guarantees that the function will decrease if take infinitely small step. Generally, you want to control how big of step you make with some kind of hyper-parameter. This hyper-parameter is called *learning rate* and I'll talk about it later. Let's now see what happens if we start at a point $x = -2$. The derivative is now equals – 4, which means, that if take a small step in positive direction our function will change proportionally to – 4, thus it will decrease. That's exactly what we want.

When $x > 0$, our derivative greater than zero and we need to go in negative direction, when $x < 0$, the derivative less than zero, we need to go in positive direction. We always need to take a step in the direction which is opposite of derivative. Let's apply the same idea to gradient. Gradient is vector which points to some direction in space. It actually point to the direction of the steepest increase of the function. Since we want minimize our function, we'll take a step in the opposite direction of gradient.

In neural network we think of inputs x, and outputs y as fixed numbers. The variable with respect to which we're going to be taking our derivatives are weights w, since these are the values we want to change to improve our network. If we compute the gradient of the loss function w.r.t our weights and take small steps in the opposite direction of gradient our loss will gradually decrease until it converges to some local minima. This algorithms is called Gradient Descent. The rule for updating weights on each iteration of Gradient Descent is the following:

$$w_j = w_j - l\,r\,\partial \frac{L}{\partial w_j}$$

lr in the notation above means learning rate. It's there to control how big of a step we're taking each iteration. It is the most important hyper-parameter to tune when training neural networks.

5.3.1. THE ARTIFICIAL NEURAL NETWORK

To build a good Artificial Neural Network (ANN) you will need the following ingredients

Ingredients:

- Artificial Neurons (processing node) composed of:
  - ✓ (many) **input** neuron(s) connection(s) (dendrites)
  - ✓ a **computation unit** (nucleus) composed of:
    - a **linear function** $(ax + b)$
    - an **activation function** (equivalent to the the **synapse**)
  - ✓ an **output** (axon)

Preparation to get an ANN for image classification training:

1. Decide on the number of output classes (meaning the number of image classes – for example two for cat vs dog).

2. Draw as many computation units as the number of output classes (congrats you just create the Output Layer of the ANN).
3. Add as many Hidden Layers as needed within the defined architecture. Hidden Layers are just a set of neighboured Compute Units, they are not linked together.
4. Stack those Hidden Layers to the Output Layer using Neural Connections
5. It is important to understand that the Input Layer is basically a layer of data ingestion
6. Add an Input Layer that is adapted to ingest your data (or you will adapt your data format to the pre-defined architecture)
7. Assemble many Artificial Neurons together in a way where the output (axon) an Neuron on a given Layer is (one) of the input of another Neuron on a subsequent Layer. As a consequence, the Input Layer is linked to the Hidden Layers which are then linked to the Output Layer (as shown in the picture below) using Neural Connections

Training an Artificial Neural Network (ANN) requires just a few steps:

1. First an ANN will require a random weight initialization
2. Split the dataset in batches (batch size)
3. Send the batches 1 by 1 to the GPU
4. Calculate the forward pass (what would be the output with the current weights)
5. Compare the calculated output to the expected output (loss)
6. Adjust the weights (using the learning rate increment or decrement) according to the backward pass (backward gradient propagation).

5.4. STOCHASTIC GRADIENT DESCENT

What is Gradient Descent?

Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea is to tweak parameters iteratively in order to minimize the cost function. An important parameter of

Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

What is the objective of Gradient Descent?

Gradient, in plain terms means slope or slant of a surface. So gradient descent literally means descending a slope to reach the lowest point on that surface. Let us imagine a two dimensional graph, such as a parabola in the figure below.

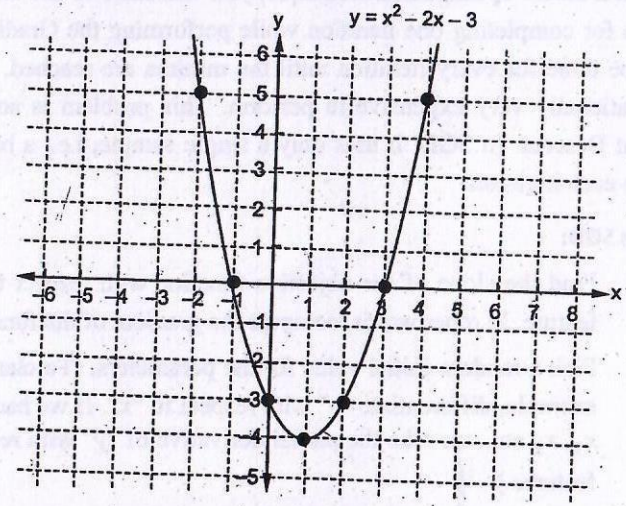


Fig. 5.11.

A parabolic function with two dimensions (x , y)

In the above graph, the lowest point on the parabola occurs at $x = 1$. The objective of gradient descent algorithm is to find the value of "x" such that "y" is minimum. "y" here is termed as the objective function that the gradient descent algorithm operates upon, to descend to the lowest point.

5.4.1. TYPES OF GRADIENT DESCENT:

Typically, there are three types of Gradient Descent:

- Batch Gradient Descent
- Stochastic Gradient Descent

❖ Mini-batch Gradient Descent

Stochastic Gradient Descent (SGD):

In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima are reached. Hence, it becomes computationally very expensive to perform. This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration.

Steps in SGD:

1. Find the slope of the objective function with respect to each parameter / feature. In other words, compute the gradient of the function.
2. Pick a random initial value for the parameters. (To clarify, in the parabola example, differentiate "y" with respect to "x". If we had more features like x_1, x_2 etc., we take the partial derivative of "y" with respect to each of the features.)
3. Update the gradient function by plugging in the parameter values.
4. Calculate the step sizes for each feature as : step size = gradient * learning rate.
5. Calculate the new parameters as : new params = old params -step size
6. Repeat steps 3 to 5 until gradient is almost 0.

Stochastic Gradient Descent using Python

The SGD algorithm is used in several loss functions. Simply put, it is used to minimize a cost function by iterating a gradient-based weight update. Instead of looking at the full dataset, the weight update is applied to batches randomly extracted from it, which is why it is also known as mini-batch gradient descent.

Below is the process of the stochastic gradient descent algorithm:

1. The algorithm starts at a random point by initializing the weights with random values
2. Then it calculates the gradients at that random point
3. Then it moves in the opposite direction of the gradient
4. The process continues to repeat itself until it finds the point of minimum loss

```
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score
samples = 500
x, y = make_classification(n_samples=samples, n_features=2,
                           n_informative=2, n_redundant=0,
                           n_clusters_per_class=1)
SGD_classifier = SGDClassifier(loss="perceptron", learning_rate="optimal",
                               n_iter_no_change=10)
print(cross_val_score(SGD_classifier, x, y, scoring="accuracy", cv=10).mean())
```

5.5. ERROR BACKPROPAGATION, FROM SHALLOW NETWORKS TO DEEP NETWORKS

Backpropagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter. Similarly here we also use gradient descent algorithm using Backpropagation.

Backpropagation defines the whole process encompassing both the calculation of the gradient and its need in the stochastic gradient descent. Technically,

backpropagation is used to calculate the gradient of the error of the network concerning the network's modifiable weights. The characteristics of Backpropagation are the iterative, recursive and effective approach through which it computes the updated weight to increase the network until it is not able to implement the service for which it is being trained. Derivatives of the activation service to be known at network design time are needed for Backpropagation.

Backpropagation is widely used in neural network training and calculates the loss function for the weights of the network. Its service with a multi-layer neural network and discover the internal description of input-output mapping. It is a standard form of artificial network training, which supports computing gradient loss function concerning all weights in the network. The backpropagation algorithm is used to train a neural network more effectively through a chain rule method. This gradient is used in a simple stochastic gradient descent algorithm to find weights that minimize the error. The error propagates backward from the output nodes to the inner nodes.

The training algorithm of backpropagation involves four stages which are as follows

- **Initialization of weights** – There are some small random values are assigned.
- **Feed-forward** – Each unit X receives an input signal and transmits this signal to each of the hidden unit $Z_1, Z_2, \ldots Z_n$. Each hidden unit calculates the activation function and sends its signal Z_1 to each output unit. The output unit calculates the activation function to form the response of the given input pattern.
- **Backpropagation of errors** – Each output unit compares activation Y_k with the target value T_k to determine the associated error for that unit. It is based on the error, the factor $\delta\delta_k$ (K = 1, ... *m*) is computed and is used to distribute the error at the output unit Y_k back to all units in the previous layer. Similarly the factor $\delta\delta_j$(*j* = 1, ... *p*) is compared for each hidden unit Z_j.
- It can update the weights and biases.

Consider the following Back propagation neural network example diagram to understand:

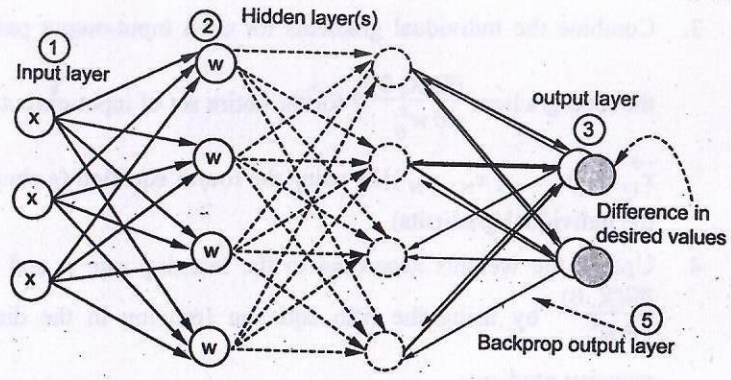


Fig. 5.12.

The General Algorithm

The backpropagation algorithm proceeds in the following steps, assuming a suitable learning rate \ alpha α and random initialization of the parameters w_{ij}^k:wijk:

Definition

1. Calculate the forward phase for each input-output pair ($\vec{x_d}$, y_d) and store the results $\hat{y}_d$, a_j^k and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.
2. Calculate the backward phase for each input-output pair ($\vec{x_d}$, y_d) and store the results $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k connecting node in layer $k-1$ to node j in layer k by proceeding from layer m, the output layer, to layer1, the input layer.
   (*a*) Evaluate the error term for the final layer δ_1^m by using the second equation.
   (*b*) Backpropogate the error terms for the hidden layers δ_j^k, working backwards from the final hidden layer $k = m-1$, by repeatedly using the third equation.
   (*c*) Evaluate the partial derivatives of the individual error E_d with respect to w_{ij}^k by using the first equation.

3. Combine the individual gradients for each input-output pair $\frac{\partial E_d}{\partial w_{ij}^k}$ to get the total gradient $\frac{\partial E(X, \theta)}{\partial w_{ij}^k}$? for the entire set of input-output pairs X = { ($\vec{x_1}$, y_1) , ..., ($\vec{x_N}$, y_N)}by using the fourth equation (a simple average of the individual gradients).
4. Update the weights according to the learning rate α and total gradient $\frac{\partial E(X, \theta)}{\partial w_{ij}^k}$ by using the fifth equation (moving in the direction of the negative gradient).

How Backpropagation Algorithm Works

Inputs X, arrive through the preconnected path

1. Input is modeled using real weights W. The weights are usually randomly selected.
2. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
3. Calculate the error in the outputs

   Error B = Actual Output – Desired Output
4. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
5. Keep repeating the process until the desired output is achieved

Why We Need Backpropagation?

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Types of Backpropagation

There are two types of Backpropagation which are as follows –

Static Back Propagation – In this type of backpropagation, the static output is created because of the mapping of static input. It is used to resolve static classification problems like optical character recognition.

Recurrent Backpropagation – The Recurrent Propagation is directed forward or directed until a specific determined value or threshold value is acquired. After the certain value, the error is evaluated and propagated backward.

Key Points

- Simplifies the network structure by elements weighted links that have the least effect on the trained network
- You need to study a group of input and activation values to develop the relationship between the input and hidden unit layers.
- It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.
- Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
- Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.

5.6. UNIT SATURATION (AKA THE VANISHING GRADIENT PROBLEM)

The vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks.

The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network. Since the gradients control how much the network learns during training, if the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

The problem:

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

Why:

Certain activation functions, like the sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

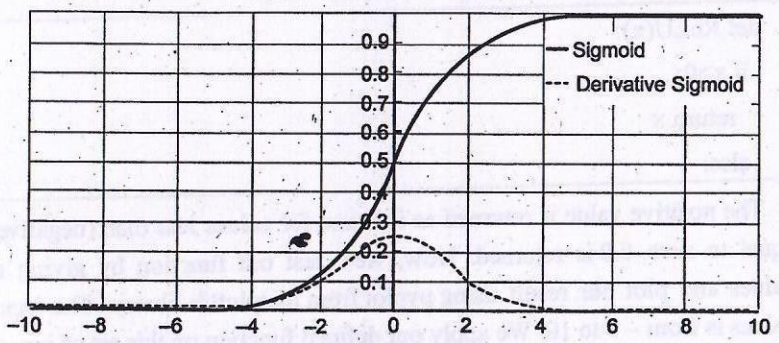


Fig. 5.13.

The sigmoid function and its derivative

As an example, Image 1 is the sigmoid function and its derivative. Note how when the inputs of the sigmoid function becomes larger or smaller (when $|x|$ becomes bigger), the derivative becomes close to zero.

Why it's significant:

For shallow network with only a few layers that use these activations, this isn't a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively. Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.

However, when *n* hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases

exponentially as we propagate down to the initial layers. A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

Solution:

The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative. Residual networks are another solution, as they provide residual connections straight to earlier layers. The residual connection directly adds the value at the beginning of the block, x, to the end of the block ($F(x) + x$). This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.

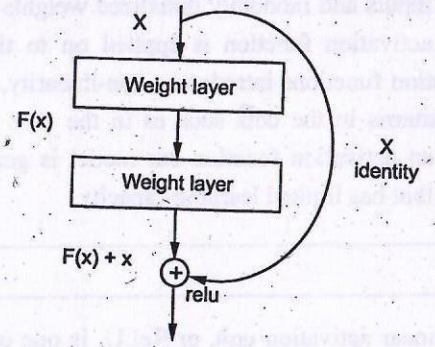


Fig. 5.14.

A residual block

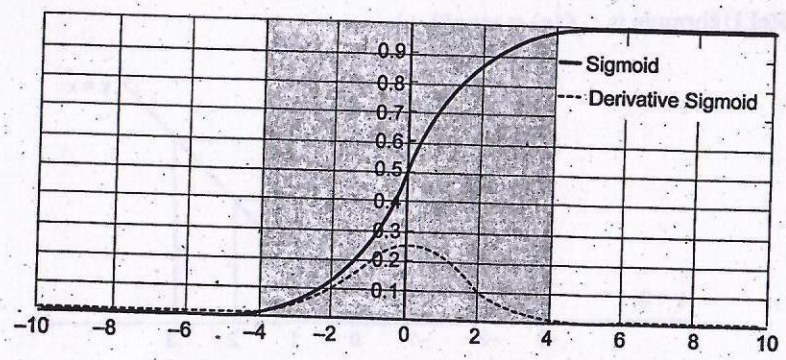


Fig. 5.15. Sigmoid function with restricted inputs

Finally, batch normalization layers can also resolve the issue. As stated before, the problem arises when a large input space is mapped to a small one, causing the derivatives to disappear. In Image 1, this is most clearly seen at when $|x|$ is big. Batch normalization reduces this problem by simply normalizing the input so $|x|$ doesn't reach the outer edges of the sigmoid function. As seen in Image 3, it normalizes the input so that most of it falls in the green region, where the derivative isn't too small.

What is an activation function?

Activation function is a simple mathematical function that transforms the given input to the required output that has a certain range. From their name they activate the neuron when output reaches the set threshold value of the function. Basically they are responsible for switching the neuron ON/OFF. The neuron receives the sum of the product of inputs and randomly initialized weights along with a static bias for each layer. The activation function is applied on to this sum, and an output is generated. Activation functions introduce a non-linearity, so as to make the network learn complex patterns in the data such as in the case of images, text, videos or sounds. Without an activation function our model is going to behave like a linear regression model that has limited learning capacity.

5.7. RELU

The rectified linear activation unit, or ReLU, is one of the few landmarks in the deep learning revolution. It's simple, yet it's far superior to previous activation functions like sigmoid or tanh.

ReLU formula is : $f(x) = \max(0, x)$

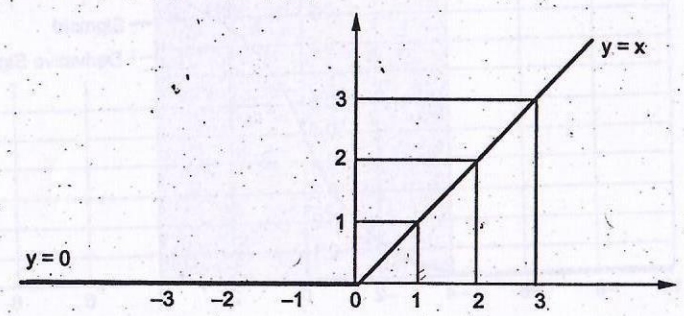


Fig. 5.16.

Both the ReLU function and its derivative are monotonic. If the function receives any negative input, it returns 0; however, if the function receives any positive value x, it returns that value. As a result, the output has a range of 0 to infinite. ReLU is the most often used activation function in neural networks, especially CNNs, and is utilized as the default activation function.

Implementing ReLU function in Python

We can implement a simple ReLU function with Python code using an if-else statement as,

```
def ReLU(x):
 if x>0:
  return x
 else:
```

The positive value is returned as it is and for values less than (negative values) or equal to zero, 0.0 is returned. Now, we'll test out function by giving some input values and plot our result using pyplot from matplotlib library. The input range of values is from – 5 to 10. We apply our defined function on this set of input values.

```
from matplotlib import pyplot

def relu(x):
  return max(0.0, x)

input = [x for x in range(-5, 10)]
# apply relu on each input
output = [relu(x) for x in input]
# plot our result
pyplot.plot(series_in, series_out)
pyplot.show()
```

We see from the plot that all the negative values have been set to zero, and the positive values are returned as it is. Note that we've given a set of consecutively increasing numbers as input, so we've a linear output with an increasing slope.

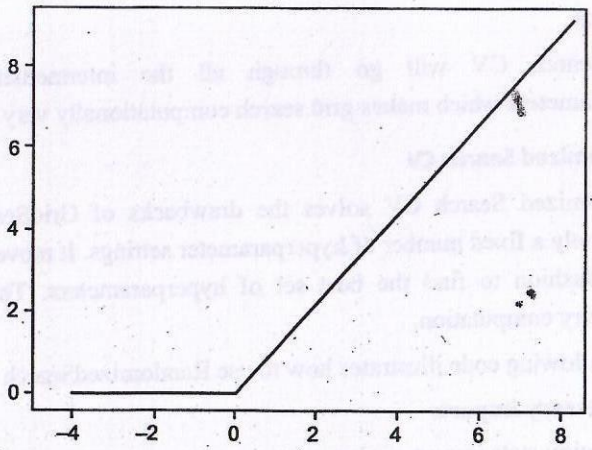


Fig. 5.17.

Advantages of ReLU:

ReLU is used in the hidden layers instead of Sigmoid or tanh as using sigmoid or tanh in the hidden layers leads to the infamous problem of "Vanishing Gradient". The "Vanishing Gradient" prevents the earlier layers from learning important information when the network is backpropagating. The sigmoid which is a logistic function is more preferrable to be used in regression or binary classification related problems and that too only in the output layer, as the output of a sigmoid function ranges from 0 to 1. Also Sigmoid and tanh saturate and have lesser sensitivity.

Some of the advantages of ReLU are:

- **Simpler Computation**: Derivative remains constant i.e 1 for a positive input and thus reduces the time taken for the model to learn and in minimizing the errors.
- **Representational Sparsity**: It is capable of outputting a true zero value.
- **Linearity**: Linear activation functions are easier to optimize and allow for a smooth flow. So, it is best suited for supervised tasks on large sets of labelled data.

Disadvantages of ReLU:

- **Exploding Gradient**: This occurs when the gradient gets accumulated, this causes a large differences in the subsequent weight updates. This as a result causes instability when converging to the global minima and causes instability in the learning too.
- **Dying ReLU**: The problem of "dead neurons" occurs when the neuron gets stuck in the negative side and constantly outputs zero. Because gradient of 0 is also 0, it's unlikely for the neuron to ever recover. This happens when the learning rate is too high or negative bias is quite large.

5.8. HYPERPARAMETER TUNING

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters. However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyperparameters include:

The penalty in Logistic Regression Classifier i.e. L_1 or L_2 regularization

The learning rate for training a neural network.

The C and sigma hyperparameters for support vector machines.

The *k* in k-nearest neighbors.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

Grid Search CV

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values. For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of C = 0.3 and Alpha = 0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.

C	Alpha = 0.1	Alpha = 0.2	Alpha = 0.3	Alpha = 0.5
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

The following code illustrates how to use GridSearchCV

```
# Necessary imports
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
# Creating the hyperparameter grid
c_space = np.logspace(-5, 8, 15)
param_grid = {'C': c_space}
# Instantiating logistic regression classifier
logreg = LogisticRegression()
# Instantiating the GridSearchCV object
logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)
logreg_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Best score is {}".format(logreg_cv.best_score_))
```

Output:

Tuned Logistic Regression Parameters: {'C': 3.7275937203149381} Best score is 0.7708333333333334

Drawback:

GridSearch CV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

9. Randomized Search CV

Randomized Search CV solves the drawbacks of GridSearch CV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

The following code illustrates how to use RandomizedSearch CV

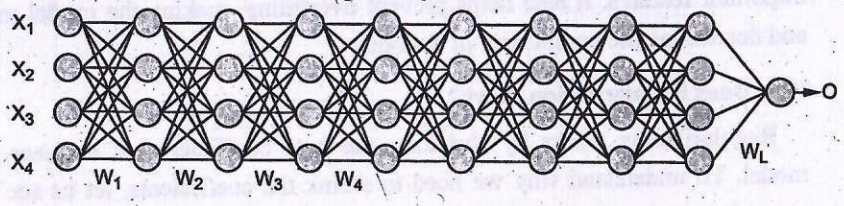
```
# Necessary imports
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

# Creating the hyperparameter grid
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

# Instantiating Decision Tree classifier
tree = DecisionTreeClassifier()
# Instantiating RandomizedSearchCV object
tree_cv = RandomizedSearchCV(tree, param_dist, cv = 5)
tree_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

Output:

Tuned Decision Tree Parameters: {'min_samples_leaf': 5, 'max_depth': 3, 'max_features': 5, 'criterion': 'gini'} Best score is 0.7265625

5.9. BATCH NORMALIZATION

Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape. -Generally, when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale. The reason we normalize is partly to ensure that our model can generalize appropriately. Now coming back to Batch normalization, it is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer. A typical neural network is trained using a collected set of input data called batch. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

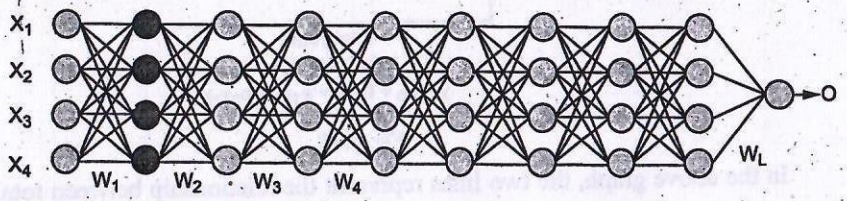


L = Number of layers

Bias = 0

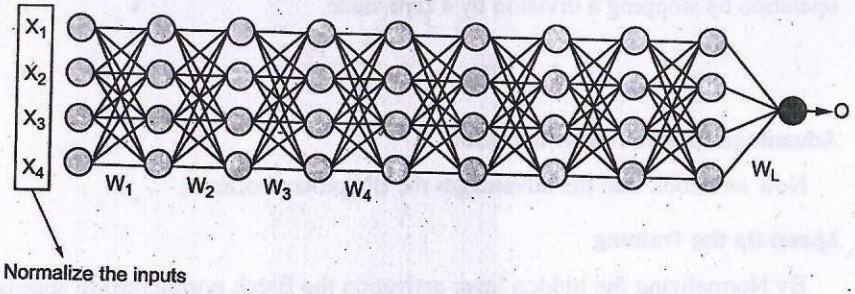
Activation Function : Sigmoid

Initially, our inputs X_1, X_2, X_3, X_4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W



$$h_1 = \sigma(W_1 X)$$

Similarly, this transformation will take place for the second layer and go till the last layer L as shown in the following image.



$$h_1 = \sigma(W_1 X)$$

$$h_2 = \sigma(W_2 h_1) = \sigma(W_2 \sigma(W_1 X))$$

$$O = \sigma(W_L h_{L-1})$$

Although, our input X was normalized with time the output will no longer be on the same scale. As the data go through multiple layers of the neural network and L activation functions are applied, it leads to an internal co-variate shift in the data.

How does Batch Normalization work?

Since by now we have a clear idea of why we need Batch normalization, let's understand how it works. It is a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.

Normalization of the Input

Normalization is the process of transforming the data to have a mean zero and standard deviation one. In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

Here, m is the number of neurons at layer h. Once we have meant at our end, the next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

Further, as we have the mean and the standard deviation ready. We will normalize the hidden activations using these values. For this, we will subtract the mean from

each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).The smoothing term(ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(\text{norm})} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

Advantages of Batch Normalization

Now let's look into the advantages the BN process offers.

Speed Up the Training

By Normalizing the hidden layer activation the Batch normalization speeds up the training process.

Handles internal covariate shift

It solves the problem of internal covariate shift. Through this, we ensure that the input for every layer is distributed around the same mean and standard deviation. If you are unaware of what is an internal covariate shift, look at the following example.

Internal covariate shift

Suppose we are training an image classification model, that classifies the images into Dog or Not Dog. Let's say we have the images of white dogs only, these images will have certain distribution as well. Using these images model will update its parameters.

Smoothens the Loss Function

Batch normalization smoothens the loss function that in turn by optimizing the model parameters improves the training speed of the model.

5.10. REGULARIZATION

The Problem of Overfitting

So, before diving into regularization, let's take a step back to understand what bias-variance is and its impact. Bias is the deviation between the values predicted by the model and the actual values whereas, variance is the difference between the predictions when the model fits different datasets.

When a model performs well on the training data and does not perform well on the testing data, then the model is said to have high generalization error. In other words, in such a scenario, the model has low bias and high variance and is too complex. This is called overfitting. Overfitting means that the model is a good fit on the train data compared to the data, as illustrated in the graph above. Overfitting is also a result of the model being too complex.

What Is Regularization in Machine Learning?

Regularization is one of the key concepts in Machine learning as it helps choose a simple model rather than a complex one. We want our model to perform well both on the train and the new unseen data, meaning the model must have the ability to be generalized. Generalization error is "a measure of how accurately an algorithm can predict outcome values for previously unseen data." Regularization refers to the modifications that can be made to a learning algorithm that helps to reduce this generalization error and not the training error. It reduces by ignoring the less important features. It also helps prevent overfitting, making the model more robust and decreasing the complexity of a model.

How Does Regularization Work?

Regularization works by shrinking the beta coefficients of a linear regression model. To understand why we need to shrink the coefficients, let us see the below example:

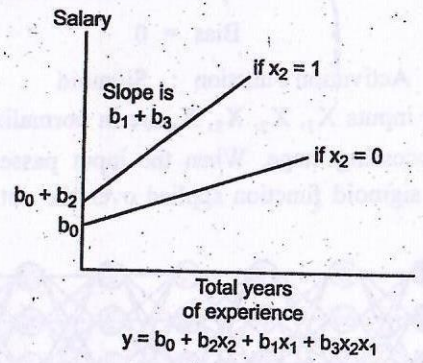


Fig. 5.18.

In the above graph, the two lines represent the relationship between total years of experience and salary, where salary is the target variable. These are slopes indicating

the change in salary per unit change in total years of experience. As the slope $b_1 + b_3$ decreases to slope b_1, we see that the salary is less sensitive to the total years of experience. By decreasing the slope, the target variable (salary) became less sensitive to the change in the independent X variables, which increases the bias into the model. Remember, bias is the difference between the predicted and the actual values.

With the increase in bias to the model, the variance (which is the difference between the predictions when the model fits different datasets.) decreases. And, by decreasing the variance, the overfitting gets reduced. The models having the higher variance leads to overfitting, and we saw above, we will shrink or reduce the beta coefficients to overcome the overfitting. The beta coefficients or the weights of the features converge towards zero, which is known as shrinkage.

What Is the Regularization Parameter?

For linear regression, the regularization has two terms in the loss function:

The Ordinary Least Squares (OLS) function, and

The penalty term

It becomes :

$$\text{Loss function}_{\text{regularization}} = \text{Loss function}_{\text{ols}} + \text{Penalty term}$$

The goal of the linear regression model is to minimize the loss function. Now for Regularization, the goal becomes to minimize the following cost function:

$$\sum_{i=1}^{n} (y_{act} - y_{pred})^2 + \text{penalty}$$

Where, the penalty term comprises the regularization parameter and the weights associated with the variables. Hence, the penalty term is:

$$\text{penalty} = \lambda * w$$

where,

λ = Regularization parameter

w = weight associated with the variables; generally considered to be the $L-p$ norms

The regularization parameter in machine learning is λ : It imposes a higher penalty on the variable having higher values, and hence, it controls the strength of the penalty term. This tuning parameter controls the bias-variance trade-off.

λ can take values 0 to infinity. If $\lambda = 0$, then means there is no difference between a model with and without regularization.

Regularization Techniques in Machine Learning

Each of the following techniques uses different regularization norms (L-p) based on the mathematical methodology that creates different kinds of regularization. These methodologies have different effects on the beta coefficients of the features. The regularization techniques in machine learning as follows:

(a) Ridge Regression

The Ridge regression technique is used to analyze the model where the variables may be having multicollinearity. It reduces the insignificant independent variables though it does not remove them completely. This type of regularization uses the L_2 norm for regularization.

$$\text{Cost function} = \sum_{i=1}^{n} (y_{act} - y_{pred})^2 + \lambda \cdot \| w \|_2^2$$

(b) Lasso Regression

Least Absolute Shrinkage and Selection Operator (or LASSO) Regression penalizes the coefficients to the extent that it becomes zero. It eliminates the insignificant independent variables. This regularization technique uses the L1 norm for regularization.

$$\text{Cost function} = \sum_{i=1}^{n} (y_{act} - y_{pred})^2 + \lambda \cdot \| w \|_1$$

(c) Elastic Net Regression

The Elastic Net Regression technique is a combination of the Ridge and Lasso regression technique. It is the linear combination of penalties for both the L_1 -norm and L_2 -norm regularization.

The model using elastic net regression allows the learning of the sparse model where some of the points are zero, similar to Lasso regularization, and yet maintains the Ridge regression properties. Therefore, the model is trained on both the L_1 and L_2 norms.

The cost function of Elastic Net Regression is:

$$\text{Cost function} = \sum_{i=1}^{n} (y_{act} - y_{pred})^2 + \lambda_{ridge} \cdot \| w \|_2^2 + \lambda_{lasso} \cdot \| w \|_1$$

When to Use Which Regularization Technique?

The regularization in machine learning is used in following scenarios:

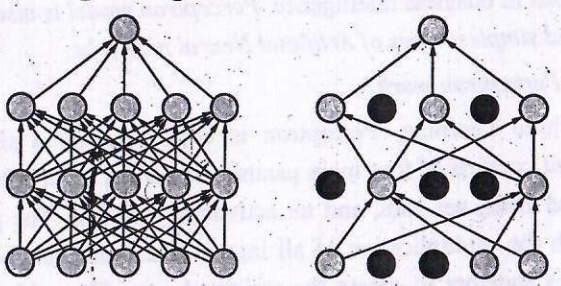
Ridge regression is used when it is important to consider all the independent variables in the model or when many interactions are present. That is where collinearity or codependency is present amongst the variables.

Lasso regression is applied when there are many predictors available and would want the model to make feature selection as well for us.

When many variables are present, and we can't determine whether to use Ridge or Lasso regression, then the Elastic-Net regression is your safe bet.

5.11. DROPOUT

"Dropout" in machine learning refers to the process of randomly ignoring certain nodes in a layer during training. In the figure below, the neural network on the left represents a typical neural network where all units are activated. On the right, the red units have been dropped out of the model — the values of their weights and biases are not considered during training.



(a) Standard Neural Net (b) After applying dropout

Fig. 5.19.

Dropout is used as a regularization technique - it prevents overfitting by ensuring that no units are codependent (more on this later).

Common Regularization Methods

Common regularization techniques include:

Early stopping: stop training automatically when a specific performance measure (eg. Validation loss, accuracy) stops improving

Weight decay: incentivize the network to use smaller weights by adding a penalty to the loss function (this ensures that the norms of the weights are relatively evenly distributed amongst all the weights in the networks, which prevents just a few weights from heavily influencing network output)

Noise: allow some random fluctuations in the data through augmentation (which makes the network robust to a larger distribution of inputs and hence improves generalization)

Model combination: average the outputs of separately trained neural networks (requires a lot of computational power, data, and time)

Dropout remains an extremely popular protective measure against overfitting because of its efficiency and effectiveness.

How Does Dropout Work?

When we apply dropout to a neural network, we're creating a "thinned" network with unique combinations of the units in the hidden layers being dropped randomly at different points in time during training. Each time the gradient of our model is updated, we generate a new thinned neural network with different units dropped based on a probability hyperparameter p. Training a network using dropout can thus be viewed as training loads of different thinned neural networks and merging them into one network that picks up the key properties of each thinned network. This process allows dropout to reduce the overfitting of models on training data.

This graph, taken from the paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" by Srivastava et al., compares the change in classification error of models without dropout to the same models with dropout (keeping all other hyperparameters constant). All the models have been trained on the MNIST dataset.

It is observed that the models with dropout had a lower classification error than the same models without dropout at any given point in time. A similar trend was observed when the models were used to train other datasets in vision, as well as speech recognition and text analysis. The lower error is because dropout helps prevent overfitting on the training data by reducing the reliance of each unit in the hidden layer on other units in the hidden layers.

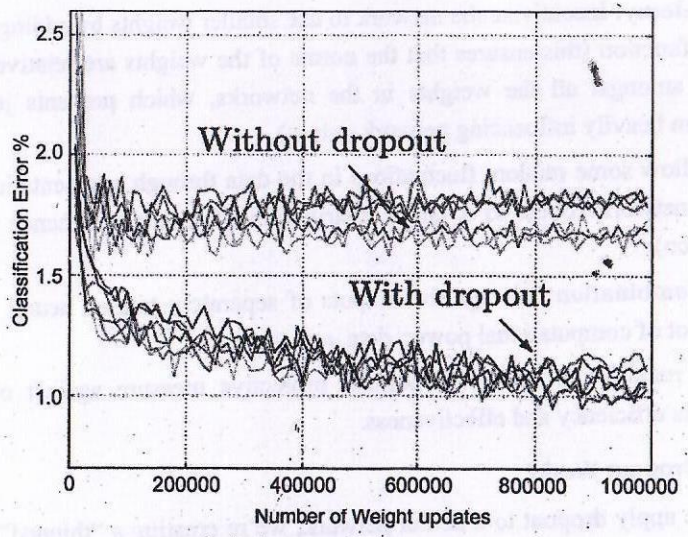


Fig. 5.20.

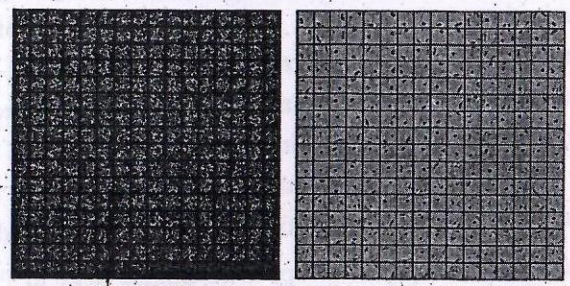


Fig. 5.21.

It can be observed in figure a that the units don't seem to pick up on any meaningful feature, whereas in figure b, the units seem to have picked up on distinct edges and spots in the data provided to them. This indicates that dropout helps break co-adaptations among units, and each unit can act more independently when dropout regularization is used. In other words, without dropout, the network would never be able to catch a unit A compensating for another unit B's flaws. With dropout, at some point unit A would be ignored and the training accuracy would decrease as a result, exposing the inaccuracy of unit B.

The Downside of Dropout

Although dropout is clearly a highly effective tool, it comes with certain drawbacks. A network with dropout can take 2 – 3 times longer to train than a standard network. One way to attain the benefits of dropout without slowing down training is by finding a regularizer that is essentially equivalent to a dropout layer. For linear regression, this regularizer has been proven to be a modified form of L_2 regularization.

TWO MARKS QUESTIONS AND ANSWERS (PART-A)

1. ***What is the Perceptron model in Machine Learning?***

   Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence. *Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks.*

2. ***How does Perceptron work?***

   In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function *'f'* to obtain the desired output. This activation function is also known as the **step function** and is represented by ***'f'***.

3. ***List down the advantages of Multi-Layer Perceptron:***

   - A multi-layered perceptron model can be used to solve complex non-linear problems.
   - It works well with both small and large input data.
   - It helps us to obtain quick predictions after the training.
   - It helps to obtain the same accuracy ratio with large as well as small data.

4. ***Compare the types of Activation Functions.***

Activation Functions

Sigmoid	**Leaky ReLU**
$\sigma(x=\frac{1}{1+e^{-x}}$	$\max(0.1\,x, x)$
tanh	**Maxout**
$\tanh(x)$	$\max(w_1^T\,x + b_1, w_2^T\,x + b_2)$
ReLU	**ELU**
$\max(0,x)$	$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

5. ***Define an Activation function***

An **activation function** is a **function** that is added to an **artificial neural network** in order to help the network learn **complex patterns in the data**. When comparing with a neuron-based model that is in our brains, the **activation function** is at the end deciding what is to be fired to the next neuron. The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern. Thus the activation function is an important part of an artificial neural network.

6. ***How do you train a neural network?***

In the process of training, we want to start with a bad performing neural network and wind up with network with high accuracy. In terms of loss function, we want our loss function to much lower in the end of training. Improving the network is possible, because we can change its function by adjusting weights. We want to find another function that performs better than the initial one.

7. ***What is the objective of Gradient Descent?***

Gradient, in plain terms means slope or slant of a surface. So gradient descent literally means descending a slope to reach the lowest point on that surface

8. ***List the types of Gradient Descent:***

Typically, there are three types of Gradient Descent:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

9. ***Define a Stochastic Gradient Descent (SGD):***

In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

10. ***Write an algorithm for stochastic gradient descent***

1. The algorithm starts at a random point by initializing the weights with random values
2. Then it calculates the gradients at that random point
3. Then it moves in the opposite direction of the gradient
4. The process continues to repeat itself until it finds the point of minimum loss

11. ***What do you mean by back propogation?***

Backpropagation defines the whole process encompassing both the calculation of the gradient and its need in the stochastic gradient descent. Technically, backpropagation is used to calculate the gradient of the error of the network concerning the network's modifiable weights. The characteristics of Backpropagation are the iterative, recursive and effective approach through which it computes the updated weight to increase the network until it is not able to implement the service for which it is being trained

12. ***What is a Backpropagation of error?***

Each output unit compares activation Y_k with the target value T_k to determine the associated error for that unit. It is based on the error, the factor $\delta\delta_k(K = 1, \ldots . m)$ is computed and is used to distribute the error at the output unit Y_k back to all units in the previous layer. Similarly the factor $\delta\delta_j(j = 1, \ldots . p)$ is compared for each hidden unit Z_j.

13. ***How Backpropagation Algorithm Works***

- Inputs X, arrive through the preconnected path
- Input is modeled using real weights W. The weights are usually randomly selected.

- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- Calculate the error in the outputs
  - $Error_B$ = Actual Output – Desired Output
- Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

14. *List the Types of Backpropagation*

There are two types of Backpropagation which are as follows –

Static Back Propagation – In this type of backpropagation, the static output is created because of the mapping of static input. It is used to resolve static classification problems like optical character recognition.

Recurrent Backpropagation – The Recurrent Propagation is directed forward or directed until a specific determined value or threshold value is acquired. After the certain value, the error is evaluated and propagated backward.

15. *What is Unit saturation (vanishing gradient problem)*

The vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks.

The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network. Since the gradients control how much the network learns during training, if the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

16. *Define ReLU*

The rectified linear activation unit, or ReLU, is one of the few landmarks in the deep learning revolution. It's simple, yet it's far superior to previous activation functions like sigmoid or tanh.

ReLU formula is : $f(x) = \max(0, x)$

17. *State Hyperparameter tuning*

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters. However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

18. *What is Randomized Search CV?*

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

19. *Clarify the concept Regularization in Machine Learning?*

Regularization is an application of Occam's Razor. It is one of the key concepts in Machine learning as it helps choose a simple model rather than a complex one. We want our model to perform well both on the train and the new unseen data, meaning the model must have the ability to be generalized. Generalization error is "a measure of how accurately an algorithm can predict outcome values for previously unseen data."

Regularization refers to the modifications that can be made to a learning algorithm that helps to reduce this generalization error and not the training error. It reduces by ignoring the less important features. It also helps prevent overfitting, making the model more robust and decreasing the complexity of a model.

20. *Define Dropout.*

"Dropout" in machine learning refers to the process of randomly ignoring certain nodes in a layer during training. In the figure below, the neural network on the left represents a typical neural network where all units are activated. On the right, the red units have been dropped out of the model — the values of their weights and biases are not considered during training.

21. *How Does Dropout Work?*

When we apply dropout to a neural network, we're creating a "thinned" network with unique combinations of the units in the hidden layers being dropped randomly at different points in time during training. Each time the gradient of our model is updated, we generate a new thinned neural network

with different units dropped based on a probability hyperparameter p. Training a network using dropout can thus be viewed as training loads of different thinned neural networks and merging them into one network that picks up the key properties of each thinned network. This process allows dropout to reduce the overfitting of models on training data.

22. ***What is the Downside of Dropout?***

Although dropout is clearly a highly effective tool, it comes with certain drawbacks. A network with dropout can take 2 – 3 times longer to train than a standard network. One way to attain the benefits of dropout without slowing down training is by finding a regularizer that is essentially equivalent to a dropout layer. For linear regression, this regularizer has been proven to be a modified form of L_2 regularization.

PART-B & C

1. Explain the Basic Components of Perceptron.
2. Demonstrate the Multi-Layered Perceptron Model:
3. Elaborate Activation functions in detail.
4. Clarify how to train a neural network.
5. Explain Stochastic gradient descent.
6. What is back propogation? How Backpropagation Algorithm Works?
7. Write in detail the vanishing gradient problem
8. Discuss on ReLU.
9. Demonstrate the strategies for Hyperparameter tuning.
10. Explain the Regularization Techniques in Machine Learning.
11. How Does Dropout Work? Discuss.

****

MODEL QUESTION PAPERS

Model Question Paper - 1

B.E./B.Tech DEGREE EXAMINATION.,

Fourth Semester

CS3491 – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

(Regulations 2021)

Time: Three Hours **Maximum: 100 Marks**

Answer ALL Questions

PART – A (10 × 2 = 20 Marks)

1. ***What is A.I ?***

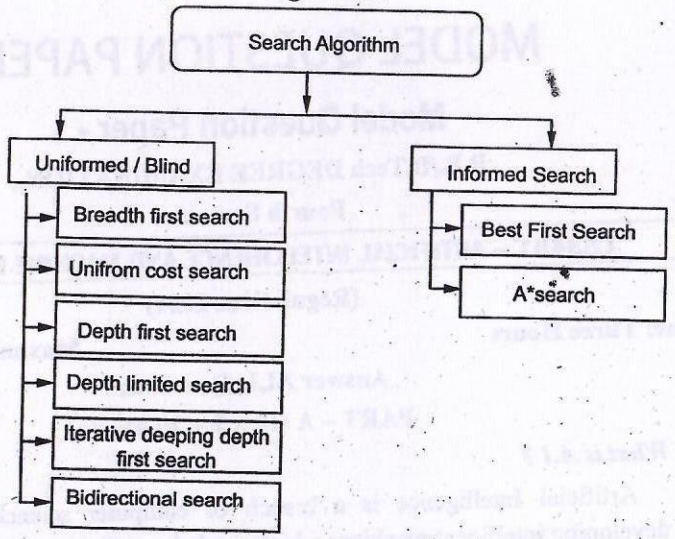
Artificial Intelligence is a branch of computer science that deals with developing intelligent machines which can behave like human, think like human, and has ability to take decisions by their own.

Artificial Intelligence is a combination of two words Artificial and Intelligence, which refers to man-made intelligence. Therefore, when machines are equipped with man-made intelligence to perform intelligent tasks similar to humans, it is known as Artificial Intelligence.

2. ***Different Between Super A.I & Weak A.I?***

S.No	Weak AI	Super AI
1	Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence.	Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI.
2	Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.	Some key characteristics of strong AI include capability include the ability to think, to reason, solve the puzzle, make judgments, plan, learn, and communicate by its own.

3. ***What are the Types of Search Algorithm?***



4. ***What is Probabilistic reasoning?***

- Probabilistic reasoning is a form of knowledge representation in which the concept of probability is used to indicate the degree of uncertainty in knowledge. In AI, probabilistic models are used to examine data using statistical codes.
- Probabilistic reasoning is using logic and probability to handle uncertain situations. An example of probabilistic reasoning is using past situations and statistics to predict an outcome.

5. ***How does Gradient Descent work?***

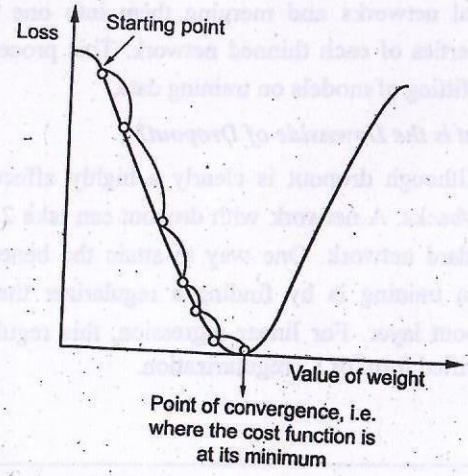
Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

$$Y = mX + c$$

Where '*m*' represents the slope of the line, and '*c*' represents the intercepts on the y-axis.

The starting point (shown in below figure) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate

the steepness of this slope. Further, this slope will inform the updates to th parameters (weights and bias).



6. ***Type of Generative Models?***

- Naive Bayes
- Hidden Markov Models
- Autoencoder
- Boltzmann Machines
- Variational Autoencoder
- Generative Adversarial Network

7. ***Write applications of Naïve Bayes Classifier?***

- It is used for **Credit Scoring.**
- It is used in **medical data classification.**
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis.**

8. ***What do you mean by Unsupervised learning.***

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These

algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition

9. ***Define a Stochastic Gradient Descent (SGD):***

In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

10. ***Write an algorithm for stochastic gradient descent.***

1. The algorithm starts at a random point by initializing the weights with random values
2. Then it calculates the gradients at that random point
3. Then it moves in the opposite direction of the gradient
4. The process continues to repeat itself until it finds the point of minimum loss

PART - B (5 × 13 = 65 Marks)

11. (*a*) Write about CSP with suitable example?

Ans: Refer Page No.1.36

[OR]

(*b*) Explain the types of Uninformed search algorithms?

Ans: Refer Page No.1.13

12. (*a*) Explain Bayes' theorem with example.

Ans: Refer Page No.2.5

[OR]

(*b*) Illustrate causal network with neat diagram.

Ans: Refer Page No.2.22

13. (*a*) Write in detail about Generative Modelling?

Ans: Refer Page No.3.46

[OR]

(*b*) Explain in detail about Naïve Bayes Classifier Algorithm?

Ans: Refer Page No.3.47

14. (*a*) Detail the Steps to create k-means clusters.

Ans: Refer Page No.4.11

[OR]

(*b*) Demonstrate Gaussian Mixture Model (GMM).

Ans: Refer Page No.4.17

15. (*a*) Explain the Regularization Techniques in Machine Learning.

Ans: Refer Page No.5.35

[OR]

(*b*) How Does Dropout Work?

Ans: Refer Page No.5.37

PART - C (1 × 15 = 15 Marks)

16. (*a*) Solve the Cryptarithmetic problem CSP – SEND + MORE = MONEY.?

Ans: Refer Page No. 1.38

[OR]

(*b*) Explain Bayes' theorem with example.

Ans: Refer Page No.2.10

****

Model Question Paper - 2

B.E./B.Tech DEGREE EXAMINATION.,

Fourth Semester

CS3491 – ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

(Regulations 2021)

Time: Three Hours **Maximum: 100 Marks**

Answer ALL Questions

PART – A (10 × 2 = 20 Marks)

1. ***List Some Applictaions of A.I?***

❖ **Game Playing:**

AI is widely used in Gaming. Different strategic games such as Chess, where the machine needs to think logically, and video games to provide real-time experiences use Artificial Intelligence.

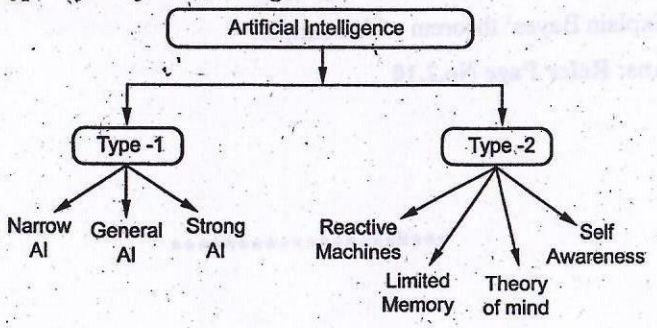
❖ **Robotics:**

Artificial Intelligence is commonly used in the field of Robotics to develop intelligent robots. AI implemented robots use real-time updates to sense any obstacle in their path and can change the path instantly. AI robots can be used for carrying goods in hospitals and industries and can also be used for other different purposes.

❖ **Healthcare:**

In the healthcare sector, AI has diverse uses. In this field, AI can be used to detect diseases and cancer cells. It also helps in finding new drugs with the use of historical data and medical intelligence.

2. ***List the types of Artificial Intelligence?***



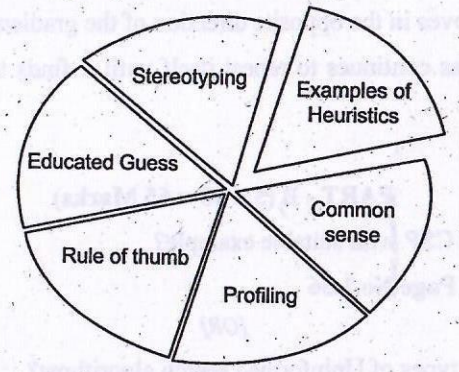
3. ***What is Blind Search and it types?***

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- ❖ Breadth-first search
- ❖ Uniform cost search
- ❖ Depth-first search
- ❖ Iterative deepening depth-first search
- ❖ Bidirectional Search

4. ***Give an real life example of Heuristics search?***



5. ***Define Bayesian Learning.***

It calculates the probability of each hypotheses, given the data and makes predictions on that basis, (i.e.) predictions are made by using all the hypotheses, weighted by their probabilities rather than by using just single "best" hypotheses.

6. ***Define Naïve Bayes model.***

In this model, the "class" variable C is the root and the "attribute" variable XI are the leaves. This model assumes that the attributes are conditionally independent of each other, given the class.

7. ***Write* simple linear regression?**

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- Y represents the output or dependent variable.
- β_0 and β_1 are two unknown constants that represent the intercept and coefficient (slope) respectively.
- ε (Epsilon) is the error term.

8. ***What is meant by Expectation Maximization (EM) intuition?***

The Expectation-Maximization algorithm is performed exactly the same way. In fact, the optimization procedure we describe above for GMMs is a specific implementation of the EM algorithm. The EM algorithm is just more generally and formally defined (as it can be applied to many other optimization problems).

So the general idea is that we are trying to maximize a likelihood (and more frequently a log-likelihood), that is, we are trying to solve the following optimization problem:

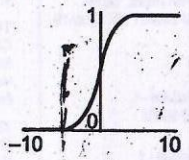
$$\max_{\Theta} \log P(X \mid \Theta) = \max_{\Theta} \log \left[\prod_i P(x_i \mid \Theta) \right] = \max_{\Theta} \sum_i \log (P(x_i \mid \Theta))$$

9. ***Compare the types of Activation Functions.***

Activation Functions

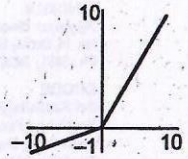
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



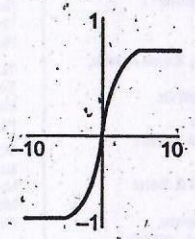
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

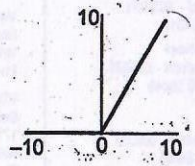


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

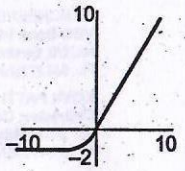
ReLU

$\max(0,x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



10. ***How do you train a neural network?***

In the process of training, we want to start with a bad performing neural network and wind up with network with high accuracy. In terms of loss function, we want our loss function to much lower in the end of training. Improving the network is possible, because we can change its function by adjusting weights. We want to find another function that performs better than the initial one.

PART - B (5 × 13 = 65 Marks)

11. (*a*) Explain Hill Climbing Search algorithm.

Ans: Refer Page No.1.29

[OR]

(*b*) What are the application of A.I .

Ans: Refer Page No.1.7

12. (*a*) Explain Working of Naïve Bayes' Classifier with example.

Ans: Refer Page No.2.11

[OR]

(*b*) Explain in detail about approximate inference in Bayesian network.

Ans: Refer Page No.2.21

13. (*a*) Explain in detail about Maximum Margin Classifier?

Ans: Refer Page No.3.52

[OR]

(*b*) Write in detail about Decision tree?

Ans: Refer Page No.3.55

14. (*a*) Compare the Advanced Ensemble techniques.

Ans: Refer Page No.4.4

[OR]

(*b*) Explain the Working of Unsupervised Learning.

Ans: Refer Page No.4.11

15. (*a*) Clarify how to train a neural network.

Ans: Refer Page No.5.11

[OR]

(*b*) Explain Stochastic gradient descent.

Ans: Refer Page No.5.16

PART - C (1 × 15 = 15 Marks)

16. (*a*) Explain the types of Uninformed search and Informed search algorithm?

Ans: Refer Page No.1.13

[OR]

(*b*) Demonstrate Gaussian Mixture Model (GMM).

Ans: Refer Page No.4.17

****

CSE 0811