F.Y. MCA

(TWO YEARS PATTERN)

SEMESTER - II (CBCS)

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

SUBJECT CODE: MCA22

Dr. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai

Dr. Prakash Mahanwar
Director,
IDOL, University of Mumbai

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology IDOL,
Univeristy of Mumbai – 400098

Course Co-ordinator : **Shyam Mohan T.**
Department - MCA, IDOL
University of Mumbai- 400098

Course Writers : **Dr. Ghayathri J**
Associate Professor,
Kongu Arts and Science College (Autonomous)

: **Ms. Jayalalita**
Assistant Professor,
NES Ratnam College of Arts, Science and Commerce

: **Dr. R Jayakarthik, M.Sc., M.Phil., Ph.D**
Associate Professor,
Department of Computer Science
Saveetha College of Liberal Arts & Sciences
SIMATS Deemed to be University
Saveetha Nagar,Thandalam, Chennai

: **Mr.Sandeep Kamble**
Assistant Professor,
Cosmopolitan's Valia College

: **Ms. Hema Darne**
Assistant Professor,
Dr. Moonje Institute of Management and Computer Studies, Nashik

: **Dr. D.S.Rao**
Professor
(CSE) & Associate Dean(Student Affairs)
Koneru Lakshmaiah Education Foundation.
KL H (Deemed to be University), Hyderabad.

: **Dr. Ruchi Gupta**
Assistant Professor,
Rizvi college of Arts, Science and Commerce

May 2022, Print - 1

Published by : Director,
Institute of Distance and Open Learning,
University of Mumbai,
Vidyanagari,Mumbai - 400 098.

DTP composed and Printed by:
Mumbai University Press
Vidyanagari, Santacruz (E), Mumbai- 400098

CONTENTS

F.Y. MCA

(TWO YEARS PATTERN)

SEMESTER - II (CBCS)

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

SYLLABUS

Module	Detail Content	Hrs
1	**Module1:Introduction:** Artificial Intelligence, Application of AI, AI Problems, Problem Formulation, Intelligent Agents, Types of Agents, Agent Environments, PEAS representation for an Agent, Architecture of Intelligent agents. Reasoning and Logic, Prepositional logic, First order logic, Using First-order logic, Inference in First-order logic, forward and Backward Chaining **Self-Learning topics: Expert systems**	6
2	**Module2: Search Strategies:** Solving problems by searching, Search- Issues in The Design of Search Programs, Un-Informed Search- BFS, DFS; Heuristic Search Techniques: Generate-And-Test, Hill Climbing, Best-First Search, A* Algorithm, Alpha beta search algorithm, Problem Reduction, AO*Algorithm, Constraint Satisfaction, Means-Ends Analysis **Self-Learning topics: Tabu search**	8
3	**Module3:Artificial Neural Networks** : Introduction, Activation Function, Optimization algorithm- Gradient decent, Networks- Perceptrons, Adaline, Multilayer Perceptrons , Backpropogation Algorithms Training Procedures, Tuning the Network Size **Self-Learning topics: Maxnet algorithm**	6

4	**Module4: Introduction to ML**: Machine Learning basics, Applications of ML,Data Mining Vs Machine Learning vs Big Data Analytics. Supervised Learning- Naïve Base Classifier, , Classifying with k-Nearest Neighbour classifier, Decision Tree classifier, Naive Bayes classifier. Unsupervised Learning - Grouping unlabeled items using k-means clustering, Association analysis with the Apriori algorithm Introduction to reinforcement learning **Self-Learning topics: Density Based Clustering,K-medoid**	4
5	**Module5:Forecasting and Learning Theory :** Non-linear regression, Logistic regression, Random forest, Baysian Belief networks, Bias/variance tradeoff, Tuning Model Complexity, Model Selection Dilemma Clustering **:** Expectation-Maximization Algorithm, Hierarchical Clustering, Supervised Learning after Clustering, Choosing the number of clusters, Learning using ANN **Self-Learning topics: Maximum Likelihood Estimation**	6
6	**Module6:Kernel Machines & Ensemble Methods** Introduction, Optimal Separating Hyperplane, Separating data with maximum margin, Support Vector Machine (SVM), Finding the maximum margin, The Non-Separable Case: Soft Margin Hyperplane, Kernel Trick, Defining Kernels Ensemble Methods : Mixture Models, Classifier using multiple samples of the data set, Improving classifier by focusing on error, weak learner with a decision stump, Bagging , Stacking, Boosting ,Implementing the AdaBoost algorithm, Classifying with AdaBoostBootstrapping and cross validation **Self-Learning topics: SMO Algorithm**	8
7	**Module7:Dimensionality Reduction: Introduction, Subset Selection, Principal Components Analysis, Multidimensional Scaling, Linear Discriminant Analysis.** **Self-Learning topics; Feature selection – feature ranking and subset selection**	2

1

INTRODUCTION

Unit Structure

1.0 OBJECTIVES

- To understand the AI foundations and applications.
- To describe the Problem types and characteristics of AI problems.
- To acquire knowledge about agents and its types.
- To Study agent environment and architecture.
- To Gain Knowledge about the performance measures of AI agents.

1.1 INTRODUCTION

We, human-beings are having mental capacity to understand the physical world around us. As per the intelligence of each person he/she understand the environment and act according to that. Artificial Intelligence is

developed to build intelligent entities (Russell et. al. 2015) and to understand the real world problems as well. It is one of the latest modern technologies and started its march from 1956 when the name 'Artificial Intelligence' was coined by John McCarthy, an American Scientist.

Nowadays, AI is one of the essential technologies for the human life style. Artificial Intelligence spreads its wings over different fields from general-purpose areas, such as learning and perception to such specific tasks as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. A1 is potentially supportive to the human to work on intellectual tasks and make them systematic. Hence, AI makes human life more ease.

1.2 AN OVERVIEW

1.2.1 What is Artificial Intelligence?

Artificial Intelligence can be defined based on thought process, human performance, ideal performance as well as behaviour of the system. The following are the definitions and the example systems for the various types of AI.

i) Systems that think like humans

"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 . . ." (Bellman, 1978)

ii) Systems that act like humans

"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990)

"The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)

iii) Systems that think rationally:

"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)

iv) Systems that act rationally

"Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)

"A1 . . .is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

1.2.2 Foundations of AI

Some of the important disciplines that contributed ideas, viewpoints, and techniques to AI are Philosophy, Mathematics, Economics, Neuroscience, Psychology, Computer Engineering, Cybernetics Linguistics and Sociology

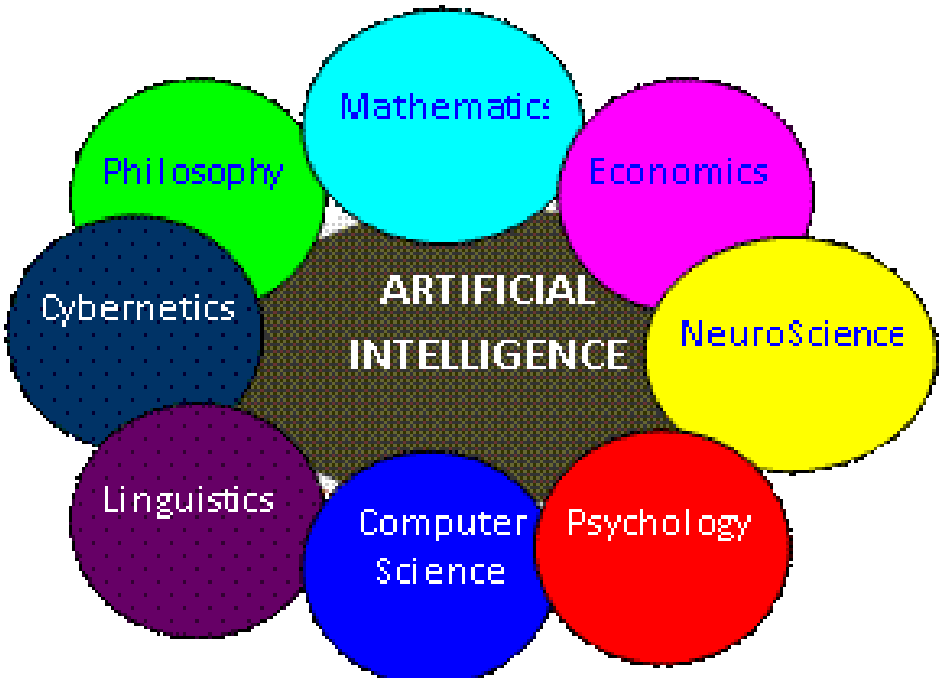


Fig. 1.2.1 Foundations of AI

Philosophy :

Can formal rules be used to draw valid conclusions?
How does the mental mind arise from a physical brain?
Where does knowledge come from?
How does knowledge lead to action?

Mathematics:

What are the formal rules to draw valid conclusions?
What can be computed?
How do we reason with uncertain information?

Economics :

How should we make decisions so as to maximize payoff?
How should we do this when others may not go along?
How should we do this when the payoff may be fix in the future?

Neuroscience

How do brains process information?

Psychology

How do humans and animals think and act?

Computer engineering

How can we build an efficient computer?

Control theory and Cybernetics

How can artifacts operate under their own control?

Linguistics

How does language relate to thought?

1.2.3 History of AI

The first work that is now generally recognized as A1 was done by Warren McCulloch and Walter Pitts (1943).

Donald Hebb (1949) demonstrated a simple updating rule for modifying the connection strengths between neurons. His rule, now called **Hebbian learning,** remains an influential model to this day

In 1956, American computer scientist John McCarthy organised the Dartmouth Conference, at which the term 'Artificial Intelligence' was first adopted

In 1951, a machine known as Ferranti Mark 1 successfully used an algorithm to master checkers. Subsequently, Newell and Simon developed General Problem Solver algorithm to solve mathematical problems. Also in the 50s John McCarthy, often known as the father of AI, developed the LISP programming language which became important in machine learning.

In the late 1960s, computer scientists worked on Machine Vision Learning and developing machine learning in robots. WABOT-1, the first 'intelligent' humanoid robot, was built in Japan in 1972.

From the mid 1970s to the mid 1990s, computer scientists dealt with an acute shortage of funding for AI research. These years became known as the 'AI Winters'.

In 1997, IBM's Deep Blue defeated became the first computer to beat a reigning world chess champion, Garry Kasparov.

In the past 20 years, Amazon, Google, Baidu, and others leveraged machine learning to their huge commercial advantage.

1.3 APPLICATIONS OF AI

AI has applications in all fields of human study, such as finance and economics, environmental engineering, chemistry, computer science, and so on. Some of the applications of AI are listed and illustrated below:

- Perception
- Machine vision
- Speech understanding

- Touch sensation
- Robotics
- Natural Language Processing
- Natural Language Understanding
- Speech Understanding
- Language Generation
- Machine Translation
- Autonomous Planning and scheduling
- Expert Systems
- Machine Learning
- Theorem Proving
- Symbolic Mathematics
- Game Playing

Autonomous planning and scheduling:

A hundred million miles from Earth, program became the first on-board autonomous NASA's Remote Agent planning program control the scheduling of operations for a spacecraft (Jonsson et al., 2000).

Game playing:

IBM's Deep Blue computer program to defeat the world champion Garry Kasparov in a chess match(Goodman and Keene, 1997).

Diagnosis:

Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician in several areas of medicine.

Logistics Planning:

During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation.The AI planning techniques allows a plan to be generated in hours that would have taken weeks with older methods.

Robotics:

Eases the human work in plenty of areas which requires more efficiency and which are more hard and dangerous.

Machine Translation:

Traffic prediction, Image recognition, online fraud detection, virtual personal assistants are some of the applications supported by machine learning in this todays' online/internet world.

- In addition to the above said applications Web agents, Personal desktop agent, recommendersystems,e-commerce agent use the artificial intelligence techniques to solve the problems and for seamless, effective implementation

1.4 AI PROBLEMS

1.4.1 Problem Definitions

A problem is defined by its elements and their relations. To provide a formal description of a problem, we need to do following: A formal problem should

- Define a state space that contains all the possible configurations of the relevant objects, including some impossible ones.
- Specify one or more states that describe possible situations, from which the problem-solving process may start. These states are called initial states.
- Specify one or more states that would be acceptable solution to the problem. These states are called goal states.
- Specify a set of rules that describe the actions (operators) available. The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found.

1.4.2 Problem Space

A directed graph represents problem space. The nodes of graph represent search state and paths represent the operators applied to change the state To reduce complexity problem space is represented as a tree.

A tree usually decreases the complexity of a search at a cost. Duplication of the nodes increases the cost in graph which is not so in tree.

- A tree is a graph in which any two vertices are connected by exactly one path.
- Connected graph with no cycles is a tree.

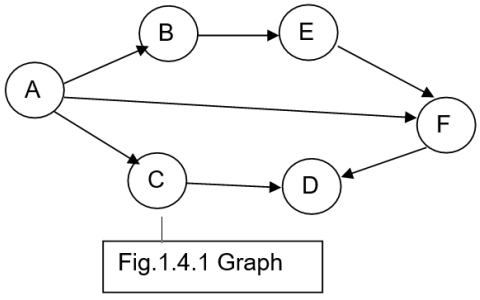


Fig.1.4.1 Graph

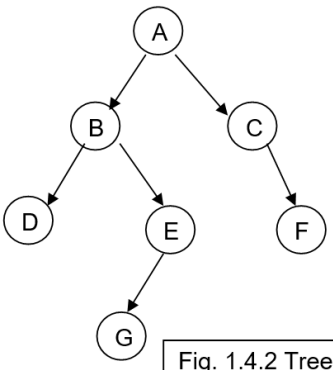


Fig. 1.4.2 Tree

States

A state is a representation of elements at a given moment.

- A problem is defined by its elements and their relations.
- At each instant of a problem, the elements have specific descriptors and relations;

- the descriptors give the information about how to select elements from the all possible states. The special states are Initial state and final (goal) state

State Change:

The successor function moves one state to another state.

Successor Function:

It is a description of possible actions; a set of operators.
Is a transformation function on a state representation, which converts that state into another state?
Represents the conditions of applicability of a state and corresponding transformation function

State Space

A State space is the set of all states reachable from the initial state.

Definitions of terms:

A state space forms a graph (or map) in which the nodes are states and the arcs between nodes are actions.

In state space, a path is a sequence of states connected by a sequence of actions. The solution of a problem is part of the map formed by the state space.

1.4.3 Problem Characteristics

To select appropriate problem solving method, first the problem has to be analysed in various dimensions. These dimensions are referred to as problem characteristics. They are as follows:

1. **Is the problem decomposable into a set of independent smaller or easier sub-problems?**

   A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used.

   symbolic integration: Decomposable Problem

   $\int x^2+x+sin^2 x cos^2 x dx$

   This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results we can find the complete solution.

   Blocks world problem: Non-Decomposable problem

   Start and goal state are given as,

   Solution: Achieved by moving blocks in a sequence to reach goal state

   Solution steps: Interdependent and cannot be decomposed in sub problems.

These two examples, symbolic integration and the blocks world illustrate the difference between decomposable and non-decomposable problems.

2. Can solution steps be ignored or at least undone if they prove unwise?

Three categories of problems:

(i) ignorable, (ii) recoverable and (iii) Irrecoverable. This classification is with reference to the steps of the solution to a problem.

Theorem proving – Ignorable

We may later find that it is of no use. We can still proceed further, since nothing is lost by this redundant step.

This is an example of ignorable solutions steps.

problems can be solved using a simple control structure that never backtracks

8 puzzle problem - Recoverable

While moving from the start state towards goal state, we may make some stupid move but we can backtrack and undo the unwanted move.

This only involves additional steps and the solution steps are recoverable.

problems can be solved by a slightly more complicated control strategy that allows backtracking

Chess Game - Irrecoverable

- If a wrong move is made, it can neither be ignored nor be recovered.
- The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps.
- problems will need to be solved by a system that expends a great deal of effort making each decision

3. Is the problem's universe predictable?

Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards).

Certain outcome problems

- Planning could be done to generate a sequence of operators that guarantees to lead to a solution.

- Planning helps to avoid unwanted solution steps.
- Example: Water jug problem

Uncertain outcome problems

- Problems do not guarantee a solution
- Very expensive since the number of solution paths to be explored increases exponentially
- Example: Playing cards

4. Is a good solution to the problem obvious without comparison to all other possible solutions?

There are two categories of problems

Any path problem and Best path problem.

Any path problem : we are satisfied with the solution, irrespective of the solution path taken.

Example : water jug and 8 puzzle problems

Best path problem

Any solution is acceptable but we want the best path solution.

In any –path problems, by heuristic methods we obtain a solution and we do not explore alternatives.

Traveling sales man problem – shortest path problem

5. Is the desired solution a state of the world or a path to a state?

State of the world- Solution:

- Finding the interpretation but not the record of the processing by which the interpretation is found.
- Example: The problem of natural language processing. Finding a consistent interpretation for the sentence "The bank president ate a dish of pasta salad with the fork"

A path to a state – Solution:

- It is not sufficient to report that we have solved, but the path that we found to the state
- The statement of a solution to this problem must be a sequence of operations that produces the final state.
- Example: Water jug problem

6. What is the role of knowledge?

- The size of the knowledge base available for solving the problem does matter in arriving at a good solution.

- Take for example the game of playing chess, just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution.
- Additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program

The solution would then be realistic.

7. Does the task require interaction with a person?

The problems can again be categorized under two heads.

Solitary

- Computer will be given a problem description and will produce an answer
- No intermediate communication and with the demand for an explanation of the reasoning process.

Example: Simple theorem proving

Conversational,

- There will be intermediate communication between a person and the computer
- Provides additional assistance to the computer or additional information to the user, or both.

Example: Medical diagnosis

1.4.4 Production system

- It provides appropriate structures for performing and describing search processes.
- A production system has four basic components such as:
- A set of rules with left side determines the applicability of the rule and a right side describes the operation to be performed if the rule is applied.
- A database of current facts established during the process of inference.
- A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.

A rule firing module

The production rules operate on the knowledge database.
Each rule has a precondition—that is, either satisfied or not by the knowledge database.
If the precondition is satisfied, the rule can be applied.

Application of the rule changes the knowledge database.

The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the knowledge database is satisfied.

Example: Eight puzzle (8-Puzzle) The 8-puzzle is a 3 × 3 array containing eight square pieces, numbered 1 through 8, and one empty space. A piece can be moved horizontally or vertically into the empty space, in effect exchanging the positions of the piece and the empty space. There are four possible moves, UP (move the blank space up), DOWN, LEFT and RIGHT. The aim of the game is to make a sequence of moves that will convert the board from the start state into the goal state:

1	2	3
8	X	4
7	6	5

Table 1.4.1 Initial State

2	8	3
1	6	4
7	X	5

Table 1.4.2 Goal State

Operator sequence UP, UP, LEFT, DOWN, RIGHT

Example: Missionaries and Cannibals

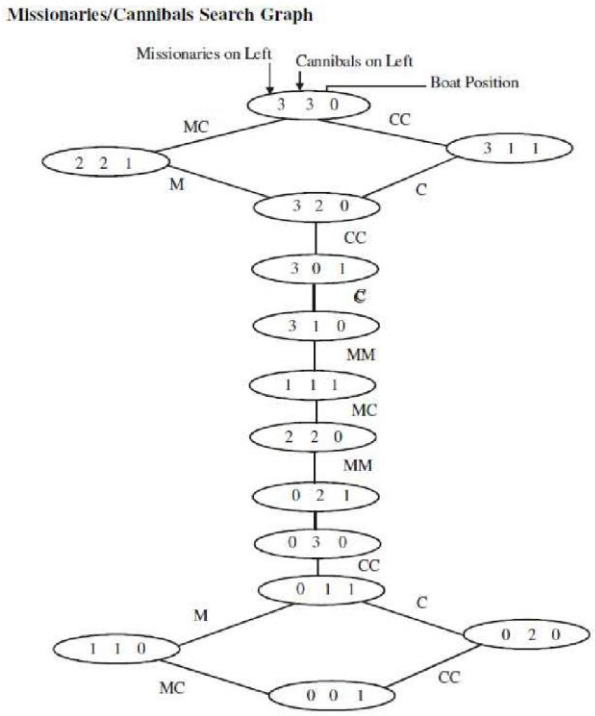
- This problem uses state space search for planning under constraints.
- Three missionaries and three cannibals wish to cross a river using a two person boat.
- If at any time the cannibals outnumber the missionaries on either side of the river, they will eat the missionaries.
- The boat should have at least one person while rowing.
- Boat trips should be performed based on without losing any missionaries everyone should cross the river.

State representation:

1. BOAT position: original (B) or final (Nil) side of the river.
2. Number of Missionaries and Cannibals on the original side of the river.
3. Start is (B 3 3).
4. Goal is (Nil 0 0)

Operators:

(MM 2 0)	**Two missionaries cross the river**
(MC 1 1)	**One missionary and one cannibal cross the river**
(CC 0 2)	**Two cannibals cross the river**
(M 1 0)	**One missionary cross the river**
(C 0 1)	**One cannibal cross the river**



1.5 INTELLIGENT AGENTS

1.5.1 **Agent Definition**

An agent perceives its environment through sensors and acting upon the environment through actuators to achieve the goals. In addition, an intelligent agent is capable of learning from the environment.

Example :

Intelligent Agent: *Thermostat* – It regulates the room temperature automatically.

The common procedure to be followed by the agents is:

1. An AI agent must have the ability to perceive the environment.
2. The observation must be used to make decisions.
3. Decision should result in an action.
4. The action taken by an AI agent must be a rational action.

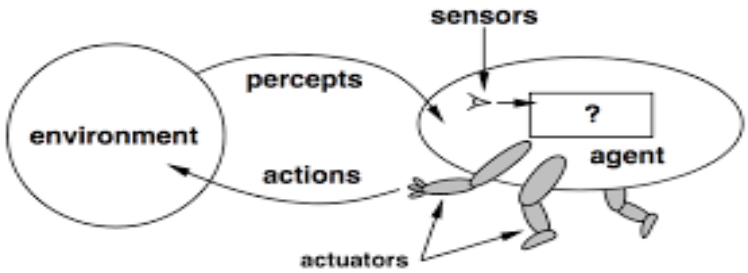


Fig. 1.5.1

Example: Vacuum-Agent
It percepts the location and status
Example :
[location, status]; location may be A or B; status: dirty =1;clean =0;
Actions: Left, Right, Suck, NoOp
function VacAgent([location,status])
returns an action
Actions:
if status = 1 then return Suck
else if location = A then return Right
else if location = B then return Left

Categories of Agent:

- Human agent
- Robotic agent
- Software agent

Human Agent :

People are intelligent agents with the sensors eyes, ears, nose, skin and mouth. The hands, legs, mouth, and other body parts are actuators

Robotic agent :

Robots use cameras and infrared range finders for sensors and various motors for actuators

Software agent :

Software agents consider keystrokes, file contents, received network packages as sensors and the displays on the screen, files, sent network packets as actuators

1.5.2 Problem Formulation

Problem formulation aims to reach the goal by deciding actions to be carried out and status to be considered. . For example, if the agent were to consider the action to be at the level of "move the left foot by one inch" or "turn the steering wheel by 1 degree left", there would be too many steps for the agent to leave the parking lot, let alone to Bucharest. In general, we need to abstract the state details from the representation.

1. The **initial state** of the agent. In this case, the initial state can be described as *In: Arad*
2. The possible **actions** available to the agent, corresponding to each of the state the agent resides in. For example, ACTIONS(*In: Arad*) = {*Go: Sibiu, Go: Timisoara, Go: Zerind*}
3. The **transition model** describing what each action does. Let us represent it by RESULT(s, a) where s is the state the action is currently in and a is the action performed by the agent. In this example, RESULT(*In: Arad, Go: Zerind*) = *In: Zerind.*
4. The **goal test**, determining whether the current state is a goal state. Here, the goal state is {*In: Bucharest*}
5. The **path cost** function, which determines the cost of each path, which is reflecting in the performance measure. For the agent trying to reach Bucharest, time is essential, so we can set the cost function to be the distance between the places. By convention, we define the cost function as *c(s, a, s')*, where s is the current state and a is the action performed by the agent to reach state s'.

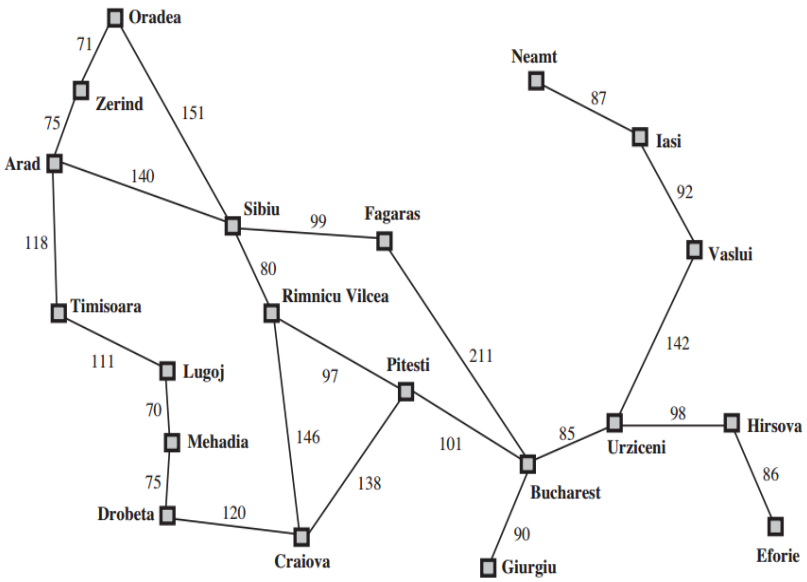


Figure 1.5.2. : A simplified road map of part of Romania.

The problem is to travel from Arad to Bucharest in a day. For the agent, the goal will be to reach Bucharest the following day. Courses of action that doesn't make agent to reach Bucharest on time can be rejected without further consideration, making the agent's decision problem simplified.

The initial state, the actions and the transition model together define the **state space** of the problem — the set of all states reachable by any sequence of actions. Figure 1.5.2 is the graphical representation of the state space of the traveling problem. A **path** in the state space is a sequence of states connected by a sequence of actions.

The **solution** to the given problem is defined as the sequence of actions from the initial state to the goal states. The quality of the solution is measured by the cost function of the path, and an **optimal solution** has the lowest path cost among all the solutions.

1.5.3 Types of Agents

Agents can be grouped into five categories based on their degree of perceived intelligence and capability.

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agents

Simple Reflex Agents (SRA)

Simplest agents.

These agents take decisions on the basis of the current percepts

Ignore the past states

Succeed only in the fully observable environment.

Works on Condition-action rule which maps the current state to action.

Such as a Room Cleaner agent, it works only if there is dirt in the room.

Demerits

They have very limited intelligence

They do not have knowledge of non-perceptual parts of the current state

Too big to generate and store.

Not adaptive to changes in the environment.

```
Function Simple-Reflex-Agent(percept) returns
action
static: rules, a set of condition-action rules
state ← Interpret-Input(percept)
rule ← Rule-Match(state, rules)
action ← Rule-Action[rule]
```

Ex: if car-in-front-is-braking then initiate- braking

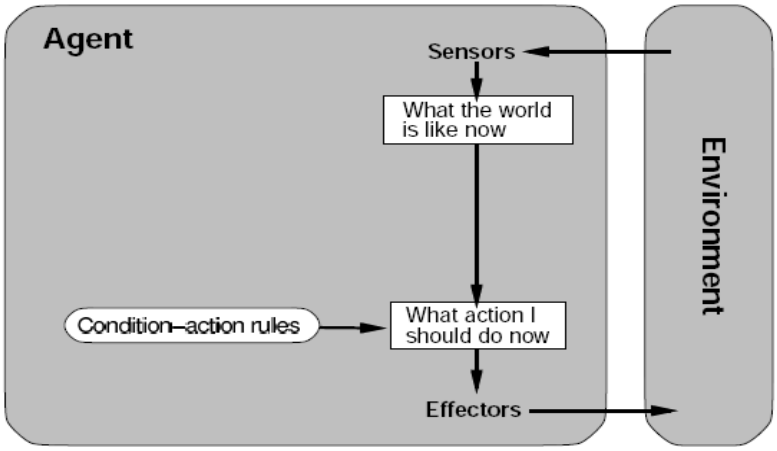


Fig. 1.5.3. Simple Reflex Agents

Model Based Reflex Agents (MBRA)

These agents work in a partially observable environment, and track the situation.

Two important factors

Model: Knowledge about "how things happen in the world,"

Internal State: Representation of the current state based on percept history.

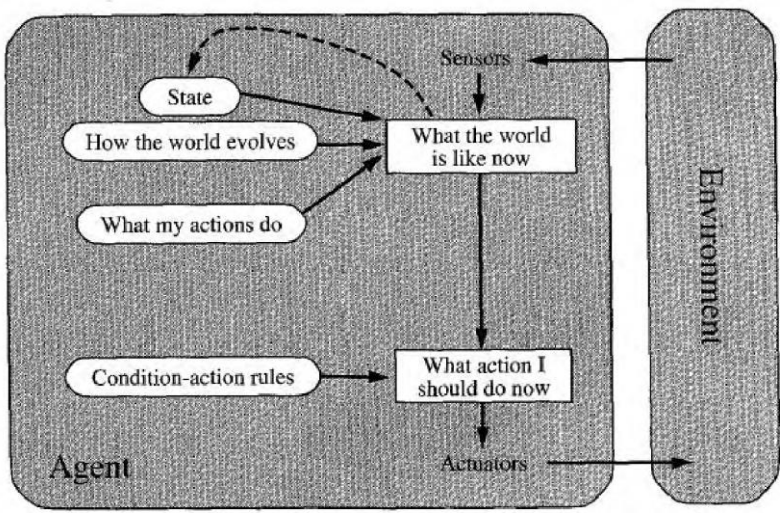


Fig 1.5.4 A model-based reflex agent

- Actions performed based on the model

Updating the agent state requires information about:

- How the world evolves
- How the agent's action affects the world

Demerits

Not having information about goal state

Goal Based Agents (GBR)

Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.

They choose an action, so that they can achieve the goal.

These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not.

Such considerations of different scenario are called searching and planning, which makes an agent proactive.

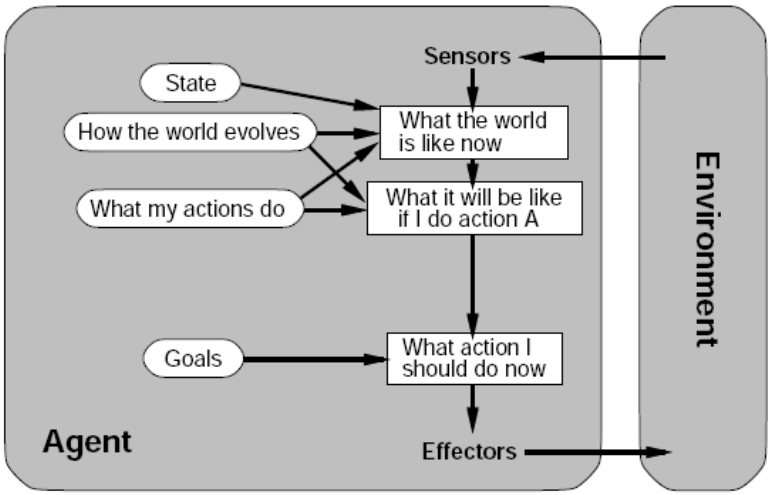


Fig 1.5.5 A model-based, goal-based agent.

Types of Goal-based agents

Goal based agents consider the long-term actions and the desirability of the outcome, which is easier to train and is adaptable to the changing environment.

There are two kinds of goal-based agents:

(i) Problem-solving agents

(ii) Planning agents.

i) Problem-solving agents

This agent considers each states of the world as indivisible.

No internal structure of the states visible to the problem-solving algorithms.

ii) Planning agents

This type of agents split up each state into variables and establishes relationship between them.

Utility Based Agents (UBA)

- These agents are similar to the goal-based agent
- Provides an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Act based on the best way to achieve the goal.
- These are useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- This agent maps each state to a real number to check how efficiently each action achieves the goals.

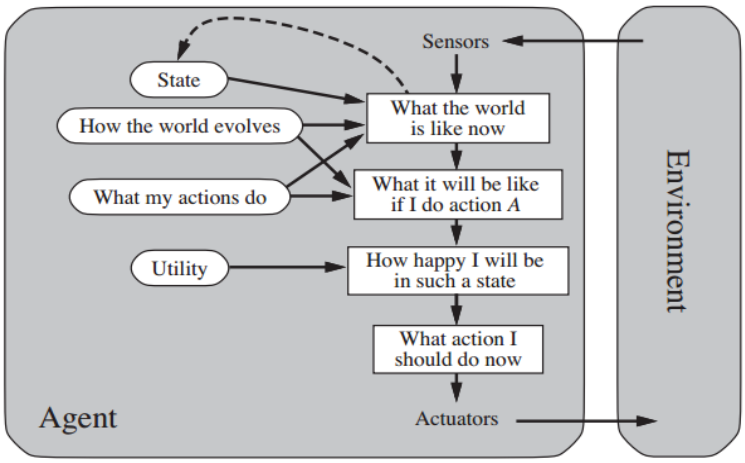


Fig 1.5.6 A utility-based agent.

Learning Agents (LA)

A learning agent can learn from its past experience. It has the learning capabilities.

The actions are performed using basic knowledge and then the actions are further updated through automatic learning.

Components of a learning agent:

a. **Learning element:** Makes improvements by learning from environment

b. **Critic:** Takes feedback from critic to find and fix the performance standard.

c. **Performance element:** Selects the external actions

d. **Problem generator:** Suggests actions that will lead to new and informative experiences.

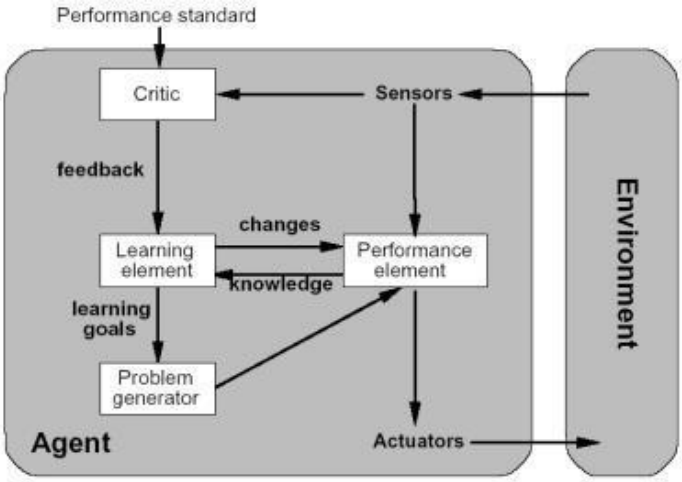


Fig 1.5.7 Learning agent.

1.6 AGENT ENVIRONMENTS

Agent

Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators as shown in fig.1.6.1.

- The perceptual input of the agent is 'Percept' at any given instant and the perceived history is known as 'Percept sequence'.
- An agent's choice of action at any given instant can depend on the entire percept sequence observed to date
- An agent's behavior is described by the agent function which maps from percept histories to actions:

[$f: P^* \rightarrow A$]

We can imagine tabulating the agent function that describes any given agent (External characterization)

Internally, the agent function will be implemented by an agent program which runs on the physical architecture to produce f

agent = architecture + program

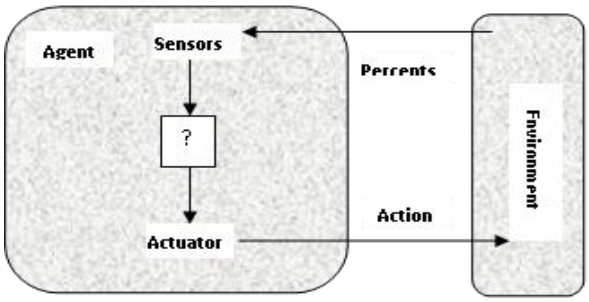


Figure 1.6.1 Agents interact with environments through sensors and actuators.

Example Scenario:

- A particular world has just two locations: squares P1 and P2, shown in Fig. 1.6.2.

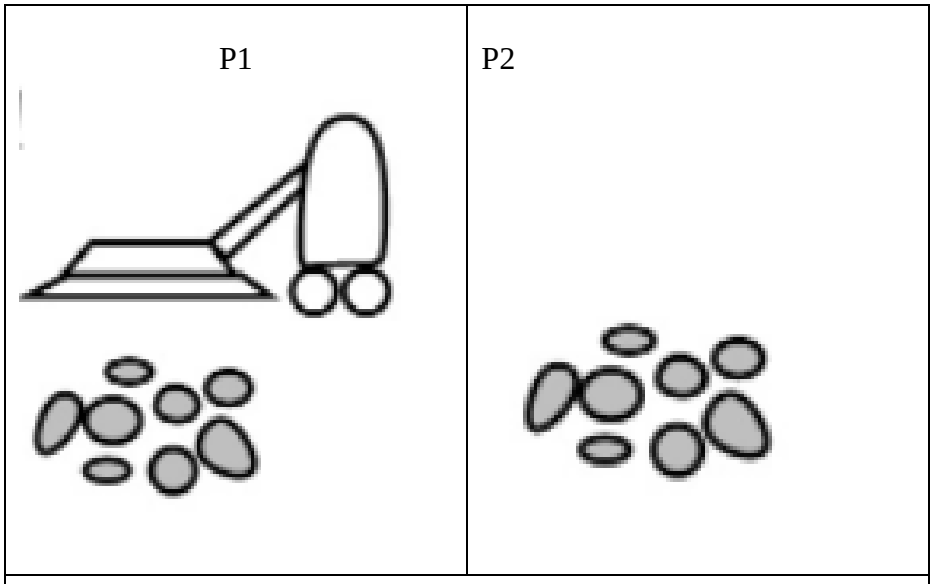


Fig. 1.6.2 A vacuum-cleaner World

- The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing.

The agent function checks if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in Figure 1.6.3

Percept Sequence	**Action**
[P1, clean] [P2,clean] [P1,dirty] [P2,dirty] [P1, clean], [P1,clean] [P1,clean],[P1,dirty] [P1, clean], [P1,clean], [P1, clean] [P1, clean], [P1,clean], [P1, dirty]	Right Suck Left Suck Right Suck Right Suck

Fig. 1.6.3 A vacuum-cleaner World function tabulation

1.7 PEAS REPRESENTATION FOR AN AGENT

A performance measure represents the criterion for success of an agent's behavior.

PEAS stands for Performance Measures, Environment, Actuators, and Sensors.

Task Envornment : PEAS forms the task environment.

Performance Measure: The objective functions to judge the performance of the agent. For example, Minimizing the trip time, so that cost can be reduced.

Environment: The real environment where the agent has to do actions.

Actuators: Output of the Agents. These are the tools, equipment or organs which are using agent to perform actions in the environment.

Sensors: Input to the Agents. These are tools, of the organs using which, agent captures the state of the environment.

PEAS Descriptor for Automated Car Driver:

Performance Measure: minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort. Automated system should be able to

- Drive the car safely without dashing anywhere.
- Maintain the optimal speed depending upon the surroundings.
- Give a comfortable journey to the end user.

Environment:

Roads: Automated car driver should be able to drive on any kind of a road ranging from city roads to highway.

Traffic conditions: Different sort of traffic conditions for different type of roads.

Actuators:

Steering wheel: used to direct car in desired directions.

Accelerator, gear: To increase or decrease speed of the car.

Sensors: Take input from environment in car driving example cameras, sonar system etc.

The following table 1.7.1 shows the PEAS of some real world agents.

Agent Type	Performance Measure	Environment	Accuators	Sensors
Taxi driver	Safe: fast, legal,	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odomete, accelerometer, engine sensors, key board
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery Controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Table. 1.7.1 Examples of agent types and their PEAS descriptions.

Properties of task environments

Fully observable vs. partially observable

Fully observable : if the sensors detect all aspects that are *relevant* to the choice of action

partially observable : if noisy and inaccurate sensors or parts of the state are missing from the sensor data.

Deterministic vs. stochastic

Deterministic

- If the next state of the environment is completely determined by the current state
- The action executed by the agent

Stochastic : If the environment is deterministic except for the actions of other agents.

Episodic vs. sequential

- The agent's experience is divided into atomic episode
- The current decision could affect all future decisions.

Static vs, dynamic

- If the environment can change while an agent is deliberating, then the environment is dynamic for that agent; otherwise, it is static.

Discrete vs. continuous

- The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent

Single agent vs. multiagent

- Single agent to perform the task or more than one agent to perform the task.

The sample task environment and their characteristics are shown in the Table1.7.2.

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Table 1.7.2 Examples of task environments and their characteristics

1.8 ARCHITECTURE OF INTELLIGENT AGENTS

The agent architectures consist of all the models such as simple reflexive model, Goal based reflexive model, learning model etc. Reflex responses

are needed for situations in which time is of the essence, whereas knowledge-based deliberation allows the agent to plan ahead. A complete agent must be able to do both, using hybrid architecture.

Property of hybrid architectures: The boundaries between different decision components are not fixed.

Compilation: Continually converts declarative information at the deliberative level into more efficient representations, eventually reaching the reflex level.

Example: Agent architectures SOAR (Laird et al., 1987) and THEO (Mitchell, 1990)

The reflexive mechanism is depicted in Fig. 1.8.1

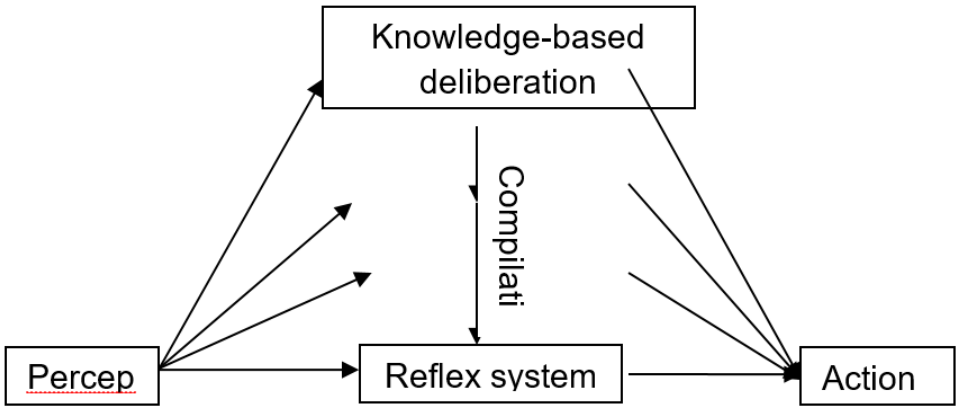


Fig. 1.8.1 Reflexive Mechanism

There are two techniques that work in general decision-making situations i) Anytime algorithm ii) Decision-theoretic meta reasoning

i) Anytime algorithms

In anytime algorithm the output quality improves gradually over time, so that it has a reasonable decision ready whenever it is interrupted. These algorithms are controlled by a meta level decision procedure.

Example: Game playing

ii) Decision-theoretic meta reasoning

This method applies the theory of information value to the selection of computations. The value of a computation depends on both its cost can be used to design better search algorithms and to guarantee that the algorithms have the anytime property.

Meta reasoning is expensive,

Compilation methods can be applied so that the overhead is small compared to the costs of the computations being controlled.

1.9 SUMMARY

The artificial intelligence is defined as*"The exciting new effort to make computers think . . . machines with minds, in the full and literal sense."*. The foundations include the philosophy, Mathematics, Economics, Neuroscience, Psychology etc. These fields based the foundation for the evolution of artificial Intelligence.

Applications of Artificial intelligence from game playing to robotics are explained. To select appropriate problem solving method, first the problem has to be analysed in seven dimensions along with suitable example.

Problem spaces, and production system which provides appropriate structures for performing and describing search processes are explained with illustration

An agent perceives its environment through sensors and acting upon the environment through actuators to achieve the goals. There are five types of agents; Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents, Utility-Based Agents and Learning Agents to perform the automated tasks.

PEAS does the performance measure of the agents.

1.10 REFERENCES

1. Russell, S. and Norvig, P, Artificial Intelligence: A Modern Approach, Third Edition, Prentice- Hall, 2010
2. Artificial Intelligence, Elaine Rich, Kevin Knight, Shivasankar B. Nair, The McGrawHill publications, Third Edition, 2009.
3. George F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Pearson Education, 6th ed., 2009.
4. https://towardsdatascience.com

1.11 UNIT END EXERCISES

1. Define Artificial Intelligence.
2. Explain the impact of the various fields in AI.
3. Elucidate the History of AI
4. What is problem space?
5. Write the characteristics of problems.
6. Describe the production system with suitable examples.
7. What is an agent?
8. What are the components of Agents?
9. Summarize the types of agents.
10. Explain the architecture of intelligent agents.?

❄❄❄❄❄❄❄

2

EXPERT SYSTEMS

Unit Structure

2.0 OBJECTIVES

1. To explore the knowledge of Logic, reasoning and its representation
2. To study about the knowledge based agents
3. To represent sentences in propositional Logic
4. To describe the illustration of first order logic
5. To explain forward and backward reasoning

2.1 INTRODUCTION

Artificial Intelligence is the ability to acquire, understand and apply the knowledge to achieve the given goals. AI opens the path to new technologies and the drastic change and modernization in all the fields like life science, astronomy, medical, social science and business as well. Perhaps we say AI machines and softwares are intelligent, however to make them as such, the specific domain knowledge should be fed into the machine or software. Knowledge representation is compassionate to make

this happen. The propositional and first order logic representation is used to represent the sentences in the form of knowledge.

The automation of the processes are derived through the intelligent agents which entails the AI problem solving and knowledge representation concepts. The propositional and first order logic concepts for intelligent agents are discussed in this chapter.

2.2 REASONING AND LOGIC

The knowledge based processing, logic representation and reasoning types are described in this section**.**

2.2.1 Knowledge based Agents

A knowledge base is a set of sentences comprised of facts.

Knowledge Base (KB), is the essential component of knowledge-based agent.

Knowledge representation language is used to express each sentence.

The assertions about the world are represented by the sentences.

Deriving new sentence from old sentence is known as Inference.

A generic knowledge based agent has corresponding operations and actions as given in Fig. 2.2.1 and Table 1 gives the key components of Procedure KB-AGENT

```
function KB-AGENT(percept) returns an action
static: KB, a knowledge base
t, a counter, initially 0, indicating time
TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
action ← ASK(KB, MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action,t))
t ←t + 1
return action
```

Fig. 2.2.1 A generic knowledge based agent

Operations	TELL, ASK
Functions	MAKE-PERCEPT-SENTENCE, MAKE-ACTION-QUERY, MAKE-ACTION-SENTENCE
TELL	Used to add new sentence to KB
ASK	Querying the KB

MAKE-PERCEPT-SENTENCE	Constructs a sentence asserting that the agent perceived the given percept at the given time
MAKE-ACTION-QUERY	Constructs a sentence that asks what action should be performed at the current time
MAKE-ACTION-SENTENCE	Constructs a sentence asserting that the chosen action was executed
Table 1 Procedure KB-AGENT key components	

Two levels of agent program are:

i) *Knowledge level:* Specifies only what the agent knows and what its goals are

ii) *Implementation level*:Specifies how it works

Two types of approaches for knowledge base:

i) *Declarative Approach:* The agent's initial program, before it starts to receive percepts, is built by adding one by one the sentences that represent the designer's knowledge of the environment.

ii) *Procedural Approach:* Encodes desired behaviors directly as program code

2.2.2 Logic

- Knowledge bases consist of sentences.
- These sentencesare expressed according to the syntax of the representation language, which specifies all thesentences that are well formed.
- The concept of syntax is, in ordinary arithmetic expression: 'a *b =10' is a well-formed sentence, where as 'a5b* =' is not well-formed sentence.
- Logic defines the semantics of the language.
- Semantics do the actions with the "meaning" of sentences.
- In logic, the definition is more precise.

The semantics of the language defines the truth of each sentence with respect to each possible world. For example, the usual semantics adopted for arithmetic specifies that the sentence "a + b = 4" is true in a world where a is 2 and b is 2, but false in a world where a is 1 and b is 1.1. In standard logics, every sentence must be either true or false in each possible world, there is no "in between" 2

Model: This term is used in place of "possible world."; "m is a model of α" to mean that sentence α is true in model m.

Entailment : The relation of logical entailment between sentences is the idea that a sentence *follows logically* from another sentence.

In mathematical notation, α ⊨ β to mean that the sentence α entails the sentence β

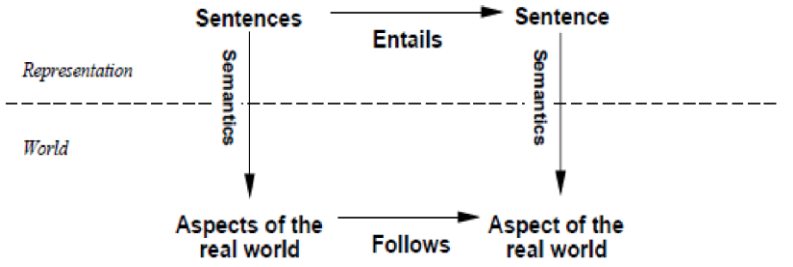


Fig 2.2.2 Sentences are physical configurations of the agent, and reasoning is a processof constructing new physical configurations from old ones.

Types of logics in Artificial Intelligence

1. Deductive logic
2. Inductive logic

1) Deductive logic

In deductive logic, the complete evidence is provided about the truth of the conclusion made. Here, the agent uses specific and accurate premises that lead to a specific conclusion. An example of this logic can be seen in an expert system designed to suggest medicines to the patient.

2) Inductive logic

In Inductive logic, the reasoning is done through a 'bottom-up' approach. The agent takes specific information and then generalizes it for complete understanding. An example of this can be seen in the natural language processing.

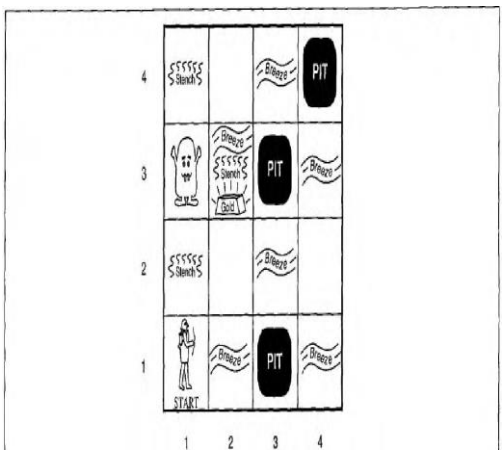
2.2.3 The Wumpus World

- The Wumpus world (Game) is a cave consisting of rooms connected by passage ways.
- Wumpus,waits somewhere in the cave to eat anyone who enters its room.
- The Wumpus can be shot by an agent, but the agent has only one arrow.
- Some rooms contain bottomless pits that will trap anyone who wanders into these rooms.

- Mitigating feature of living in this environment is the possibility of finding a heap of gold. [Originally suggested by Michael Genesereth, presented by Russell and Norvig, 1995)

PEAS Description

- Performance measure
- Gold: +1000
- Death: -1000
- Per step: -1
- For using the arrow: -10
- Environment
- Squares adjacent to Wumpus are smelly
- Squares adjacent to pit are breezy
- Glitters, iff gold is in the same square
- Gold is picked up by reflex
- Agent will bump if it walks into a wall
- Shooting kills Wumpus
- Shooting uses up the only arrow in straight line of agent facing
- Grabbing picks up gold if in same square
- Actuators::
- Move Forward
- Turn Left : 90 Degree
- Turn, Right : 90 Degree
- Grab : to pick up gold
- Shoot
- Release
- Climb (Finally, only from square[1,1])
- Sensors:
- Stench, Breeze, Glitter, Bump, Scream



2.2.3 Wumpus World. Agent in bottom left corner, facing right

Agent is in [1,1] and that [1,1] is a safe square.

The first percept is *[None, None, None, None, None]*, so, neighboring squares are safe.

B(Breezy) and OK marked in the appropriate squares.

From the fact that there was no stench or breeze in [1,1],

The agent can infer that [1,2]and [2,1] are free of dangers.

If agent decides to move forward to [2,1], giving the scene in Figure 2.2.5

The agent detects a breeze in [2,1], so there must be a pit in a neighboring square.

There must be a pit in [2,2] or [3,1] or both.

The notation PIT in Figure 2.2.5 indicates a possible pit in those squares.

At this point, thereis only one known square that is OK and has not been visited yet.

So the agent willturn around, go back to [I, 11, and then proceed to [1,2].

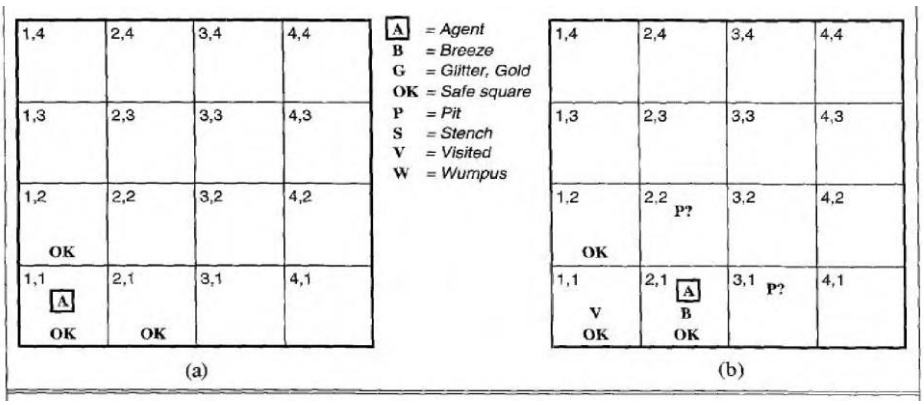


Figure 2.2.4 The first step taken by the agent in the wumpus world. The initial situation, after percept [None, None, None, None, None]

Figure 2.2.5 After one move, with percept [None, Breeze, None, None, None].

The new percept in [1,2] is *[Stench, None, None, None, None]*, resulting in the state of knowledge shown in Figure 2.2.6

The stench in [1,2] so, must be a wumpus nearby.

But the wumpus cannot be in [1,1], by the rules of the game, and it cannot be in [2,2]

Therefore, the agent can infer that the wumpus is in [1,3]. The notation W! indicates this. Moreover, the lack of a *Breeze* in [1,2] so, there is no pit in [2,2].

Agent inferred that there must be a pit in either [2,2] or [3,1], so this means it must be in [3,1].

The agent has now proved to itself that there is neither a pit nor a wumpus in *[2,2],* soit is OK to move there.

Agent turns and moves to *[2,3],* given in Figure 2.2.7

In [2,3], the agent detects a glitter, so it should grab the gold and thereby end the game.

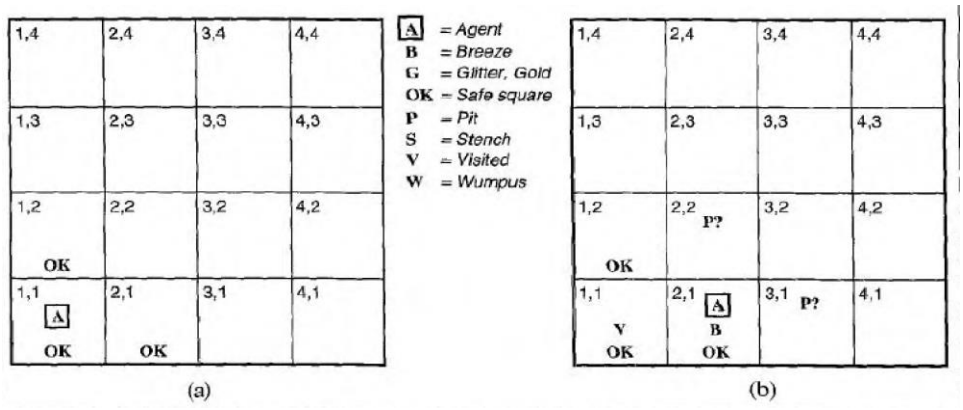


Figure 2.2.6 Two later stages in the progress of the agent. (a) After the third move, with percept [Stench, None, None, None, None].

Figure 2.2.7After the fifth move, with percept [Stench, Breeze, Glitter, None, None].

2.2.4 Reasoning

The reasoning is the mental process of deriving logical conclusion and making predictions from available knowledge, facts, and beliefs. "**Reasoning is a way to infer facts from existing data**." It is a general process of thinking rationally, to find valid conclusions.

In artificial intelligence, the reasoning is essential so that the machine can also think rationally as a human brain, and can perform like a human.

Types of Reasoning

In artificial intelligence, reasoning can be divided into the following categories:

- Deductive reasoning
- Inductive reasoning
- Abductive reasoning
- Common Sense Reasoning
- Monotonic Reasoning
- Non-monotonic Reasoning
- Deductive reasoning:

Deductive reasoning is deducing new information from logically related known information.

It is the form of valid reasoning, which means the argument's conclusion must be true when the premises are true.

Deductive reasoning is a type of propositional logic in AI, and it requires various rules and facts.

It is sometimes referred to as top-down reasoning, and contradictory to inductive reasoning.

Example:

Premise-1: All the human eats veggies
Premise-2: Mala is human.
Conclusion: Mala eats veggies.

- **Inductive Reasoning:**

  Inductive reasoning is a form of reasoning to arrive at a conclusion using limited sets of facts by the process of generalization.

  It starts with the series of specific facts or data and reaches to a general statement or conclusion.

  Inductive reasoning is a type of propositional logic, which is also known as cause-effect reasoning or bottom-up reasoning.

  In inductive reasoning historical data or various premises to generate a generic rule, for which premises support the conclusion are used.

  In inductive reasoning, premises provide probable supports to the conclusion, so the truth of premises does not guarantee the truth of the conclusion.

Example:

- Premise: All the tigers we have seen in the zoo are smart
- Conclusion: Therefore, we can expect all the tigers to be smart.

Abductive reasoning

It is a form of logical reasoning which starts with single or multiple observations then seeks to find the most likely explanation or conclusion for the observation.
It is an extension of deductive reasoning, but in abductive reasoning, the premises do not guarantee the conclusion.

Example:

Implication: Cricket ground is wet if it is raining
Axiom: Cricket ground is wet.
Conclusion It is raining.

- Common Sense Reasoning

  It is an informal form of reasoning, which can be gained through experiences.

  Common Sense reasoning simulates the human ability to make presumptions about events which occurs on every day.

Example:

1. One person can be at one place at a time.
2. If I put my hand in a fire, then it will burn.

The above two statements are the examples of common sense reasoning which a human mind can easily understand and assume.

- Monotonic Reasoning:

  In monotonic reasoning, once the conclusion is taken, then it will remain the same even if some other information is added to existing information in the knowledge base. In monotonic reasoning, adding knowledge does not decrease the set of prepositions that can be derived.

  To solve monotonic problems, the valid conclusion can be derived only from the available facts, and it will not be affected by new facts.

Example:

- Earth revolves around the Sun.

It is a true fact, and it cannot be changed even if another sentence is added in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Advantages of Monotonic Reasoning:

- In monotonic reasoning, each old proof will always remain valid.
- If deduce some facts from available facts, then it will remain valid for always.

Disadvantages of Monotonic Reasoning:

- The real world scenarios using Monotonic reasoning cannot be represented
- Hypothesis knowledge cannot be expressed with monotonic reasoning, which means facts should be true.
- Derive conclusions only from the old proofs, so new knowledge from the real world cannot be added.

Non-monotonic Reasoning

Example**:** Let suppose the knowledge base contains the following knowledge:

- All birds can fly
- Kiwi cannot fly
- Pattu is a bird

It is true that Pattu can fly.

Adding one another sentence into knowledge base "**Pattu is a Kiwi**" concludes "Pattu cannot fly", so it invalidates the above conclusion.

Advantages of Non-monotonic reasoning:

- It is used for real-world systems such as Robot navigation.
- Finding probabilistic facts or making assumptions is possible in this reasoning.
- Disadvantages of Non-monotonic Reasoning:
- In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
- It cannot be used for theorem proving.

2.3 PREPOSITIONAL LOGIC

Propositional logic is a very simple language consisting of proposition symbols andlogical connectives. It can handle propositions that are known true, known false, orcompletely unknown.

Syntax of propositional logic

It defines allowable sentences.

Atomic sentences

The indivisible syntactic elements consist of a single proposition symbol**.** Each such symbol stands for a proposition that can be true or false.

Names for symbols are in uppercase such as P, Q and R. The names are arbitrary but are often chosen to have some mnemonic value to the reader

There are two proposition symbols with fixed meanings: True is the always-true proposition and False is the always-false proposition.

Complex sentences

These are constructed from simpler sentences using logical connectives.

- Logical constants: true, false
- Propositional symbols: P, Q, S, (atomic sentences)
- Wrapping parentheses: (…)
- Sentences are combined by connectives:

Connective symbol	**Logic**	**Description**
∧	and	[conjunction]
∨	or	[disjunction]
⇒	implies	[implication / conditional]
⇔	is equivalent	[biconditional]
¬	not	[negation]

- **Literal**: atomic sentence or negated atomic sentence

Examples of Propositional LogicSentences

- P means “It is hot.”
- Q means “It is humid.”
- R means “It is raining.”
- (P ∧ Q) → R is “If it is hot and humid, then it is raining”
- Q → P is “If it is humid, then it is hot”

Sentence →Atomic Sentence | Complex Sentence Atomic Sentence → True | False |Symbol Symbol →P | Q | R Complex Sentence → ¬Sentence | (Sentence ∧ Sentence) | (Sentence ∨ Sentence) | (Sentence ==> Sentence) | (Sentence <==>Sentence)
Figure 2.3.1A BNF (Backus–Naur Form) grammar of sentences in propositional logic

Semantics

The semantics defines the rules for determining the truth of a sentence with respect to a particular model. In propositional logic, a model simply fixes the truth value, true or false for every proposition symbol.

For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is

m_l = { $P_{1,2}$= *false*, $P_{2,2}$=*false*, $P_{3,1}$= *true*}2^3=8 possible models.

P,Q,R. . .

True, False (A) ,¬A ,A ∧ B ,A ∨ B,A→B ,A↔ B

Hence, the following is an example of a wff:

P ∧ Q ∨ (B ∧¬C)→A ∧ B ∨ D ∧ (¬E)

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 2.3.2 : Truth table for Logical connectors

2.3.1. Theorem Proving

- *Logical equivalence*:**T**wo sentences α and β are logically equivalent if they are true in the same set of models.
- *Validity:*A sentence is valid if it is true in *all* models.

- Valid sentences are also called *tautologies*;sentences that are necessarily true.
- *Deduction Theorem*: For any sentences α and β, α ⊨ β if and only if the sentence (α⇒ β) is valid.
- Satisfiablility a sentence is satisfiable if it istrue in some model.
- Determining satisfiablity in propositional logic is SAT which is proved to be NP-complete.
- Proof by contradiction: α ⊨ β if and only if thesentence ¬(α ∧ β) is unsatisfiable.Its also known by proof by refutation or proof by contradiction.

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Figure 2.3.3 Standard logical equivalences. The symbols a, P, and y stand for arbitrary sentences of propositional logic.

Inference and proofs

Common rule:

Rule of Modus Ponens $\dfrac{\alpha \Rightarrow \beta,\ \beta \quad \alpha \Rightarrow \beta,\ \beta}{\therefore \beta \qquad \therefore \beta}$

And-Elimination

- Finding a proof can be efficient since irrelevant propositions can be ignored βα⇒β,ααΛβ propositions can be ignored.
- Monotonicity says that the set of entailed sentences can only increase as information is added to KB

Example to prove $\neg P_{1,2}$from $Rule_1$ through $Rule_5$:

- Applying biconditional elimination to R2 to obtain
  R6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Applying And-Elimination to obtain
  R7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Contraposition gives
  R8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$
- Modus Ponens with R8 and the percept R4 ie$\neg B_{1,1}$gives
  R9 : $\neg(P_{1,2} \vee P_{2,1})$

- De Morgan's rule gives
  R10: $\neg P_{1,2} \wedge \neg P_{2,1}$ *that is, neither* $P_{1,2}$ *nor* $P_{2,1}$ *contains a pit*

Proof problems should be defined with the steps as :

i) Initial state ii) Actions iii) Result iv) Goal

Conjunctive Normal Form (CNF)

Every sentence of propositional logic is logically equivalent to a conjunction of clauses. E.g. Convert :$B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$ to CNF:

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \alpha)$
  $(B_{1,1}(P_{1,2} \vee P_{2,1})) \wedge ((P1,2 \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Leftrightarrow$, replacing $\alpha \beta$ with $\neg \alpha \vee \beta$
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Move $\neg$ inwards using de Morgan's rules and doublenegation
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Apply distributivity law ($\vee$ over $\wedge$) and flatten
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B1,1) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution algorithm: Proof by contradiction, i.e., show KB $\wedge \neg\alpha$ unsatisfiable

- Definite clause – disjunction of literals, of which exactlyone is positive
- e.g. $\neg P1 \vee \neg P2 \vee \neg P3 \vee P4$
- Horn clause – a disjunction of literals at most one of which is positive
- e.g. $\neg P1 \vee \neg P2$, or $\neg P3 \vee P4$
- Can be used with forward chaining or backward chaining
- Deciding entailment is linear in the size of KB

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← {}
  loop do
    for each pair of clauses Ci, Cj in clauses do
resolvents ← PL-RESOLVE(Ci, Cj)
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Figure 2.3.4 A simple resolution algorithm for propositional logic

2.3.2. Model Checking

There are two common approaches algorithms for general propositional inference based on model checking. They are

i) A complete backtracking algorithm
ii) Local search algorithms

i) A complete backtracking algorithm

The algorithm is derived by Martin Davis and Hilary Putnam (1960).Since the version of this algorithm explained in this chapter is described by Davis, Logemann, and Loveland (1962), it is called as DPLL(Russell et. Al. 2015)

This algorithm given in Fig. 2.3.1 is for checking satisfiability (SAT).This algorithm detects whether the sentence is true or false.

Heuristics in DPLL algorithm:

Early termination

- If *any* one of the literal in a clause is true then the clause is true.
- On the other hand when a*any* clause in the sentence is false then the sentence is false.
- Example: (A ∧B)∨ (A∧C) If A is true this clause is true irrespective of state of B and C

Pure symbol heuristic

It is a symbol that appears with the same sign in all clauses of a sentence.

Making these literals *true* never make a clause *false*.

Example: (A ∨ ¬B), (¬B∨C),(C ∨A)are clauses. Here, literals A and B are pure symbols for the reason that A and B take same signs as positive and negative respectively. But C is impure as it has both signs.

Unit clause heuristic

*Unit clause:*All literals in the clause but one has been assigned false.

*Unit propagation:*Cascade of forced assignments

Example: If B=true then (¬B∨¬C) is simplified as ¬C.

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, [])
```

```
function DPLL(clauses, symbols, model) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, EXTEND(P, value, model)
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, EXTEND(P, value, model)
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, EXTEND(P, true, model)) or
         DPLL(clauses, rest, EXTEND(P, false, model))
```

Figure 2.3.5 The *DPLL* algorithm for checking satisfiability of a sentence in propositional logic

Tricks to scale up to large SAT problems:

- Component Analysis: Speed up the process by working on each component separately
- Variable and value ordering :To avoid trying always true value first, choosing the variable that appears most often in remaining clauses
- Intelligent backtracking :Chorological back tracking can be solved by backing up all the way to the relevant conflict
- Random restarts :If run is not processing properly restart with random choices which reduces the variance on the time to solution
- Clever indexing:Indexing structures must be updated dynamically to cope up with the speedup methods used in DPLL

ii) Local-search algorithms

Local search algorithms like Hill-Climbing and Simulated-Annealing algorithms can be applied directly to satisfiability problems. The right evaluation function should be applied.

- The enhanced algorithm WALKSAT is used to overcome the few drawbacks like local maxima in the other local search algorithms.
- It is most useful when expecting a solution to exist.
- In every iteration, the algorithm picks an unsatisfied clause and picks a symbol in the clause to flip.

There are two ways to randomly pick the symbol to flip:

i) min-conflicts : Minimizes the number of unsatisfied clauses in the new state

ii) random walk : Picks the symbol randomly

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move, typically around 0.5
          max-flips, number of flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max-flips do
     if model satisfies clauses then return model
     clause ← a randomly selected clause from clauses that is false in model
     with probability p flip the value in model of a randomly selected symbol from clause
     else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

Fig. 2.3.6. WALKSAT Algorithm

2.3.3 Agents based on prepositional logic

Current State of the world

The current state of the problem can be described using simple propositional logic. some of propositions used inWumpus world problem are given below.

- No Wumpus in square [1,1])
  $\square P_{1,1}$ *(no pit in square [1,1])* $\square W_{1,1}$
- Breeze next to Pit
  $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- Stench next to Wumpus
  $S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$
- At least 1 Wumpus
  $W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$
- Atleast one of them must be Wumpus-free:
  $\square W_{1,1} \vee \square W_{1,2}$
  $\neg W_{1,1} \vee \neg W_{1,3}$
  …
  $\neg W_{4,3} \vee \neg W_{4,4}$

Atemporal Variables: Symbols associated with permanent spects of the world do not need a time superscript are called as Atemporal variables.

Fluent: Associating propositions with time steps that extends to any aspect of the world that changes over time

Ex:Agent in [1,1] at time 0 FacingEast^0, WumpusAlive^0. Based on time change the fluents are applied in proposition clauses.

A Hybrid Agent

To create a hybrid agent for the wumpus world, the capacity to deduce various aspects of the state of the world may be integrated rather simply with the condition–action rules and problem-solving algorithms.

The agent program's primary body creates a plan based on a diminishing priority of goals. The program is depicted in the fig 2.3.7.

Program Sequence

1. If there is a glitter, the program first devises a strategy for grabbing the gold, returning to the original place, and climbing out of the cave.
2. Otherwise, if no current plan exists, the software plots a path to the nearest safe square it has not yet visited, ensuring that the route passes only via safe squares. A* search, not an ASK , is used to plan a route.
3. If the agent still has an arrow and there are no safe squares to investigate, the next step is to try to make a safe square by shooting at one of the available wumpus spots.
4. These are found by inquiring where ASK(KB, $\neg W_{x,y}$) is false, i.e. when it is unknown whether or not there is a Wumpus.
5. PLAN-ROUTE is used by the function PLAN-SHOT (not shown) to plan a sequence of operations that will line up this shot.
6. If this doesn't work, the program looks for a square to explore that isn't provably unsafe—that is, one for which *ASK(KB,* $\neg OK^t_{x,y}$*)* returns false.
7. If no such square exists, the mission will be impossible, and the agent will withdraw to [1, 1] and climb out of the cave.

```
function HYBRID-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench, breeze, glitter, bump, scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, initially 0, indicating time
              plan, an action sequence, initially empty
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x, y] : ASK(KB, OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
      plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
      unvisited ← {[x, y] : ASK(KB, L^{t'}_{x,y}) = false for all t' ≤ t}
      plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, HaveArrow^t) = true then
      possible_wumpus ← {[x, y] : ASK(KB, ¬W_{x,y}) = false}
      plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then //no choice but to take a risk
      not_unsafe ← {[x, y] : ASK(KB, ¬OK^t_{x,y}) = false}
      plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
      plan ← PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

```
function PLAN-ROUTE(current, goals, allowed) returns an action sequence
  inputs: current, the agent's current position
          goals, a set of squares; try to plan a route to one of them
          allowed, a set of squares that can form part of the route
  problem ← ROUTE-PROBLEM(current, goals, allowed)
  return A*-GRAPH-SEARCH(problem)
```

Fig. 2.3.7. Hybrid agent program for Wumpus world

2.4 FIRST ORDER LOGIC

2.4.1 Syntax and Semantics of First order logic

Models for first-order logic : The models of a logical language are the formal structures that constitute the possible worlds under consideration. To determine truth of any sentence, each model links the words of the logical sentences to elements of the possible world.

Thus, models for propositional logic link proposition symbols to predefined truth values. Models for first-order logic have objects.

Domain of a model is the set of objects or domain elements it contains. The domain is required to be nonempty. A relation is set of tuples of objects that are related.

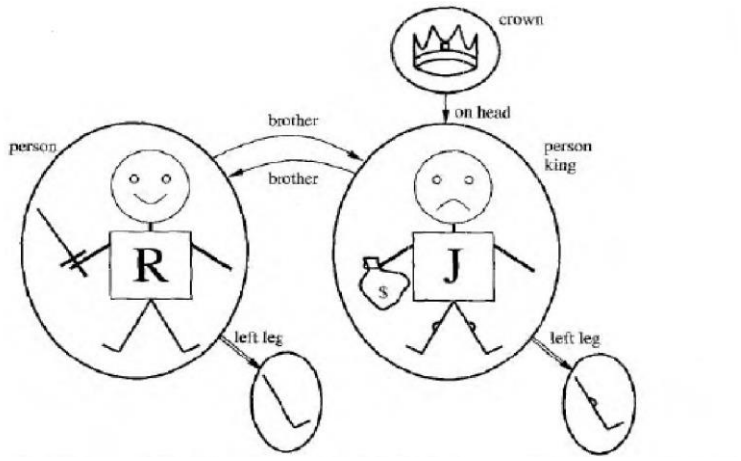


Figure 2.4.1 A model containing five objects, two binary relations, three unary relationsand one unary function, left-leg.

Figure 2.4.1 shows a model with five objects:

- Richard the Lionheart, King of England from 1189 to 1199
- His younger brother, the evil King John, who ruled from 1199 to 1215
- The left leg of Richard
- The left leg John
- A crown.

The objects in the model are related in following ways.

- Richard andJohn are brothers.
- A relation is just the set of tuples of objects that arerelated.
- The brotherhood relation in this model is the set
- *{ (Richard the Lionheart, King John), (King John, Richard the Lionheart) }*
- 'on head' relation contains just one tuple,
  (the crown, King John).

Binary Relations:

Relations those who relate objects are known as binary relations

Example :The "brother" and "on head" relations

Unary relations

Object with a property

Example: the "person" property is true of both Richard and John.

Models in first-order logic require total functions, that is, there must be avalue for every input tuple

Functions: Kinds of relationships that a given objectmust be related to exactly one object.

Example: Each person has one left leg

So the model has a unary "left leg" function that includes the following mappings:

(Richard the Lionheart) →Richard's left leg

(King John) →John's left leg

Symbols and interpretations

Symbols are the basic syntactic elements of first-order logic.

Symbols stand for objects, relations, and functions.

The symbols are of three kinds:

Constant symbols : stand for objects; Ex.: John, Richard

Predicate symbols: stand for relations; Ex.: OnHead, Person, King, and Crown

Function symbols: stand for functions. Ex.: left leg Symbols will begin with uppercase letters.

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable, \ldots\ Sentence \\
&\mid \neg\ Sentence \\
\\
AtomicSentence &\rightarrow Predicate(Term, \ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term, \ldots) \\
&\mid Constant \\
&\mid Variable \\
\\
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLeg \mid \ldots
\end{aligned}
$$

Fig 2.4.2 The syntax of first-order logic with equality

Example:

- Richard refers to Richard the Lionheart and John refers to the evil king John.
- Brother refers to the brotherhood relation
- OnHead refers to the "on head relation that holds between the crown and King John
- Person, King, and Crown refer to the sets of objects that are persons, kings, and crowns
- Left Leg refers to the "left leg" function

The truth of any sentence is determined by a model. Entailment, validity, and so on is defined in terms of all possiblemodels and all possible interpretations.

Term

A term is a logical expression that refers to an object. Ex. Constant symbols.

A complex term is just a complicated kind of name.

A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol

Example: Use LeftLeg(John) instead of using ‘King John's left leg’

Atomic sentences

An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms:

Example:

Brother(Richard, John).

Atomic sentences can have complex terms as arguments.

Example:

Married (Father(Richard), Mother(John)).

An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

Complex sentences Complex sentences can be constructed using logical Connectives, just as in propositional calculus such as:

¬Brother(LeftLeg(Richard),John)
Brother(Richard, John) ∧ Brother(John, Richard)
King (Richard) V *King (John)*
¬ King (Richard) ==>*King (John)*

Quantifiers:

Quantifiers express properties of entire collections of objects

First-order logic contains two standard quantifiers such as,

i) Universal Quantifier (∀)

ii) Existential Quantifier **(∃)**

Universal Quantification (∀)

"All kings are persons," is written in first-order logic as

∀x King(x) ⇒ Person(x)

∀ is usually pronounced "For all . . ."

"For all x, if x is a king, then z is a person." The symbol x is calleda variable.

Ground Term: A term with no variables

Different ways of Interpretation of the given sentence::

x→ Richard the Lionheart,

x→ King John,

x→ Richard's left leg,

x→ John's left leg,

x→ the crown

The universally quantified sentence *∀x King(x) ⇒Person(x)* is true in the original model if the sentence King(x) ⇒Person(x) is true under each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king ⇒Richard the Lionheart is a person.

King John is a king ⇒King John is a person.

Richard's left leg is a king ⇒Richard's left leg is a person.

John's left leg is a king ⇒John's left leg is a person.

The crown is a king ⇒The crown is a person.

Existential quantification (∃)

An existential quantifiermakes a statement about some object in the universe without naming it.

King John has a crown on his head, can be written as

∃xCrown(x) ∧OnHead(x, John)

Based on this any one of the following is true

Richard the Lionheart is a crown ∧Richard the Lionheart is on John's head;

King John is a crown ∧King John is on John's head;

Richard's left leg is a crown ∧Richard's left leg is on John's head;

John's left leg is a crown ∧John's left leg is on John's head;

The crown is a crown ∧the crown is on John's head.

The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model. Just as ⇒appears to be the natural connective to use with ∀, ∧is the natural connective to use with ∃.

Nested quantifiers

One can express more complex sentences using multiple quantifiers.

For example, “Brothers are siblings” can be written as ∀x∀y Brother (x, y) ⇒Sibling(x, y). Consecutive quantifiers of the same type can be written as one quantifier with several variables.

The sentence ‘Brothers are siblings’ can be specifies as: *∀x, ∀y Sibling(x, y) ⇔Sibling(y, x)*

Symmetric representation: *∀x, y Sibling(x, y) ⇔Sibling(y, x)*

Other Cases:

i) 'Everybody loves somebody’ can be written as ∀x∃y Loves(x, y) .

ii) ‘There is someone who is loved by everyone’ can be written as ∃y∀x Loves(x, y) .

Connections between ∀and ∃

These two **∀and** ∃quantifiers are closely connected with each other, through negation.

Example assertions:

i) “ Everyone dislikes Bitterness” is the same as asserting “ there does not exist someone who likes Bitterness” , and vice versa:

“∀x ¬Likes(x, Bitterness)” is equivalent to “¬∃x Likes(x, Bitterness)”

ii) “Everyone likes sweets” means that “ there is no one who does not like sweets” :

∀xLikes(x, sweets) is equivalent to ¬∃x ¬Likes(x, sweets)

Because ∀is really a conjunction over the universe of objects and ∃is a disjunction that they obey De Morgan’s rules.

$$\forall x\ \neg P \equiv \neg\exists x\ P \qquad \neg P \wedge \neg Q \equiv \neg(P \vee Q)$$
$$\neg\forall x\ P \equiv \exists x\ \neg P \qquad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\forall x\ P \equiv \neg\exists x\ \neg P \qquad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$
$$\exists x\ P \equiv \neg\forall x\ \neg P \qquad P \vee Q \equiv \neg(\neg P \wedge \neg Q)\ .$$

2.4.2 Using First order logic

Domain is some part of the world about which we wish to express some knowledge. Systematic representations of some simple domains are given in this section.

Assertions and queries in first-order logic

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions.

For example,

John is a king, TELL (KB, King (John)).

All kings are persons: TELL (KB, ∀x King(x) ⇒Person(x))

Richard is a person. TELL (KB, Person (Richard))

Queries: Questions asked with ASK are called queries or goals.

Example:

ASK (KB, King (John)) returns true

Any query that is logically entailed by the knowledge base should be answered affirmatively.

"ASK (KB, Person (John))" should also return true.

Substitution or binding list

To know what value of x makes the sentence true, a different function, ASKVARS is used, like*ASKVARS (KB, Person(x))*.

Two answers: {x/John} and {x/Richard} for the above query. Such an answer is called a *substitution* or *binding list.*

ASKVARS is reserved for knowledge bases with Horn clauses,

The kinship domain

"Durga is the mother of Adi" and "Adi is the father ofAjit' and rules such as "One's grandmother is the mother of one's parent."

The objects in the domain are people.The two unary predicates in the given sentences arei) Male and ii)Female.

Kinship relations: Parenthood, brotherhood, marriage, and so on

Binary predicates: Parent, Sibling, Brother,Sister,Child, Daughter, Son, Spouse, Wife, Husband, Grandparent,Grandchild, Cousin, Aunt, and Uncle.

Use functions for Mother and Father, because every person has exactly one of each of these.

Representation of each function and predicate:

i)one's mother is one's female parent:

$\forall m, c\ Mother(c) = m \Leftrightarrow Female(m) \wedge Parent(m,c)$

ii) One's husband is one's male spouse:

$\forall w, h\ Husband(h,w) \Leftrightarrow Male(h) \wedge Spouse(h,w)$

iii) Male and female are disjoint categories:

$\forall x\ Male(x) \Leftrightarrow \neg Female(x)$

iv) Parent and child are inverse relations:

$\forall p, c\ Parent(p, c) \Leftrightarrow Child(c, p)$

v)A grandparent is a parent of one's parent:

$\forall g, c\ Grandparent(g, c) \Leftrightarrow \exists p\ Parent(g, p) \wedge Parent(p, c)$

vi) A sibling is another child of one's parents:

$\forall x, y\ Sibling(x, y) \Leftrightarrow x \neq y \wedge \exists p\ Parent(p, x) \wedge Parent(p,y)$

Numbers, sets, and lists

Numbers can be used to construct large theory from a tiny kernel of axioms.

Theory of natural numbers or non-negative integers.

Predicate NatNum will be true of natural numbers;

constant symbol, 0;

function symbol, S (successor).

The PEANO axioms define natural numbers and addition.

Natural numbers are defined recursively:

NatNum(0) . ∀n NatNum(n) ⇒NatNum(S(n)) .

That is, 0 is a natural number, and for every object n, if n is a natural number, then S(n) is a natural number.

So the natural numbers are 0, S(0), S(S(0)), and so on.

Axioms to constrain the successor function are:

∀n 0 != S(n) . ∀m, n m != n ⇒ S(m) != S(n) .

Defining addition in terms of the successor function:

∀m NatNum(m) ⇒ + (0, m) = m .

∀m, n NatNum(m) ∧NatNum(n) ⇒ + (S(m), n) = S(+(m, n))

Adding 0 to any natural number m gives m itself. Addition is represented using the binary function symbol "+" in the term + (m, 0);

To make the sentences about numbers easier to read, infix notation is used.

S(n) as n + 1, so the second axiom becomes :

∀m, n NatNum (m) ∧NatNum(n) ⇒ (m + 1) + n = (m + n)+1 .

This axiom reduces addition to repeated application of the successor function.

Sets

The domain of sets is also fundamental to mathematics as well as to commonsense reasoning. Sets can be represented as individualsets, including empty sets.

Sets can be built up by:

Adding an element to a set

Taking the union or intersection of two sets.

*Operations that can be performed on sets are:*To know whether an element is a member of a set distinguishes sets from objects that are not sets.

Vocabulary of set theory: The empty set is a constant written as { }. There is one unary predicate, Set, which is true of sets.

The binary predicates are

x∈ s (x is a member of set s) s1 ⊆ s2

(set s1 is a subset, not necessarily proper, of set s2).

The binary functions are

s1 ∩ s2 (the intersection of two sets),

s1 ∪ s2 (the union of two sets),

and {x|s} (the set resulting from adjoining element x to set s).

2.5 INFERENCE IN FIRST-ORDER LOGIC

2.5.1 Forward Chaining

Starts with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.

Forward chaining approach is also called as data-driven as the goal can be reached using available data.

Forward chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
    p ← Pop(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
          decrement count[c]
          if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

Fig. 2.5.1 Forward chaining algorithm

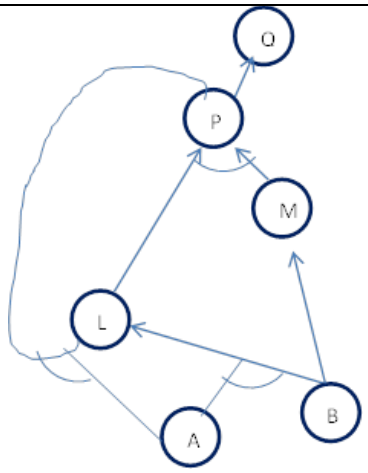
The *agenda* keeps track of symbols known to be true but not yet "processed".

The *count* table keeps track of how many premises of each implication are as yet unknown.

Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears

If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda.

A symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as P ⇒ Q and Q ⇒ P.

Horn Clauses: P ⇒Q L ∧ M ⇒ P B ∧ L ⇒ M A ∧ P ⇒ L A ∧ B ⇒ L A B	
Horn Clause	AND-OR Graph
Fig. 2.5.1. Forward chaining	

2.5.2 Backward Chaining

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

2.6 SUMMARY

Intelligent agents need know ledge about the domain to make appropriate decisions. A representation language is defined by its **syntax,** which specifies the structure of sentences, and its **semantics,** which defines the **truth** of each sentence in each **possible world** or **model**

The relationship of **entailment** between sentences is crucial to our understanding of reasoning. A sentence αentails another sentence βifβis true in all worlds where α is true. Equivalent definitions include the **validity** of the sentence and the **unsatisfiability** of the sentence α∧¬β

Propositional logic is a very simple language consisting of **proposition symbols** and **logical connectives.** It can handle propositions that are known true, known false, or completely unknown.

Domain of a model is the set of objects or domain elements it contains. The domain is required to be nonempty. A relation is set of tuples of objects that are related.

Forward Chaining starts with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.A **backward chaining** algorithm starts with the goal and works backward, chaining through rules to find known facts that support the goal.

2.7 REFERENCES

1. Russell, S. and Norvig, P, Artificial Intelligence: A Modern Approach, Third Edition, Prentice- Hall, 2010
2. Artificial Intelligence, Elaine Rich, Kevin Knight, Shivasankar B. Nair, The McGrawHill publications, Third Edition, 2009.
3. George F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Pearson Education, 6th ed., 2009.
4. https://www.geeksforgeeks.org/

2.8 UNIT END EXERCISES

1. Write a procedure for the operations of a generic knowledgebased agent
2. Explain the syntax and semantics of First order logic
3. Write and explain the model checking algorithms
4. Describe the approaches of theorem proving
5. Write short note on Hybrid agent
6. What is backward chaining? Explain.
7. Explain what is meant by “forward chaining”, and illustrate how it can be used to determine new facts
8. With suitable demonstration explain two types of Quantifiers.
9. Write short note on kinship domain
10. Discuss about role of logic and reasoning in intelligent agents.

❄❄❄❄❄❄❄

3

SEARCH STRATEGIES

Unit Structure

3.0 OBJECTIVES

This Chapter would make you understand the following concepts:

- Solving problems by searching
- Search- Issues in the Design of Search Programs,
- Un-Informed Search : BFS, DFS;
- Heuristic Search Techniques: Generate-And- Test, Hill Climbing

3.1 SOLVING PROBLEMS BY SEARCHING

To solve the problem of building a system you should take the following steps:

1. Define the problem accurately including detailed specifications and what constitutes a suitable solution.

2. Scrutinize the problem carefully, for some features may have a central affect on the chosen method of solution.
3. Segregate and represent the background knowledge needed in the solution of the problem.
4. Choose the best solving techniques for the problem to solve a solution.

***Problem solving* is a process** of generating solutions from observed data.

- a *'problem'* is characterized by a set of *goals*,
- a set of *objects*, and
- a set of *operations*.

These could be ill-defined and may evolve during problem solving.

- A **'*problem space*'** is an abstract space.
- A problem space encompasses all *valid states* that can be generated by the application of any combination of *operators* on any combination of *objects*.
- The problem space may contain one or more *solutions*. A solution is a combination of *operations* and *objects* that achieve the *goals*.
- A '***search***' refers to the search for a solution in a problem space.
- Search proceeds with different types of '*search control strategies*'.
- The *depth-first search and breadth-first search* are the two common *search strategies.*

3.1.1 AI - General Problem Solving

Problem solving has been the key area of concern for Artificial Intelligence.

Problem solving is a process of generating solutions from observed or given data. It is however not always possible to use direct methods (i.e. go directly from data to solution). Instead, problem solving often needs to use indirect or model based methods.

General Problem Solver (GPS) was a computer program created in 1957 by Simon and Newell to build a universal problem solver machine. *GPS* was based on Simon and Newell's theoretical work on logic machines. *GPS* in principle can solve any formalized symbolic problem, such as theorems proof and geometric problems and chess playing. *GPS* solved many simple problems, such as the Towers of Hanoi, that could be sufficiently formalized, but ***GPS could not solve any real-world problems***.

To build a system to solve a particular problem, we need to:

- Define the problem precisely – find input situations as well as final situations for an acceptable solution to the problem
- Analyze the problem – find few important features that may have impact on the appropriateness of various possible techniques for solving the problem

- Isolate and represent task knowledge necessary to solve the problem
- Choose the best problem-solving technique(s) and apply to the particular problem

3.1.2 Problem definitions

A problem is defined by its '*elements*' and their '*relations*'. To provide a formal description of a problem, we need to do the following:

a. Define a *state space* that contains all the possible configurations of the relevant objects, including some impossible ones.
b. Specify one or more states that describe possible situations, from which the problem- solving process may start. These states are called *initial states.*
c. Specify one or more states that would be acceptable solution to the problem.

These states are called *goal states*.

Specify a set of *rules* that describe the actions (*operators*) available.

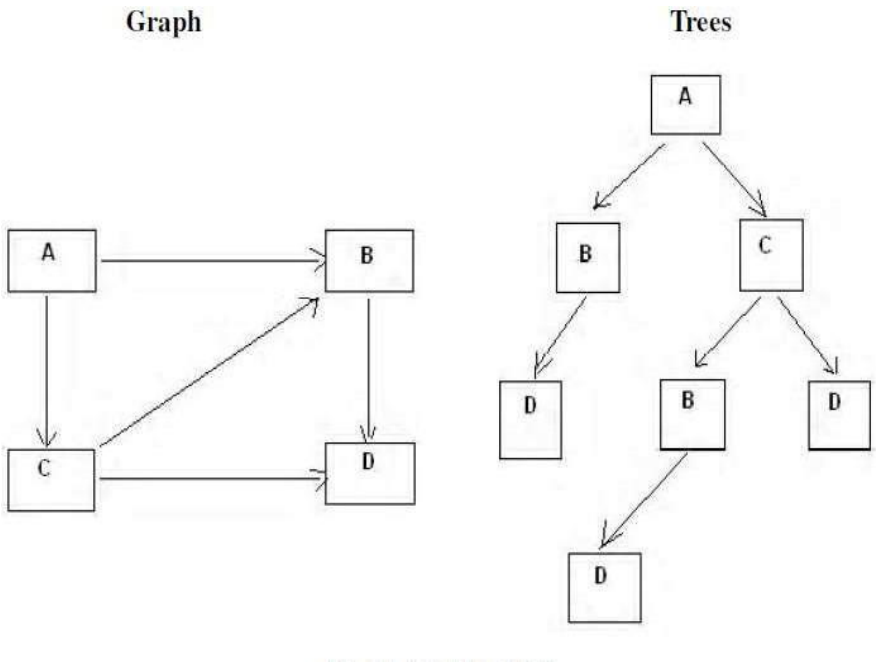
The problem can then be solved by using the *rules,* in combination with an appropriate *control strategy*, to move through the *problem space* until a *path* from an *initial state* to a *goal state* is found. This process is known as ***'search'***. Thus:

- *Search* is fundamental to the problem-solving process.
- *Search* is a general mechanism that can be used when a more direct method is not known.
- *Search* provides the framework into which more direct methods for solving subparts of a problem can be embedded. A very large number of AI problems are formulated as search problems.
- Problem space

A *problem space* is represented by a directed graph, where *nodes* represent search state and *paths* represent the operators applied to change the *state*.

To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a **tree**. A *tree* usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph,
e.g. node ***B*** and node ***D.***

A *tree is a graph* in which any two vertices are connected by exactly one path. Alternatively, any connected *graph with no cycles is a tree*.



3.1.3 Problem solving: The term, Problem Solving relates to analysis in AI. Problem solving may be characterized as a systematic search through a range of possible actions to reach some predefined goal or solution. Problem-solving methods are categorized as *special purpose* and *general purpose*.

- A *special-purpose method* is tailor-made for a particular problem, often exploits very specific features of the situation in which the problem is embedded.

- A *general-purpose method* is applicable to a wide variety of problems. One General-purpose technique used in AI is *'means-end analysis':* a step-bystep, or incremental, reduction of the difference between current state and final goal.

3.1.4 Search

Search is the systematic examination of *states* to find path from the *start / root state* to the *goal state*.

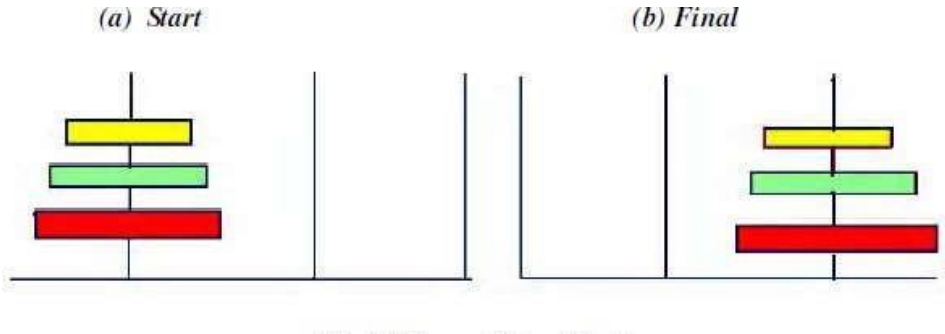
- Search usually results from a lack of knowledge.
- Search explores knowledge alternatives to arrive at the best answer.
- Search algorithm output is a solution, that is, a path from the initial state to a state that satisfies the goal test.

For general-purpose problem-solving – '***Search' is an approach***.

- Search deals with finding *nodes* having certain properties in a *graph* that represents search space.
- Search methods explore the search space 'intelligently', evaluating possibilities without investigating every single possibility.

Examples:

- For a Robot this might consist of PICKUP, PUTDOWN, MOVEFORWARD, MOVEBACK, MOVELEFT, and MOVERIGHT—until the goal is reached.
- Puzzles and Games have explicit rules: e.g., the '***Tower of Hanoi***' puzzle



This puzzle involves a set of rings of different sizes that can be placed on three different pegs.

- The puzzle starts with the rings arranged as shown in Figure 3.4(a)
- The goal of this puzzle is to move them all as to Figure 3.4(b)
- Condition: Only the top ring on a peg can be moved, and it may only be placed on a smaller ring, or on an empty peg.

In this *Tower of Hanoi* puzzle:

- Situations encountered while solving the problem are described as *states*.
- Set of all possible configurations of rings on the pegs is called *'problem space*'.

3.2 Issues in the Design of Search Programs

Each search process can be considered to be a tree traversal. The object of the search is to find a path from the initial state to a goal state using a tree. The number of nodes generated might be huge; and in practice many of the nodes would not be needed. The secret of a good search routine is to generate only those nodes that are likely to be useful, rather than having a precise tree. The rules are used to represent the tree implicitly and only to create nodes explicitly if they are actually to be of use.

The following issues arise when searching:

- The tree can be searched forward from the initial node to the goal state or backwards from the goal state to the initial state.
- To select applicable rules, it is critical to have an efficient procedure for matching rules against states.
- How to represent each node of the search process? This is the knowledge representation problem or the frame problem. In games,

an array suffices; in other problems, more complex data structures are needed.

Finally in terms of data structures, considering the water jug as a typical problem do we use a graph or tree? The breadth-first structure does take note of all nodes generated but the depth-first one can be modified.

Check duplicate nodes

1. Observe all nodes that are already generated, if a new node is present.
2. If it exists add it to the graph.
3. If it already exists, then

    a. Set the node that is being expanded to the point to the already existing node corresponding to its successor rather than to the new one. The new one can be thrown away.
    b. If the best or shortest path is being determined, check to see if this path is better or worse than the old one. If worse, do nothing.

Better save the new path and work the change in length through the chain of successor nodes if necessary.

Example: Tic-Tac-Toe

State spaces are good representations for board games such as Tic-Tac-Toe. The position of a game can be explained by the contents of the board and the player whose turn is next. The board can be represented as an array of 9 cells, each of which may contain an X or O or be empty.

- *State:*
- Player to move next: X or O.
- Board configuration:

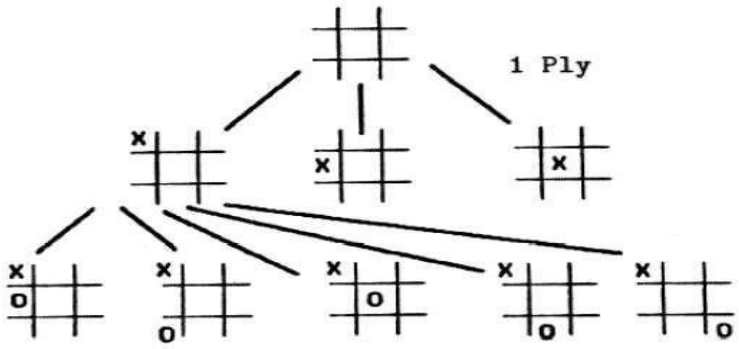
X		0
	0	
X		X

- **Operators:** Change an empty cell to X or O.
- **Start State:** Board empty; X's turn.
- **Terminal States:** Three X's in a row; Three O's in a row; All cells full.

3.2.1 Search Tree

The sequence of states formed by possible moves is called a *search tree*. Each level of the tree is called a *ply.*

Since the same state may be reachable by different sequences of moves, the state space may in general be a graph. It may be treated as a tree for simplicity, at the cost of duplicating states.



3.2.2 Solving problems using search

- Given an informal description of the problem, construct a formal description as a state space:
- Define a data structure to represent the *state*.
- Make a representation for the *initial state* from the given data.
- Write programs to represent *operators* that change a given state representation to a new state representation.
- Write a program to detect *terminal states*.
- Choose an appropriate search technique:
- How large is the search space?
- How well structured is the domain?
- What knowledge about the domain can be used to guide the search?

3.3 HEURISTIC SEARCH TECHNIQUES:

Search Algorithms

Many traditional search algorithms are used in AI applications. For complex problems, the traditional algorithms are unable to find the solutions within some practical time and space limits. Consequently, many special techniques are developed, using ***heuristic functions.***

The algorithms that use *heuristic functions* are called ***heuristic algorithms***.

- Heuristic algorithms are not really intelligent; they appear to be intelligent because they achieve better performance.
- Heuristic algorithms are more efficient because they take advantage of feedback from the data to direct the search path.
- ***Uninformed search algorithms*** or *Brute-force algorithms*, search through the search space all possible candidates for the solution checking whether each candidate satisfies the problem's statement.
- ***Informed search algorithms*** use heuristic functions that are specific to the problem, apply them to guide the search through the search space to try to reduce the amount of time spent in searching.

A good heuristic will make an informed search dramatically outperform any uninformed search: for example, the Traveling Salesman Problem (TSP), where the goal is to find is a good solution instead of finding the best solution.

In such problems, the search proceeds using current information about the problem to predict which path is closer to the goal and follow it, although it does not always guarantee to find the best possible solution. Such techniques help in finding a solution within reasonable time and space (memory). Some prominent intelligent search algorithms are stated below:

1. **Generate and Test Search**
2. **Best-first Search**
3. **Greedy Search**
4. **A* Search**
5. **Constraint Search**
6. **Means-ends analysis**

There are some more algorithms. They are either improvements or combinations of these.

- **Hierarchical Representation of Search Algorithms:** A Hierarchical representation of most search algorithms is illustrated below. The representation begins with two types of search:
- **Uninformed Search:** Also called blind, exhaustive or brute-force search, it uses no information about the problem to guide the search and therefore may not be very efficient.
- **Informed Search:** Also called heuristic or intelligent search, this uses information about the problem to guide the search—usually guesses the distance to a goal state and is therefore efficient, but the search may not be always possible.

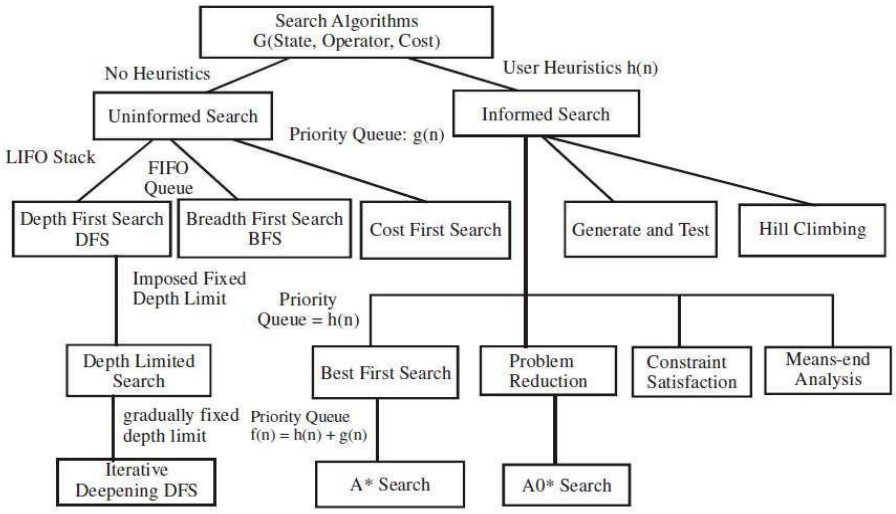
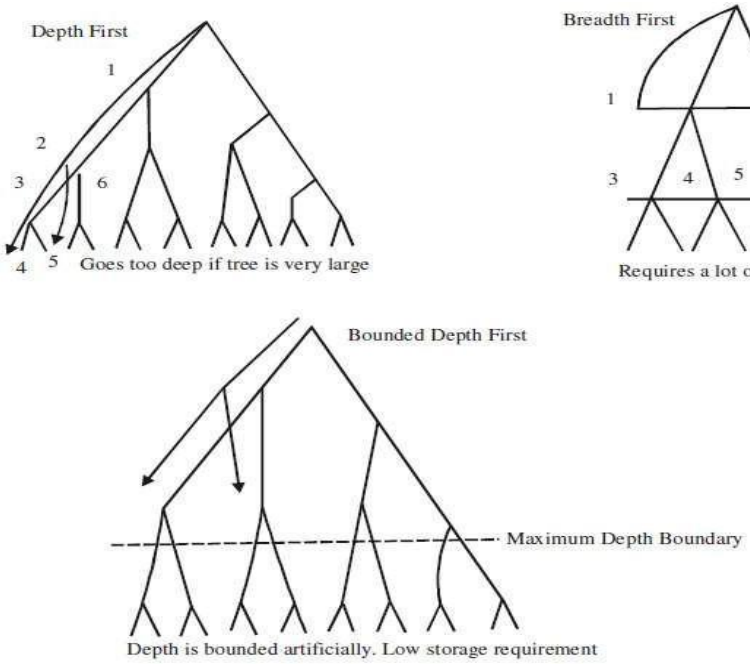


Fig. *Different Search Algorithms*

The first requirement is that it causes motion, in a game playing program,

it moves on the board and in the water jug problem, filling water is used to fill jugs. It means the control strategies without the motion will never lead to the solution.

The second requirement is that it is systematic, that is, it corresponds to the need for global motion as well as for local motion. This is a clear condition that neither would it be rational to fill a jug and empty it repeatedly, nor it would be worthwhile to move a piece round and round on the board in a cyclic way in a game. We shall initially consider two systematic approaches for searching. Searches can be classified by the order in which operators are tried: depth-first, breadth-first, bounded depth-first.



3.3.1 Breadth-first search

A Search strategy, in which the highest layer of a decision tree is searched completely before proceeding to the next layer is called *Breadth-first search (BFS).*

- In this strategy, no viable solutions are omitted and therefore it is guaranteed that an optimal solution is found.
- This strategy is often not feasible when the search space is large.

Algorithm

1. Create a variable called LIST and set it to be the starting state.
2. Loop until a goal state is found or LIST is empty, Do
    a. Remove the first element from the LIST and call it E. If the LIST is empty, quit.
    b. For every path each rule can match the state E, Do
        (i) Apply the rule to generate a new state.

(ii) If the new state is a goal state, quit and return this state.
(iii) Otherwise, add the new state to the end of LIST.

Advantages

1. Guaranteed to find an optimal solution (in terms of shortest number of steps to reach the goal).
2. Can always find a goal node if one exists (complete).

Disadvantages

1. High storage requirement: *exponential* with tree depth.

3.3.2 Depth-first search

A search strategy that extends the current path as far as possible before backtracking to the last choice point and trying the next alternative path is called *Depth-first search (DFS).*

- This strategy does not guarantee that the optimal solution has been found.
- In this strategy, search reaches a satisfactory solution more rapidly than breadth first, an advantage when the search space is large.

Algorithm

Depth-first search applies operators to each newly generated state, trying to drive directly toward the goal.

1. If the starting state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signalled:
   a. Generate a successor E to the starting state. If there are no more successors, then signal failure.
   b. Call Depth-first Search with E as the starting state.
   c. If success is returned signal success; otherwise, continue in the loop.

Advantages

1. Low storage requirement: *linear* with tree depth.
2. Easily programmed: function call stack does most of the work of maintaining state of the search.

Disadvantages

1. May find a sub-optimal solution (one that is deeper or more costly than the best solution).
2. Incomplete: without a depth bound, may not find a solution even if one exists.

Bounded depth-first search

Depth-first search can spend much time (perhaps infinite time) exploring a

very deep path that does not contain a solution, when a shallow solution exists. An easy way to solve this problem is to put a maximum depth bound on the search. Beyond the depth bound, a failure is generated automatically without exploring any deeper.

Problems:

1. It's hard to guess how deep the solution lies.
2. If the estimated depth is too deep (even by 1) the computer time used is dramatically increased, by a factor of *bextra*.
3. If the estimated depth is too shallow, the search fails to find a solution; all that computer time is wasted.

3.4 Heuristics

A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals. Some heuristics help in the search process without sacrificing any claims to entirety that the process might previously had. Others may occasionally cause an excellent path to be overlooked. By sacrificing entirety it increases efficiency. Heuristics may not find the best

solution every time but guarantee that they find a good solution in a reasonable time. These are particularly useful in solving tough and complex problems, solutions of which would require infinite time, i.e. far longer than a lifetime for the problems which are not solved in any other way.

3.4.1 Heuristic search

To find a solution in proper time rather than a complete solution in unlimited time we use heuristics. 'A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as numbers'. Heuristic search methods use knowledge about the problem domain and choose promising operators first. These heuristic search methods use heuristic functions to evaluate the next state towards the goal state. For finding a solution, by using the heuristic technique, one should carry out the following steps:

1. Add domain—specific information to select what is the best path to continue searching along.
2. Define a heuristic function h(n) that estimates the 'goodness' of a node n.
   Specifically, h(n) = estimated cost(or distance) of minimal cost path from n to a goal state.
3. The term, heuristic means 'serving to aid discovery' and is an estimate, based on domain specific information that is computable from the current state description of how close we are to a goal.

Finding a route from one city to another city is an example of a search problem in which different search orders and the use of heuristic knowledge are easily understood.

1. State: The current city in which the traveller is located.
2. Operators: Roads linking the current city to other cities.
3. Cost Metric: The cost of taking a given road between cities.
4. Heuristic information: The search could be guided by the direction of the goal city from the current city, or we could use airline distance as an estimate of the distance to the goal.

3.4.2 Heuristic search techniques

For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed, using ***heuristic functions.***

- Blind search is not always possible, because it requires too much time or Space (memory).

  Heuristics are ***rules of thumb***; they do not guarantee a solution to a problem.
- Heuristic Search is a weak technique but can be effective if applied correctly; it requires domain specific information.

3.4.2.1 Characteristics of heuristic search

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search incorporates domain knowledge to improve efficiency over blind search.
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.
- Heuristics might (for reasons) *underestimate* or *overestimate* the merit of a state with respect to goal.
- Heuristics that underestimate are desirable and called admissible.
- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient.

3.4.2.2 *Heuristic search compared with other search*

The Heuristic search is compared with Brute force or Blind search techniques below:

Comparison of Algorithms

Brute force / Blind search	**Heuristic search**
Can only search what it has knowledge	Estimates ‘distance’ to goal state

about already through explored nodes

No knowledge about how far a node Guides search process toward goal node from goal state

Prefers states (nodes) that lead close to and not away from goal state

Example: Travelling salesman

A salesman has to visit a list of cities and he must visit each city only once. There are different routes between the cities. The problem is to find the shortest route between the cities so that the salesman visits all the cities at once.

Suppose there are N cities, then a solution would be to take N! possible combinations to find the shortest distance to decide the required route. This is not efficient as with N=10 there are 36,28,800 possible routes. This is an example of *combinatorial explosion*.

There are better methods for the solution of such problems: one is called *branch* and *bound*. First, generate all the complete paths and find the distance of the first complete path. If the next path is shorter, then save it and proceed this way avoiding the path when its length exceeds the saved shortest path length, although it is better than the previous method.

3.4.2.3 Generate and Test Strategy

Generate-And-Test Algorithm

Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.

Algorithm: Generate-And-Test

1. Generate a possible solution.
2. Test to see if this is the expected solution.
3. If the solution has been found quit else go to step 1.

Potential solutions that need to be generated vary depending on the kinds of problems. For some problems the possible solutions may be particular points in the problem space and for some problems, paths from the start state.

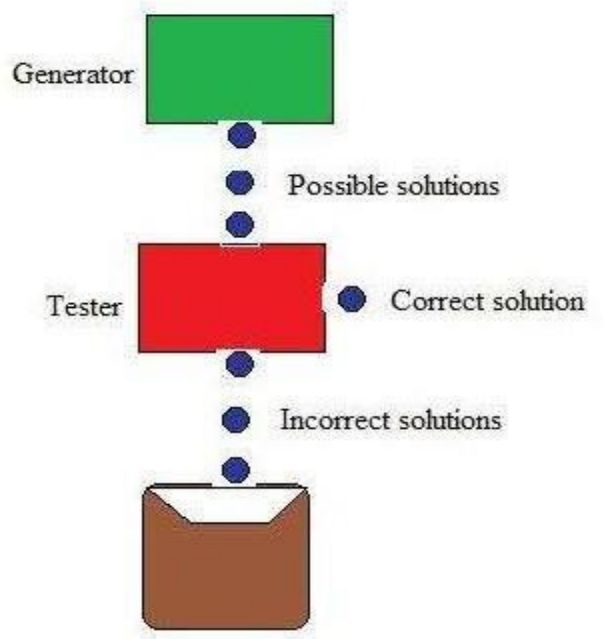


Figure: Generate And Test

Generate-and-test, like depth-first search, requires that complete solutions be generated for testing. In its most systematic form, it is only an exhaustive search of the problem space.

Solutions can also be generated randomly but solution is not guaranteed. This approach is what is known as British Museum algorithm: finding an object in the British Museum by wandering randomly.

Systematic Generate-And-Test

While generating complete solutions and generating random solutions are the two extremes there exists another approach that lies in between. The approach is that the search process proceeds systematically but some paths that unlikely to lead the solution are not considered. This evaluation is performed by a heuristic function.

Depth-first search tree with backtracking can be used to implement systematic generate-and-test procedure. As per this procedure, if some intermediate states are likely to appear often in the tree, it would be better to modify that procedure to traverse a graph rather than a tree.

Generate-And-Test And Planning

Exhaustive generate-and-test is very useful for simple problems. But for complex problems even heuristic generate-and-test is not very effective technique. But this may be made effective by combining with other techniques in such a way that the space in which to search is restricted. An AI program DENDRAL, for example, uses plan-Generate-and-test technique. First, the planning process uses constraint-satisfaction techniques and creates lists of recommended and contraindicated substructures. Then the generate-and-test procedure uses the lists

generated and required to explore only a limited set of structures. Constrained in this way, generate-and-test proved highly effective. A major weakness of planning is that it often produces inaccurate solutions as there is no feedback from the world. But if it is used to produce only pieces of solutions then lack of detailed accuracy becomes unimportant.3.4.2.4 Hill Climbing

Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence .

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where we need to minimize the distance traveled by salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time.
- A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

1. Variant of generate and test algorithm : It is a variant of generate and test algorithm. The generate and test algorithm is as follows :

1. Generate a possible solutions.
2. Test to see if this is the expected solution.
3. If the solution has been found quit else go to step 1.

Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

3. Uses the Greedy approach : At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

Types of Hill Climbing

1. Simple Hill climbing : It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.

Algorithm for Simple Hill climbing :

Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2 : Loop until the solution state is found or there are no new operators present which can be applied to current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.
b) Perform these to evaluate new state
i. If the current state is a goal state, then stop and return success.
ii. If it is better than the current state, then make it current state and proceed further.
iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3 : Exit.

2. Steepest-Ascent Hill climbing : It first examines all the neighboring nodes and then selects the node closest to the solution state as next node.

Step 1 : Evaluate the initial state. If it is goal state then exit else make the current state as initial state

Step 2 : Repeat these steps until a solution is found or current state does not change

i. Let ‘target’ be a state such that any successor of the current state will be better than it;
ii. for each operator that applies to the current state
a. apply the new operator and create a new state
b. evaluate the new state
c. if this state is goal state then quit else compare with ‘target’
d. if this state is better than ‘target’, set this state as ‘target’
e. if target is better than current state set current state to Target

Step 3 : Exit

3. Stochastic hill climbing : It does not examine all the neighboring nodes before deciding which node to select .It just selects a neighboring node at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

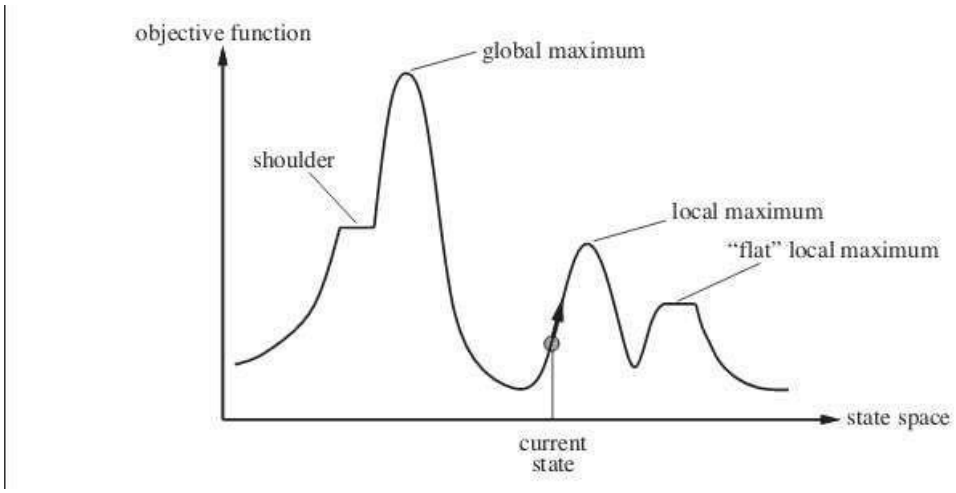
State Space diagram for Hill Climbing

State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).

X- axis : denotes the state space ie states or configuration our algorithm may reach.

Y-axis : denotes the values of objective function corresponding to to a particular state.

The best solution will be that state space where objective function has maximum value(global maximum).



Different regions in the State Space Diagram

1. Local maximum : It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here value of objective function is higher than its neighbors.
2. Global maximum : It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
3. Plateua/flat local maximum : It is a flat region of state space where neighboring states have the same value.
4. Ridge : It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
5. Current state : The region of state space diagram where we are currently present during the search.
6. Shoulder : It is a plateau that has an uphill edge. Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

1. Local maximum : At a local maximum all neighboring states have a values which is worse than than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

   To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
2. Plateau : On plateau all neighbors have same value . Hence, it is not possible to select the best direction.

   To overcome plateaus : Make a big jump. Randomly select a state

far away from current state. Chances are that we will land at a non-plateau region

3. Ridge : Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.

   To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

3.5 LETS SUM UP

- Solving problems by searching
- Search- Issues in the Design of Search Programs,
- Un-Informed Search : BFS, DFS;
- Heuristic Search Techniques: Generate-And- Test, Hill Climbing

3.6 REFERENCES

- https://www.cpp.edu/~ftang/courses/CS420/notes/uninformed%20search.pdf
- https://web.ntnu.edu.tw/~tcchiang/ai/2_Uninformed%20search%20(D).pdf
- https://web.cs.hacettepe.edu.tr/~pinar/courses/VBM688/lectures/uninformed_search.pdf
- https://www.includehelp.com/ml-ai/solving-problem-by-searching-in-artificial-intelligence.aspx
- https://www.princeton.edu/~otorres/Regression101.pdf

3.7 EXERCISES

- Solving problems by searching real time example.
- Search- Issues in the Design of Search Programs
- Take a real time example and execute Search Techniques.

❄❄❄❄❄❄❄

4

TABU SEARCH

Unit Structure

4.1 OBJECTIVES

This Chapter would make you understand the following concepts:

- Heuristic Search Techniques: Best-First Search,
- A* Algorithm, Alpha beta search algorithm, Problem Reduction,
- AO*Algorithm, Constraint Satisfaction, Means-Ends Analysis

4.2 HEURISTIC SEARCH TECHNIQUES

For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed, using ***heuristic functions.***

- Blind search is not always possible, because it requires too much time or Space (memory).

Heuristics are ***rules of thumb***; they do not guarantee a solution to a problem.

- Heuristic Search is a weak technique but can be effective if applied correctly; it requires domain specific information.

4.2.1 Best First Search (Informed Search)

In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

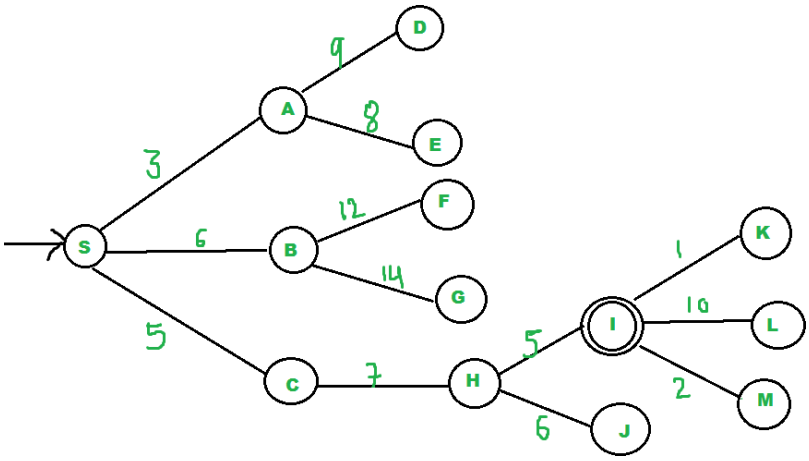
We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm:

Best-First-Search(Grah g, Node start)

1) Create an empty PriorityQueue PriorityQueue pq;
2) Insert "start" in pq. pq.insert(start)
3) Until PriorityQueue is empty u = PriorityQueue.DeleteMin

If u is the goal Exit
Else
Foreach neighbor v of u If v "Unvisited"
Mark v "Visited" pq.insert(v)
Mark v "Examined" End procedure
Let us consider below example.



We start from source "S" and search for goal "I" using given costs and Best First search.
pq initially contains S
We remove s from and process unvisited neighbors of S to pq.
pq now contains {A, C, B} (C is put before B because C has lesser cost)
We remove A from pq and process unvisited neighbors of A to pq.
pq now contains {C, B, E, D}
We remove C from pq and process unvisited neighbors of C to pq.
pq now contains {B, H, E, D}
We remove B from pq and process unvisited neighbors of B to pq.
pq now contains {H, E, D, F, G}
We remove H from pq. Since our goal "I" is a neighbor of H, we return.

Analysis :

- The worst case time complexity for Best First Search is O(n * Log n) where n is number of nodes. In worst case, we may have to visit all nodes before we reach goal. Note that priority queue is

implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time.

- Performance of the algorithm depends on how well the cost or evaluation function is designed.

4.2.2 A* Search Algorithm

A* is a type of search algorithm. Some problems can be solved by representing the world in the initial state, and then for each action we can perform on the world we generate states for what the world would be like if we did so. If you do this until the world is in the state that we specified as a solution, then the route from the start to this goal state is the solution to your problem.

In this tutorial I will look at the use of state space search to find the shortest path between two points (pathfinding), and also to solve a simple sliding tile puzzle (the 8-puzzle). Let's look at some of the terms used in Artificial Intelligence when describing this state space search.

Some terminology

A *node* is a state that the problem's world can be in. In pathfinding a node would be just a 2d coordinate of where we are at the present time. In the 8-puzzle it is the positions of all the tiles. Next all the nodes are arranged in a *graph* where links between nodes represent valid steps in solving the problem. These links are known as *edges*. In the 8-puzzle diagram the edges are shown as blue lines. See figure 1 below.

State space search, then, is solving a problem by beginning with the start state, and then for each node we expand all the nodes beneath it in the graph by applying all the possible moves that can be made at each point.

Heuristics and Algorithms

At this point we introduce an important concept, the *heuristic*. This is like an algorithm, but with a key difference. An algorithm is a set of steps which you can follow to solve a problem, which always works for valid input. For example you could probably write an algorithm yourself for

multiplying two numbers together on paper. A heuristic is not guaranteed to work but is useful in that it may solve a problem for which there is no algorithm.

We need a heuristic to help us cut down on this huge search problem. What we need is to use our heuristic at each node to make an estimate of how far we are from the goal. In pathfinding we know exactly how far we are, because we know how far we can move each step, and we can calculate the exact distance to the goal.

But the 8-puzzle is more difficult. There is no known algorithm for calculating from a given position how many moves it will take to get to the goal state. So various heuristics have been devised. The best one that I know of is known as the Nilsson score which leads fairly directly to the goal most of the time, as we shall see.

Cost

When looking at each node in the graph, we now have an idea of a heuristic, which can estimate how close the state is to the goal. Another important consideration is the cost of getting to where we are. In the case of pathfinding we often assign a movement cost to each square. The cost is the same then the cost of each square is one. If we wanted to differentiate between terrain types we may give higher costs to grass and mud than to newly made road. When looking at a node we want to add up the cost of what it took to get here, and this is simply the sum of the cost of this node and all those that are above it in the graph.

8 Puzzle

Let's look at the 8 puzzle in more detail. This is a simple sliding tile puzzle on a 3*3 grid where one tile is missing and you can move the other tiles into the gap until you get the puzzle into the goal position. See figure 1.

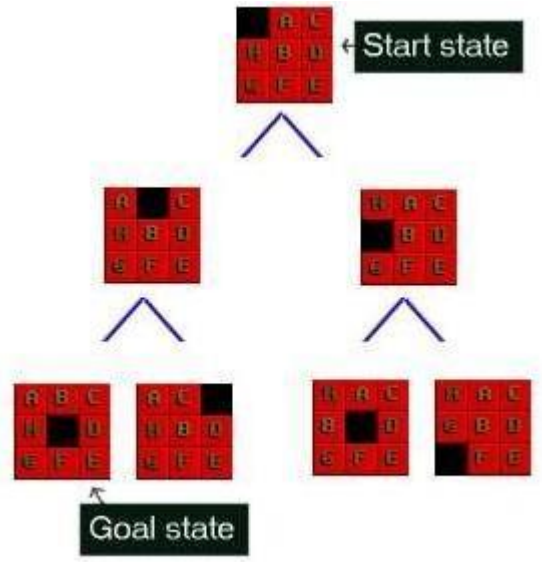


Figure 1 : The 8-Puzzle state space for a very simple example

There are 362,880 different states that the puzzle can be in, and to find a solution the search has to find a route through them. From most positions of the search the number of edges (that's the

blue lines) is two. That means that the number of nodes you have in each level of the search is 2^d where d is the depth. If the number of steps to solve a particular state is 18, then that's 262,144 nodes just at that level.

The 8 puzzle game state is as simple as representing a list of the 9 squares and what's in them. Here are two states for example; the last one is the GOAL state, at which point we've found the solution. The first is a jumbled up example that you may start from.

Start state SPACE, A, C, H, B, D, G, F, E Goal state A, B, C, H, SPACE, D, G, F, E

The rules that you can apply to the puzzle are also simple. If there is a blank tile above, below, to the left or to the right of a given tile, then you

can move that tile into the space. To solve the puzzle you need to find the path from the start state, through the graph down to the goal state.

There is example code to to solve the 8-puzzle on the github site.

4.2.3 Path finding

In a video game, or some other pathfinding scenario, you want to search a state space and find out how to get from somewhere you are to somewhere you want to be, without bumping into walls or going too far. For reasons we will see later, the A* algorithm will not only find a path, if there is one, but it will find the shortest path. A state in pathfinding is simply a position in the world. In the example of a maze game like Pacman you can represent where everything is using a simple 2d grid. The start state for a ghost say, would be the 2d coordinate of where the ghost is at the start of the search. The goal state would be where pacman is so we can go and eat him.

There is also example code to do pathfinding on the github site.

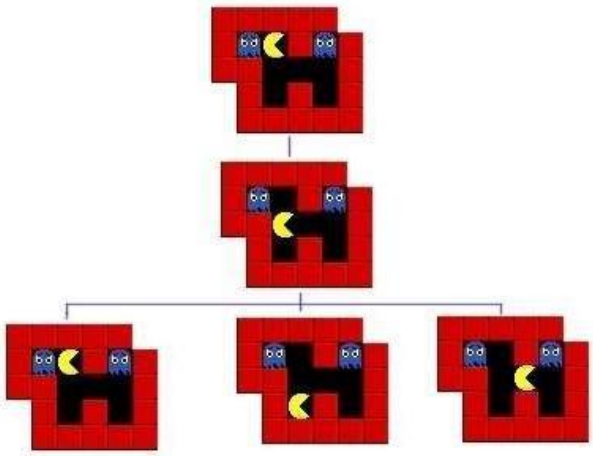


Figure: The first three steps of a pathfinding state space

Implementing A*

We are now ready to look at the operation of the A* algorithm. What we need to do is start with the goal state and then generate the graph downwards from there. Let's take the 8-puzzle in figure 1. We ask how many moves can we make from the start state? The answer is 2, there are two directions we can move the blank tile, and so our graph expands.

If we were just to continue blindly generating successors to each node, we could potentially fill the computer's memory before we found the goal node. Obviously we need to remember the best nodes and search those first. We also need to remember the nodes that we have expanded already, so that we don't expand the same state repeatedly.

Let's start with the OPEN list. This is where we will remember which nodes we haven't yet expanded. When the algorithm begins the start state is placed on the open list, it is the only state we know about and we have

not expanded it. So we will expand the nodes from the start and put those on the OPEN list too. Now we are done with the start node and we will put that on the CLOSED list. The CLOSED list is a list of nodes that we have expanded.

f = g + h

Using the OPEN and CLOSED list lets us be more selective about what we look at next in the search. We want to look at the best nodes first. We will give each node a score on how good we think it is. This score should be thought of as the cost of getting from the node to the goal plus the cost of getting to where we are. Traditionally this has been represented by the letters f, g and

h. 'g' is the sum of all the costs it took to get here, 'h' is our heuristic function, the estimate of what it will take to get to the goal. 'f' is the sum of these two. We will store each of these in our nodes.

Using the f, g and h values the A* algorithm will be directed, subject to conditions we will look at further on, towards the goal and will find it in the shortest route possible.

So far we have looked at the components of the A*, let's see how they all fit together to make the algorithm :

Pseudocode

Hopefully the ideas we looked at in the preceding paragraphs will now click into place as we look at the A* algorithm pseudocode. You may find it helpful to print this out or leave the window open while we discuss it.

To help make the operation of the algorithm clear we will look again at the 8-puzzle problem in figure 1 above. Figure 3 below shows the f,g and h scores for each of the tiles.

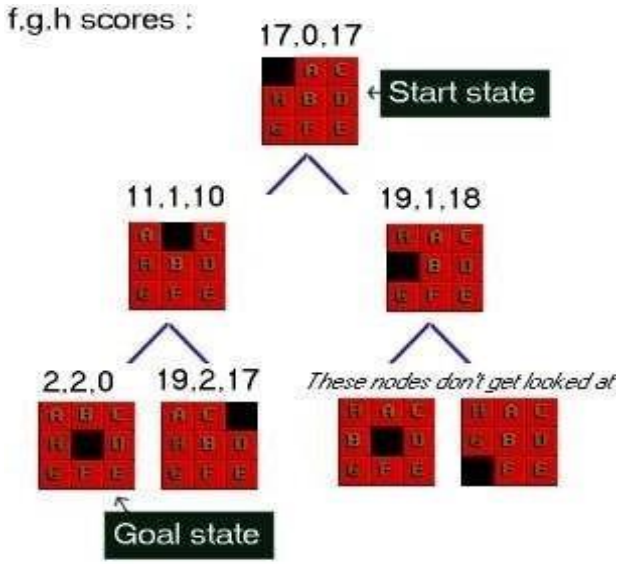


Figure : 8-Puzzle state space showing f,g,h scores

First of all look at the g score for each node. This is the cost of what it took to get from the start to that node. So in the picture the center number is g. As you can see it increases by one at each level. In some problems the cost may vary for different state changes. For example in pathfinding there is sometimes a type of terrain that costs more than other types.

Next look at the last number in each triple. This is h, the heuristic score. As I mentioned above I am using a heuristic known as Nilsson's Sequence, which converges quickly to a correct solution in many cases. Here is how you calculate this score for a given 8-puzzle state :

Advantages*:*

It is complete and optimal.

It is the best one from other techniques. It is used to solve very complex problems.

It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages*:*

This algorithm is complete if the branching factor is finite and every action has fixed cost.

The speed execution of A* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).

4.2.4 AO Search: (And-Or) Graph*

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN*:*

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE*:*

It contains the nodes that have already been processed.

6 7 : The distance from current node to goal node.

Algorithm*:*

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in

CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

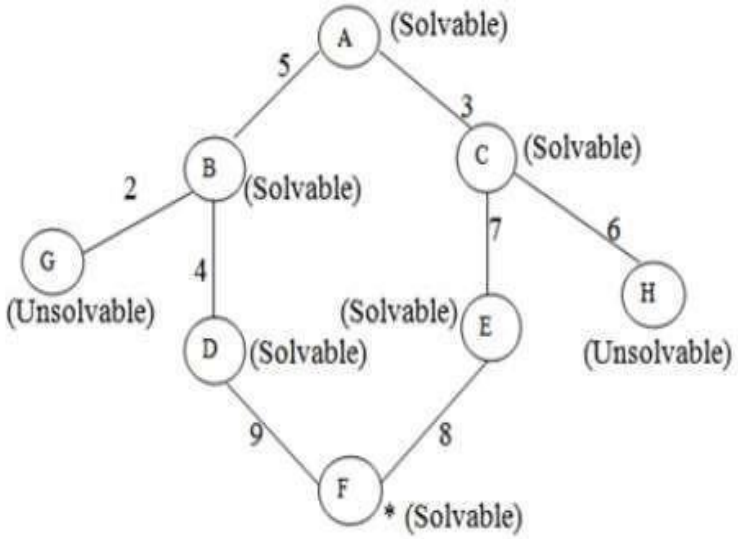
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation*:*

Let us take the following example to implement the AO* algorithm.



Figure

Step 1:

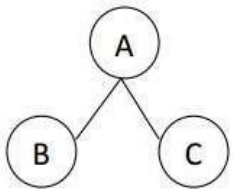
In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

i.e. OPEN = [A] CLOSE = (NULL) [ϕ] (A)

Step 2:

The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.

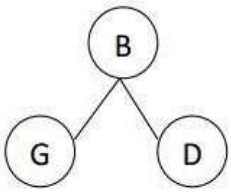
i.e. OPEN = | B | C | CLOSE = | A |

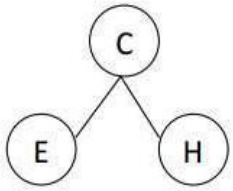


Step 3:

Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.

So OPEN = | G | D | E | | C | A | B | C |





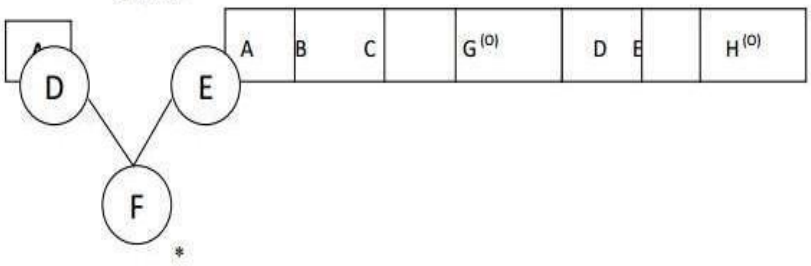
(O)

'O' indicated that the nodes G and H are unsolvable.

Step 4:

As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

i.e. OPEN = CLOSE =



Step 5:

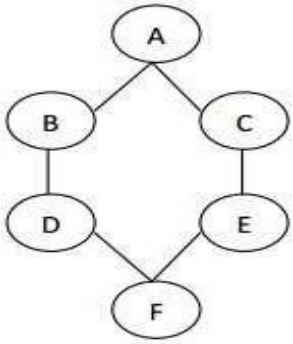
Now we have been reached at our goal state. So place F into CLOSE.

i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Figure

Advantages:

It is an optimal algorithm.

If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages*:*

Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

PROBLEM REDUCTION

Problem Reduction with AO* Algorithm.

When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a

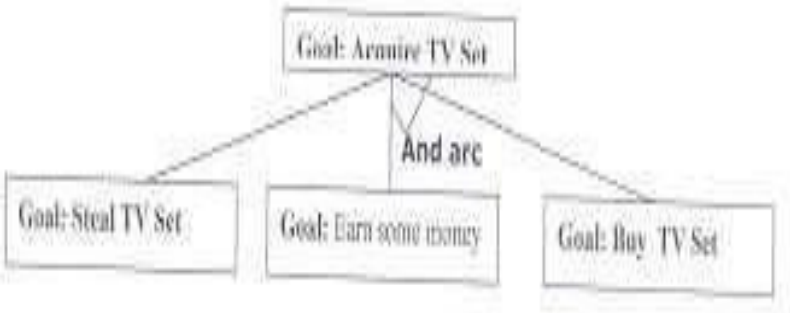


Figure shows AND - Or graph - an example.

solution, AND-OR graphs or AND - OR trees are used for representing the solution. The decomposition of the problem or problem reduction generates AND arcs. One AND are may point to any number of successor

nodes. All these must be solved so that the arc will rise to many arcs, indicating several possible solutions. Hence the graph is known as AND - OR instead of AND. Figure shows an AND - OR graph.

An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A* algorithm can not search AND - OR graphs efficiently. This can be understand from the give figure.

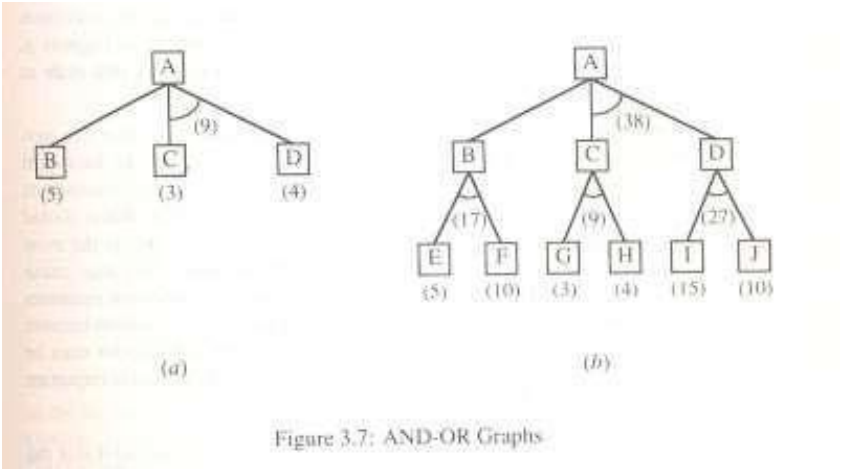


Figure 3.7: AND-OR Graphs

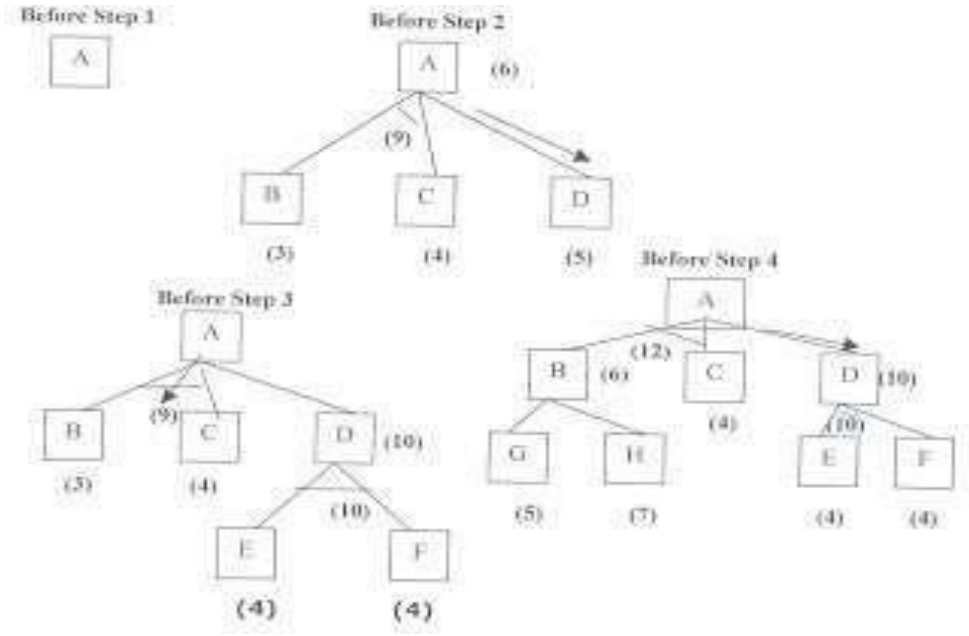
FIGURE : AND - OR graph

In figure (a) the top node A has been expanded producing two area one leading to B and leading to C-D . the numbers at each node represent the value of f ' at that node (cost of getting to the goal state from current state). For simplicity, it is assumed that every operation(i.e. applying a rule) has unit cost, i.e., each are with single successor will have a cost of 1 and each of its components. With the available information till now , it appears that C is the most promising node to expand since its f ' = 3 , the lowest but going through B would be better since to use C we must also use D' and the cost would be 9(3+4+1+1). Through B it would be 6(5+1).

Thus the choice of the next node to expand depends not only n a value but also on whether that node is part of the current best path form the initial mode. Figure (b) makes this clearer. In figure the node G appears to be the most promising node, with the least f ' value. But G is not on the current beat path, since to use G we must use GH with a cost of 9 and again this demands that arcs be used (with a cost of 27). The path from A through B, E-F is better with a total cost of (17+1=18). Thus we can see that to search an AND-OR graph, the following three things must be done.

1. traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.
2. Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f ' (cost of the remaining distance) for each of them.
3. Change the f ' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path.

The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the current best path can be selected. The working of AO* algorithm is illustrated in figure as follows:



Referring the figure. The initial node is expanded and D is Marked initially as promising node. D is expanded producing an AND arc E-F. f ' value of D is updated to 10. Going backwards we can see that the AND arc B-C is better . it is now marked as current best path. B and C have to be expanded next. This process continues until a solution is found or all paths have led to dead ends, indicating that there is no solution. An A* algorithm the path from one node to the other is always that of the lowest cost and it is independent of the paths through other nodes.

The algorithm for performing a heuristic search of an AND - OR graph is given below. Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G. G represents the part of the search graph generated so far. Each node in G points down to its immediate successors and up to its immediate predecessors, and also has with it the value of h' cost of a path from itself to a set of solution nodes. The cost of getting from the start nodes to the current node "g" is not stored as in the A* algorithm. This is because it is not possible to compute a single such value since there may be many paths to the same state. In AO* algorithm serves as the estimate of goodness of a node. Also a there should value called FUTILITY is used. The estimated cost of a solution is greater than FUTILITY then the search is abandoned as too expansive to be practical.

For representing above graphs AO* algorithm is as follows

AO* ALGORITHM:

1. Let G consists only to the node representing the initial state call this node INTT. Compute h' (INIT).

2. Until INIT is labeled SOLVED or hi (INIT) becomes greater than FUTILITY, repeat the following procedure.

(I) Trace the marked arcs from INIT and select an unbounded node NODE.

(II) Generate the successors of NODE . if there are no successors then assign FUTILITY as h' (NODE). This means that NODE is not solvable. If there are successors then for each one called SUCCESSOR, that is not also an ancester of NODE do the following

(a) add SUCCESSOR to graph G

(b) if successor is not a terminal node, mark it solved and assign zero to its h ' value.

(III) If successor is not a terminal node, compute it h' value. propagate the newly discovered information up the graph by doing the following . let S be a set of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;

(a) select a node from S call if CURRENT and remove it from S.

(b) compute h' of each of the arcs emerging from CURRENT , Assign minimum h' to CURRENT.

(c) Mark the minimum cost path a s the best out of CURRENT.

(d) Mark CURRENT SOLVED if all of the nodes connected to it through the new marked are have been labeled SOLVED. must

(e) If CURRENT has been marked SOLVED or its h ' has just changed, its new status be propagate backwards up the graph . hence all the ancestors of CURRENT are added to S.

(Refered From Artificial Intelligence TMH) **AO* Search Procedure.**

1. Place the start node on open.
2. Using the search tree, compute the most promising solution tree TP .
3. Select node n that is both on open and a part of tp, remove n from open and place it no closed.
4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.
5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.

6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.

7. Go back to step(2)

Note: AO* will always find minimum cost solution.

4.2.5 CONSTRAINT SATISFACTION:-

Many problems in AI can be considered as problems of constraint satisfaction, in which the goal state satisfies a given set of constraint. constraint satisfaction problems can be solved by using any of the search strategies. The general form of the constraint satisfaction procedure is as follows:

Until a complete solution is found or until all paths have led to lead ends, do

1. select an unexpanded node of the search graph.
2. Apply the constraint inference rules to the selected node to generate all possible new constraints.
3. If the set of constraints contains a contradiction, then report that this path is a dead end.
4. If the set of constraints describes a complete solution then report success.
5. If neither a constraint nor a complete solution has been found then apply the rules to generate new partial solutions. Insert these partial solutions into the search graph.

Example: consider the crypt arithmetic problems.

```
   SEND
+ MORE
-------

MONEY
-------
```

Assign decimal digit to each of the letters in such a way that the answer to the problem is correct to the same letter occurs more than once , it must be assign the same digit each time . no two different letters may be assigned the same digit. Consider the crypt arithmetic problem.

```
   SEND
+ MORE
-------

MONEY
-------
```

CONSTRAINTS:-

1. no two digit can be assigned to same letter.
2. only single digit number can be assign to a letter.

1. no two letters can be assigned same digit.
2. Assumption can be made at various levels such that they do not contradict each other.
3. The problem can be decomposed into secured constraints. A constraint satisfaction approach may be used.
4. Any of search techniques may be used.
5. Backtracking may be performed as applicable us applied search techniques.
6. Rule of arithmetic may be followed.

Initial state of problem.

D=?
E=?

Y=?

N=?

R=?

O=?

S=?

M=? C1=? C2=?

C1 ,C 2, C3 stands for the carry variables respectively.

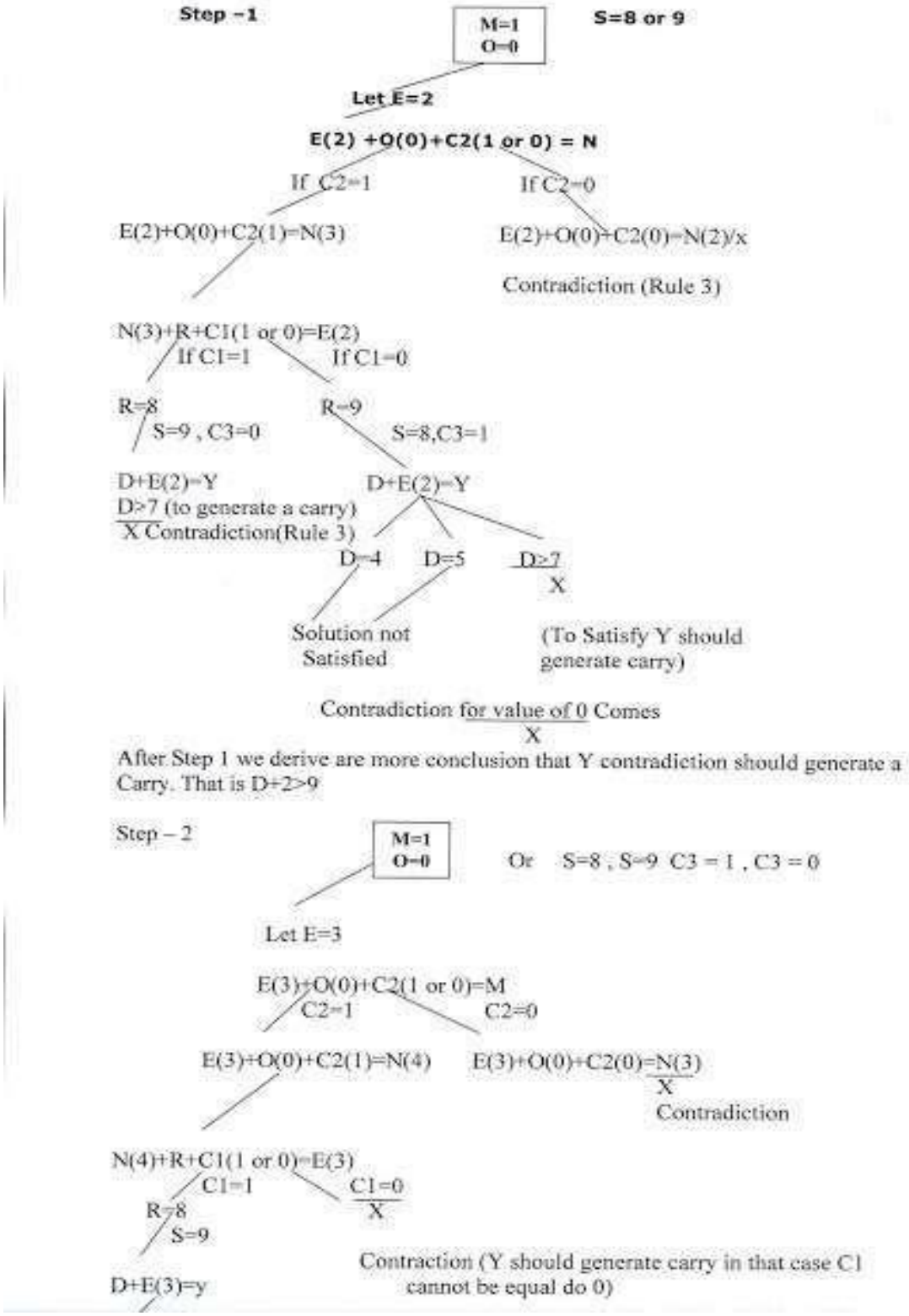
Goal State: the digits to the letters must be assigned in such a manner so that the sum is satisfied.

Solution Process:

We are following the depth-first method to solve the problem.

1. initial guess m=1 because the sum of two single digits can generate at most a carry '1'.
2. When n=1 o=0 or 1 because the largest single digit number added to m=1 can generate the sum of either 0 or 1 depend on the carry received from the carry sum. By this we conclude that o=0 because m is already 1 hence we cannot assign same digit another letter (rule no.)
3. We have m=1 and o=0 to get o=0 we have s=8 or 9, again depending on the carry received from the earlier sum.

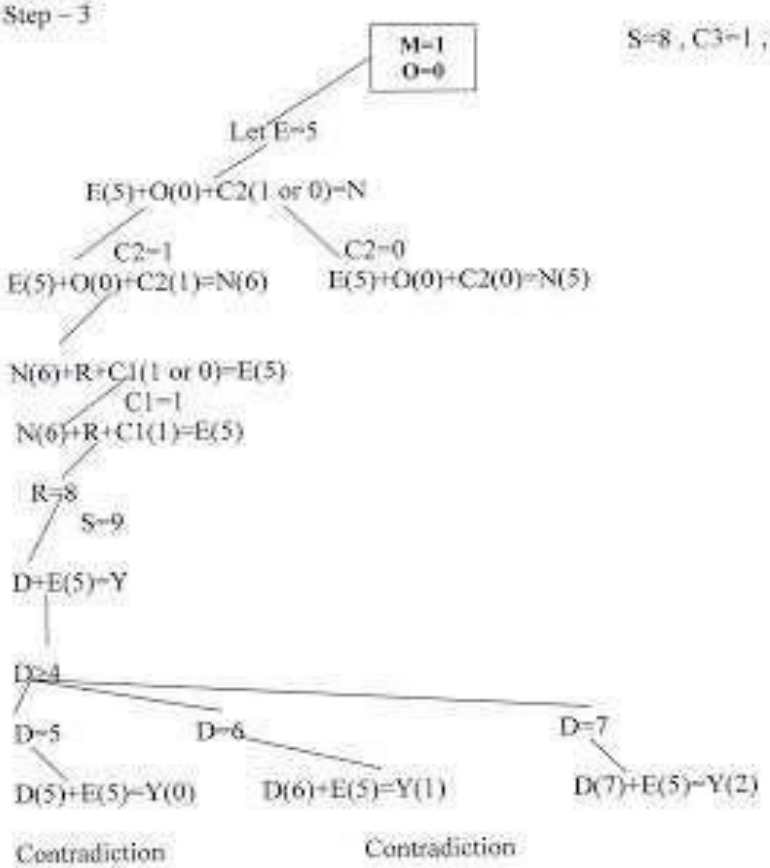
The same process can be repeated further. The problem has to be composed into various constraints. And each constraints is to be satisfied by guessing the possible digits that the letters can be assumed that the initial guess has been already made rest of the process is being shown in the form of a tree, using depth-first search for the clear understandability of the solution process.



D>6(Contradiction)

After Step 2 , we found that C1 cannot be Zero, Since Y has to generate a carry to satisfy goal state. From this step onwards, no need to branch for C1=0.

Step – 3



At Step (4) we have assigned a single digit to every letter in accordance with the constraints & production rules.

Now by backtracking , we find the different digits assigned to different letters and hence reach the solution state.

Solution State:-

Y = 2
D = 7
S = 9
R = 8
N = 6
E = 5
O = 0
M = 1
C1 = 1
C2 = 0
C3 = 0

	C3(0)	C2(1)	C1(1)	
	S(9)	E(5)	N(6)	D(7)
+	M(1)	O(0)	R(8)	E(5)
M(1)	O(0)	N(6)	E(5)	Y(2)

4.2.6 MEANS - ENDS ANALYSIS:-

Most of the search strategies either reason forward of backward however, often a mixture o the two directions is appropriate. Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems the arise when combining them together. Such a technique is called "Means - Ends Analysis".

The means -ends analysis process centers around finding the difference between current state and goal state. The problem space of means - ends analysis has an initial state and one or more goal state, a set of operate with a set of preconditions their application and difference functions that computes the difference between two state a(i) and s(j). A problem is solved using means - ends analysis by

1. Computing the current state s1 to a goal state s2 and computing their difference D12.
2. Satisfy the preconditions for some recommended operator op is selected, then to reduce the difference D12.
3. The operator OP is applied if possible. If not the current state is solved a goal is created and means- ends analysis is applied recursively to reduce the sub goal.
4. If the sub goal is solved state is restored and work resumed on the original problem.

(the first AI program to use means - ends analysis was the GPS General problem solver) means- ends analysis I useful for many human planning activities.

Consider the example of planning for an office worker. Suppose we have a different table of three rules:

1. If in out current state we are hungry , and in our goal state we are not hungry , then either the "visit hotel" or "visit Canteen " operator is recommended.
2. If our current state we do not have money , and if in your goal state we have money, then the "Visit our bank" operator or the "Visit secretary" operator is recommended.
3. If our current state we do not know where something is , need in our goal state we do know, then either the "visit office enquiry" , "visit secretary" or "visit co worker " operator is recommended.

4.3 LETS SUM UP

- Heuristic Search Techniques: Best-First Search,
- A* Algorithm, Alpha beta search algorithm, Problem Reduction,
- AO*Algorithm, Constraint Satisfaction, Means-Ends Analysis

4.4 REFERENCES

- https://www.cpp.edu/~ftang/courses/CS420/notes/uninformed%20search.pdf
- https://web.ntnu.edu.tw/~tcchiang/ai/2_Uninformed%20search%20(D).pdf
- https://web.cs.hacettepe.edu.tr/~pinar/courses/VBM688/lectures/uninformed_search.pdf
- https://www.includehelp.com/ml-ai/solving-problem-by-searching-in-artificial-intelligence.aspx
- https://www.princeton.edu/~otorres/Regression101.pdf

4.5 EXERCISES

- Solving problems by searching real time example.
- Search- Issues in the Design of Search Programs
- Take a real time example and execute Search Techniques.

❄❄❄❄❄❄❄

5

ARTIFICIAL NEURAL NETWORKS

Unit Structure

5.1 OBJECTIVES

After going through this unit, you will be able to:

- define the ANN, why we need the ANN, and architecture of ANN
- state the common characteristics in ANN
- classify different types of activation function
- explain what is Gradient decent optimization algorithm.
- Classify different types of methods

5.2 INTRODUCTION

Several developments have been made in emerging intelligent systems, some inspired by biological neural networks. Scholars from many scientific disciplines are designing artificial neural networks (ANNs) to resolve a variety of problems in pattern recognition, prediction, optimization, associative memory, and Conventional approaches have been proposed for solving these problems. Although successful applications can be found in certain well-constrained environments, none is flexible enough to perform well outside its domain. ANNs provide exciting alternatives, and many applications could benefit from using them.' We discuss the motivations behind the development of ANN" s, describe the basic biological neuron and the artificial computational model, outline network architectures and learning processes, and present some of the most commonly used ANN models. We conclude with character recognition, a successful ANN application.

5.2.1 From Biological to Artificial Neurons

Surprisingly, the ANNs have been around for a long time: first introduced in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts. In their landmark paper, 2 "Logical Calculus of Ideas Immanent in Nervous Activity," McCulloch and Pitts introduced a simplified computer model of how biological neurons can work together in the animal brain to perform complex calculations using propositional logic. This was the first structure of an artificial neural network. Since then, many other buildings have been established, as we shall see.

The first success of the ANNs until the 1960s led to the widespread belief that we would soon be talking to very intelligent machines. When it became clear that this promise would not be fulfilled (at least for a long time), funding flew elsewhere and the ANNs entered a long dark period. In the early 1980s, there was a renewed interest in ANNs as new network structures were developed and better training methods were developed. But in the 1990s, powerful mechanical teaching techniques such as Supporting Machines were popular with many researchers, as they seemed to provide better results and stronger theoretical foundations. Finally, we now see another wave of interest in the ANNs. Will this wave end as it did in the past? There are several good reasons to believe that this one is unique and will have a profound effect on our lives: There is now a huge

quantity of data available to train neural networks, and ANNs frequently outperform other ML techniques on very large and complex problems.

- The tremendous increase in computing power since the 1990s now makes it possible to train large neural networks in a reasonable amount of time. This is in part due to Moore's Law, but also thanks to the gaming industry, which has produced powerful GPU cards by the millions.
- The training algorithms have been improved. To be fair they are only slightly different from the ones used in the 1990s, but these relatively small tweaks have a huge positive impact.
- Some theoretical limitations of ANNs have turned out to be benign in practice. For example, many people thought that ANN training algorithms were doomed because they were likely to get stuck in local optima, but it turns out that this is rather rare in practice (or when it is the case, they are usually fairly close to the global optimum).
- ANNs seem to have entered a virtuous circle of funding and progress. Amazing products based on ANNs regularly make the headline news, which pulls more and more attention and funding toward them, resulting in more and more progress, and even more amazing products.

5.2.2 WHY ARTIFICIAL NEURAL NETWORKS?

The development of the new age in the present time has given the human brain many desirable features that do not exist in von Neumann or modern parallel computers. These include

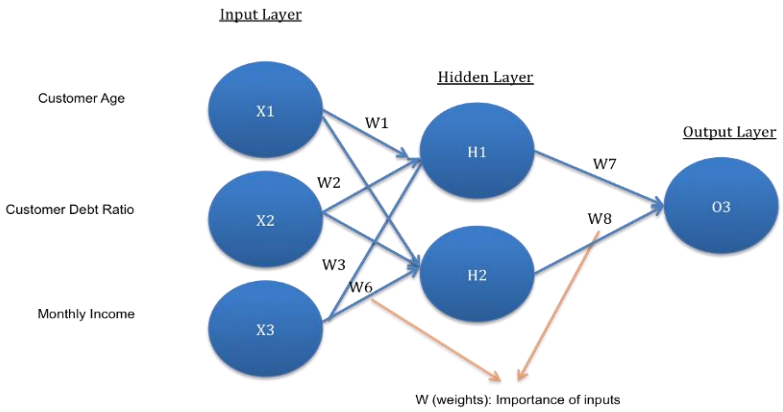
- generalization ability,
- adaptivity,
- fault tolerance, and
- learning ability,
- massive parallelism,
- distributed representation and computation,
- inherent contextual information processing,
- low energy consumption.

It contains several different processing materials (PEs) for ANN construction as an advanced machine. Each PE receives contact from it and/or other PE. Communication defines ANN topology. Signals flowing in the connection are measured by adjustable parameters called weights. PE combines all these contributions and produces an output that is an indirect (dry) sum function. The results of PEs become systemic results or are transmitted to the same PE or to others. Synthetic neural network ANN creates discriminatory functions in its PEs. ANN topology determines the number and shape of discriminatory activities. The nature of discriminatory activities changes with topology, so ANNs are considered semi-parametric class settlers. One of the average benefits of ANNs is that

they are strong enough to create 4 discriminatory jobs so that, ANNs can achieve full inclusion.

5.2.3 ANN architecture

To understand the concept of the artificial neural network, we need to understand what a neural network contains. To describe a neural network that contains a large number of activated neurons, they are called units arranged in chronological order. Let's take a look at the different types of layers found in the artificial neural network. The ANN properties of the example above can be:



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

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

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

Artificial neural network captures the input and calculate the total input scale and includes bias. This calculation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

Determines the amount of weight transferred as an input to the output function to generate output. Activation functions determine whether the node should open or not. Only those who are expelled reach the exit layer. There are distinctive activation functions available that can be applied to the sort of task we are performing.

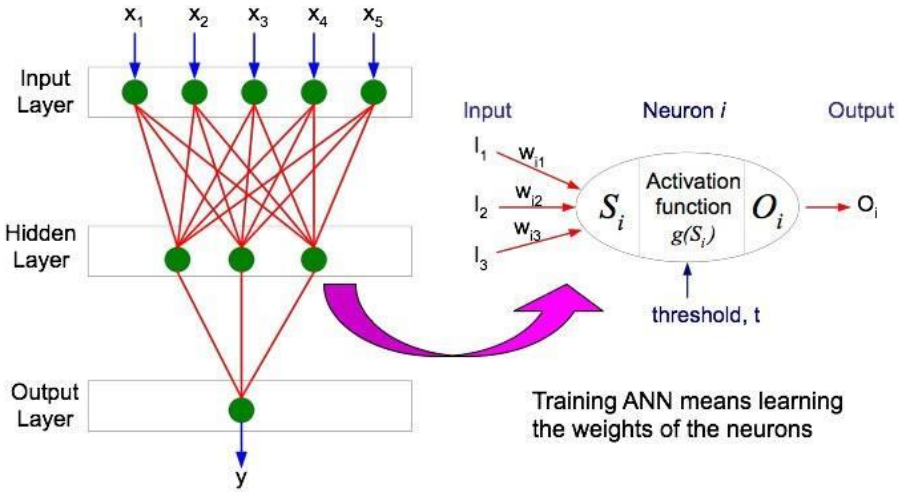
5.2.4 How does an ANN Work?

As we have seen above, the neural network contains inputs, Xs and there are certain weights assigned to this input. And naturally, we will have a model effect. This output depends on the type of business problem we have.

In the case of classifying web pages, it's a classification problem so then will have multiple classes as the output. Similar to Machine Learning, with ANN or Deep learning as well, the function is a regressor or classifier (binary or multiclass) depending on whether the output variable is continuous or categorical.

Therefore, the architecture of the Neural Network will be at least two layers: Input and Output layer. What comes between these two layers is called the Hidden Layer.

The Hidden Layer contains neurons and please note that there are no neurons in the input layer. Input layer can only be inserted, Xs. Each of these layers is the result of a previous layer and the links between each layer are shown below:



Source: faculty.juniata.edu

It can be any number of hidden layers and any number of neurons in each hidden layer. The structure of the network is defined by the user, which is why the number of hidden layers and the number of neurons in the user's vision.

Now, at the end of the day, ANN is an algorithm, so we can't build any model without having any mathematical equation behind it which brings us to the way ANN looks mathematically and what we need to solve.

A number is a combination of input lines and their weights in sequence and the learning term, which is a corrective element that weighs as much as one. The word bias is termination, why do we need a term? Because the model output cannot be zero if there are no inputs or no independent Xs or features in the model. To elaborate on that, is it possible for a store to not

have any sales if there are no factors such as categories, location, store number? We may not have these details about the store, but if the store exists it is certainly bound to make some sales even if it is a small amount.

So, the neural network equation is:

$Z = Bias + W_1X_1 + W_2X_2 + \ldots + W_nX_n$.

Z is the symbol for denotation of the above graphical representation of ANN.

W_is, are the weights or the beta coefficients

X_is, are the independent variables or the inputs

$Bias = W_0$

Each of the neurons in the hidden layer will have an equation like above which will connect between the layers and the respective weights and the bias terms. This is how the neurons get estimated and then are passed on to the next layer.

One more piece in the building of the neural network before moving to the optimizers is the activation function. The hidden layers, as well the output layer, are passed through a function called the **Activation Function.** It is an important part as it adds non-linearity to the function. It is needed because typically not every business problem can be solved linearly. So to take into account the non-linearity, we apply some mathematical transformation to the equation before the output is generated.

In the output layer, there are several activation functions and which function to use depends upon their functionality. All we need to know is that the output of the neuron is the output of the activation function.

To summarize, how does a neural network work is:

1. Each of the input-link is assigned a weight. Initially, the weights are randomly assigned. These weights are multiplied by each input value and then added together which results in the following linear combination:

$Z = W_0 + W_1X_1 + W_2X_2 + \ldots + W_nX_n$.

2. The above equation is passed through a transformation (the fancy technical word for this transformation is: the Activation or the Squashing function). The activation function depends on the type of data and problem. Hence, it is a tuning parameter for ANN.

For a binary classification problem, we know that Sigmoid is needed to transform the linear equation to a non-linear equation. Therefore, the activation function for binary classification is the Sigmoid and looks like this:

Let says, for neuron 1,

$N1 = F(Z)$

where $Z = W_0 + W_1X_1 + W_2X_2 + \ldots + W_nX_n$ which becomes:

$N1 = sigmoid(Z)$

where, sigmoid(Z) = $e^Z/(1+ e^Z)$

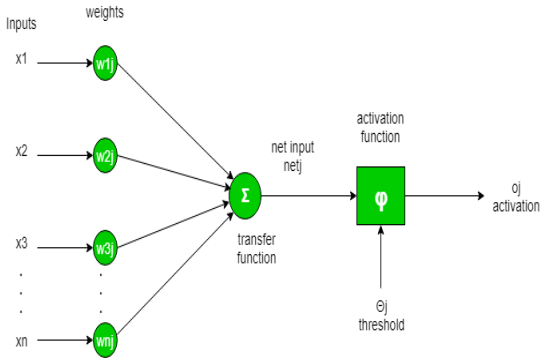
3. After applying the activation function, the output becomes:

Output = N1

If the transformed equation crosses the threshold for the Neuron then the output is class 1 else the output is class 0.

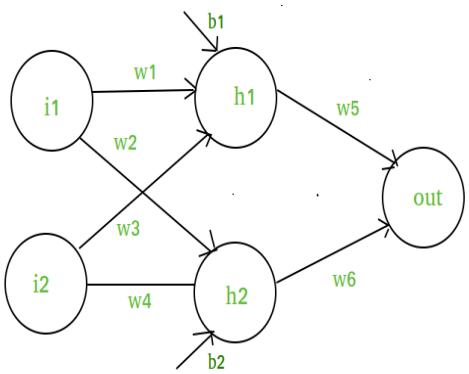
5.3 Activation Function

The activation function determines whether a neuron should be activated by calculating the amount measured and adding bias to it. The purpose of the activation function is to introduce line degradation into neuron production. We know, the neural network is made up of neurons that work in line with weight, bias, and function to coordinate their functions. In the neural network, we can review the weights and biases of neurons on the basis of error in extraction. This process is known as **back-propagation**. Activation functions make back distribution possible as gradients are provided with errors to update weights and biases. The activation function creates an indirect change in the inputs that enable it to learn and perform more complex tasks.



Mathematical proof:

Suppose we have a Neural net like this:
Elements of the Activation function Structure:
Layer 1 i.e. Hidden layer: -



z(1) = w(1)X + b(1)
a(1) = z(1)
Here,

- z(1) is the vectorized output of layer 1
- w(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4
- X be the vectorized input features i.e. i1 and i2
- b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2
- a(1) is the vectorized form of any linear function.

Layer 2 i.e. output layer: -

// **Note :** Input for layer 2 is output from layer 1
z(2) = W(2)a(1) + b(2)
a(2) = z(2)

Calculation at Output layer:

// Putting value of z(1) here
z(2) = (W(2) * [W(1)X + b(1)]) + b(2)
z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]
Let,
[W(2) * W(1)] = W
[W(2)*b(1) + b(2)] = b
Final output : z(2) = W*X + b

Which is again a linear function

This view also results in inline function even after inserting a hidden layer, so we can conclude that no matter how many hidden layers we attach to the neural network, all layers will behave the same way because the two linear functions are the function of the line itself. A neuron cannot learn about the function of the line just attached to it. The non-line function will allow it to read according to the error of w.r.t. So, we need work to make it work

5.3.1 MODIFICATIONS OF ACTIVATION FUNCTION: -

1) Linear Function: -

- **Equation:** Linear function has the equation like to as of a straight line i.e., **y = ax**
- No matter how many layers we have, if all are linear in nature, the ultimate activation function of the last layer is nothing but just a linear function of the input of the first layer.
- **Uses: Linear activation function** is used at just one place i.e., output layer.
- **Range:** -inf to +inf

- **Issues:** If we will differentiate linear function to bring non-linearity, the result will no longer depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

2) Sigmoid Function: -

- It is a function that is plotted as an **S-shaped** graph.
- **Uses:** Usually used in the output layer of binary categorization, the result can be predicted easily to be ***1*** if the value is greater than **0.5** and ***0*** otherwise. where the result is either 0 or 1, as the value for sigmoid function lies between 0 and 1 only so,
- **Value Range:** 0 to 1
- **Equation:** $A = 1/(1 + e^{-x})$
- **Nature:** Non-linear. Notice that X values lie between -2 to 2, Y values are very steep. This means small changes in x would also bring about large changes in the value of Y.

3) Tanh Function: -

The activation that works nearly continually better than the sigmoid function is the Tanh function also recognized as the **Tangent Hyperbolic function**. It's really a mathematically shifted form of the sigmoid function. Both are analogous and can be consequent from each other.

- **Equation:** $f(x) = \tanh(x) = 2/(1 + e^{-2x}) - 1$
- OR
- $\tanh(x) = 2 * \text{sigmoid}(2x) - 1$
- **Value Range:** -1 to +1
- **Uses:** Usually used in hidden layers of a neural network as its values lie between **-1 to 1** hence the mean for the hidden layer comes out to be 0 or very close to it, hence helping in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.
- **Nature:** non-linear

4) RELU: - (*Rectified linear unit*)

Stands for the ***Rectified linear unit.*** It is the most widely used activation function. Mainly executed in *hidden layers* of Neural network.

- **Equation:** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range:** [0, inf)
- **Uses:** RELU is less computationally exclusive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation. In simple words, RELU learns *much faster* than the sigmoid and Tanh function.

- **Nature:** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the RELU function.

5) **SoftMax Function:** The SoftMax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

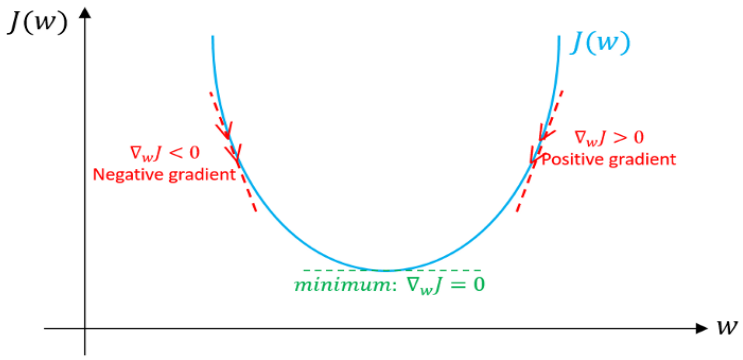
- **Nature:** non-linear
- **Uses:** Usually used when trying to handle multiple classes. The SoftMax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:** The SoftMax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

5.4 OPTIMIZATION ALGORITHM- GRADIENT DESCENT

The objective of any optimizer is to abate the loss function or the error term. The loss function evaluates the distance of the observed value from the predicted value. A loss function must have two qualities: it must be constant and differentiable at each point. To minimize the loss created from any model, we need two effects:

1) The scale that is by how much amount to decrease or increase, and
2) The path in which to move

Gradient Descent has been doing a fairly good job in helping with these two necessities. How Gradient Descent helps is:



Source: miro.medium.com

Using Gradient Descent, we get the formula to update the weights or the beta coefficients of the equation we have in the form of $Z = W_0 + W_1X_1 + W_2X_2 + \ldots + W_nX_n$.

$W_{new} = W_{old} – (\alpha * dL/dw)$

where,

W_{new} = the new weight of Xi

W_{old} = the old weight of the Xi

α = learning rate

dL/dw is the partial derivative of the loss function for each of the Xs. It is the rate of change of the loss function to the change in weight.

Learning rate and dL/dw help us with the two requirements to minimize the loss function:

- **Learning rate:** answers the magnitude part. It controls the update of the weights by telling how much amount to increase or decrease.
- **dL/dw:** conveys how much the parameter must increase or decrease. It indicates the direction by its sign.

It is an iterative process to find the parameters (or the weights) that converge with the optimum solution. The optimum solution is where the loss function is minimized.

Now, as we know there can be many hidden layers and neurons in the neural network. We do not consider the same weight of the same number. However, weights are attached to each neuron in each layer, and that from the output layer returns to the original input layer. In such a case, Gradient Descent weights are missing in two key areas:

Gradient Descent gets stuck at Local Minima

- The Gradient Descent gets stuck at the local minima. The result for this is using **Stochastic Gradient with Momentum**, which routines the **weighted sum of gradients** to benefit to get out of the local minima.
- The learning rate does not change in Gradient Descent
- The learning rate in Gradient Descent is constant throughout the training process for all the parameters. This can slow the convergence. As the remedy for this, we change the optimizer from Gradient Descent to RMSProp.

Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum. A good way to make sure gradient descent runs properly is by plotting the cost function as the optimization runs. Put the number of iterations on the x-axis and the value of the cost function on the y-axis. This helps you see the value of your cost function after each iteration of gradient descent, and provides a way to easily spot how appropriate your learning rate is. You can just try different values for it and plot them all together. There are three popular types of gradient descent that mainly differ in the amount of data they use:

5.4.1 BATCH GRADIENT DESCENT

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

Let $h_\theta(x)$ be the hypothesis for linear regression. Then, the cost function is given by:
Let Σ represents the sum of all training examples from i=1 to m.

$J_{train}(\theta) = (1/2m)\ \Sigma(h_\theta(x^{(i)}) - y^{(i)})^2$

Repeat

{

$\theta j = \theta j - (\text{learning rate}/m) * \Sigma(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

For every j =0 …n

}

Where $x_j^{(i)}$ Represents the j^{th} feature of the i^{th} training example. So if *m* is very large(e.g. 5 million training samples), then it takes hours or even days to converge to the global minimum. That's why for large datasets, it is not recommended to use batch gradient descent as it slows down the learning.

5.4.2 STOCHASTIC GRADIENT DESCENT

By contrast, stochastic gradient descent (SGD) does this for each training example within the dataset, meaning it updates the parameters for each training example one by one. One advantage is the frequent updates allow us to have a pretty detailed rate of improvement. Randomly shuffle the data set so that the parameters can be trained evenly for each type of data. As mentioned above, it takes into consideration one example per iteration.

Hence,
Let $(x^{(i)},y^{(i)})$ be the training example

$Cost(\theta, (x^{(i)},y^{(i)})) = (1/2)\ \Sigma(h\theta(x^{(i)}) - y^{(i)})^2$

$J_{train}(\theta) = (1/m)\ \Sigma\ Cost(\theta, (x^{(i)},y^{(i)}))$

Repeat

{

For i=1 to m{

$\theta_j = \theta_j - (\text{learning rate}) * \Sigma(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

For every j =0 …n

}

}

5.4.3 MINI-BATCH GRADIENT DESCENT

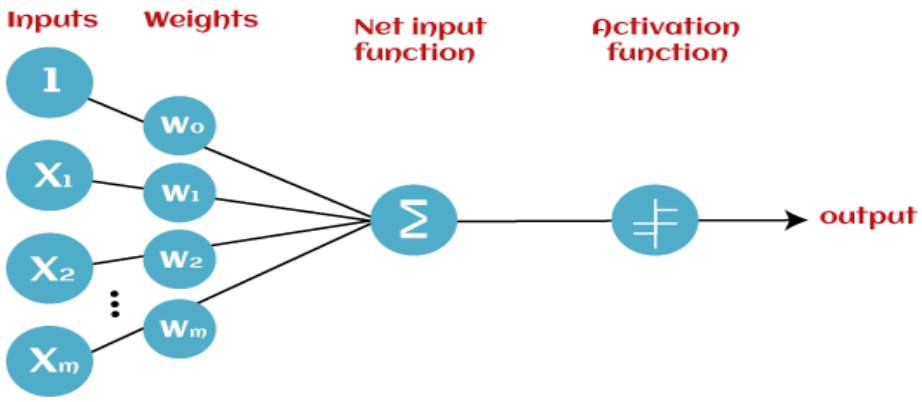
Mini-batch gradient descent is the go-to scheme since it's a grouping of the notions of SGD and batch gradient descent. It basically splits the training dataset into small batches and performs an update for each of those batches. This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

Say b be the no of examples in one batch, where b < m. Assume b = 10, m = 100; **Note:** However, we can adjust the batch size. It is generally kept as the power of 2. The reason behind it is that some hardware such as GPUs achieves better run time with common batch sizes such as the power of 2.

Repeat {
For i=1,11, 21,..,91
Let Σ be the summation from i to i+9 denoted by k.
$\theta_j = \theta_j - (\text{learning rate/size of (b)}) * \Sigma(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$
For every j =0 …n
}

5.5 NETWORKS- PERCEPTRONS

In machine learning and artificial intelligence, Perceptron is the most widely used term for all people. It is the first phase of machine learning and in-depth learning technology, consisting of a set of weights, input values or points, and limits. Perceptron is the architect of the Artificial Neural Network. Perceptron to develop specific strategies for acquiring enterprise data capabilities or business intelligence. Perceptron is a machine learning algorithm used for supervised reading of various binary class dividers. This algorithm is suitable for neurons to learn important things and process them one by one during preparation. The Perceptron model is also maintained as one of the best and simplest forms of Artificial Neural networks. But a supervised learning algorithm for binary class dividers. Therefore, we can view it as a single-layer neural network with four main components, namely, input values, weights and Bias, total volume, and opening function. Mr. Frank Rosenblatt has developed the perceptron model as a binary split system consisting of three main components. These are as follows



- **Input Nodes or Input Layer:**

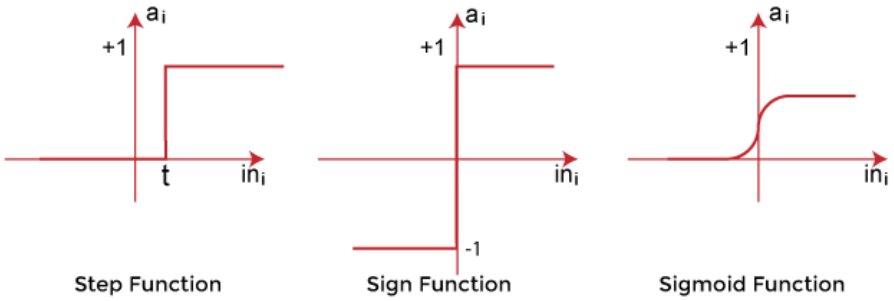
  This is the primary module of Perceptron which takes the initial statistics into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

  Weight parameter represents the strength of the connection between units. This is the alternative most important constraint 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:**

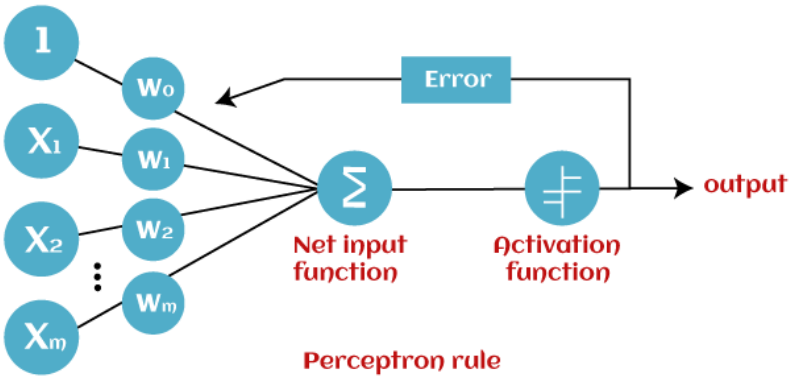
  These are the last and important components that help determine whether a neuron will burn or not. The Activation function can be viewed primarily as a step function.



A data scientist uses the activation function to make a flexible decision based on statements of various problems and to create the results you want. The activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by assessing whether the learning process is slow or has progressive or explosive gradients.

5.5.1 Process of Perceptron work

In Machine Learning, Perceptron is considered to be a single-layer neural network consisting of four main parameters named input values (Input nodes), weights and Bias, total cost, and activation function. The perceptron model starts by multiplying all the input values and their weights, and then adding these values together to form a measured value.



Then this measured amount is used in the 'f' activation function to find the output you want. This activation function is also known as a step function and is represented by 'f'. This action step or Activity function plays an

important role in ensuring that the output is mapped between the required values (0,1) or (-1,1). It is important to note that the input weight indicates the strength of the node. Similarly, the input value of the input provides the ability to change the function curve upwards or downwards.

Perceptron model works in two important steps as follows:

Step-1

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

∑wi*xi = x1*w1 + x2*w2 +…wn*xn

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

∑wi*xi + 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(∑wi*xi + b)

5.5.2 Types of Perceptron Models

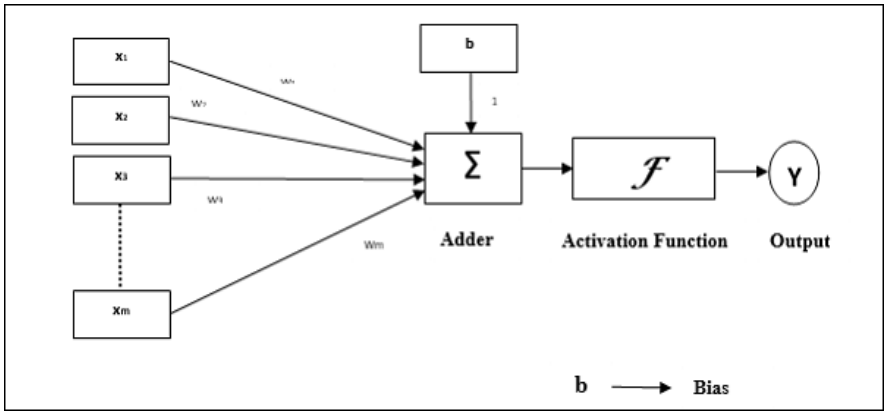
Based on the layers, Perceptron models are divided into two types. These are as follows:

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

1. 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.

Operational characteristics of the perceptron consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1. The following figure gives a schematic representation of the perceptron.



Perceptron thus has the following three basic elements −

- **Links** − It would have a set of connection links, which carries a weight including a bias always having weight 1.
- **Adder** − It adds the input after they are multiplied with their respective weights.
- **Activation function** − It limits the output of neurons. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.

Training Algorithm

Perceptron networks can be trained for single output units as well as multiple output units.

Training Algorithm for Single Output Unit

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate αα

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.
Step 3 − Continue step 4-6 for every training vector **x**.
Step 4 − Activate each input unit as follows −
xi=si(i=1ton)xi=si(i=1ton)
Step 5 − Now obtain the net input with the following relation −
yin=b+∑inxi.wiyin=b+∑inxi.wi
Here **'b'** is bias and **'n'** is the total number of input neurons.
Step 6 − Apply the following activation function to obtain the final output.
f(yin)=⎧⎩⎨10−1ifyin>θif−θ⩽yin⩽θifyin<−θf(yin)={1ifyin>θ0if−θ⩽yin⩽θ−1ifyin<−θ
Step 7 − Adjust the weight and bias as follows −

Case 1 − if **y ≠ t** then,
wi(new)=wi(old)+αtxiwi(new)=wi(old)+αtxi
b(new)=b(old)+αtb(new)=b(old)+αt
Case 2 − if **y = t** then,
wi(new)=wi(old)wi(new)=wi(old)
b(new)=b(old)b(new)=b(old)
Here **'y'** is the actual output and **'t'** is the desired/target output.
Step 8 − Test for the stopping condition, which would happen when there is no change in weight.

2. Multi-Layered Perceptron Model:

A multilayer net is a net with one or more layers of nodes between the input units and output units. 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 been 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. 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; if w.x+b>0

otherwise, f(x)=0

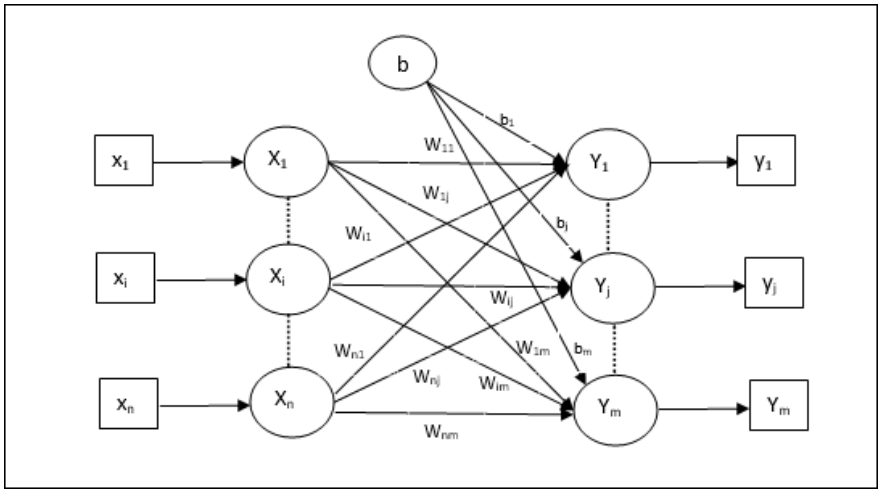
- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

A perceptron model has limitations as follows:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly

Training Algorithm

The following diagram is the architecture of perceptron for multiple output classes.



Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate αα

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.
Step 3 − Continue step 4-6 for every training vector **x**.
Step 4 − Activate each input unit as follows −
xi=si(i=1ton)xi=si(i=1ton)
Step 5 − Obtain the net input with the following relation −
yin=b+∑inxiwijyin=b+∑inxiwij
Here **'b'** is bias and **'n'** is the total number of input neurons.
Step 6 − Apply the following activation function to obtain the final output for each output unit **j = 1 to m** −
f(yin)=⎧⎩⎨10−1ifyinj>θif−θ⩽yinj⩽θifyinj<−θf(yin)={1ifyinj>θ0if−θ⩽yinj⩽θ−1ifyinj<−θ
Step 7 − Adjust the weight and bias for **x = 1 to n** and **j = 1 to m** as follows −

Case 1 − if $y_j \neq t_j$ then,

wij(new)=wij(old)+αtjxiwij(new)=wij(old)+αtjxi

bj(new)=bj(old)+αtjbj(new)=bj(old)+αtj

Case 2 − if $y_j = t_j$ then,

wij(new)=wij(old)wij(new)=wij(old)

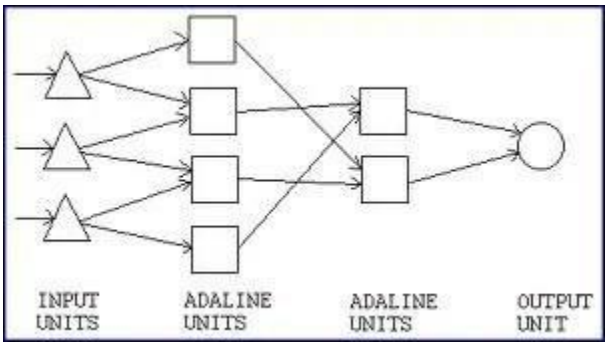
bj(new)=bj(old)bj(new)=bj(old)

Here **'y'** is the actual output and **'t'** is the desired/target output.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight.

5.6 ADALINE

Adaptive Linear Neuron (Adaline) means Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit. Adaline is a single-unit neuron, which receives input from several units and also from one unit, called bias. An Adeline model consists of trainable weights. The inputs are of two values (+1 or -1) and the weights have signs (positive or negative).



Initially, random weights are assigned. The net input calculated is applied to a quantizer transfer function (possibly activation function) that restores the output to +1 or -1. The Adaline model compares the actual output with the target output and with the bias and adjusts all the weights. It was developed by Widrow and Hoff in 1960. Some important points about Adaline are as follows −

- It uses bipolar activation function.
- It uses the delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

Mathematically, the ADALINE is described by:

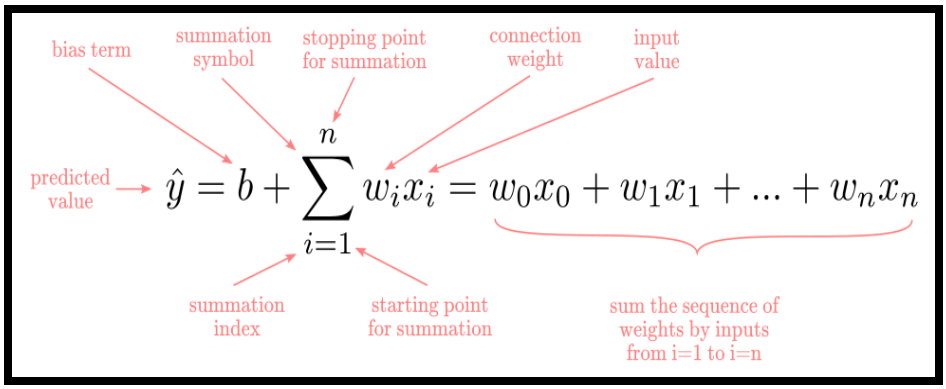
- a linear function that aggregates the input signal
- a learning procedure to adjust connection weights

Depending on the problem to be approached, a threshold function, as in the McCulloch-Pitts and the perceptron, can be added. Yet, such function is not part of the learning procedure, therefore, it is not strictly necessary to define an ADALINE.

5.6.1 Linear aggregation function

The linear aggregation function is the same as in the perceptron:

For a real-valued prediction problem, this is enough.



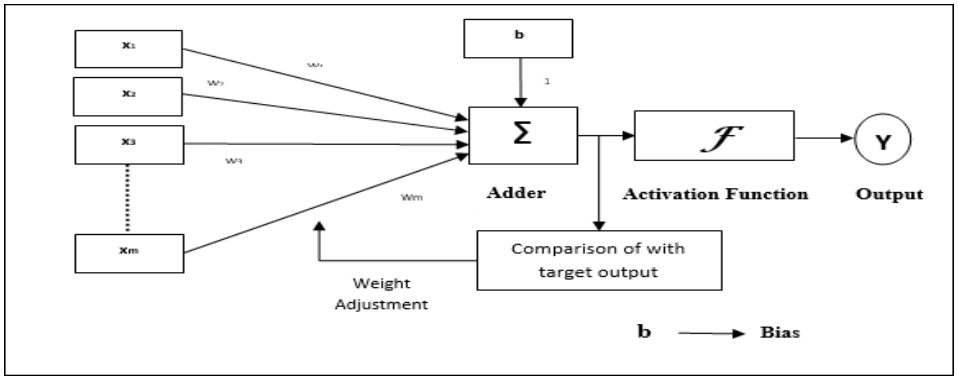
5.6.2 Threshold decision function

When dealing with a binary classification problem, we will still use a threshold function, as in the perceptron, by making the sign of the linear function as:

y^′=f(y^)={+1,−1,if y^ > 0 otherwise
where y^ is the output of the linear function.

Architecture

The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of the training algorithm, the weights and bias will be updated.



5.6.3 Training Algorithm

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate αα

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue steps 3-8 when the stopping condition is not true.
Step 3 − Continue steps 4-6 for every bipolar training pair **s:t**.
Step 4 − Activate each input unit as follows −
xi=si(i=1ton)xi=si(i=1ton)
Step 5 − Obtain the net input with the following relation −
yin=b+∑inxiwiyin=b+∑inxiwi
Here **'b'** is bias and **'n'** is the total number of input neurons.
Step 6 − Apply the following activation function to obtain the final output −
f(yin)={1−1ifyin⩾0ifyin<0f(yin)={1ifyin⩾0−1ifyin<0
Step 7 − Adjust the weight and bias as follows −
Case 1 − if **y ≠ t** then,
wi(new)=wi(old)+α(t−yin)xiwi(new)=wi(old)+α(t−yin)xi
b(new)=b(old)+α(t−yin)b(new)=b(old)+α(t−yin)
Case 2 − if **y = t** then,
wi(new)=wi(old)wi(new)=wi(old)
b(new)=b(old)b(new)=b(old)
Here **'y'** is the actual output and **'t'** is the desired/target output.
(t−yin)(t−yin) is the computed error.
Step 8 − Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

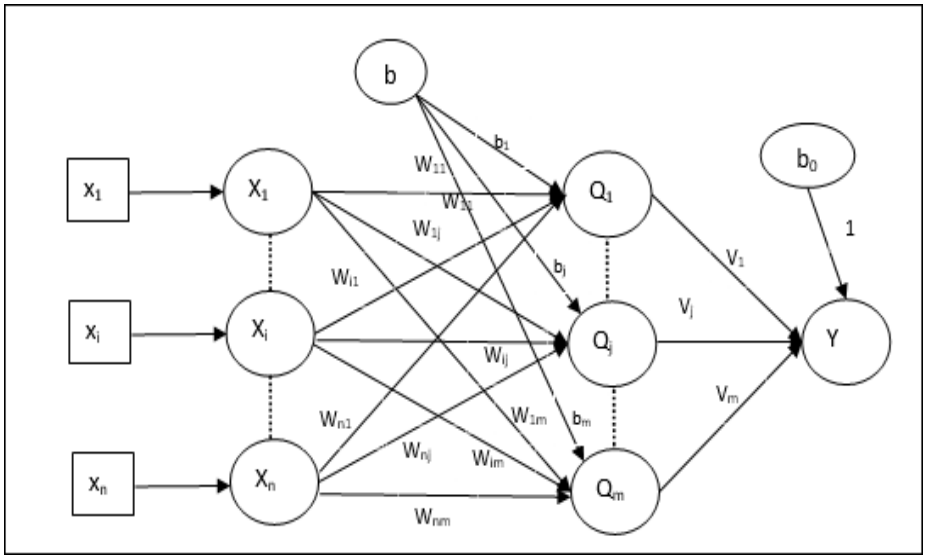
5.7 MULTILAYER PERCEPTRON'S

MALN(Madaline)which stands for Multiple Adaptive Linear Neuron, is a network system that consists of many Adaline's in parallel. It will have a single output unit. Some important points about Madaline are as follows −

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.
- The weights and the bias between the input and Adaline layers, as we see in the Adaline architecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and biases of 1.
- Training can be done with the help of the Delta rule.

5.7.1 Architecture

The architecture of Madaline consists of **"n"** neurons of the input layer, **"m"** neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.



5.7.2 Training Algorithm

By now we know that only the weights and bias between the input and the Adaline layer are to be adjusted, and the weights and bias between the Adaline and the Madaline layer are fixed.

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate αα

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-7 for every bipolar training pair **s:t**.

Step 4 − Activate each input unit as follows −

xi=si(i=1ton)xi=si(i=1ton)

Step 5 − Obtain the net input at each hidden layer, i.e. the Adaline layer with the following relation −

Qinj=bj+∑inxiwijj=1tomQinj=bj+∑inxiwijj=1tom

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output at the Adaline and the Madaline layer −

f(x)={1−1ifx⩾0ifx<0f(x)={1ifx⩾0−1ifx<0

Output at the hidden (Adaline) unit

Qj=f(Qinj)Qj=f(Qinj)

Final output of the network

y=f(yin)y=f(yin)

i.e. yinj=b0+∑mj=1Qjvjyinj=b0+∑j=1mQjvj

Step 7 − Calculate the error and adjust the weights as follows −

Case 1 − if **y ≠ t** and **t = 1** then,

wij(new)=wij(old)+α(1−Qinj)xiwij(new)=wij(old)+α(1−Qinj)xi

bj(new)=bj(old)+α(1−Qinj)bj(new)=bj(old)+α(1−Qinj)

In this case, the weights would be updated on $\mathbf{Q_j}$ where the net input is close to 0 because **t = 1**.

Case 2 − if **y ≠ t** and **t = -1** then,

wik(new)=wik(old)+α(−1−Qink)xiwik(new)=wik(old)+α(−1−Qink)xi

bk(new)=bk(old)+α(−1−Qink)bk(new)=bk(old)+α(−1−Qink)

In this case, the weights would be updated on $\mathbf{Q_k}$ where the net input is positive because **t = -1**.

Here **'y'** is the actual output and **'t'** is the desired/target output.

Case 3 − if **y = t** then

There would be no change in weights.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

The Multilayer Perceptron (MLP) is a current supervised learning technique in ANN whose architecture has been utilized for several forecasting problems in the literature. It is a distributed mathematical model enthused by the actions of the human brain and nervous system. The MLP basically consists of three layers; the input layer, the hidden layer, and the output layer. The hidden layer may have one or more activation functions (s).

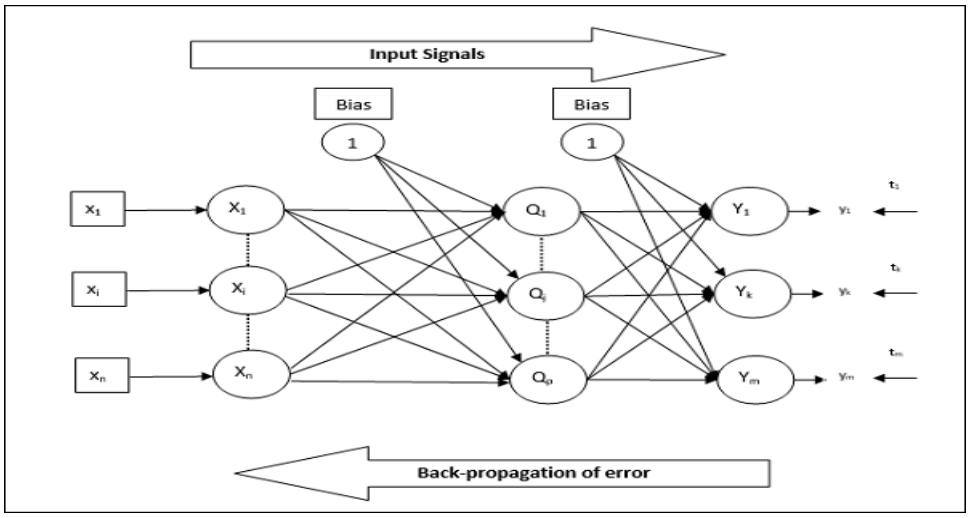
5.8 Backpropagation Algorithms

Back Propagation Neural (BPN) is a multi-layered neural network that includes an input layer, at least one hidden layer, and an output layer. As its name suggests, back distribution will occur in this network. The error listed in the output layer, by comparing the target output with the actual output, will be streamed back to the input layer.

5.8.1 Architecture

As shown in the diagram, the BPN configuration has three interconnected layers that weigh on them. The hidden layer, as well as the exit layer, is also biased, its weight remains 1, for them. As can be seen in the diagram, BPN performance is in two stages. One section sends a signal from the

input layer to the output layer, and another phase back transmits the error from the output layer to the input layer.



5.8.2 Training Procedures

For training, BPN will use a binary sigmoid activation function. The training of BPN will have the subsequent three stages.

- **Phase 1** − Feed Forward Segment
- **Phase 2** − Back Propagation of error
- **Phase 3** − Updating of weights

All these steps will be determined in the algorithm as follows

Step 1 − Set the following to start the training −

- Weights
- Learning rate αα

For easy computation and easiness, take some insignificant random values.

Step 2 − Continue steps 3-11 when the ending condition is not true.
Step 3 − Continue steps 4-10 for every keeping fit pair.

Phase 1

Step 4 - Each input unit receives a xi signal and sends it to a hidden unit for all = 1 to n

Step 5 − Calculate net-input in a hidden area via the resulting relationships –

Qinj=b0j+∑i =1nxivijj=1topQinj=b0j+∑i=1nxivijj=1top

Here b0j is a hidden unit bias, Vij is the weight of the j unit of the hidden layer from the i-unit of the input layer.

Now combine the output of the net by inserting the following opening functionQj=f(Qinj)Qj=f(Qinj)

Send these output signals of the hidden layer units to the output layer units.

Step 6 − Calculate the net input at the output layer unit via the subsequent relation −

yink=b0k+∑j=1pQjwjkk=1tomyink=b0k+∑j=1pQjwjkk=1tom

Here $\mathbf{b_{0k}}$ is the bias on the output unit, $\mathbf{w_{jk}}$ is the weight on **k** unit of the output layer coming from **j** unit of the hidden layer.

Calculate the net output by applying the following activation function

yk=f(yink)yk=f(yink)

Phase 2

Step 7 − Compute the error-correcting term, in correspondence with the target pattern received at each output unit, as follows −

δk=(tk−yk)f′(yink)δk=(tk−yk)f′(yink)

On this basis, update the weight and bias as follows −

Δvjk=αδkQijΔvjk=αδkQij

Δb0k=αδkΔb0k=αδk

Then, send δkδk back to the hidden layer.

Step 8 − Now each hidden unit will be the sum of its delta inputs from the output units.

δinj=∑k=1mδkwjkδinj=∑k=1mδkwjk

Error term can be calculated as follows −

δj=δinjf′(Qinj)δj=δinjf′(Qinj)

On this basis, update the weight and bias as follows −

Δwij=αδjxiΔwij=αδjxi

Δb0j=αδjΔb0j=αδj

Phase 3

Step 9 − Each output unit ***(y_kk = 1 to m)*** updates the weight and bias as follows −

vjk(new)=vjk(old)+Δvjkvjk(new)=vjk(old)+Δvjk

b0k(new)=b0k(old)+Δb0kb0k(new)=b0k(old)+Δb0k

Step 10 − To each output unit ***(z_j = 1 to p)*** bring up-to-date the weight and bias as follows − wij(new)=wij(old)+Δwijwij(new)=wij(old)+Δwij

b0j(new)=b0j(old)+Δb0jb0j(new)=b0j(old)+Δb0j

Step 11 − Checked for the ending state, which may be either the numeral of epochs got or the target output matches the actual output.

Generalized Delta Learning Rule Delta rule works only for the output layer. On the further hand, the generalized delta rule, also called a back-propagation rule, is a way of generating the chosen values of the hidden layer. Similarly, we can calculate the other weight values as well. After that, we will again propagate forward and calculate the output. Again, we will calculate the error. If the error is minimum, we will stop right there, else we will again propagate backward and update the weight values. This process will keep on repeating until the error becomes minimum.

5.9 Summary

Basics of neural networks and their implementation on the automatic acquisition of process planning knowledge have been introduced. This approach overcomes the time complexity associated with the earlier attempts using machine learning techniques. The example demonstrated here shows the potential of the approach for use on real-world problems. The neural network approach uses a single methodology for generating useful inferences, rather than using explicit generalization rules. Because the network only generates inferences as needed for a problem, there is no need to generate and store all possible inferences ahead of time. In this unit, we have discussed the fundamental concepts of artificial neural networks and their various real-life especially medical applications. We have also discussed how supervised pattern recognition differs from the unsupervised pattern recognition process. And also discussed that his unit includes the four basic operations to design and run a Hopfield network. These four operations are–learning, initialization (of the network), iteration until convergence, and finally outputting the output vector.

5.10 List of References

REFERENCE BOOKS

1. James A. Freeman, David M.Skapura, "Neural Networks-Algorithms, Applications, and Programming Techniques", Pearson Education.
2. Fredric M. Ham, Ivica Kostanic, "Principles of Neurocomputing for the science of Engineering", TMCH.
3. B. YEGNANARAYANA, "Artificial Neural Networks" Prentice-Hall of India Dob& Mnm New Delhi - 11 0 001 2005.
4. **Elaine Rich, Kevin Knight, & Shivashankar** B Nair, Artificial Intelligence, McGraw Hill, 3rd ed.,2009
5. The ANN Book | R. M. Hristev |

Bibliography

1. This definition of a neural network is adapted from Aleksander and Morton (1990).
2. Konar, Amit (1999), Artificial Intelligence and Soft Computing: Behavioral & Cognitive Modeling of the Human Brain, CRC Press, Boca Raton, Florida, 2000.
3. According to Kuffler et al. (1984), the term "receptive field" was coined originally by Sherrington (1906) and reintroduced by Hartline (1940). In the context of a visual system,
4. J.A. Anderson and E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge, Mass., 1988.
5. J. Hertz, A. Krogh, and R.G. Palmer, Introduction to the "Theory of Neural Computation, Addison-Wesley, Reading, Mass.,1991.

6. Arasaratnam, I., and S. Haykin, 2009."Cubature Kalman filters," IEEE Trans.Automatic Control, vol. 54, June.

7. Arasaratnam, I., S. Haykin, and R.J. Elliott, 2007. "Discrete-time nonlinear-filtering algorithms using Gauss–Hermite quadrature," Proc. IEEE, vol. 95, pp. 953–977.

5.11 Glossary

Further Readings

Maxent algorithm, pattern recognition, Fuzzy set, Radial-basis functions, Regularization theory, Neuro dynamics, dynamic driven neural network.

Model Questions

1. Define ANN and Neural computing. Distinguish between Supervised and Unsupervised Learning.
2. Draw the basic topologies for (a) Nonrecurrent and (b) Recurrent Networks and distinguish between them.
3. Define Adaptive System and Generalization.
4. List some applications of ANNs.
5. What are the design parameters of ANN?
6. Explain the three classifications of ANNs based on their functions. Explain them in brief.
7. Define Learning and Learning Law. Distinguish between Learning and Training.
8. How can you measure the similarity of two patterns in the input space?
9. Mention the linear and nonlinear activation functions used in Artificial Neural Networks.
10. Explain in Detail how weights are adjusted in the different types of Learning Law.(Both supervised and Unsupervised)
11. Write short notes on the following.
    a. Learning Rate Parameter
    b. Momentum
    c. Stability
    d. Convergence
    e. Generalization
12. Draw the model of the MP (McCulloch Pitts) neuron and state its characteristics.
13. What are the two approaches to add a bias input?
14. Distinguish between linearly separable and nonlinearly separable problems. Give examples.

15. Define the Perceptron convergence theorem.
16. What is the XOR problem?
17. What is perceptron? Write the differences between Single Layer Perceptron (SLP) and Multilayer Perceptron (MLP).
18. Define the minimum disturbance principle.
19. Consider a 4 input, 1 output parity detector. The output is 1 if the number of inputs is even. Otherwise, it is 0. Is this problem linearly separable? Justify your answer.
20. What is a-LMS algorithm?
21. Draw the ADALINE implementation for AND and OR functions.
22. Draw the architecture of a single layer perceptron (SLP) and explain its operation. Mention its advantages and disadvantages.
23. Draw the architecture of a Multilayer perceptron (MLP) and explain its operation. Mention its advantages and disadvantages.
24. Write the algorithm of generalized delta rule (Back Propagation Algorithm).
25. What is backpropagation? Explain the backpropagation training algorithm with the help of a one-hidden layer feed-forward network.
26. Explain the effect of momentum terms and the number of samples used for training in back propagation algorithm.
27. What are Hopfield networks? Explain discrete Hopfield networks in detail.
28. What is simulated annealing? Briefly explain move - generation and move - acceptance.

❄❄❄❄❄❄❄

6

INTRODUCTION TO ML

Unit Structure

6.0 MACHINE LEARNING BASICS

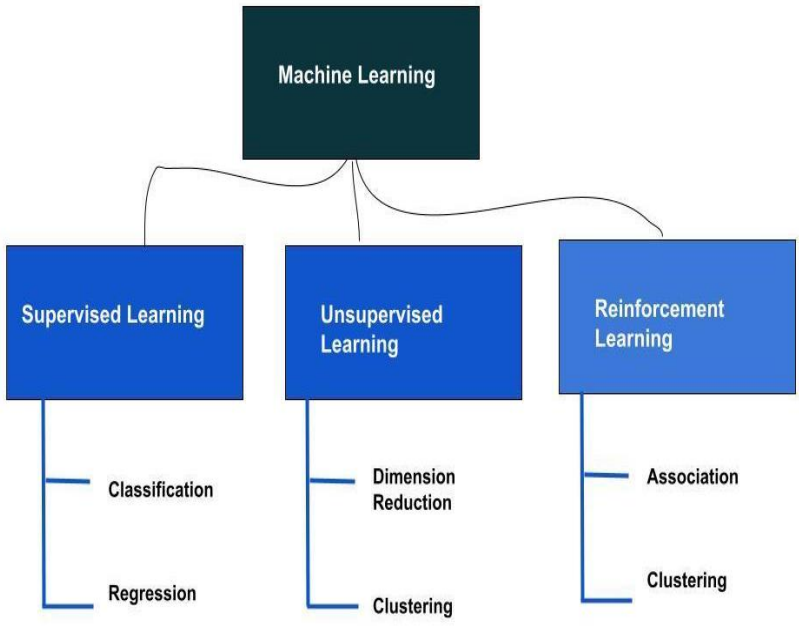
- Machine learning(ML) is nothing but making machines or computers learn.
- The natural ability of learning is present in humans which is not available in machines or computers. ML aims to do the same.
- Till now everything was handled by humans but now something new invention took place called machine learning which can behave and think like human
- Machine Learning is a branch of Artificial Intelligence (AI) and computer science. Machine learning uses algorithms and data to learn, predict everything.
- Even the problems that are faced in speech recognition or robotics can be solved using Machine learning.

- We, as humans, can solve problems by recognizing it easily but for machines to do the same it becomes quite difficult as they can't think like us.
- Intelligence should be there in machines as it is to adapt and learn according to the changing environment.
- The machine learning is not only the database related problem solving technique but also it belongs to artificial intelligence
- It can also play an important role in solving vision problems. We, as humans, can recognize humans very easily even if their hairstyle, outlook or dresses are changed in a very easy and effortless manner. This we do very easily as it's natural for us this same can be achieved through machines.
- Machine learning is nothing but making the machine give the accurate result using data and past experience.
- In Machine learning the target is known as label
- Machine learning makes applications more accurate.
- Self driving car, fraud detection are some of the example where the machine learning is getting involved
- There are four basic types of Machine learning approach which are as follows:

Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

- Supervised learning:
  - In supervised learning the training data is provided and based on this training data the algorithm is applied and further process is initiated.
  - Both the inputs and outputs will be specified in advance.
- Unsupervised learning:
  - Trained data set is not provided in this.
  - From the allotted data it will try to guess the pattern and other requirements which are needed.
  - Like supervised learning, unsupervised learning can’t be directly put in the classification and regression techniques.
  - Identifying the patterns is done here
- Semi-supervised learning:
  - This one is mixture of supervised learning and unsupervised learning
  - If needed then the trained data will be feeded in it and it is not necessary that it has to use only the feeded data but also it is free to use and explore any data by itself.
  - Here the less number of tagged data and many number of untagged data is considered.
- Reinforcement learning:

- In reinforcement learning no prior trained data is provided so it is bound to make decisions on the basis of past experience.
- Completing the task is the aim so it thinks and decides about what should be done to gain result in the given task.



Types of Machine learning

6.1 APPLICATIONS OF ML

6.1.1 Classification

- Machine learning can be used in classification.
- Classification is nothing but making your task to fall under some classes.
- For example let's say we want to detect whether the person is having COVID or not. In this case we have two classes with us one is COVID is present other is COVID is not present.
- So the machine learning system should have the capacity to find this on the basis of its previous datas and details that it has.
- This can also be meant as the prediction done using its previous experience and classifying it.

6.1.2 Pattern Recognition

- Pattern recognition is nothing but a trend.
- The machine should recognize the trend i.e if a particular thing is happening one by one then the machine should be able to recognize the next thing that can happen.
- Pattern recognition can be seen everywhere, we use it in our daily life.

- One good example is let's say a mother teaching alphabets to her child so every day she teaches her “A”, “B”,..... and so on. At some point as soon as the mother starts telling these alphabets the child automatically detects the next alphabet after much repetition done by the mother. This way of recognizing the next alphabet is known as pattern recognition.

6.1.3 Learning Associations

- Learning association is a data mining technique used by machine learning system
- The target of this learning association is to detect the relationship associated with a big group of datas.
- Given the large data set, the machine learning system will find out which specific item will occur when.
- A good example for this is when a person buys a brea in the shop and it is common that whenever a person buys milk he will also buy jam. That is the prediction done using learning association that whenever a person buys something he will buy some other product along with it.
- This kind of detection is nothing but learning association

6.1.4 Financial Services

- To get prevented from frauds and data analysis machine learning will be a boon
- Machine learning may be combined with data mining techniques to find the risks, high investments capability etc.

6.1.5 Government

- Machine learning plays a very important role in the government sector.
- There can be many situations like inflation, disaster when quick decisions need to be made by the government, in such scenarios machine learning comes into the picture.

6.1.6 Transportation

- Finding the best route, predicting it, analyzing and finding different routes etc. can be done by machine learning.
- Smart transportation has been initiated in the transportation sector which involves machine learning.
- Travel time, traffic information and even future traffic conditions that may arise can be found using machine learning techniques

6.1.7 Oil and gas.

- Oil and gas industries involve a very very huge amount of datas which is one of the crucial parts so it has become a major issue.
- Machine learning along with artificial intelligence can solve this.

6.2 DATA MINING VS MACHINE LEARNING VS BIG DATA ANALYTICS

6.2.1 Data mining vs Machine Learning

Data Mining	Machine Learning
From large amount of data information is fetched	From large amount of data information is fetched and also past experience is used
It is dependent on human	Not dependent on human
Humans manually insert the data for their use.	System doesn't takes any data from the user but it learns itself
Only data is present which is unstructured	Data as well as algorithms are present
Data mining can't learn or adapt	Machine learning can learn by itself and adapt too.
Models are created to use in data mining techniques	Models and algorithms are created by machine learning to apply and get used it artificial intelligence
Human effort is involved	Human efforts are not involved
Data mining is used in cluster analysis.	Machine learning can be used in fraud detection, computer design etc.
Can be used only in some parts	Can be used and applied in vast area

6.2.2 Machine learning vs Big data analytics

Machine Learning	Big Data analytics
Machine learning will not only use the current available data it can also learn new things and make decision accordingly	It is dependent on the current available experience and will make decision
It uses algorithms	It uses classifications
It's main goal is to learn	It's goal is to find pattern
Extract meaning and develop or	Will extract information from the

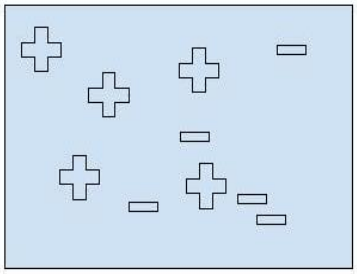
apply the algorithms	existing dataset
Tools involves machine learning algorithms and analytical models	When it comes to tools it involves analytics tools
Scope wise semi supervised, supervised and unsupervised learning is involved	Scope wise predictive and risk analysis is involved
Machine learning involves applying algorithms and coding.	Data analytics is like purification of data were whole data is considered then modified, cleaned and final data is considered

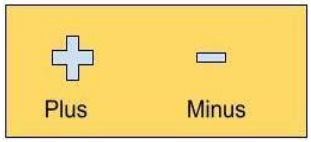
6.3 SUPERVISED LEARNING- NAÏVE BASE CLASSIFIER

6.3.1 Supervised learning

- In supervised learning the machine will already have the prior details regarding the matter
- This prior information is called as trained data
- Using this trained data the machine can take help and predict the result easily.
- Machines easily predict the output with supervised learning techniques.
- The main aim of this supervised learning is to map out the input value say i using the output value say j.
- In this we have both raw input data as well as the results.
- Mapping function is j=f(i)
- Kids learning under the supervision of parents is one example of supervised learning.
- Supervised learning techniques can be used for classification, filtering etc.
- The trained dataset has their own names or tags associated with it.
- So with the help of this naming scheme it becomes easy for the machines to identify the data uniquely.
- Refer the below diagram to understand a about how name tag done on supervised learning
- Various algorithms and techniques are supported by supervised learning
    - Naive Bayes: It is a classification step.
    - Linear regression: It is used to identify independent and dependent variable out of which it will try to find the outcome.

- K-nearest neighbor: It is a non parametric based algorithm in which nearest data is considered by applying distance formulas
- Support vector machine: This algorithm uses both classification and regression techniques.

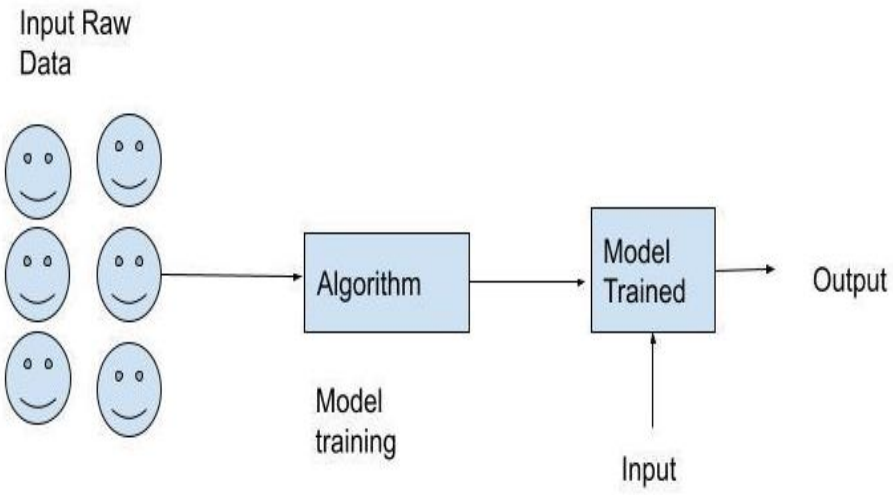




+

- Supervised learning involves following steps:

1. First the problem is considered.
2. Now the data that belongs to the problem is collected.
3. The available dataset is taken.
4. The available dataset is analyzed with the data that belong to the problem about which the solution has to be made.
5. The data is also splitted up for testing and validation.
6. Algorithms will be applied.
7. The final output will be found and if it matches the need then accuracy will be achieved.



- There are 2 types of supervised learning:
    1. Regression
    2. Classification

6.3.1.1 Application of supervised learning

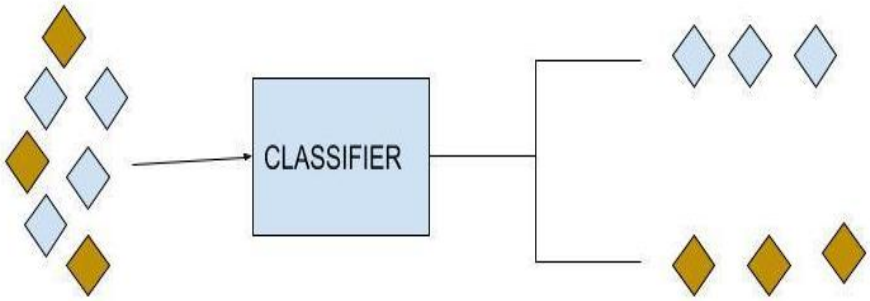
- Understanding someone's statements which can be verbal or written one, like people may complain, suggest, give opinion etc.
- Almost all ecommerce websites and applications send recommendations to us about the newly launched products or about the products on which we are interested.

6.3.1.2 Regression

- It is applied in those problems where input and output has relationship
- Also it is mostly used in real examples like “Marks”, “salary” etc.
- Some regression algorithms are as follows:
    - Linear regression
    - Bayesian Linear Regression
    - Non Linear Regression et.
- Predicting the age of a person is another example of regression.

6.3.1.3 Classification

- When categories are involved it uses classification supervised learning technique like for example
    - “pink” - “green”
    - “YES”-””No”
    - “Male”-”Female” etc.
- With the observed values it will try to classify and will find the solution
- The main role that is involved here is classifying the data and predicting the outcome.
- Refer the below figure to identify the difference between how the classification and regression looks like.
- In a simple way it can be said that it is a way in which the recognition of data is done by identifying its uniqueness by observing it then categorizing and classifying it.
- Classification in machine learning involves the usage of tagged data which is involved and input is taken and with the help of trained data it categorizes it using predetermined categories.

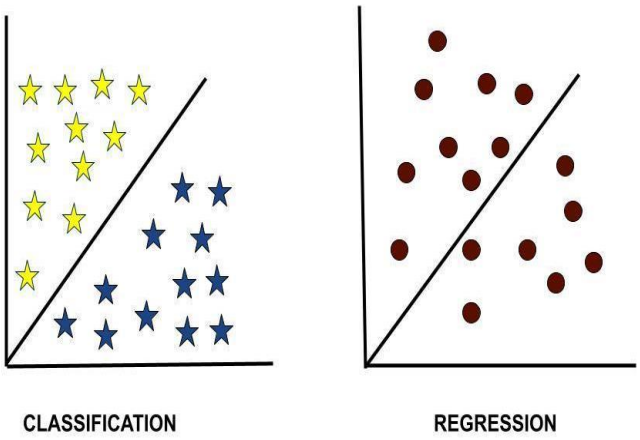


6.3.1.4 Advantages of supervised learning

- Supervised learning allows us to gather data and also it uses its earlier experience to find the result.
- Using previous experience the optimized results are achieved.
- Many real world problems that are happening currently can be solved by using supervised learning mechanisms.

6.3.1.5 Disadvantages of supervised learning

- Applying this algorithm on those data set which is having very high collection of data is difficult.
- As it needs experience, we need to train it. Training it is not that easy as we need to feed whole stuff inside it.



6.3.2 Naive base classifier

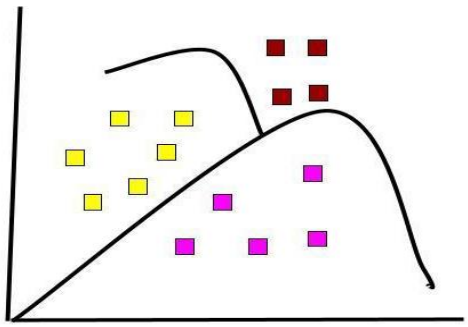
- It is one of the varieties of classification algorithm which uses Bayes' Theorem.
- As it falls under classification algorithm technique it too involves categorization.
- The Nive Bayes theorem involves following formula:

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

Where:

P(A|B): A occurrence when B occurs.
P(A): A occurrence
P(B): B occurrence
P(B|A): B occurrence when A occurs.

- Naive Base classification can be used in many situations as follows:
    - Identifying the different parts of our face like eyes, nose, ears etc,.
    - Which is nothing but face recognition. Here the classification of faces can be done.
    - Climate change recognition can be done using Naive Base algorithm like weather it's going to be hot, humid, cold etc,. Whether the climate will be good or bad etc.



NAIVE BAYES ALGORITHM

- Naive Base classification is mainly used in data which involves texts. So text based classification is widely done through this algorithm.
- The reason behind the name Naive Base is as follows:
- Naive is nothing but something which is not dependent on each other. Say it needs to identify a pencil then the categories that are involved for identification purposes are color, shape, size etc. Now if you will observe carefully then these color, shapes or size these features are not dependent on each other. So this is the reason why it is called Naive.
- The word Base or Bayes is used because it is made up of Bayes algorithms. Hence the name Naive Bayes classification.

- **Advantages of Naive Bayes classification:**
  - It is very speedy in nature.
  - It can be applied in binary.
  - In comparison with other algorithms this algorithm is more accurate.
  - Text classification will mostly be done with this classification.
- **Disadvantage of Naive Bayes classification**
  - If there is any involvement between the features of a data, naive bayes classification algorithm fails to identify it as it takes or considers all features independently. So it may fail when the relationships are involved.

Types of Naive Bayes algorithm

- Bernoulli Naive Bayes
  - In this the prediction related variables will be boolean in nature which is one of the default criteria
  - Which means true or false will be used.
- Multinomial Naive Bayes
  - Whenever document classification comes into picture this particular type of algorithm is applied
  - For example if you want to find out weather the document that is present belongs to students or staff then you may use this algorithm to find the result.
  - It will sort and will give you the final result.
  - It will take help of words present in it also the frequency is getting utilized
- Gaussian Naive Bayes
- In this the continuous value will be considered and sorted.

❄❄❄❄❄❄❄

7

K-NEAREST NEIGHBOR, DECISION TREE AND NAIVE BAYES CLASSIFICATION

Unit Structure

7.1 CLASSIFYING WITH K-NEAREST NEIGHBOR CLASSIFIER

- In this classification technique it assumes data which is near to the environment.
- It is also written as KNN which means K-Nearest Neighbor
- It falls under supervised machine learning algorithm
- The reason for having K-Nearest neighbor is as follows:

Imagine there are two categories saly category X and category Y and one another new data point say x1 got introduced, now it should be categorized under which type? This decision is made by the K-Nearest Neighbor classifier.

- Steps for applying K-Nearest Neighbor are:
  - Step 1: K is selected first
  - Step 2: The distance with the neighbors are found using Euclidean distance formula
  - Step 3: Based on the Euclidean distance formula nearest neighbors are chosen.

- Step 6: After choosing the nearest neighbor, count the number of each category and choose the one with maximum value.
- Step 5: It's ready.

- So the majority is considered and accordingly it is made.
- Consider the above diagram in which the data is categorized into star and triangle then the new point was created and its nearest neighbors are chosen with the help of Euclidean formula so after that it changed itself to triangle.
- It is also called a lazy learner classification algorithm.
- It is very difficult to find the correct point because if the newly found point is improper then it may lead to underestimated or overestimated results.
- The less value of k will lead to the noise whereas high value will lead to the expensiveness.
- Many prefer choosing the value by applying square root of N, where n is the total number of samples present in it.
- In order to gain accuracy many values of K may be taken into consideration and after that we may try to choose the one which is giving the best result. As choosing the point is one of the important aspects in K Nearest Neighbor as it may affect the result in the upcoming steps it is better to go ahead with many options instead of sticking with one value and trying to go on.
- It is non-parametric in nature, i.e is it does not take any prior value that to be implemented in the algorithm. For example if you consider linear regression then in this algorithm many assumptions and values should be applied prior to its algorithm execution whereas in K-Nearest Neighbor such parameters are not required.
- It will continuously change because it is not dependent on pre pre-made fixed data set. So the data may change from time to time and its details may change again and again but still the K-Nearest Neighbor will still work on it and will be able to achieve the end result easily as it is very adaptive in nature.
- It can be applied on binary as well as multi class problems. Actually in reality many algorithms are supported to be implemented only on binary based problems and many don't support multi class based issues. But K-Nearest Neighbor does wonders in this matter.
- K is a constant value which is defined by the user.
- It can be applied by using any of the one as formula mentioned below:
- Euclidean Distance Formula

$$d = \sqrt{[\ (x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

where,

- (x_1, y_1) are the coordinates of one point.
- (x_2, y_2) are the coordinates of the other point.
- d is the distance between (x_1, y_1) and (x_2, y_2).

- Manhattan Distance

$$Mdist = |x2 - x1| + |y2 - y1|$$

7.1.1 Advantages of K-Nearest Neighbor

- It doesn't needs any training period when it comes to most other way what happens is the data is considered which is nothing but training data and using the it analyses to make decision where as K-Nearest Neighbor will do it in real time it will not collect or have any training data for reference it will just do on the spot and will continue. This behavior of K Nearest Neighbor makes it very fast.
- Implementing KNN is very easy
- As it does not require any prior training data so it becomes easy to update dataset we can add new data very easily whenever needed in the dataset
- Implementing this algorithm doesn't require many steps.
- If the data is changing then accordingly it will adapt itself as it doesn't refer to any training data set.
- The use of different types of distance metrics makes it easy to apply like in the K-Nearest algorithm we can apply Euclidean, Minkowski, Manhattan distance etc.
- It is very simple, you just need to share the whole dataset. It will go through it and by applying an algorithm it will find the best suitable values.
- One very big and important advantage of K-Nearest Neighbor is that it can be applied on both a variety of problems i.e Classification problem as well as regression problems.
- It very flexible in nature, i.e it supports many varieties of distance formula that is available in the market like: Euclidean Distance, Manhattan Distance, Minkowski Distance etc.

7.1.2 Disadvantages of K-nearest Neighbor

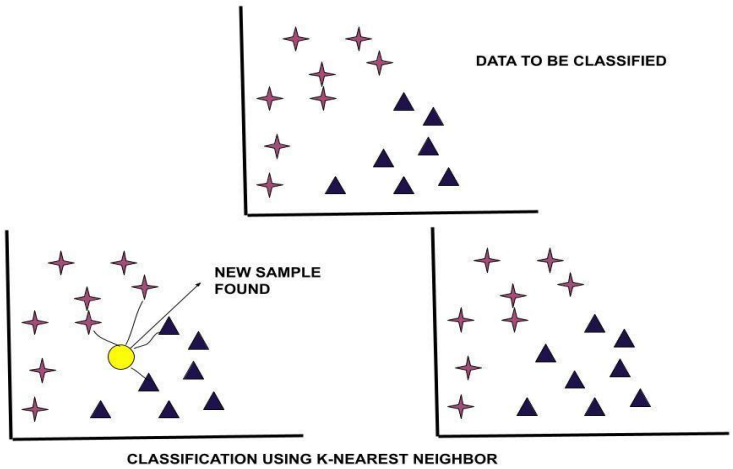
- It can't find the errors or missing data so it is manually found and filtration techniques have to be implemented.
- If the size of the dataset is high then it is very difficult to apply K-Nearest Neighbor because as soon as a new point is found it has to

go through all the nearest neighbors around it and find the distance. So this becomes a big problem and disadvantage.

- When applying KNN in a large dimensional data set it may fail in finding the best points.
- If the data will increase then the speed of calculation done by the K-Nearest Neighbor algorithm will go down.
- If the newly found value is improper then it may lead to over fit or under fit.
- Homogeneous outlook is needed because it can't take that variety of data; it needs to be applied only on those with less scale, distance and data.
- If the segregated data is not balanced then it will lead to a great issue, for example consider we have two varieties X and Z and imagine that X is majorly present whereas Z is very less in population so if the newly point is jotted using K-Nearest Neighbor then it will obviously convert itself under X.
- If a problem of missing value arises then it doesn't know how to deal with such kinds of problems.

7.1.3 KNN application

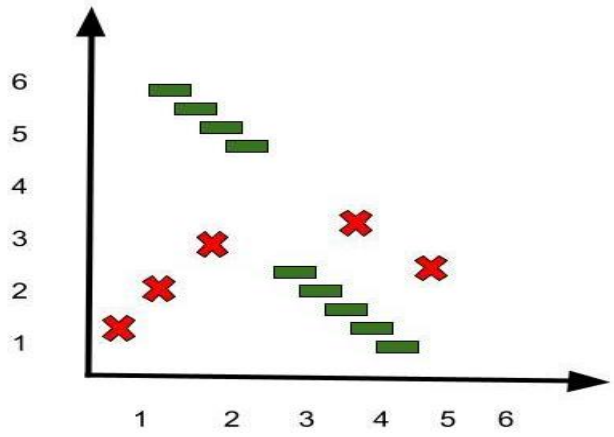
- Medicine
- Online shopping
- Data mining
- Agriculture etc.



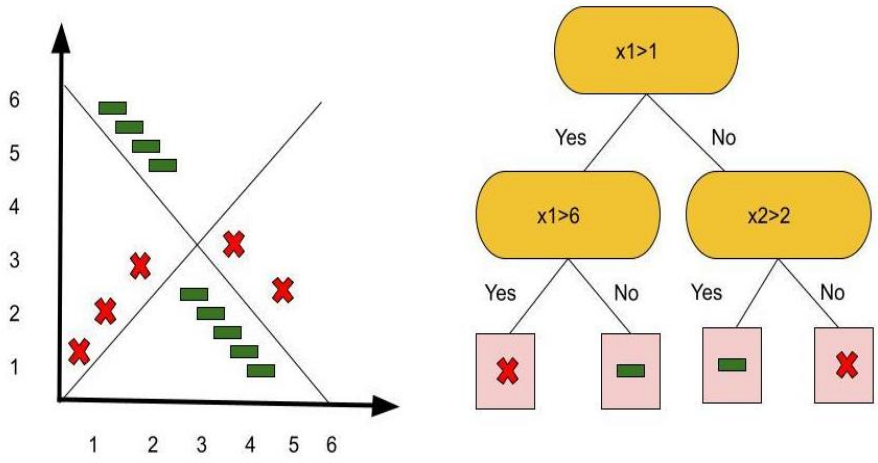
7.2 DECISION TREE CLASSIFIER

- It falls under supervised learning algorithms.
- It uses a group of protocols to make decisions like humans make decisions in real life.
- It uses non parametric methods.

- Decision trees are more famous in classification than in regression. I.e. it is more often used in classification.
- It can be used for both classification and regression
- In the decision tree the split of regions is applied which makes it useful to use and make decisions.
- The decision tree consists of terminal leaves and internal division nodes.
- Each node under decision let us consider it as m will implement function for testing as fm(x)
- From this outcome will be gained and branch labeling will be there.
- Refer the below figure where the decision tree has been created from the data set provided
- Also it is non parametric in nature which means the parameters are not provided prior instead the tree is made or it grows little by little by analyzing it, it will learn and will add up the branches, leaves etc.
- Decision tree is one of the most popular machine learning algorithms.
- It does have the capacity to work on missing and noisy data. Because of its robust nature it is still existing as it is one of the oldest algorithms too.
- When it comes to the decision tree the root node is taken first i.e the top most note is created then one by one it will go to the next levels and the remaining nodes will be created.
- This kind of creation of decision tree technique is also known as binary splitting in a recursive manner.
- The main and first step in decision tree is to apply classification
- Like the below figure the data will be given, the first step is to classify it by dividing it into a set of modules which can be seen in the second figure below once this much of steps are done then decision tree algorithm can be applied.



Classifying the initial data set



Example of Decision tree made out of the data set provided

7.2.1 Decision Tree terminologies

- Parent node: Root node is the parent node
- Child nodes: The successor node are the child nodes
- Root node: It is the first node which is located at the top of the decision tree, it represents the whole data set.
- Leaf node: These are end nodes of the decision tree, after reaching till leaf node further segregation is not possible.
- Splitting: It is a mechanism of dividing the root or the decision node according to the condition given.
- Sub tree or the branch: it is the splitted branch.
- Pruning: Removing a particular branch from the tree is known as pruning

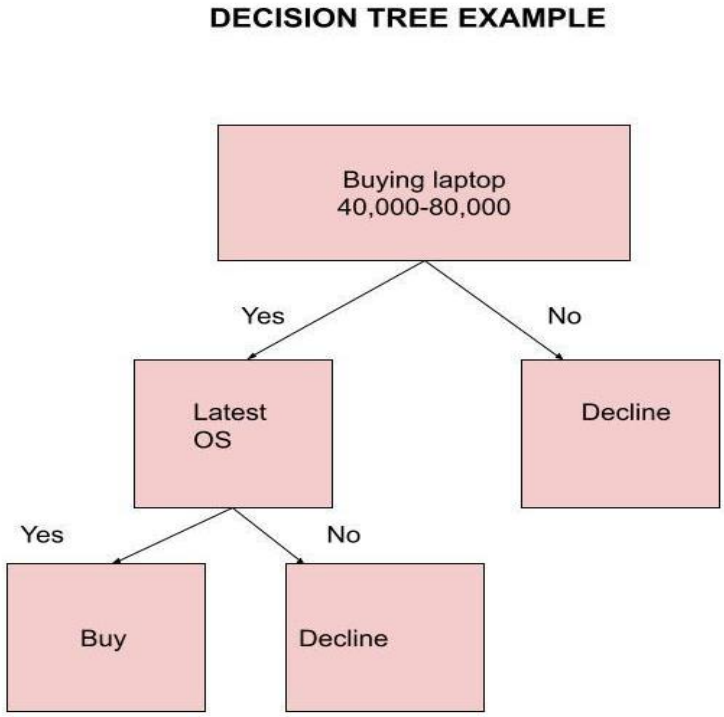
7.2.2 Working of Decision tree

- It starts with the root node then it will go through the classification that has been applied in the dataset.
- After going through the classification it will go ahead with comparison between the datas
- Based on the data it will move further if it is coming out to be of one form then accordingly one node will be created otherwise it will move on with another node.
- Like this comparison will continue again and again till it goes or reaches the end part
- End nodes are nothing but leaf nodes after which no other branches will be created.

7.2.3 Decision tree example

- Above image is a good example of a decision tree.
- The aim is to decide whether to proceed with buying a Laptop or not.

- So initially the whole data is considered which contains the laptop price, its configurations etc.
- Now the main root node is created on the basis of price, so now if you refer to the above diagram a range is taken into consideration, say 40,000 to 80,000.
- Now the checking is done on this root node, if it satisfies and gives the result as yes then it proceeds further with another node or if not satisfied then another node in the right is created and it stops there. So in this case we have created 2 nodes for yes and no. If it comes yes then it proceeds with the next node where the OS version is getting checked otherwise a declined node is initiated.
- Similarly it proceeds further with the next node, now if the node condition satisfies, i.e. if the OS i.e operating system version is latest then it moves to the next node i.e buying otherwise its stopped.So in our case if yes is getting initiated then the next node which is pointing for buying will be done otherwise it will be declined.So in this way the decision tree helps in finding out the solution in a very simple manner and in a very accurate way.



7.2.4 Univariate Trees

- In this univariate tree, the input dimensions are considered and one of them is considered to further branches at a time.
- It will consider the attribute and accordingly the branches will be created.
- For example if we consider the attribute vegetable and its associated data like eg: vegetable ϵ { carrot, tomato, drumstick} so in this case the attribute vegetable will have 3 branches.

7.2.5 Multivariate Trees

- When it comes to the univariate tree we considered one input and we splitted it but in the multivariate tree each and every input dimensions are considered.

7.3 Naive Bayes classifier

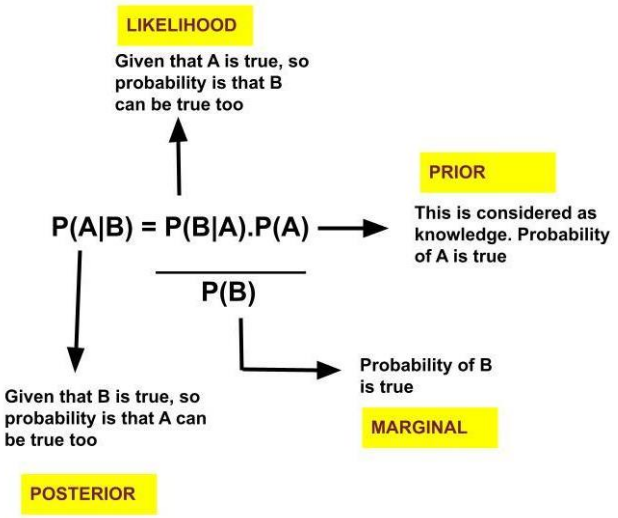
- It is a classification technique which uses Bayes theorem.
- It is a machine learning model.
- It is mostly preferred on those kinds of data which are very large in nature.
- It uses a probability approach to find the solution.
- The term “naive” is used here because it considers many features of it and basically these features are independent of each other.
- For example, consider tomatoes if we apply Naive Bayes. The main thing is the features so lets say its round in shape, red in color, small in size etc. Even though all these together will give you the end result but still they are independent in nature.
- This algorithm is very very popular, the main reason is it is very simple to code and understand.

7.3.1 Conditional Probability

- Before proceeding further it is very important to understand conditional probability.
- It is nothing but the value divided by total probability occurrence.
- For example, if you are going to roll a dice then the conditional probability occurrence of it is, the total number of faces are 6 so 1/6= 0.166
- So in this way conditional probability occurrence of different scenarios can be found.

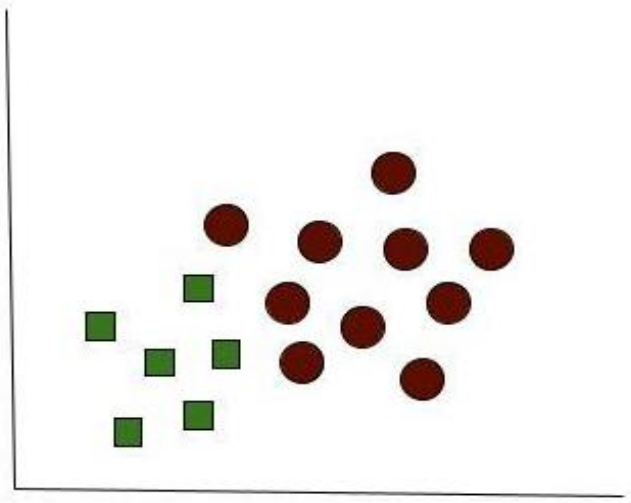
7.3.2 The Bayes rule

- It considers two value known and unknown value
- On the basis of given evidence it will try to find the solution.
- Posterior: When the evidence is collected this will do an update.
- Prior: Before the consideration of evidence the probability is applied.
- Likelihood: The belief is considered true and probability is applied.
- Marginal: Under any kind of situation the evidence probability is applied.
- It is named after Thomas Bayes



7.3.3 Naive Bayes classification

- For the formula for Naive Bayes classification refer chapter number 6



- Consider the above figure that contains two shapes square and circle.
- If you will observe properly, the number of square shapes is less than the number of circles.
- So now after applying naive bayes formula it will automatically consider the newly created one under the circle category as its in major form.
- Our target is to classify the newly formed option under any of the one category. So it is going to consider and classify this new one under the circle.
- This belief of classifying it under the major category is known as prior probability.
- Which means deciding something on the basis of previous experience.

- So the formula can be written in this way:
- Prior probability of circle = number of circle objects

  ———————————

  Total number of objects

- Prior probability of square= number of square objects

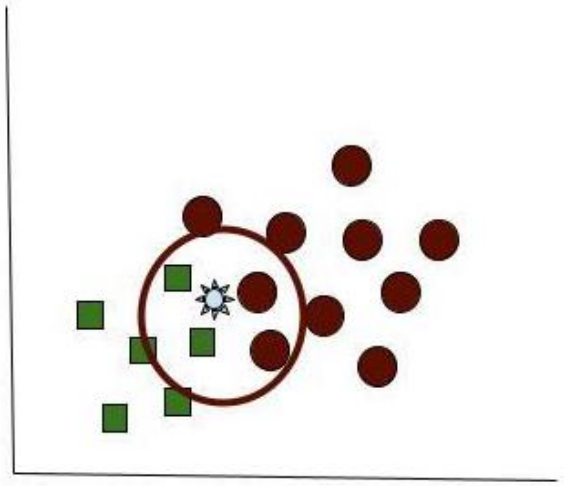
  ———————————

  Total number of objects

- In our example we have 16 objects in total, out of which 6 are squares and the remaining 10 are circles.
- So by putting these values in our prior probability formula we get this:

  Prior probability of circle = 10/16

  Prior probability of square=6/16
- Also similarly posterior probability formula can be applied which will help in finding the solution
- The below figure is an example of the Naive Bayes classification in which the formula is applied and the result is found accordingly.



7.3.4 Advantages of Naive Bayes Classification

- It is very fast in applying predictions and finding the solution.
- When it comes to text classification in comparison with other algorithms Naive Bayes has high success rates.
- Performance in categorical inputs are better in comparison with other types.

❄❄❄❄❄❄❄

8

UNSUPERVISED LEARNING AND REINFORCEMENT LEARNING

Unit Structure

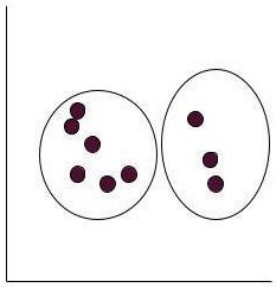
8.1 UNSUPERVISED LEARNING - GROUPING UNLABELED ITEMS USING K-MEANS CLUSTERING

- It is also known as unsupervised machine learning
- It uses the algorithms provided by machine learning algorithms to apply clustering and other techniques.
- In this, supervising the model by users is not required.
- In other words we can say that it doesn't helps it to find any solution, it has to do everything on its own.
- The pattern identification is done by it.
- This pattern recognition is done in the following manner, let's take an example to understand the same.Say we have vegetables now using our unsupervised learning the vegetables will be grouped on the basis of color, size, shape etc. Once groupism on this category is done then further groupism will be done by observing it in much deeper fashion. This is done by applying a lot of observations.
- So from this example we can say that we had an untagged set of data and we segregated it on the basis of its pattern. Such technique is nothing but unsupervised learning.
- In comparison with supervised learning, unsupervised learning techniques can tolerate more complex tasks easily.

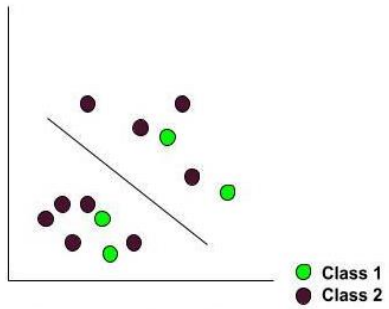
8.1.1 Why use unsupervised learning?

- Noting each and every kind of data in the dataset is not an easy task. Unsupervised learning doesn't require annotated data.
- In many situations like data mining techniques it may not be known about the count of the number of classes and numbers, this can be found using unsupervised learning.
- It can support all kinds of unknown patterns.
- It can easily categorize the data.
- Extracting unlabelled data is easier as labeled data needs to be done manually.

8.1.2 Difference between supervised and unsupervised Learning



Unsupervised Learning

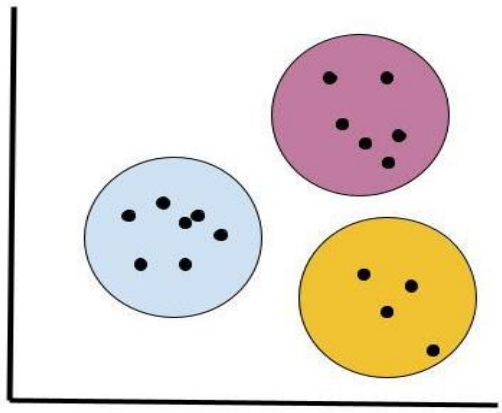


Supervised Learning

Supervised learning	**Unsupervised learning**
1. In this the goal is to predict output using the trained data set.	1. In this the goal is to find patterns from unknown data set.
2. Data are labeled in supervised learning	2. Data are unlabelled in unsupervised learning
3. It takes feedback directly	3. It doesn't takes any feedback
4. It predicts output	4. It finds the patterns
5. The different types of algorithms included in supervised learning are: Decision tree, linear regression etc.	5. The different types of algorithms included in unsupervised learning are: KNN, clustering etc.
6. It's very accurate	6. Less accurate in comparison with supervised learning
7. Supervised learning has regression and classification	7. Unsupervised learning has association and clustering

8.1.3 Clustering

- It is one of the important techniques which is used in unsupervised learning.
- Categorizing unlabeled data is called clustering.
- To understand this lets take a example, let say two person wants to learn about drawing now the drawings can be grouped on the basis of many things say first person grouped the drawing on the basis of themes like scenery, cartoon etc. where's the second person did grouping on the basis type of drawings like sketch, watercolor, poster paint etc. Even though both did the same job, the approach and groping was different. This is nothing but clustering. Here the whole data will be in front of you. You have to segregate it.
- If the datas are labeled then it will be called classification.
- So in simple terms we can say that clustering is a machine learning approach that does grouping of data.
- In this the similar data points are grouped together called clusters, refer the below image for the same.
- It may consider shape, behavior, size, color, dimensions etc. based on such criteria it will group the items that have similar properties.
- After the creation of a cluster each cluster will have its own unique id associated with it called cluster-ID.
- Machine learning will use cluster-ID to apply further evaluation.



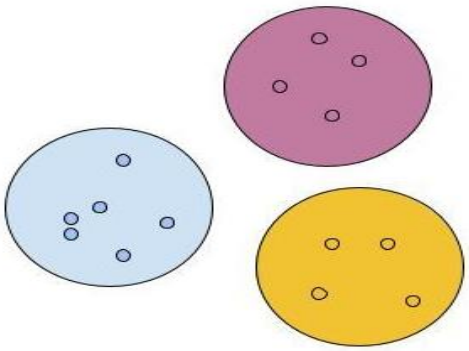
Clustering

8.1.3.1 Clustering approaches

1. Distribution based
2. Density based
3. Fuzzy based
4. Centroid based

1. Distribution based:

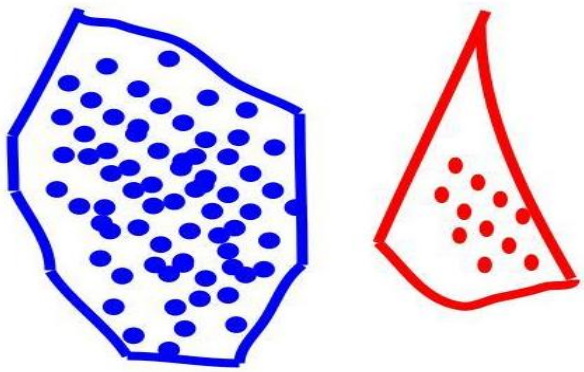
- In this the data will belong to a particular cluster.
- In other words we can say that the data set belongs to the particular distribution.
- In the figure below we can see the distribution based clustering where the distribution of data is done in 3 forms and clustering of the same is achieved.
- Gaussian distribution term is also used sometimes in this to apply distribution based clustering.



Distribution based Clustering

2. Density based clustering:

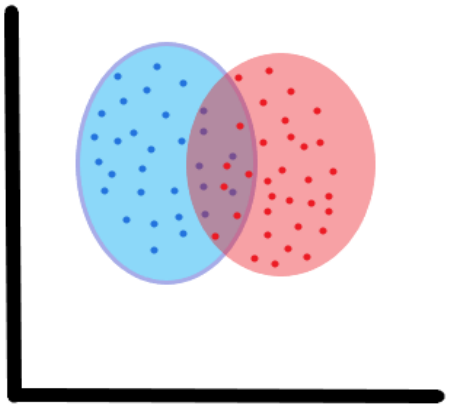
- In this the clustering is done on the basis of the amount of data.
- Those areas that have a high amount of similar data will be groubed together forming clusters in that particular region.
- The dense area is considered and that area is grouped together.
- Below figure shows the visual scenario of how the cluster happens in this.
- As you can see that over concentration of blue and red dots are there so they are grouped together and it is giving a cluster formation.



Density based clustering

3. Fuzzy based:

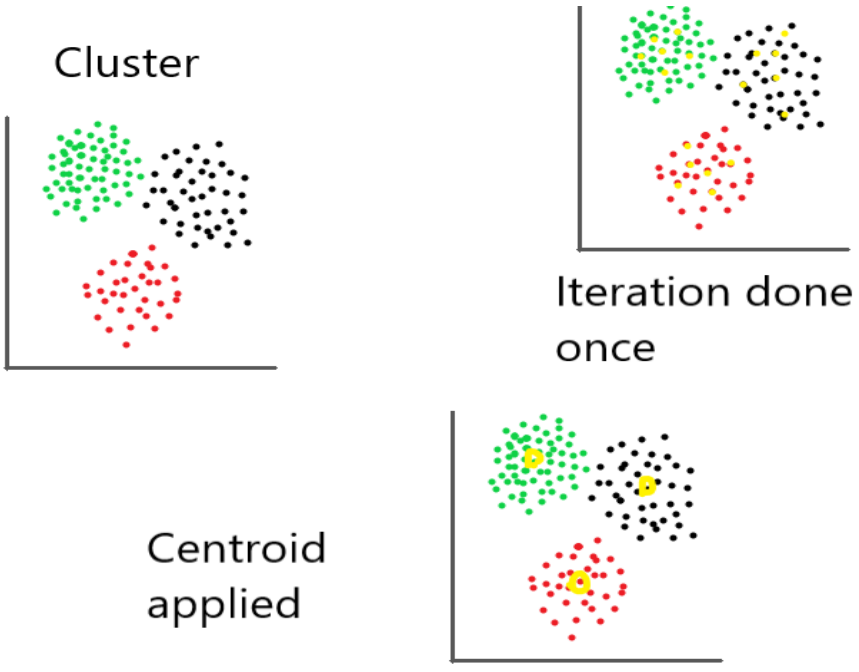
- It is also called as soft clustering
- In this fuzzy based clustering what happens is it may happen that the data present in one group of cluster is overlapping with another group of cluster.
- This overlapped part also forms one cluster group.
- This kind of groupism is known as fuzzy based clustering.



FUZZY BASED CLUSTERING

4. Centroid based:

- In this the cluster is formed and multiple centroids are created as shown in the figure below.
- Then iteration is applied again and again to find the best possible point which may give the best possible result.

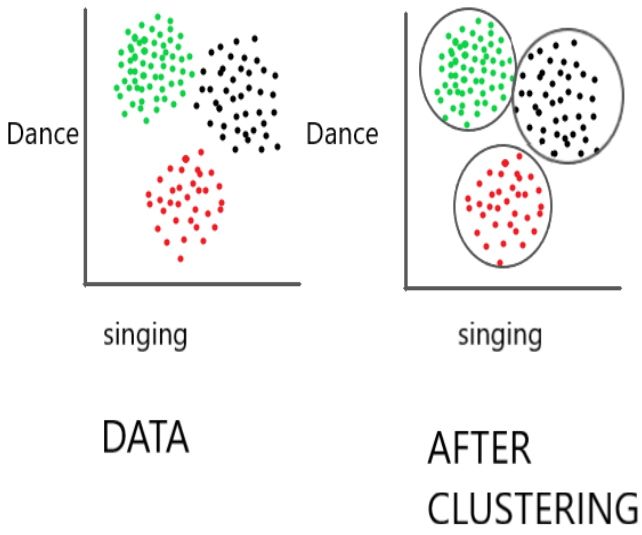


8.1.4 K-means Clustering algorithm

- It is a unsupervised based clustering algorithm
- It is used to solve clustering problems for machine learning.
- The ‘K’ is nothing but a number.
- It differentiates the whole set of data which are unlabeled into groups and forms clusters.
- The value of k is nothing but the number of clusters. For example if we say K=6 that means there are 6 groups of clusters present.
- It doesn’t need any training data sets.
- To understand the K-means algorithm, let's take an example of arts, so we have arts which provide dance, singing etc. many such kinds of courses. Now we need to differentiate or group students on the basis of their choice. Such kind of grouping is nothing but clustering. Also if we differentiate among them on the basis of score then K-means comes into the picture.
- Kindly refer below diagram to understand the same.

8.1.4.1 Steps for applying K-means clustering algorithm

- Step 1: Consider the whole data set.
- Step 2: Differentiate the data set into groups.
- Step 3: Apply cluster formation by randomly grouping it.
- Step 4: Make a centroid and check the variance and similarity between the data.
- Step 5: Rechange the centroid location according to the requirement of the updated cluster.
- Step 6: Repeat the step number 4 again and again until and unless you get the proper cluster formation
- Step 8: End.



8.2 ASSOCIATION ANALYSIS WITH THE APRIORI ALGORITHM INTRODUCTION TO REINFORCEMENT LEARNING

8.2.1 Association analysis

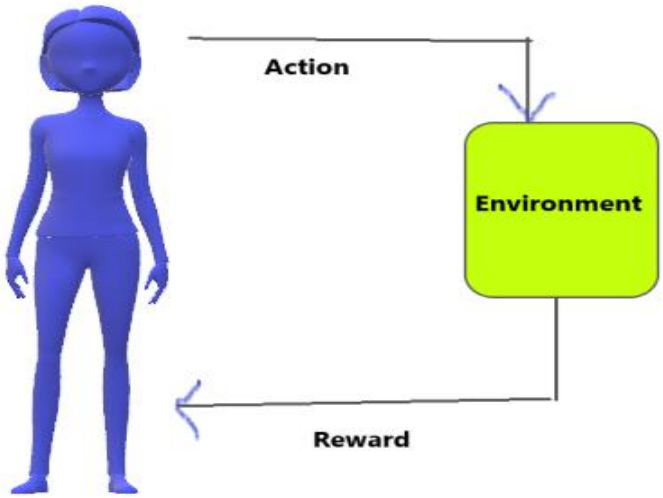
- Consider a hypermarket, this market contains all kinds of items in it and it has to analyze the behavior of its customer.
- It happens that when a particular item is bought some item which is in relation to it is also purchased, for example if a person buys a bread then the chances of buying jam with it is very high.
- This example is nothing but association analysis.
- It's main goal is to find the relationship among the datas in large data sets.
- Frequent item set is maintained which contains those data details that is in relationship and that comes together frequently

8.2.2 Apriori algorithm

- In this algorithm a frequent item is considered and two categories are also considered called support and support threshold
- If the item falls under support threshold then anyhow even its sub combination will fall under frequent consumption.

8.2.3 Reinforcement learning

- In this algorithm the machine will receive the data from its environment and from this it will try to make a decision.
- This decision will be on a trial and error basis.



❄❄❄❄❄❄❄

9

FORECASTING AND LEARNING THEORY

Unit Structure

9.1 NON-LINEAR REGRESSION

9.1.1 Introduction

Linear regression models provide a rich and flexible framework that suits the needs of many analysts. However, linear regression models are not appropriate for all situations. There are many problems in engineering and the sciences where the response variable and the predictor variables are related through a known **non-linear** function. This leads to a **non-linear regression model**. When the method of least squares is applied to such models, the resulting normal equations are nonlinear and, in general, difficult to solve. The usual approach is to directly minimize the residual sum of squares by an iterative procedure. In this chapter we describe estimating the parameters in a nonlinear regression model and show how to make appropriate inferences on the model parameters. We also illustrate computer software for non-linear regression.

Non-linear regression:

Nonlinear regression is a form of regression analysis in which data is fit to a model and then expressed as a mathematical function. Simple linear regression relates two variables (X and Y) with a straight line (y = mx + b), while nonlinear regression relates the two variables in a nonlinear (curved) relationship.

The goal of the model is to make the sun of the squares as small as possible. The sum of squares is a measure that tracks how far the Y observations vary from the nonlinear (curved) function that is used to predict Y.

It is computed by first finding the difference between the fitted nonlinear function and every Y point of data in the set. Then, each of those

differences is squared. Lastly, all of the squared figures are added together. The smaller the sum of these squared figures, the better the function fits the data points in the set. Nonlinear regression uses logarithmic functions, trigonometric functions, exponential functions, power functions, Lorenz curves, Gaussian functions, and other fitting methods.

Note: Both linear and nonlinear regression predict Y responses from an X variable (or variables).

Nonlinear regression is a curved function of an X variable (or variables) that is used to predict a Y variable

Nonlinear regression can show a prediction of population growth over time.

Nonlinear regression modelling is similar to linear regression modelling in that both seek to track a particular response from a set of variables graphically. Nonlinear models are more complicated than linear models to develop because the function is created through a series of approximations (iterations) that may stem from trial-and-error. Mathematicians use several established methods, such as the Gauss-Newton method and the Levenberg-Marquardt method.

Often, regression models that appear nonlinear upon first glance are actually linear. The curve estimation procedure can be used to identify the nature of the functional relationships at play in your data, so you can choose the correct regression model, whether linear or nonlinear. Linear regression models, while they typically form a straight line, can also form curves, depending on the form of the linear regression equation. Likewise, it's possible to use algebra to transform a nonlinear equation so that mimics a linear equation—such a nonlinear equation is referred to as "intrinsically linear."

Example of Nonlinear Regression:

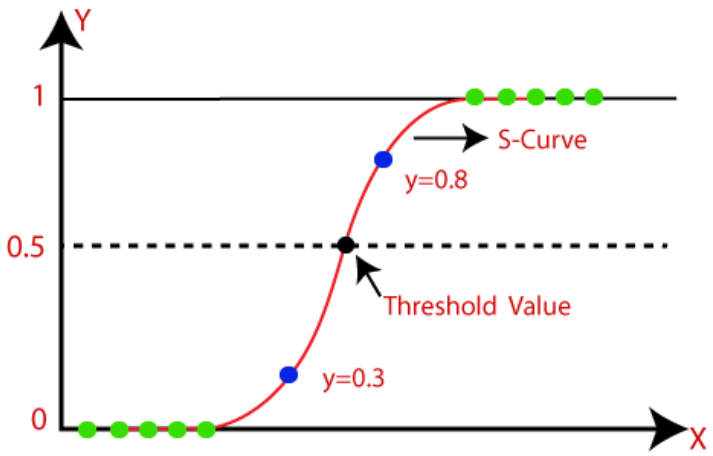
One example of how non-linear regression can be used is to predict population growth over time. A scatterplot of changing population data over time shows that there seems to be a relationship between time and population growth, but that it is a nonlinear relationship, requiring the use of a non-linear regression model. A logistic population growth model can provide estimates of the population for periods that were not measured, and predictions of future population growth. Independent and dependent variables used in non-linear regression should be quantitative. Categorical variables, like region of residence or religion, should be coded as binary variables or other types of quantitative variables.

In order to obtain accurate results from the non-linear regression model, you should make sure the function you specify describes the relationship between the independent and dependent variables accurately. Good starting values are also necessary. Poor starting values may result in a model that fails to converge, or a solution that is only optimal locally,

rather than globally, even if you've specified the right functional form for the model.

9.2 LOGISTIC REGRESSION

1. Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
2. Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either **Yes** or **No,** 0 or 1, true or False, etc. but instead of giving the exact value as **0** and **1**, **it gives the probabilistic values which lie between 0 and 1**.
3. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
4. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
5. The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
6. Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
7. Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.

- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- The above equation is the final equation for Logistic Regression.

9.2.1 Type of Logistic Regression:

Logistic Regression can be classified into three types:

1. **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
2. **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"

3. **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

9.3 RANDOM FOREST

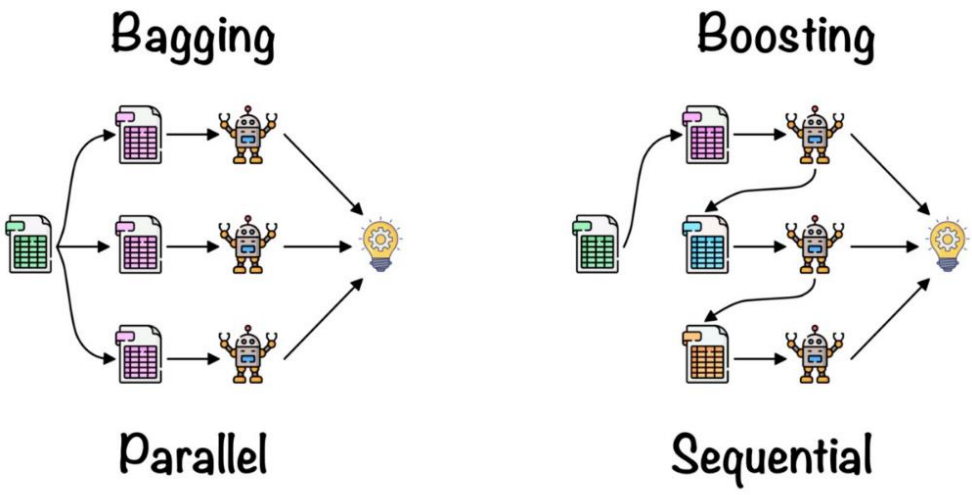
Random forest is a **Supervised Machine Learning Algorithm** that is **used widely in Classification and Regression problems**. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

Working of Random Forest Algorithm:

Before understanding the working of the random forest, we must look into the ensemble technique. ***Ensemble*** simply means combining multiple models. Thus, a collection of models is used to make predictions rather than an individual model.

Ensemble uses two types of methods:

1. **Bagging**– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.
2. **Boosting**– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST

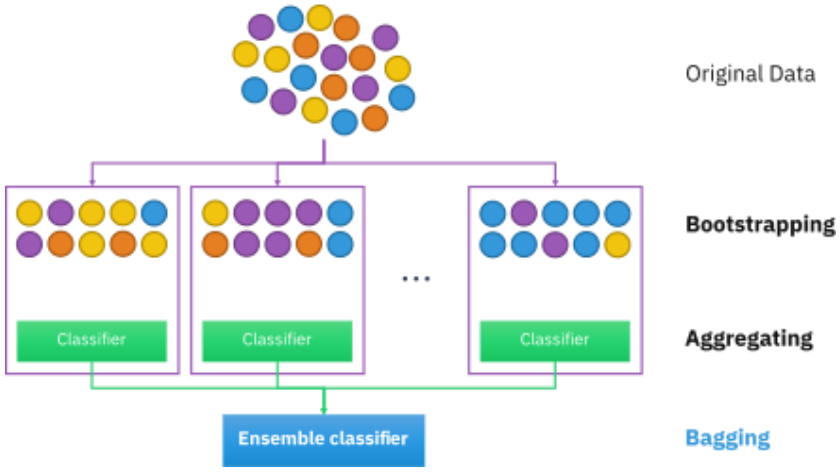


As mentioned earlier, Random forest works on the Bagging principle. Now let's dive in and understand bagging in detail.

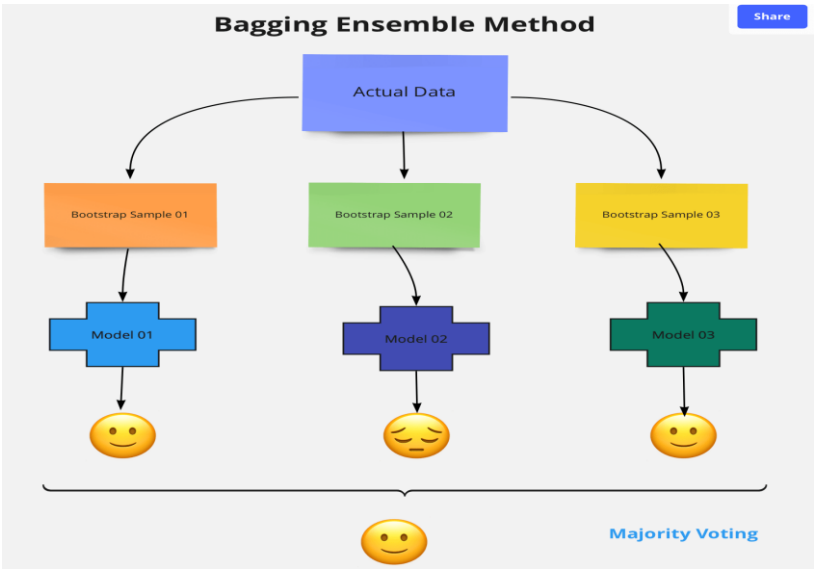
Bagging:

Bagging, also known as ***Bootstrap Aggregation*** is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as ***row sampling***. This step of row sampling with replacement is called ***bootstrap***. Now each

model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as ***aggregation***.



Now let's look at an example by breaking it down with the help of the following figure. Here the bootstrap sample is taken from actual data (Bootstrap sample 01, Bootstrap sample 02, and Bootstrap sample 03) with a replacement which means there is a high possibility that each sample won't contain unique data. Now the model (Model 01, Model 02, and Model 03) obtained from this bootstrap sample is trained independently. Each model generates results as shown. Now Happy emoji is having a majority when compared to sad emoji. Thus, based on majority voting final output is obtained as Happy emoji.



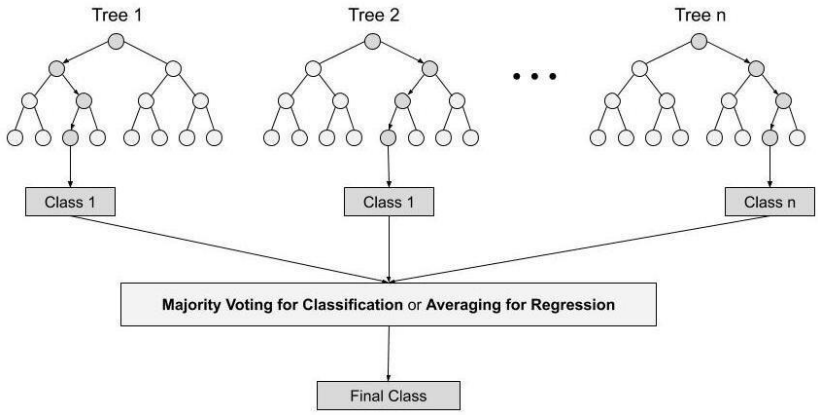
Steps involved in random forest algorithm:

Step 1: In Random forest n number of random records are taken from the data set having k number of records.

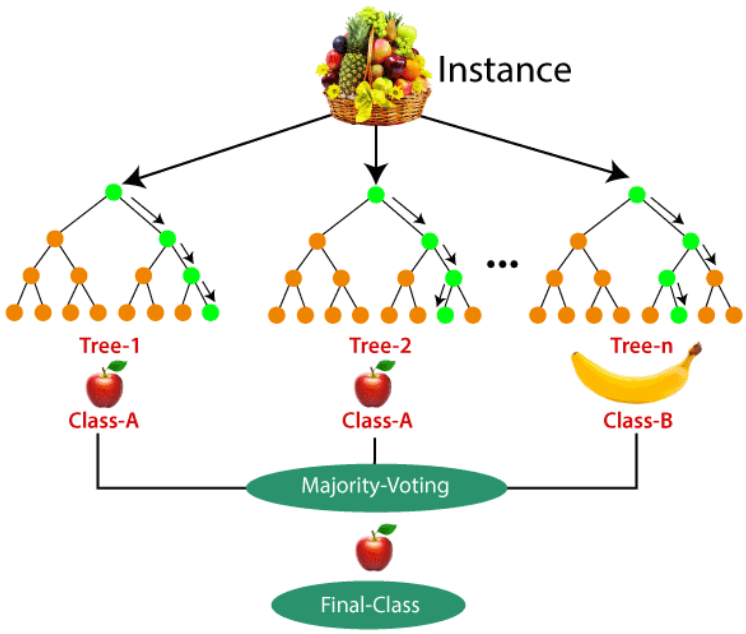
Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging*** for Classification and regression respectively.



For example: consider the fruit basket as the data as shown in the figure below. Now n number of samples are taken from the fruit basket and an individual decision tree is constructed for each sample. Each decision tree will generate an output as shown in the figure. The final output is considered based on majority voting. In the below figure you can see that the majority decision tree gives output as an apple when compared to a banana, so the final output is taken as an apple.



9.3.1 Important Features of Random Forest

1. **Diversity-** Not all attributes/variables/features are considered while making an individual tree, each tree is different.
2. **Immune to the curse of dimensionality-** Since each tree does not consider all the features, the feature space is reduced.
3. **Parallelization-**Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. **Train-Test split-** In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
5. **Stability-** Stability arises because the result is based on majority voting/ averaging.

9.4 BAYESIAN BELIEF 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**.

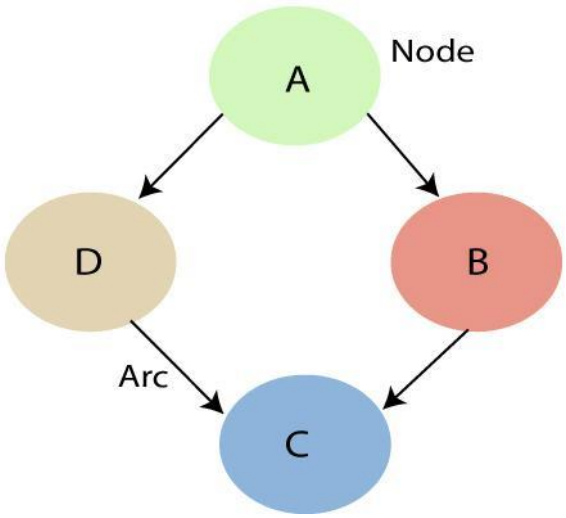
Bayesian Network can be used for building models from data and experts' opinions, and it consists of two parts:

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:



Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.

Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.
These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other

In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.

If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.

Node C is independent of node A.

The Bayesian network has mainly two components:

1. **Causal Component**
2. **Actual numbers**

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

9.5 BIAS/VARIANCE TRADE OFF

Whenever we discuss model prediction, it's important to understand prediction errors (bias and variance). There is a tradeoff between a model's ability to minimize bias and variance. Gaining a proper understanding of these errors would help us not only to build accurate models but also to avoid the mistake of overfitting and underfitting.

What is bias?

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

What is variance?

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

Mathematically

Let the variable we are trying to predict as Y and other covariates as X. We assume there is a relationship between the two such that

$Y=f(X) + e$

Where e is the error term and it's normally distributed with a mean of 0.

We will make a model f^(X) of f(X) using linear regression or any other modelling technique.

So, the expected squared error at a point x is

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

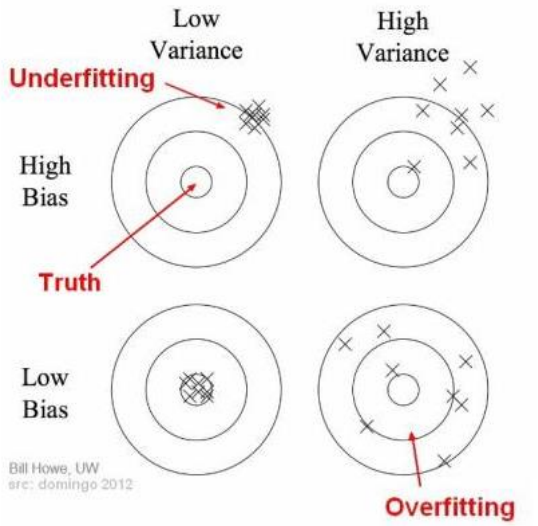
The Err(x) can be further decomposed as

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Err(x) is the sum of Bias², variance and the irreducible error.
Irreducible error is the error that can't be reduced by creating good models. It is a measure of the amount of noise in our data. Here it is important to understand that no matter how good we make our model, our data will have certain amount of noise or irreducible error that can not be removed.

Bias and variance using bulls-eye diagram

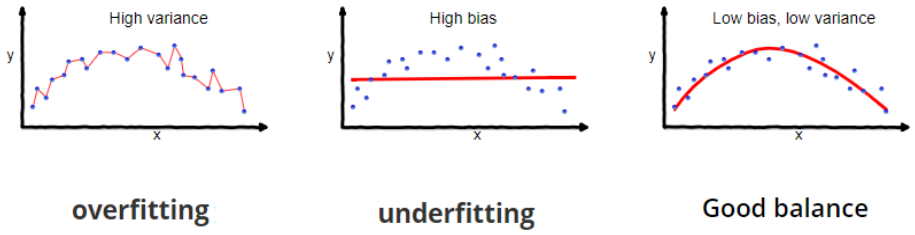


In the above diagram, center of the target is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse. We can repeat our process of model building to get separate hits on the target.

In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.

In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high

variance. These models are very complex like Decision trees which are prone to overfitting.



Why is Bias Variance Tradeoff?

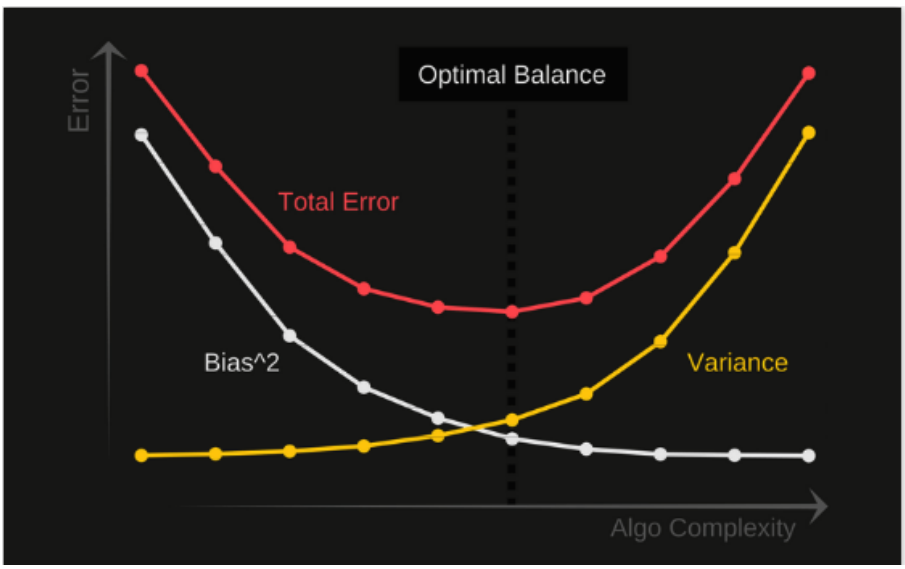
If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.

Total Error

To build a good model, we need to find a good balance between bias and variance such that it minimizes the total error.

Total Error = Bias^2 + Variance + Irreducible Error



An optimal balance of bias and variance would never overfit or underfit the model.

Therefore, understanding bias and variance is critical for understanding the behaviour of prediction models.

9.6 TUNING MODEL COMPLEXITY

Tuning is usually a trial-and-error process by which you change some hyperparameters (for example, the number of trees in a tree-based algorithm or the value of alpha in a linear algorithm), run the algorithm on the data again, then compare its performance on your validation set in order to determine which set of hyperparameters results in the most accurate model.

All machine learning algorithms have a "default" set of hyperparameters, which Machine Learning Mastery defines as "a configuration that is external to the model and whose value cannot be estimated from data." Different algorithms consist of different hyperparameters. For example, regularized regression models have coefficients penalties, decision trees have a set number of branches, and neural networks have a set number of layers. When building models, analysts and data scientists choose the default configuration of these hyperparameters after running the model on several datasets.

While the generic set of hyperparameters for each algorithm provides a starting point for analysis and will generally result in a well-performing model, it may not have the optimal configurations for your particular dataset and business problem. In order to find the best hyperparameters for your data, you need to tune them.

Tuning is the process of maximizing a model's performance without overfitting or creating too high of a variance. In machine learning, this is accomplished by selecting appropriate "hyperparameters."

Hyperparameters can be thought of as the "dials" or "knobs" of a machine learning model. Choosing an appropriate set of hyperparameters is crucial for model accuracy, but can be computationally challenging. Hyperparameters differ from other model parameters in that they are not learned by the model automatically through training methods. Instead, these parameters must be set manually. Many methods exist for selecting appropriate hyperparameters. This post focuses on three:

- Grid Search
- Random Search
- Bayesian Optimization

Grid Search

Grid Search, also known as parameter sweeping, is one of the most basic and traditional methods of hyperparametric optimization. This method involves manually defining a subset of the hyperparametric space and exhausting all combinations of the specified hyperparameter subsets. Each combination's performance is then evaluated, typically using cross-validation, and the best performing hyperparametric combination is chosen.

Random Search

Random search methods resemble grid search methods but tend to be less expensive and time consuming because they do not examine every possible combination of parameters. Instead of testing on a predetermined subset of hyperparameters, random search, as its name implies, randomly selects a chosen number of hyperparametric pairs from a given domain and tests only those. This greatly simplifies the analysis without significantly sacrificing optimization. For example, if the region of hyperparameters that are near optimal occupies at least 5% of the grid, then random search with 60 trials will find that region with high probability (95%).

Bayesian Optimization

The idea behind Bayesian Optimization is fundamentally different from grid and random search. This process builds a probabilistic model for a given function and analyses this model to make decisions about where to next evaluate the function. There are two main components under the Bayesian optimization framework.

A prior function that captures the behaviour of the unknown objective function and an observation model that describes the data generation mechanism.

A loss function, or an acquisition function, that describes how optimal a sequence of queries are, usually taking the form of regret.

The most common selection for a prior function in Bayesian Optimization is the Gaussian process (GP) prior. This is a particular kind of statistical model where observations occur in a continuous domain.

In a Gaussian process, every point in the defined continuous input space is associated with a normally distributed random variable. Additionally, every finite linear combination of those random variables has a multivariate normal distribution.

Why is Model Tuning Important?

Model tuning allows you to customize your models so they generate the most accurate outcomes and give you highly valuable insights into your data, enabling you to make the most effective business decisions.

❄❄❄❄❄❄❄

10

MODEL SELECTION DILEMMA CLUSTERING

Unit Structure

10.1 INTRODUCTION

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as **"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."**

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, colour, behaviour, etc., and divides them as per the presence and absence of those similar patterns. It is an unsupervised learning method; hence no supervision is provided to the algorithm, and it deals with the un-labelled dataset. After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

10.2 EXPECTATION-MAXIMIZATION ALGORITHM

Expectation-Maximization algorithm can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

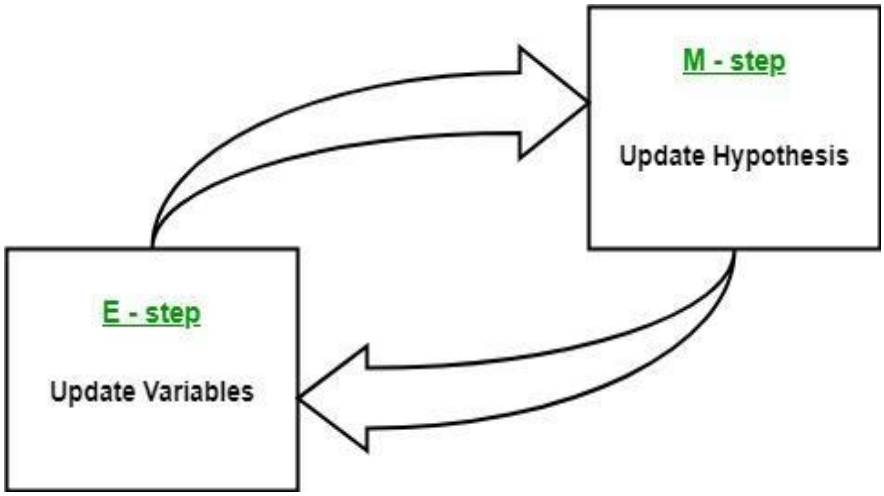
Algorithm:

Given a set of incomplete data, consider a set of starting parameters.

Expectation step (E – step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.

Maximization step (M – step): Complete data generated after the expectation (E) step is used in order to update the parameters.

Repeat step 2 and step 3 until convergence.



The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

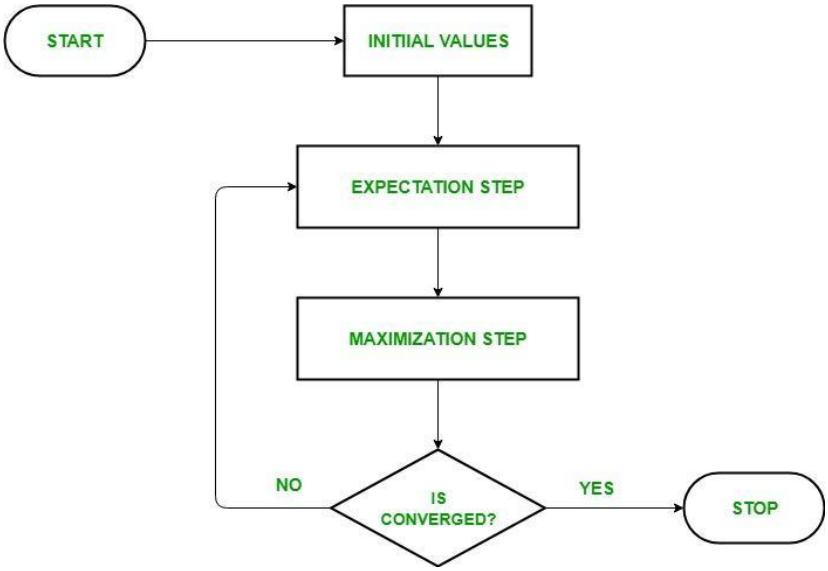
Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.

The next step is known as “Expectation” – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.

The next step is known as “Maximization”-step or *M-step*. In this step, we use the complete data generated in the preceding “Expectation” – step in order to update the values of the parameters. It is basically used to update the hypothesis.

Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. “Expectation” – step and “Maximization” – step until the convergence occurs.

Flow chart for EM algorithm –



Usage of EM algorithm:

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm:

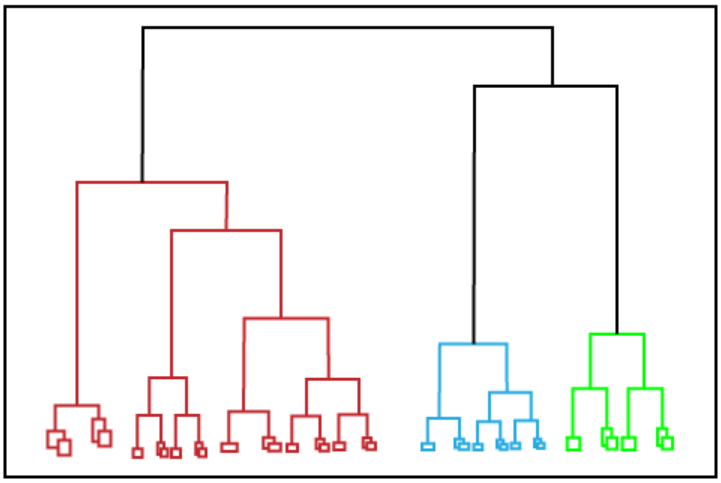
- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

Disadvantages of EM algorithm:

- It has slow convergence.
- It makes convergence to the local optima only.
- It requires both the probabilities, forward and backward (numerical optimization requires only forward probability).

10.3 HIERARCHICAL CLUSTERING

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



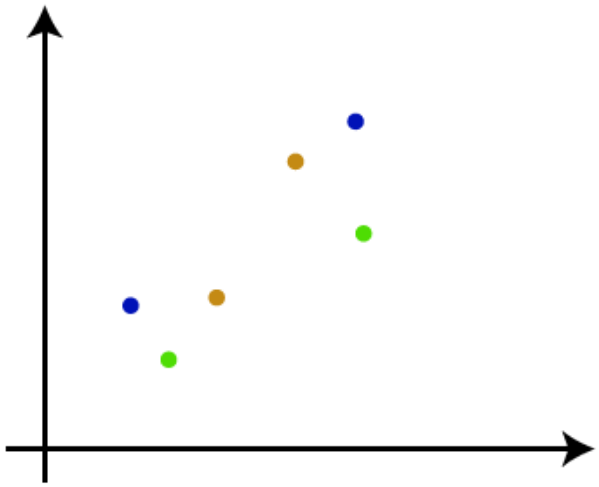
The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

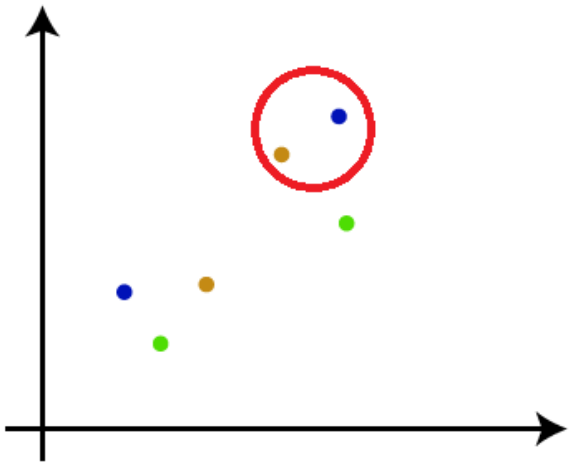
How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

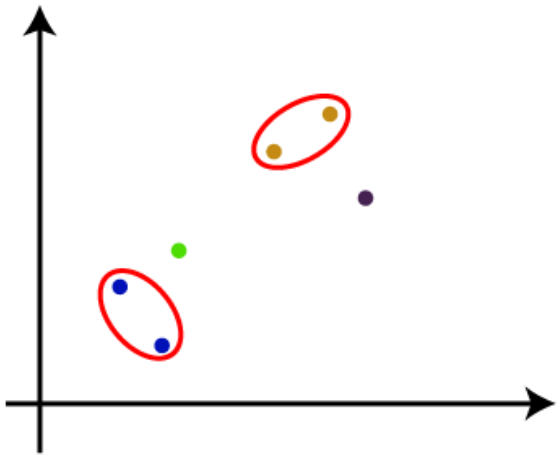
Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



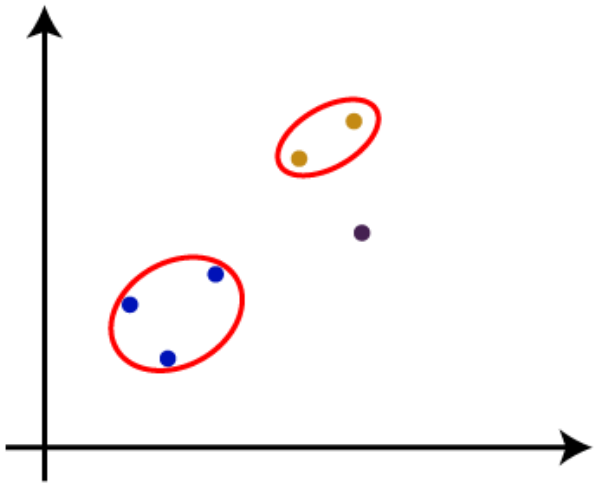
Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

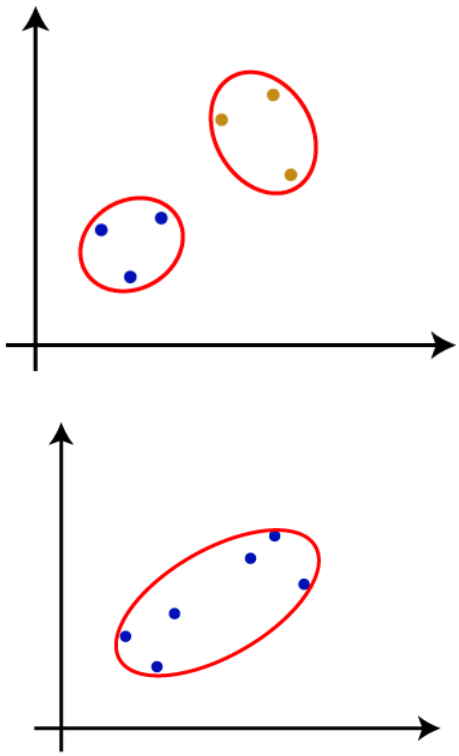


Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.



Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



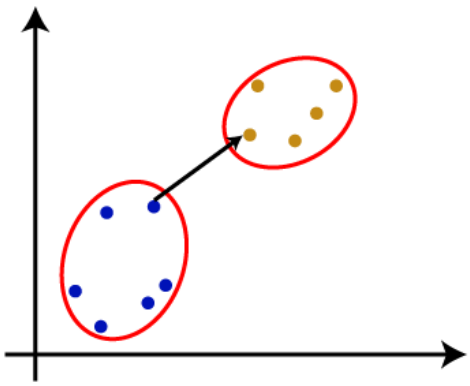


Step-5: Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

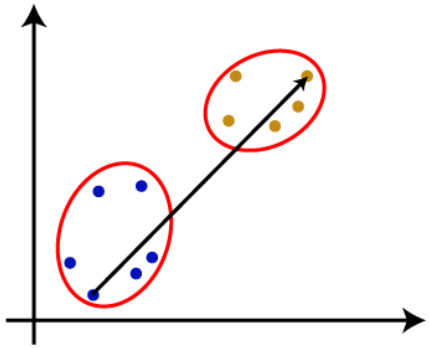
Measure for the distance between two clusters:

The **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

Single Linkage: It is the Shortest Distance between the closest points of the clusters. Consider the below image:

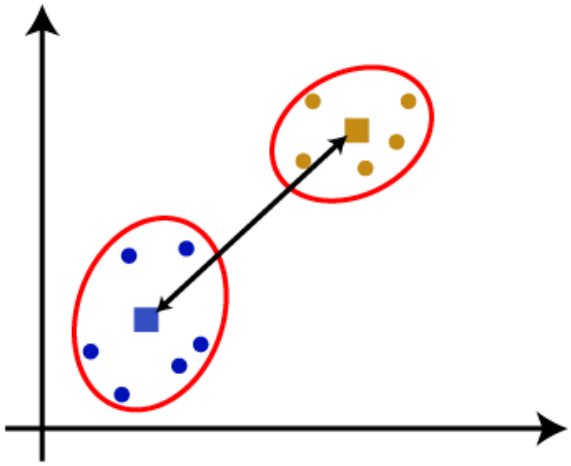


Complete Linkage: It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



Average Linkage: It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

Centroid Linkage: It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:



From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

10.4 SUPERVISED LEARNING AFTER LEARNING

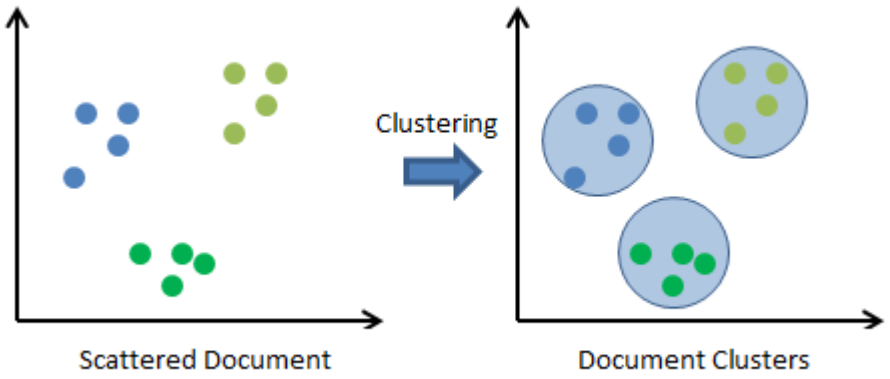
After Supervised Learning algorithms, it's time to have a look at the most popular Unsupervised method. Here, we present to you - ***Clustering***, and it's variants.

Let's look at it is simplicity here:

Everybody does this, right?

In our daily life, we group different activities according to their utility. This grouping is what you need to learn.

Clustering algorithm **does not** predict an outcome or target variable but can be used to improve predictive model. Predictive models can be built for clusters to *improve the accuracy of our prediction*.

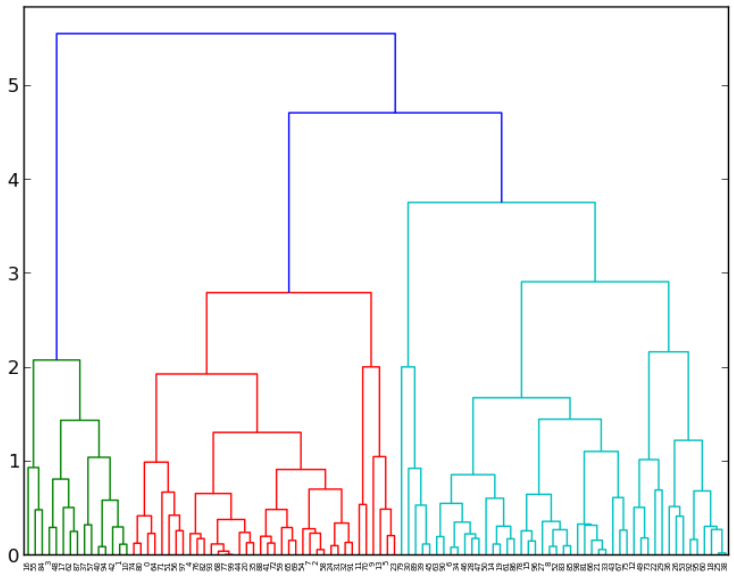


Types of Clustering

There exist more than 100 clustering algorithms as of today. Some of the commonly used are **k-Means**, **Hierarchical**, DBSCAN and OPTICS. Two of these have been covered here:

1. Hierarchical Clustering

It is a type of connectivity model clustering which is based on the fact that data points that are closer to each other are more similar than the data points lying far away in a data space. As the name speaks for **itself, the hierarchical clustering forms the hierarchy of the clusters that can be studied by visualising *dendogram*.**



Dendogram

How to measure closeness of points?

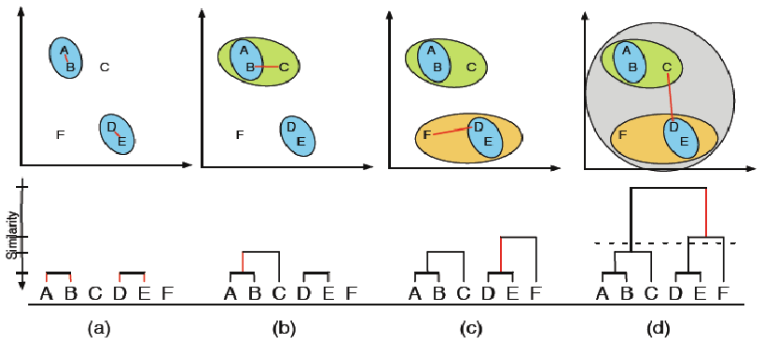
- **Euclidean distance**: ||a-b||2 = √(Σ(ai-bi))
- **Squared Euclidean distance**: ||a-b||22 = Σ((ai-bi)²)
- **Manhattan distance**: ||a-b||¹ = Σ|ai-bi|
- **Maximum distance**:||a-b||^inf = maxi|ai-bi|
- **Mahalanobis distance**: √((a-b)T S-1 (-b)) {where, s : covariance matrix}

How to calculate distance between two clusters?

1. **Centroid Distance**: Euclidean distance between mean of data points in the two clusters
2. **Minimum Distance**: Euclidean distance between two data points in the two clusters that are closest to each other
3. **Maximum Distance** : Euclidean distance between two data points in the two clusters that are farthest to each other

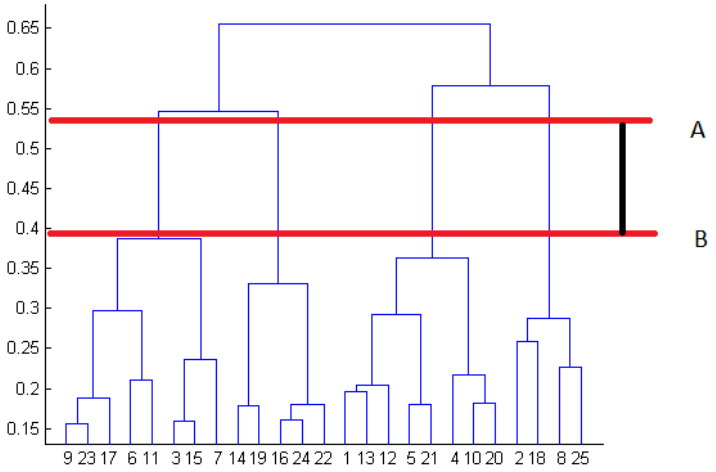
Algorithm Explained

1. Let there be ***N*** data points. Firstly, these *N* data points are assigned to *N* different clusters with one data point in each cluster.
2. Then, two data points with **minimum euclidean distanc**e between them are merged into a single cluster.
3. Then, two clusters with **minimum centroid distance** between them are merged into a single cluster.
4. This ***process is repeated*** until we are left with a single cluster, hence forming hierarchy of clusters.



How many clusters to form?

1. **Visualising dendogram:** Best choice of no. of clusters is *no. of vertical lines that can be cut by a horizontal line*, that can transverse maximum distance vertically without intersecting other cluster. For eg., in the below case, best choice for no. of clusters will be ***4***.
2. **Intuition** and prior knowledge of the data set.



Focus on A and B.

Good Cluster Analysis

- **Data-points within same cluster share similar profile**: Statistically, check the standard deviation for each input variable in each cluster. A perfect separation in case of cluster analysis is rarely achieved. Hence, even ***one standard deviation distance*** between two cluster means is considered to be a good separation.

- **Well spread proportion of data-points among clusters**: There are no standards for this requirement. But a minimum of 5% and maximum of 35% of the total population can be assumed as a safe range for each cluster.

```
In [23]: from sklearn.cluster import AgglomerativeClustering
         Hclustering = AgglomerativeClustering(n_clusters= 5, affinity = 'euclidean',linkage = 'ward')
         Hclustering.fit(x_train2)
```

Implementation in Python!

K-Means Clustering

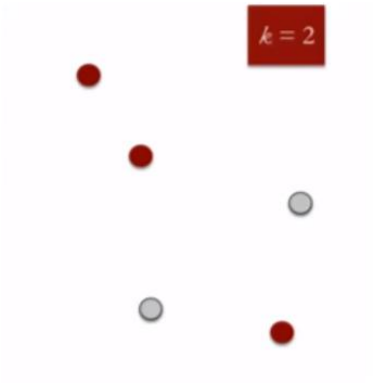
One of the simplest and most widely used unsupervised learning algorithm. It involves a simple way to classify the data set into fixed no. of ***K*** clusters . The idea is to define ***K*** centroids, one for each cluster.

The final clusters depend on the initial configuration of centroids. So, they should be initialized as far from each other as possible.

K-Means is *iterative* in nature and *easy* to implement.

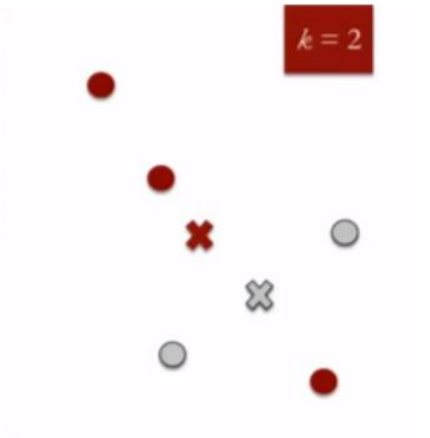
Algorithm Explained

Let there be ***N*** data points. At first, ***K*** centroids are initialised in our data set representing *K* different clusters.



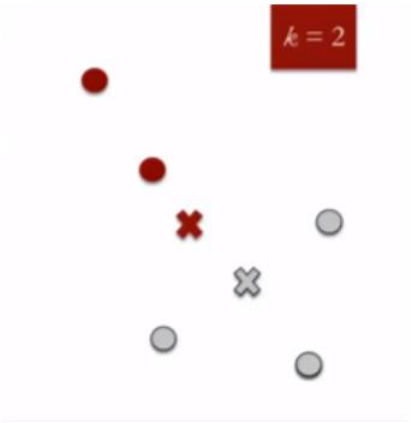
Step 1: N = 5, K = 2

Now, each of the ***N*** data points are assigned to closest centroid in the data set and merged with that centroid as a single cluster. In this way, every data point is assigned to one of the centroids.

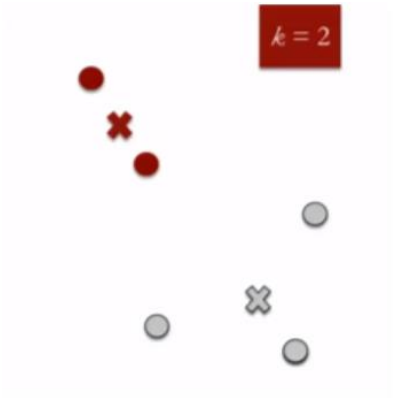


Step 2: Calculating the centroid of the 2 clusters

Then, ***K*** cluster centroids are recalculated and again, each of the ***N*** data points are assigned to the nearest centroid.



Step 3: Assigning all the data points to the nearest cluster centroid
Step 3 is repeated until no further improvement can be made.



Step 4: Recalculating the cluster centroid. After this step, no more improvement can be made.
In this process, a loop is generated. As a result of this loop, K centroids change their location step by step until no more change is possible.

This algorithm aims at minimising the **objective function**:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

It represents the sum of **euclidean distance** of all the data points from the cluster centroid which is minimised.

```
In [258]: #K-Means Clustering Algorithm applied with clusters varying from 1 to 11
          #Distortions is nothing but within the cluster SSE and can be computed using inbuilt km_inertia
          distortions2 = []
          for i in range(1,11):
              km1 = KMeans(n_clusters = i,n_init=10,max_iter=300,random_state=25)
              km1.fit(x_train2)
              distortions2.append(km1.inertia_)
```

Implementation in Python!

How to initialize K centroids?

1. **Forgy:** Randomly assigning K centroid points in our data set.
2. **Random Partition:** Assigning each data point to a cluster randomly, and then proceeding to evaluation of centroid positions of each cluster.
3. **KMeans++:** Used for ***small*** data sets.
4. **Canopy Clustering:** Unsupervised pre-clustering algorithm used as preprocessing step for K-Means or any Hierarchical Clustering. It helps in speeding up clustering operations on ***large data sets***.

10.5 CHOOSING THE NUMBER OF CLUSTERS

Determining the **optimal number of clusters** in a data set is a fundamental issue in partitioning clustering, such as k-means clustering, which requires the user to specify the number of clusters k to be generated. Unfortunately, there is no definitive answer to this question. The optimal number of clusters is somehow subjective and depends on the method used for measuring similarities and the parameters used for partitioning.

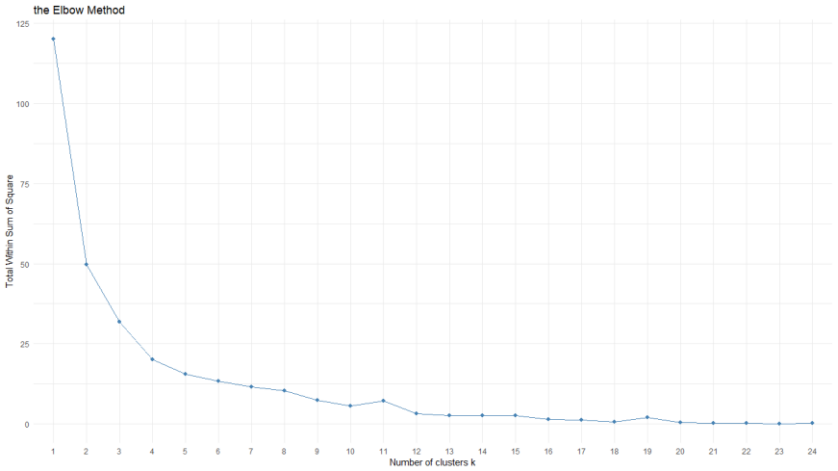
Determining Optimal Number of Clusters

A variety of measures have been proposed in the literature for evaluating clustering results. The term **clustering validation** is used to design the procedure of evaluating the results of a clustering algorithm. There are more than thirty indices and methods for identifying the optimal number of clusters so I'll just focus on a few here including the very neat **cluster** package.

1. The "Elbow" Method

Probably the most well-known method, the elbow method, in which the sum of squares at each number of clusters is calculated and graphed, and the user looks for a change of slope from steep to shallow (an elbow) to determine the optimal number of clusters. This method is inexact, but still potentially helpful.

```
set.seed(31)
# function to compute total within-cluster sum of squares fviz_nbclust(mammals_scaled, kmeans, method = "wss", k.max = 24) + theme_minimal() + ggtitle("the Elbow Method")
```

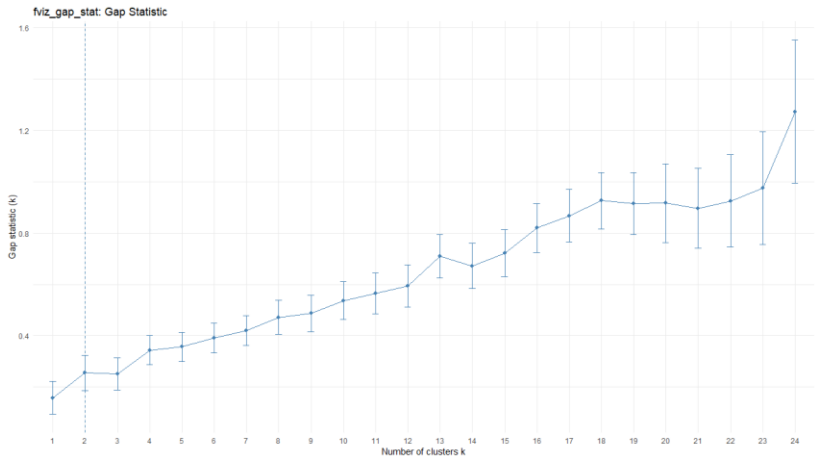


The Elbow Curve method is helpful because it shows how increasing the number of the clusters contribute separating the clusters in a meaningful way, not in a marginal way. The bend indicates that additional clusters beyond the third have little value (Elbow method is fairly clear, if not a naïve solution based on intra-cluster variance. The gap statistic is more sophisticated method to deal with data that has a distribution with no obvious clustering (can find the correct number of *k* for globular, Gaussian-distributed, mildly disjoint data distributions).

2. The Gap Statistic

The gap statistic compares the total within intra-cluster variation for different values of **k** with their expected values under null reference distribution of the data. The estimate of the optimal clusters will be value that maximize the gap statistic (*i.e.*, that yields the largest gap statistic). This means that the clustering structure is far away from the random uniform distribution of points.

```
gap_stat <- clusGap(mammals_scaled, FUN = kmeans, nstart = 30, K.max = 24, B = 50)

fviz_gap_stat(gap_stat) + theme_minimal() + ggtitle("fviz_gap_stat: Gap Statistic")
```

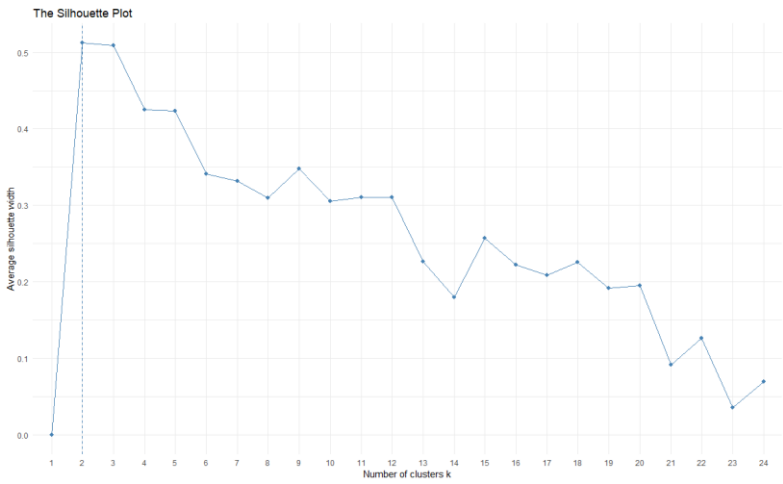


The gap stats plot shows the statistics by number of clusters (**k**) with standard errors drawn with vertical segments and the optimal value of **k** marked with a vertical dashed blue line. According to this observation **k = 2** is the optimal number of clusters in the data.

3. The Silhouette Method

Another visualization that can help determine the optimal number of clusters is called a silhouette method. Average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for **k**.

```
fviz_nbclust(mammals_scaled, kmeans, method = "silhouette", k.max = 24) + theme_minimal() + ggtitle("The Silhouette Plot")
```



10.6 LEARNING USING ANN

What Is Learning in ANN?

Basically, learning means to do and adapt the change in itself as and when there is a change in environment. ANN is a complex system or more precisely we can say that it is a complex adaptive system, which can change its internal structure based on the information passing through it.

Why Is It important?

Being a complex adaptive system, learning in ANN implies that a processing unit is capable of changing its input/output behaviour due to the change in environment. The importance of learning in ANN increases because of the fixed activation function as well as the input/output vector, when a particular network is constructed. Now to change the input/output behaviour, we need to adjust the weights.

Classification

It may be defined as the process of learning to distinguish the data of samples into different classes by finding common features between the samples of the same classes. For example, to perform training of ANN, we have some training samples with unique features, and to perform its

testing we have some testing samples with other unique features. Classification is an example of supervised learning.

Neural Network Learning Rules

We know that, during ANN learning, to change the input/output behaviour, we need to adjust the weights. Hence, a method is required with the help of which the weights can be modified. These methods are called Learning rules, which are simply algorithms or equations. Following are some learning rules for the neural network −

1. Hebbian Learning Rule

This rule, one of the oldest and simplest, was introduced by Donald Hebb in his book *The Organization of Behavior* in 1949. It is a kind of feed-forward, unsupervised learning.

Basic Concept − This rule is based on a proposal given by Hebb, who wrote −

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

From the above postulate, we can conclude that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times.

Mathematical Formulation − According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step.

$\Delta w_{ji}(t)=\alpha x_i(t).y_j(t)$

Here, $\Delta w_{ji}(t)$ increment by which the weight of connection increases at time step t

α= the positive and constant learning rate

$x_i(t)$ = the input value from pre-synaptic neuron at time step t

$y_i(t)$ = the output of pre-synaptic neuron at same time step t

2. Perceptron Learning Rule

This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function, introduced by Rosenblatt.

Basic Concept − As being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output. If there is any difference found, then a change must be made to the weights of connection.

Mathematical Formulation:

To explain its mathematical formulation, suppose we have 'n' number of finite input vectors, xn, along with its desired/target output vector tn, where n = 1 to N.

Now the output 'y' can be calculated, as explained earlier on the basis of the net input, and activation function being applied over that net input can be expressed as follows −

$$y = f(y_{in}) = \begin{cases} 1, & y_{in} > \theta \\ 0, & y_{in} \leqslant \theta \end{cases}$$

Where θ is threshold.

The updating of weight can be done in the following two cases:

Case I: when t ≠ y, then

w(new)=w(old)+tx

Case II: when t = y, then

No change in weight

3. Delta Learning Rule Widrow−HoffRule

It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square LMSLMS method, to minimize the error over all training patterns. It is kind of supervised learning algorithm with having continuous activation function.

Basic Concept − The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.

Mathematical Formulation

To update the synaptic weights, delta rule is given by:

Δwi=α. xi. ej

Here Δwi = weight change for i^{th} pattern;

α= the positive and constant learning rate;

xi = the input value from pre-synaptic neuron;

ej = (t−yin), the difference between the desired/target output and the actual output yin

The above delta rule is for a single output unit only.

The updating of weight can be done in the following two cases:

Case-I: when t ≠ y, then

w(new)=w(old)+Δww(new)=w(old)+Δw

Case-II: when t = y, then

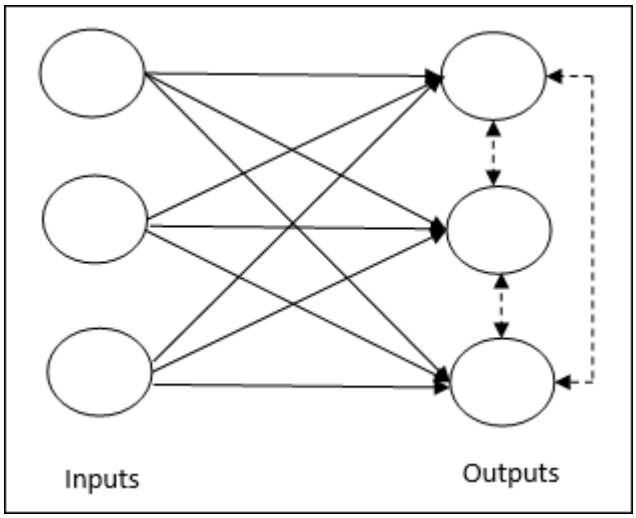
No change in weight

4. Competitive Learning Rule Winner−takes−all

It is concerned with unsupervised training in which the output nodes try to compete with each other to represent the input pattern. To understand this learning rule, we must understand the competitive network which is given as follows −

Basic Concept of Competitive Network − This network is just like a single layer feedforward network with feedback connection between outputs.

The connections between outputs are inhibitory type, shown by dotted lines, which means the competitors never support themselves.



Basic Concept of Competitive Learning Rule − As said earlier, there will be a competition among the output nodes. Hence, the main concept is that during training, the output unit with the highest activation to a given input pattern, will be declared the winner. This rule is also called Winner-takes-all because only the winning neuron is updated and the rest of the neurons are left unchanged.

Mathematical formulation:

Following are the three important factors for mathematical formulation of this learning rule −

Condition to be a winner − Suppose if a neuron yk wants to be the winner then there would be the following condition –

$$y_k = \begin{cases} 1 & if\ v_k > v_j\ for\ all\ j,\ j \neq k \\ 0 & otherwise \end{cases}$$

It means that if any neuron, say Yk , wants to win, then its induced local field theoutputofsummationunit say vk must be the largest among all the other neurons in the network.

Condition of sum total of weight − Another constraint over the competitive learning rule is, the sum total of weights to a particular output neuron is going to be 1. For example, if we consider neuron k then −

$$\sum_j w_{kj} = 1 \qquad for\ all\ k$$

Change of weight for winner − If a neuron does not respond to the input pattern, then no learning takes place in that neuron. However, if a particular neuron wins, then the corresponding weights are adjusted as follows

$$\Delta w_{kj} = \begin{cases} -\alpha(x_j - w_{kj}), & if\ neuron\ k\ wins \\ 0, & if\ neuron\ k\ losses \end{cases}$$

Here α is the learning rate.

This clearly shows that we are favoring the winning neuron by adjusting its weight and if there is a neuron loss, then we need not bother to re-adjust its weight.

5. Outstar Learning Rule

This rule, introduced by Grossberg, is concerned with supervised learning because the desired outputs are known. It is also called Grossberg learning.

Basic Concept − This rule is applied over the neurons arranged in a layer. It is specially designed to produce a desired output d of the layer of p neurons.

Mathematical Formulation

The weight adjustments in this rule are computed as follows

$\Delta wj = \alpha\ (d - wj)$

Here **d** is the desired neuron output and **α** is the learning rate.

❄❄❄❄❄❄❄

11

KERNEL MACHINES & ENSEMBLE METHODS

Unit Structure

11.0 OBJECTIVES

The term "kernel" refers to a set of mathematical functions used in Support Vector Machine to provide a window through which data can be manipulated. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface can transform to a linear equation in a higher number of dimension spaces.

11.1 INTRODUCTION

Kernels or kernel techniques (also known as Kernel functions) are a collection of distinct sorts of pattern analysis algorithms. They are used in conjunction with a linear classifier to tackle a non-linear issue. SVM (Support Vector Machines), which are utilized in classification and regression issues, utilizes Kernels Methods. The SVM employs a "Kernel Trick," in which the data is processed and an optimal boundary for the various outputs is found.

11.2 AN OVERVIEW

What is the definition of a kernel function?

*"Machine Learning is the field of study that gives computers the ability to learn without being **explicitly** programmed."*

*A computer program is said to learn from **experience** E with respect to some **task** T and some **performance measure** P, if its performance on T, as measured by P, improves with experience E." — Tom Mitchell, Carnegie Mellon University*

So, if you want your program to forecast traffic patterns at a busy intersection (task T), you may feed it data from previous traffic patterns (experience E) into a machine learning algorithm, and if it "learns," it will be better at predicting future traffic patterns (performance measure P).

We call supervised learning one of the different types of machine learning jobs (SL). This is a case in which you enter data for which you already know answers (for example, to forecast whether a dog is of a certain breed, we load in millions of canine information/properties such as type, height, skin colour, body hair length, and so on). These traits are referred to identify as 'features' in ML jargon. A data instance is a single entry in this list of features, whereas the collection of everything is the Training Data, which forms the basis of your prediction. For example, if you know a dog's skin colour, body hair length, height, and other characteristics, you can predict the breed it will most likely belong to.

We need to know what a support vector machine is before we can move on to kernels. Support Vector Machines, or SVMs, are supervised learning models with associated learning algorithms that analyse data for classification (classifications mean knowing what belongs to what; for example, an apple belongs to the class 'fruit,' while a dog belongs to the class 'animals,' as shown in fig.1).

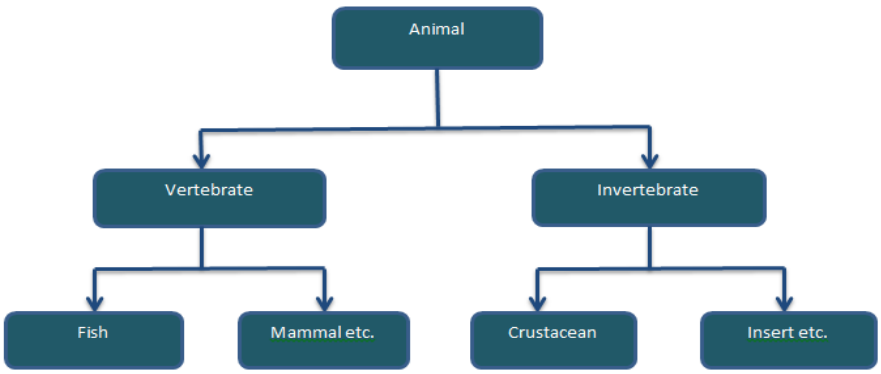


Figure. 1

11.3 OPTIMAL SEPARATING HYPERLANE

SVM Hyperplane Separation

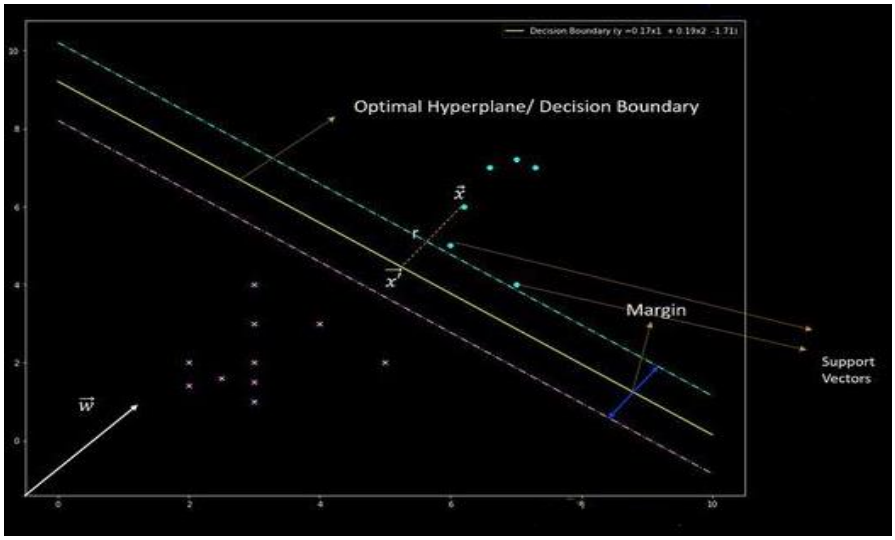
The supervised machine learning algorithm Support Vector Machine is utilized in both classification and regression of models. The concept is straightforward located a plane or a boundary that divides data into two classes.

Supporting Characteristics:

Support vectors are data points around the decision border that are the most difficult to categorize, and they are the key to SVM being the best decision surface. The ideal hyperplane is derived from the function class with the smallest capacity, i.e., the smallest number of independent features/parameters.

Hyperplane Separation:

An example of a scatter plot is shown below:



Given a linearly separable data set, the best separating hyperplane in a binary classification issue is the one that correctly classifies all of the data while being the furthest away from the data points. The hyperplane that maximizes the margin, defined as the distance between the hyperplane and the closest data point, is said to be the hyperplane that maximizes the margin.

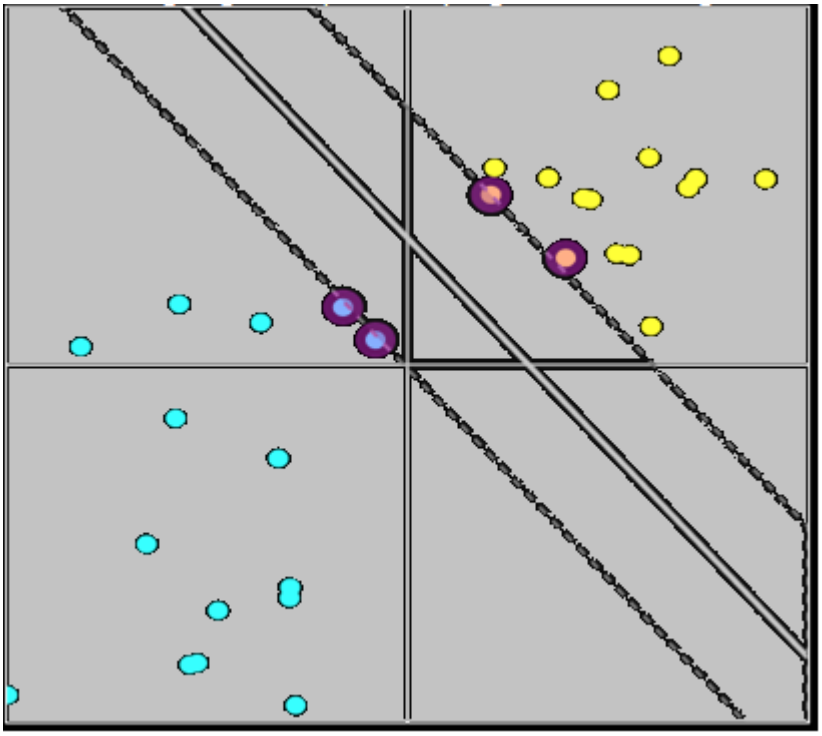
The following diagram illustrates the concept of this classifier's optimality. The same distribution as the training data is used to draw new test points. As a result, if the separating hyperplane is far from the data points, previously unseen test points will almost certainly fall far away from the hyperplane or in the margin. As a result, the wider the margin, the less likely the points will be on the incorrect side of the hyperplane.

The problem of finding the best separating hyperplane can be expressed as a convex quadratic programming problem that can be solved using well-known methods.

The optimal separating hyperplane should not be confused with the Bayes classifier, which is the best classifier for a given problem regardless of available data but impossible to achieve in practice, whereas the optimal separating hyperplane is only the best linear classifier one can produce given a specific data set.

One of the key concepts of support vector machines is the optimal separation hyperplane. It gives rise to the so-called support vectors, which are data points that lie on the hyperplane's margin border. The hyperplane is supported by these points because they provide all of the necessary information to compute the hyperplane: deleting other points has no effect on the optimal separating hyperplane. Using this fact as an example, one can add points to the data set without affecting the hyperplane as long as the points are outside the margin.

The ideal separation hyperplane and its margin for a two-dimensional data set are shown in the graph below. The highlighted points on the margin boundary are the support vectors.



Generate a new data set

The optimal separating hyperplane parameters :
$\boldsymbol{w}$= [-0.23, -0.24]
b= 0.76

The optimal separating hyperplane has been found with a margin of 2.98 and 5 support vectors.
This hyperplane could be found from these 5 points only.

Draw a random test point

Margin of a classifier

Given a linearly separable data set $\{(x_i,y_i)\}_{i=1}^{N}$ with input vectors $x_i \in X \subseteq R^d$ and labels $y_i \in \{-1,+1\}$, the **margin** of a classifier $f(x)=\text{sign}(g(x))$ with separating surface H is

$\gamma = \min_{i \in \{1,\ldots,N\}} \text{dist}(x_i,H)$

For a linear classifier, with $g(x)=\langle w,x \rangle + b$, H is a separating hyperplane:

$H=\{x \in X : \langle w,x \rangle + b = 0\}$

and the distance can be calculated as follows:

$$\text{dist}(x_i,H)=|\langle w,x_i\rangle+b|\|w\|2$$

(the use of the $\ell 2$-norm is implied by the classical inner product in Rd).

The optimal separating hyperplane

The optimal separating hyperplane $H*$ is the one that maximizes the margin:

$$(w*,b*)=\text{argmax}_{w,b}\ \min_{i\in\{1,\ldots,N\}}|\langle w,x_i\rangle+b|\|w\|2$$

Solving this optimization problem is difficult, in particular due to the scaling by the inverse of $\|w\|2$, which makes the problem nonconvex. In addition, the problem is ill-posed due to an infinite number of solutions (obtained by scaling the parameters of one of them).

A possible alternative would be to fix $\|w\|2$, but again this leads to a nonconvex optimization problem (with a quadratic equality constraint). A better idea is to consider the so-called **canonical hyperplane** by fixing the numerator. That is, we impose the constraint

$$\min_{i\in\{1,\ldots,N\}}|\langle w,x_i\rangle+b|=1$$

when searching for the parameters $w*$ and $b*$. Given that we search for a classifier that correctly classifies all the data, we have, for all $i\in\{1,\ldots,N\}$, $\text{sign}(\langle w,x_i\rangle+b)=y_i$, and $y_i(\langle w,x_i\rangle+b)=|\langle w,x_i\rangle+b|$. Thus, the constraint becomes

$$\min_{i\in\{1,\ldots,N\}}y_i(\langle w,x_i\rangle+b)=1$$

Assuming the canonical form of the hyperplane, the margin is now given by

$$\gamma=1\|w\|2\min_{i\in\{1,\ldots,N\}}|\langle w,x_i\rangle+b|=1\|w\|2$$

In this case, **maximizing the margin is equivalent to minimizing the norm of** w; and $H*$ can be found by solving

$$(w*,b*)=\text{argmin}_{w,b}\ \text{s.t.}\ \ 12\|w\|22\min_{i\in\{1,\ldots,N\}}y_i(\langle w,x_i\rangle+b)=1$$

(minimizing $12\|w\|22$ amounts to minimizing $\|w\|$ but is more easily handled from an optimization viewpoint). And since we minimize $\|w\|2$, we can relax the constraint to

$$(w*,b*)=\text{argmin}_{w,b}\ \text{s.t.}\ \ 12\|w\|2y_i(\langle w,x_i\rangle+b)\geq 1,\ i=1,\ldots,N.$$

At this point, we formulated the determination of the optimal separating hyperplane as an optimization problem known as a convex quadratic program for which efficient solvers exist.

Support vectors

The support vectors are the data points x_i that absolutely coincide with the margin border and hence satisfy

$$y_i(w,x_i+b)=1.$$

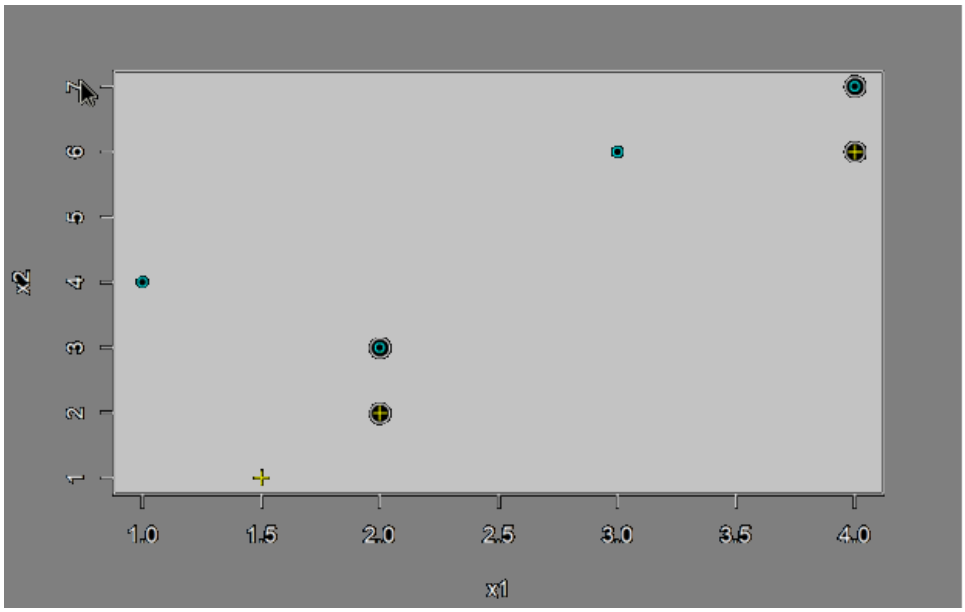
These points (when taken together with their labels) give enough data to compute the best separation hyperplane. To see this, consider that removing a non-active constraint from the quadratic programme above is equivalent to deleting another point from the data set.

Furthermore, as long as the points are accurately classified by H and outside the margin, one can add as many points to the data set as desired without influencing H. This is essentially the same as applying restrictions to the optimization problem and narrowing the viable set. The answer to the optimization problem does not change as long as the previous solution H remains feasible.

11.4 SEPARATING DATA WITH MAXIMUM MARGIN

Separators with the Largest Margin

In a two-dimensional space, a Maximal Margin Separator is a hyperplane (in this case, a line) that entirely separates two classes of observations while leaving the maximum space between the line and the closest observation. The support vectors are the closest observations. The support vectors are the circled locations in the graph below. Because the support vectors are evenly spaced in this case, the most effective margin separator would be a line that runs halfway between each pair of support vectors and matches their slope. (In the case of three support vectors, the line will run parallel to the slope of the side with two support vectors.)



Identifying the line

We can determine the equation for the hyperplane by first determining the equation for the line because we have a very simple plot and know what our support vectors are.

x2 = m*x1 + b

Compare and contrast the blue cross point (2, 2) with the red circle point (2, 3). It's worth noting that a position midway between them (vertically) would be (2, 2.5). On our maximal margin separator, this is point 1. Compare and contrast the blue cross and red circle observations (4, 6). (4, 7). It's worth noting that a position midway between those two would be (4, 6.5). On our maximal margin separator, this is point number two.

By dividing x22 – x21 by x12 – x11, we can now compute the slope. This equates to (6.5-2.5)/(4-2) = 2. Our slope is x2 = 2 * x1 + b, which we can substitute for m in the equation:

x2 = 2 * x1 + b

We know what our points are. We can sub in either one to find our intercept (b). Subbing in the point at (4, 6.5), we get:

6.5 = 2 * 4 + b

or

6.5 = 8 + b

We can subtract 8 from both sides to get b:

6.5 – 8 = b – 8

-1.5 = b

So now we know that our line equation is:

x2 = 2 * x1 + -1.5

A hyperplane's equation

$beta_0 + (beta_1 * x_1) + (beta_2 * x_2) = 0$

with the proviso that $(beta_1^2 + beta_2^2) = 1$

It's important to note that the hyperplane equation must equal zero. As a result, all points above the hyperplane are positive, whereas all locations below it are negative. We can then categorize our points into positive or negative categories based on which class they belong to.

Let's take this in steps:

First let's fill in what we know from our point on the hyperplane and our linear equation. $beta_0$ is our intercept, so fill that in.

$-1.5 + beta_1 x_1 + beta_2 x_2 = 0$

$beta_1$ is our slope, so fill that in.

$-1.5 + 2 x_1 + beta_2 x_2 = 0$

We know a point on our hyperplane is (4, 6.5), so we can fill in x_1 and x_2

$-1.5 + 2 * 4 + beta_2 * 6.5 = 0$

In this case, $beta_2 = -1$.

Our hyperplane equation, discounting the caveat is:

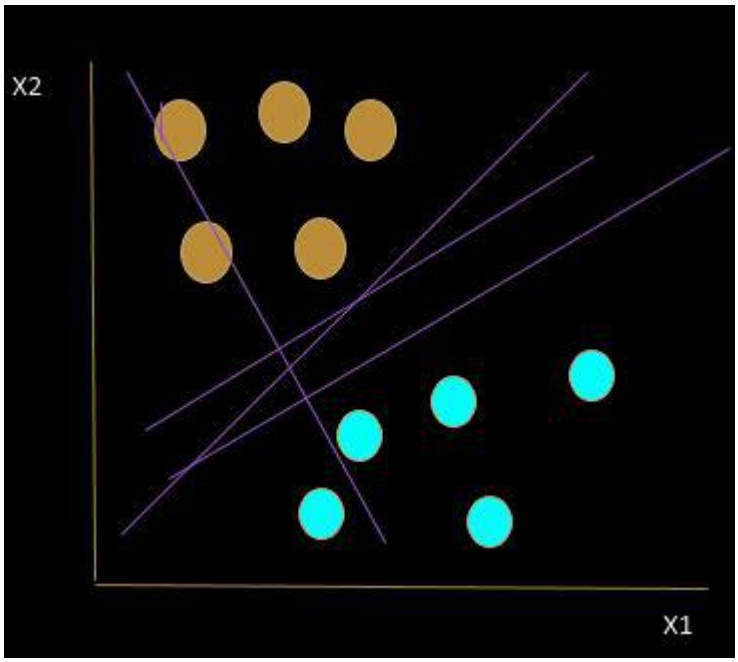
$-1.5 + 2(x_1) + -1(x_2) = 0$

11.5 SUPPORT VECTOR MACHINE (SVM)

Algorithm for Support Vector Machines

SVM is a supervised machine learning technique that may be used for both classification and regression. Though we might also argue regression difficulties, categorization is the best fit. The goal of the SVM algorithm is to find a hyperplane in an N-dimensional space that categorizes data points. The hyperplane's size is determined by the number of features. If there are only two input characteristics, the hyperplane is merely a line. When the number of input features reaches three, the hyperplane transforms into a two-dimensional plane. When the number of features exceeds three, it becomes impossible to imagine.

Consider two independent variables, x1, and x2, as well as one dependent variable, either a blue or a red circle.

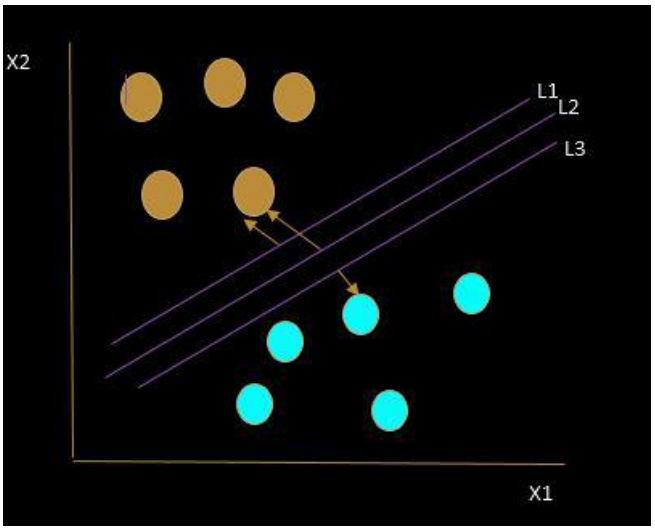


Linearly Separable Data points

It's evident from the diagram above that there are several lines (our hyperplane is a line because we're only examining two input features, x1 and x2) that separate our data points or classify them into red and blue circles. So, how do we pick the best line, or more broadly, the optimal hyperplane, to separate our data points?

Choosing the most appropriate hyper-plane:

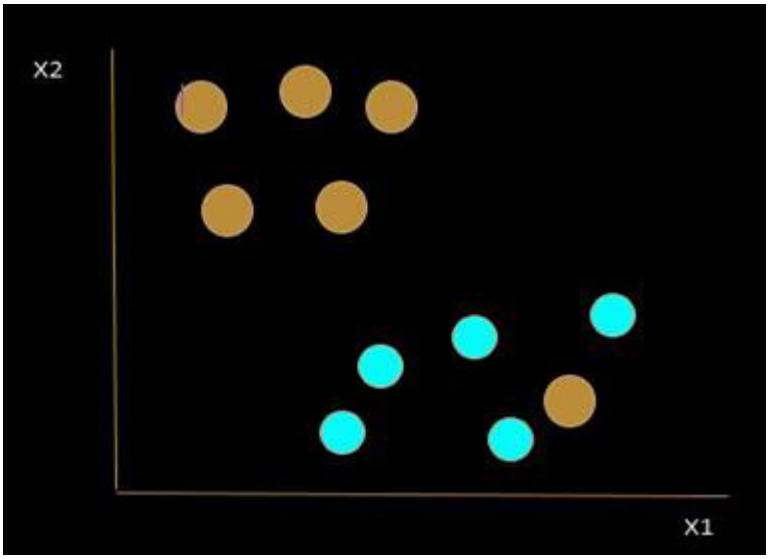
The hyperplane that represents the greatest separation or margin between the two classes is a viable choice as the best hyperplane..



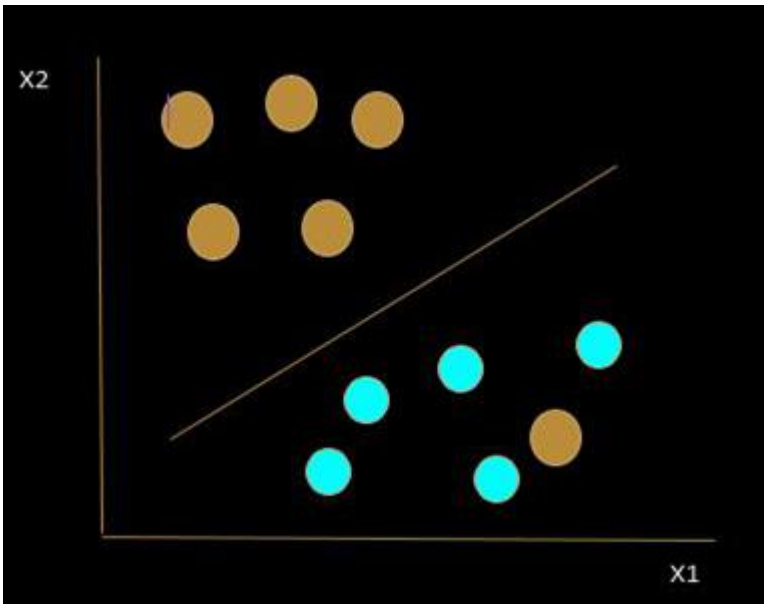
As a result, we select the hyperplane with the greatest distance between it and the nearest data point on each side. The maximum-margin

hyperplane/hard margin is known if such a hyperplane exists. So, based on the diagram above, we'll go with L2.

Let's take a look at a scenario like the one below.



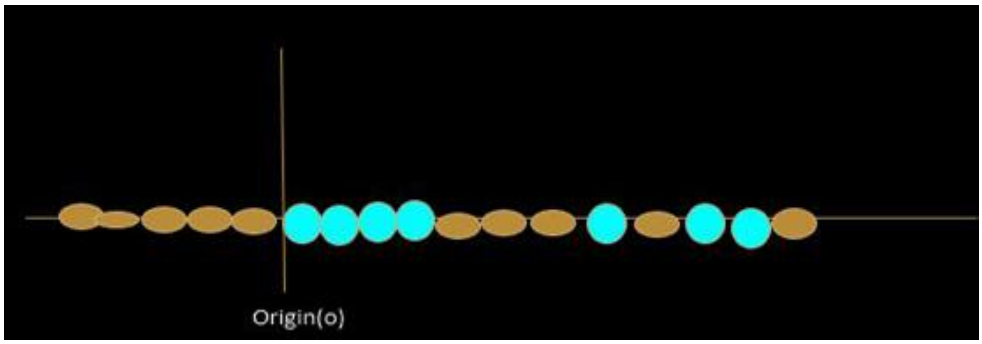
We have one blue ball within the red ball's boundary. So, how does SVM categorize the information? It's that easy! An outlier of blue balls is the blue ball in the border of red ones. The SVM algorithm can ignore outliers while locating the optimum hyperplane that maximizes the margin. Outliers are not a problem with SVM.



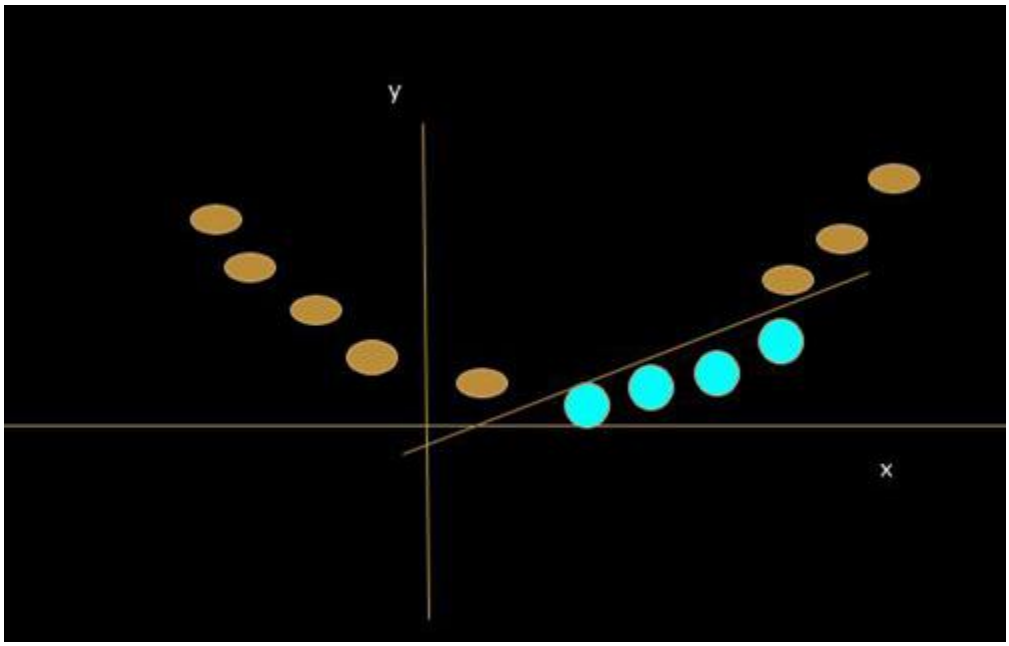
So, for this type of data, SVM finds the greatest margin, as it has for previous data sets, and then adds a penalty each time a point crosses the margin. In these instances, the margins are referred to as soft margins. The SVM tries to minimize *(1/margin+∧(∑penalty))*. when the data set has a soft margin. A common penalty is the loss of a hinge. There will be no hinge loss if there are no infractions. If a violation is proportionate to the

distance of the violation, the loss is proportional to the distance of the violation.

We've been discussing linearly separable data (the group of blue balls and red balls can be separated by a straight line/linear line) up until now. What should you do if your data isn't linearly separable?



Assume our data is as depicted in the diagram above. SVM overcomes this by utilizing a kernel to create a new variable. We establish a new variable Y_i as a function of distance from the origin o and call it a point xi on the line. If we plot this, we get something like this.



The new variable y is formed as a function of distance from the origin in this scenario. Kernel refers to a non-linear function that introduces a new variable.

The SVM kernel is a function that takes a low-dimensional input space and transforms it into a higher-dimensional space, converting a not-separable problem into a separable problem. It is most beneficial in cases with non-linear separation. Simply explained, the kernel does some fairly sophisticated data transformations before determining the best method for separating the data based on the labels or outputs specified.

SVM has the following advantages:

- It is effective in high-dimensional scenarios.
- It saves memory by using a subset of training points termed support vectors in the decision function.
- Different kernel functions can be supplied for the decision functions, including custom kernels.

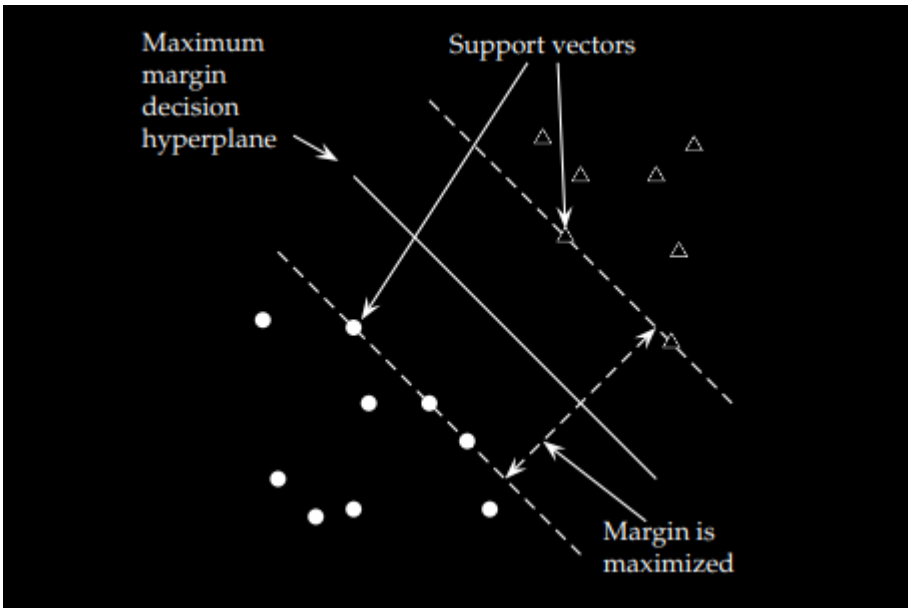
11.6 FINDING THE MAXIMUM MARGIN

Maximal Margin Classifier

The best hyperplane defined in the (rare) circumstance where two classes are linearly separable is the maximal margin classifier. It may be possible to build a p-dimensional hyperplane $h(X)=\beta0+\beta1X1+\beta2X2\cdots+\beta pXp=xTi\beta+\beta0=0h(X)=\beta0+\beta1X1+\beta2X2\cdots+\beta p$ $Xp=xiT\beta+\beta0=0$ given a $n\times pn\times p$ data matrix XX with a binary response variable specified as $y\in[-1,1]y\in[-1,1]$. so that each class's observations are on opposing sides of the hyperplane If is restricted to be a unit vector, $||\beta||=\sum\beta2=1||\beta||=\sum\beta2=1$, then the product of the hyperplane and response variables is positive perpendicular distances from the hyperplane, the smallest of which is known as the hyperplane margin, MM.

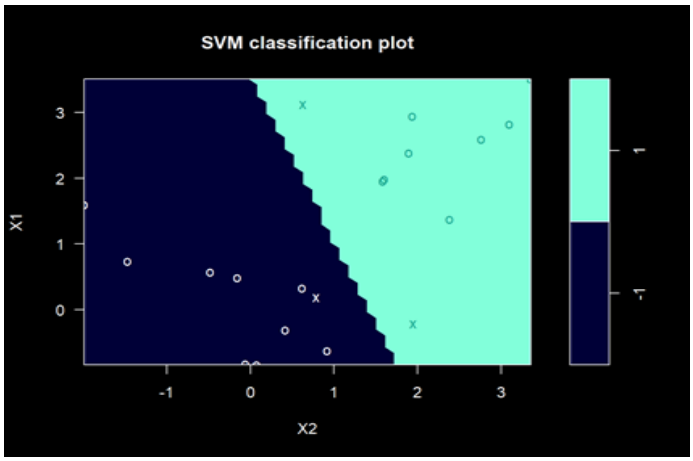
$yi(x'i\beta+\beta0)\geq M.$ $yi(xi'\beta+\beta0)\geq M.$

The hyperplane with the maximum margin, $max\{M\}max\{M\}$, is the maximal margin classifier, subject to $||\beta||=1||\beta||=1$. A separate hyperplane is uncommon. In fact, even if a separating hyperplane exists, the resulting margin is likely to be too small. The maximum margin classifier is shown below.

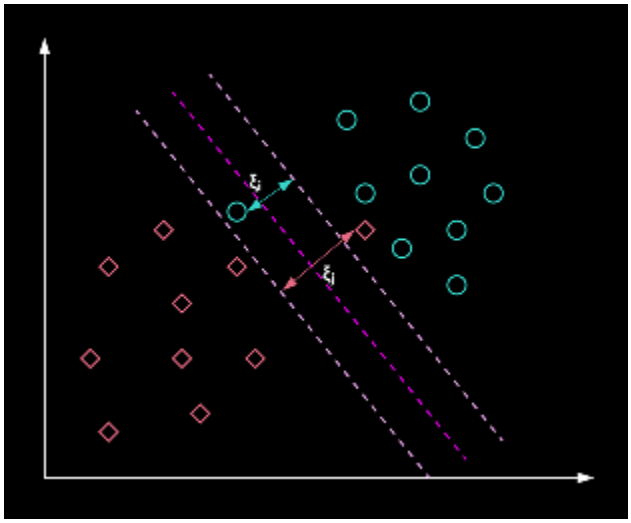


Maximum marginal classifier

Two linearly separable classes, $y \in [-1,1] y \in [-1,1]$, are characterised by two features, X1X1 and X2X2, in the data set. The coding is trivial; all that matters is that the visualisation is created.



11.7 THE NON-SEPARABLE CASE: SOFT MARGIN HYPERLANE



Introduction

One of the most common classification approaches, Support Vector Machine (SVM), tries to reduce the frequency of misclassification mistakes directly. Although there are many resources available to learn the basics of how Support Vector Machines (SVMs) function, SVMs require advanced ideas in practically all real-world applications (when the data is linearly inseparable).

Linear Inseparability

Let's explain the need for Soft Margin and Kernel Trick before proceeding on to the topics. Let's say we have some data that can be represented in 2D space as follows:

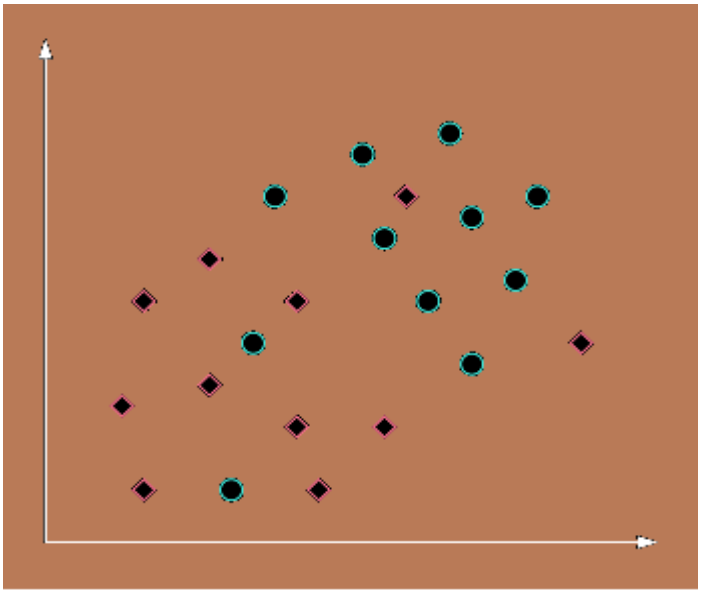


Figure 1: Data representation where the two classes are not linearly separable

The data is linearly inseparable, as shown in the image. There is no precise linear decision boundary that can perfectly separate the data. In higher-dimensional representations, we can have a similar problem. This is due to the fact that the characteristics we get from the data frequently don't contain enough information to clearly distinguish the two classes. In many real-world applications, this is usually the case. Fortunately, experts have previously devised strategies to deal with such circumstances. Let's have a look at what they are and how they function.

Formulation of a Soft Margin

This concept is built on a simple premise: allow SVM to make a fixed number of mistakes while keeping the margin as large as feasible to ensure that other points are correctly identified. SVM's objective can be changed to accomplish this.

Motivation

Let's go over the reasons for having this type of formulation in a little more detail. As previously stated, data in practically all real-world applications is linearly inseparable. To avoid over fitting, we might not want to use a decision boundary that precisely separates the data in rare circumstances where the data is linearly separable. Take a look at the illustration below:

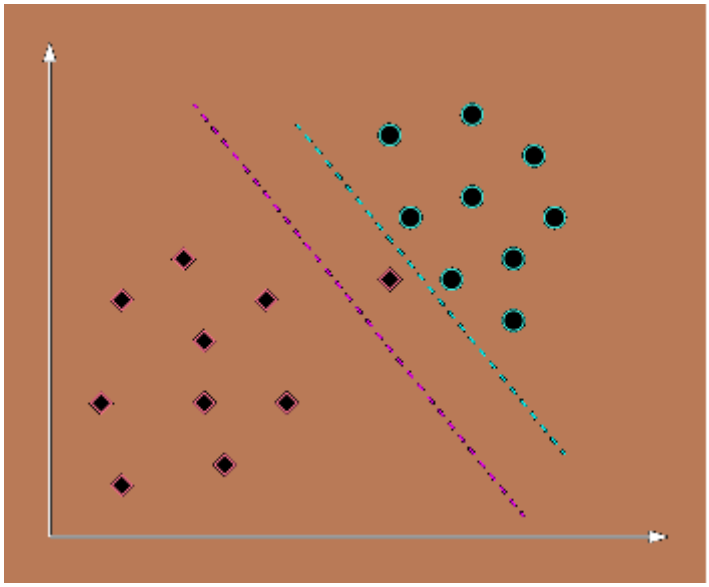


Figure 2: Which decision boundary is better? Red or Green?

All of the training spots are neatly separated by the red decision boundary. Is it, nevertheless, a smart idea to have a decision boundary with such a small margin? Do you believe a decision boundary like this will generalize well to unknown data? The solution is as follows:

No. The green decision boundary has a larger margin, thus it can generalize well to unknown data. In this regard, soft margin formulation could aid in avoiding the over fitting issue.

What Does It Mean (Mathematically)?

Let's look at how we can change our goal to achieve the desired actions. In this new setting, we'd like to reduce the following goal:

$$L = \frac{1}{2}\|w\|^2 + C(\#\ of\ mistakes)$$

Equation 1

This is different from the second term's intended goal. C is a hyperparameter that determines the trade-off between maximizing margins and reducing errors. When C is small, classification errors are less important, and the focus is more on maximizing the margin, but when C is large, the focus is on preventing misclassification at the expense of maintaining a small margin.

However, it's worth noting that not all errors are created equal. Data points on the wrong side of the decision boundary that are far away should be penalized more than those that are closer. Let's look at how this could be done with the help of the diagram below.

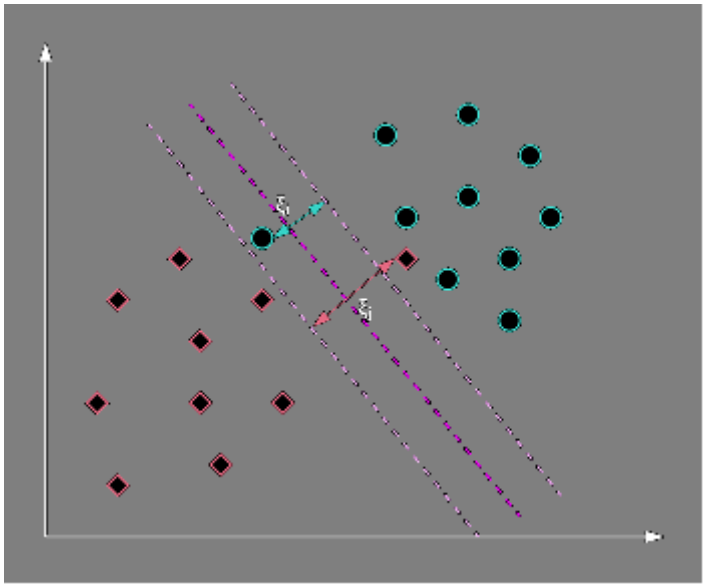


Figure 3: The penalty incurred by data points for being on the wrong side of the decision boundary

The objective is to establish a slack variable ξ_i for each data point x _i. If x _i is on the incorrect side of the margin, the value of ξ_i equals the distance between x_i and the associated class's margin, otherwise zero. As a result, the points on the incorrect side of the margin would be penalized more.

According to this concept, each data point x_i must meet the following constraint:

$$y_i(\vec{w}.\vec{x}_i + b) \geq 1 - \xi_i$$

Equation 2

In this case, the left-hand side of the inequality might be compared to classification confidence. A confidence value of ≥ 1 indicates that the classifier properly classified the point. If the confidence score is less than one, the classifier failed to identify the point properly, resulting in a linear penalty of ξ_i.

Our goal is to minimize the following function given these constraints:

$$L = \frac{1}{2}\|\vec{w}\|^2 + C\sum_i \xi_i + \sum_i \lambda_i(y_i(\vec{w}\ .\ \vec{x}_i + b) - 1 + \xi_i)$$

Equation 3

Where we used the Lagrange Multiplier concepts to optimize the loss function under constraints. Consider the objective of SVM, which deals with linearly separable cases (as given below).

$$L = \frac{1}{2}\|\vec{w}\|^2 + \sum_i \lambda_i(y_i(\vec{w}.\vec{x}_i + b) - 1)$$

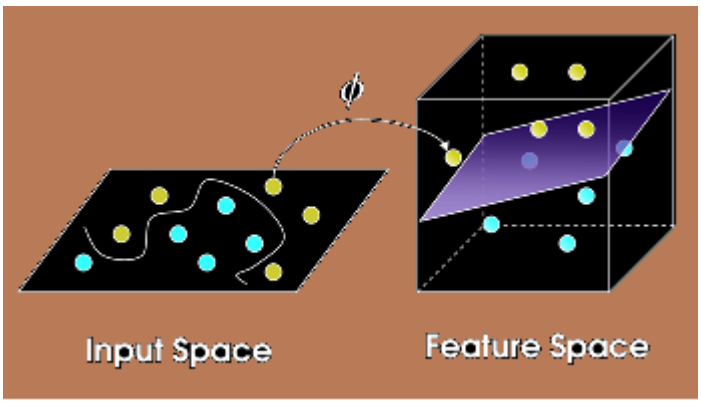
Equation 4

Only the ξ I terms are added to the updated target; everything else remains the same.

Note that λ_is corresponding to locations closest to the margin and on the wrong side of the margin (i.e. having non-zero ξ_i would be non-zero in the final solution since they play a significant role in locating the decision boundary, thus making them the support vectors.

11.8 KERNEL TRICK

Support Vector Classification's Kernel Trick



The kernel trick appears to be one of the most perplexing concepts in statistics and machine learning; at first glance, it appears to be genuine mathematical sorcery, not to mention a lexical ambiguity (does kernel refer to a non-parametric way to estimate a probability density (statistics), the set of vectors v for which a linear transformation T maps to the zero vector — i.e. T(v) = 0 (linear algebra), the set

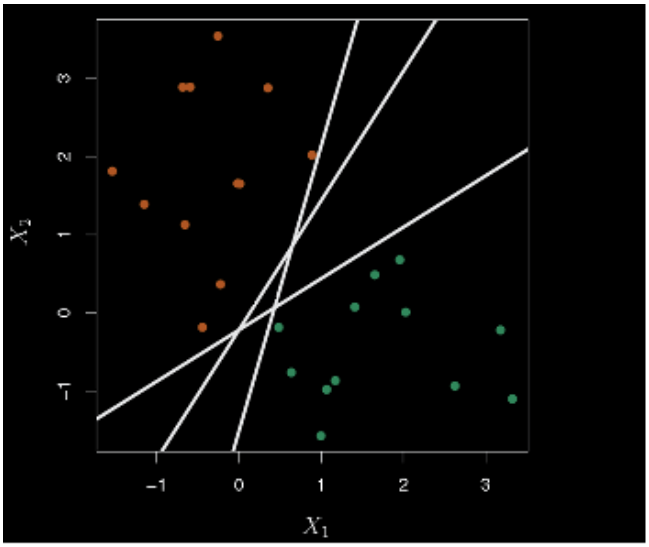
Although there are some challenges to grasping the kernel trick, understanding how kernels are utilized in support vector classification is critical. It's vital to understand for practical reasons: implementing support vector classifiers necessitates providing a kernel function, and there are no established, generic methods for determining which kernel would work best for your specific data.

The kernel trick highlights some core notions about multiple ways to represent data and how machine learning algorithms "understand" these diverse data representations on a more conceptual level. Finally, the kernel trick's seeming mathematical sleight of hand warrants deeper investigation into what it signifies.

Support Vector Classification: An Overview

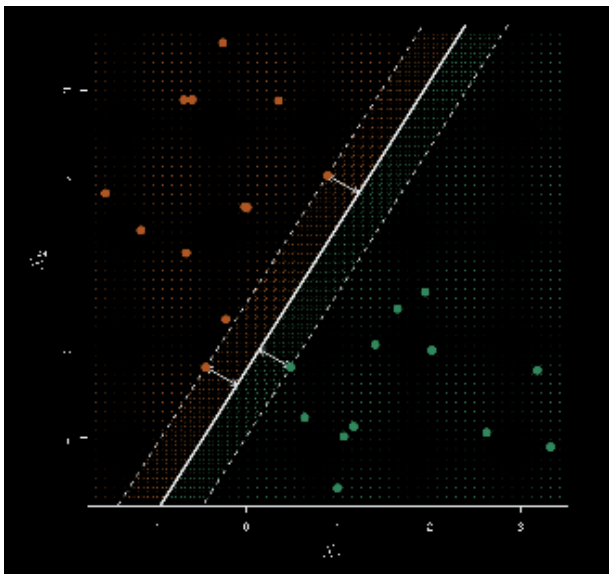
Support vector classification is based on how one may attempt to categorize data points into several target classes in a fairly natural way. If the classes in our training data can be split by a line or some other boundary, we can simply classify the data based on which side of this decision boundary it falls.

We can partition the data in the following 2-d example with any of the three lines, and then assign classes based on whether the observation is above or below the line. The data are two-dimensional vectors defined by the features X1 and X2, with class labels of y = 1 (blue) or y = 0 (orange) (red).



An example dataset demonstrating the linear separation of classes.

Training a linear support vector classifier is an optimization issue, as is practically every problem in machine learning and life. We want to maximize the margin, which is the distance between the nearest pair of data points from opposite classes. The data observations that "support," or decide, the decision boundary are referred to as support vectors. We determine the maximal margin hyperplane, or optimal separating hyperplane, to train a support vector classifier, which optimally separates the two classes to generalise to new data and produce correct classification predictions.



The points on the dashed lines are the support vectors. The margin, shown by the arrows, is the distance between the dashed and solid lines.

In higher dimensions, support vector machines are even more difficult to interpret. It's considerably more difficult to picture how the data can be separated linearly and what the decision border will look like. In p-dimensions, a hyperplane is a p-1 dimensional "flat" subspace that is contained within the larger p-dimensional space. In two dimensions, the hyperplane is nothing more than a line. The hyperplane is a standard 2-d plane in three dimensions. In terms of mathematics, we have the following:

Equation of a Hyperplane in 2 dimensions

$$\beta_0 + \beta_1 \mathbf{X_1} + \beta_2 \mathbf{X_2} = 0$$

Equation of a Hyperplane in p-dimensions

$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} = 0$$

We have points on the hyperplane:

$$\mathbf{X} = [\mathbf{X_1}, \ldots, \mathbf{X_p}]^T$$

where $\mathbf{X}$ is a feature vector

We can use this to make classifications by considering one class defined by:
$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} > 0$$

With the other class defined by:
$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} < 0$$

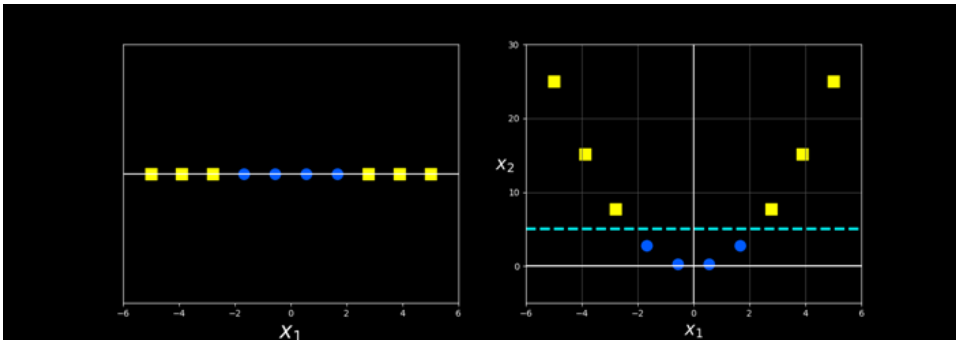
The classification rules are defined by the decision boundary and the equations defining a hyperplane

The concept of linearly separable data is used in support vector classification. When data is not entirely linearly separable, "soft margin" classification can allow certain classification errors on the training data. However, data is rarely linearly separable, and to build a support vector classifier, we must transform the data into a higher-dimensional space.

Non-linear transformations

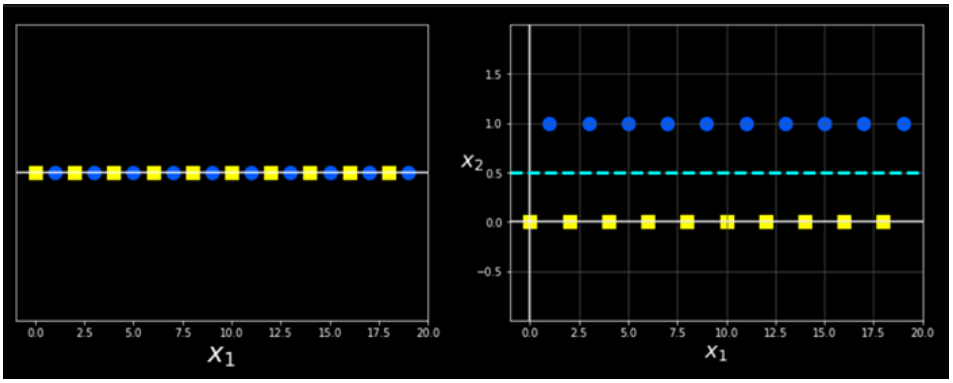
We apply transformations to the data if it is not linearly separable in the original, or input, space. These transformations translate the data from the original space into a higher dimensional feature space. The goal is for the classes to be linearly separable in this higher dimensional feature space after the transformation to the higher dimensional space. After that, we may fit a decision boundary between the classes and make predictions. In this higher-dimensional space, the decision boundary will be a hyperplane.

Higher-dimensional data is difficult to visualize, so we'll start with some 1-dimensional data manipulations. The image on the left shows our original data points in this case. This data is not linearly separable in one dimension, but after applying the transformation $\phi(x) = x^2$ to our feature space and adding this second dimension, the classifications become linearly separable.



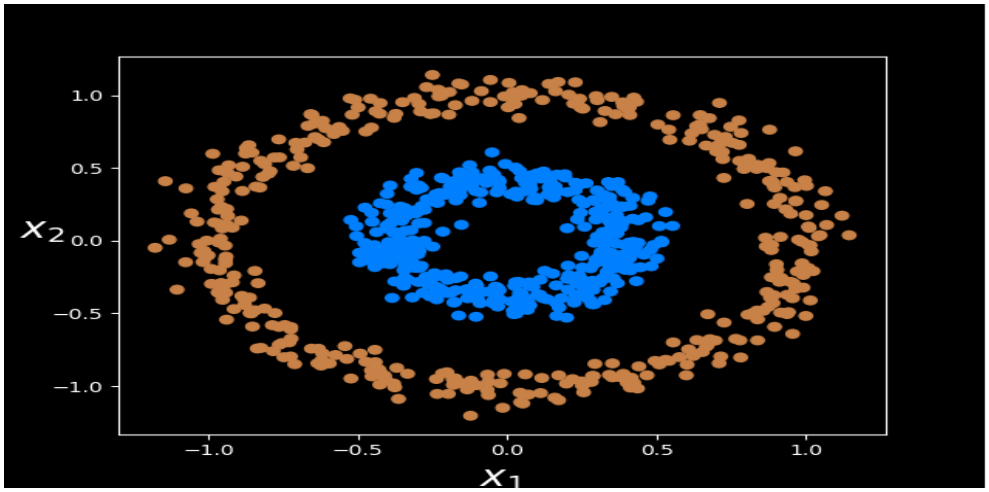
After a quadratic translation to 2-dimensions, this data becomes linearly separable. For the time being, we're only looking at transformations of the original data to higher dimensions that allow the data to be separated linearly. These are merely functions, and there are a plethora of others that can map data to any number of higher dimensions.

The transformation $\phi(x) = x \bmod 2$ is used here.



In two dimensions, this transformation allows us to linearly divide the even and odd X1 values..

Now consider a situation in which our original data is not linearly separable in two dimensions. Our original data, which cannot be linearly separated, is shown below.

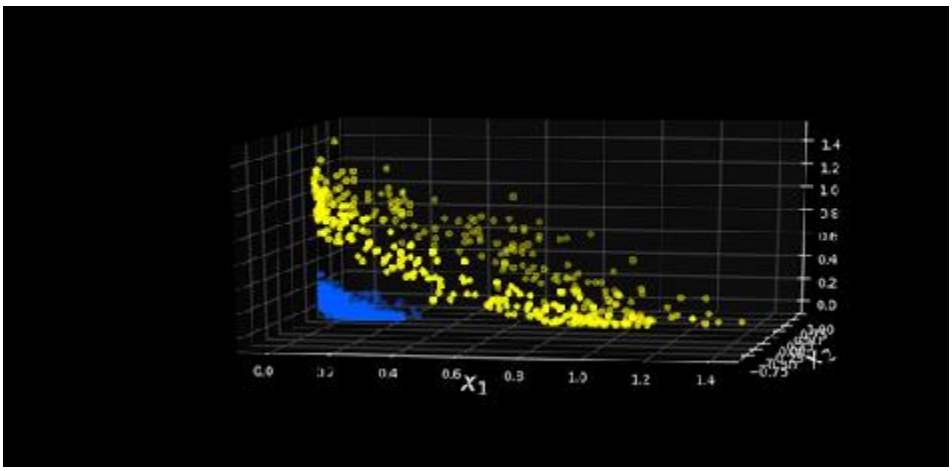


After the following transformation:

Equation 5 Second-degree polynomial mapping

$$\phi(\mathbf{x}) = \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} {x_1}^2 \\ \sqrt{2}\,x_1 x_2 \\ {x_2}^2 \end{bmatrix}$$

Our data becomes linearly separable (by a 2-d plane) in 3-dimensions.



After applying the 2nd-degree polynomial transformation on linearly separable data in 3-d,

There are a variety of transformations that allow data to be split linearly into higher dimensions, but not all of them are kernels. The kernel function has a peculiar trait that makes it very effective in training support vector models, and this property is sometimes referred to as the kernel trick for optimizing non-linear support vector classifiers.

- **THE KERNEL TRICK**

We've seen how higher-dimensional transformations can help us separate data so that classification predictions can be made. It appears that we will have to operate on the higher dimensional vectors in the modified feature space to train a support vector classifier and maximize our objective function. In real-world applications, data may contain numerous features, and transformations using multiple polynomial combinations of these features will result in extremely large and prohibitive processing costs.

This problem can be solved using the kernel trick. Instead of explicitly applying the transformations (x) and representing the data by these transformed coordinates in the higher dimensional feature space, kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations x (with the original coordinates in the lower dimensional space).

The data set × is represented in kernel techniques by a n x n kernel matrix of pairwise similarity comparisons, with the entries (i j) specified by the kernel function: k(xi, xj). The mathematical attribute of this kernel function is unique. As a modified dot product, the kernel function is used. We have the

11.9 DEFINING KERNEL

DEFINITION OF THE KERNEL

A kernel function is a function that accepts vectors in the original space as input and returns the dot product of the vectors in the feature space.

In more formal terms, if we have data X, Z ϵ X and a map K^N then K(X,Z) = <ϕ(X),ϕ(Z)>

Is a kernel function

The dot product of the transformed vectors in the higher dimensional space is returned by our kernel function, which accepts inputs in the original lower-dimensional space. There are additional theorems that state that such kernel functions must exist under specific conditions.

Consider that each coordinate of the transformed vector ϕ(x) is just some function of the coordinates in the corresponding lower-dimensional vector x to help explain how the kernel function is equal to the dot product of the transformed vectors.

The kernel method for the 2nd-degree polynomial, for example, is shown below, and this transformation was represented in 3-d in a previous picture. The coordinates of the converted vectors are functions of the two components x1 and x2. As a result, the dot product will only have components x1 and x2. The kernel function also accepts x1 and x2 as arguments and returns a real value. In addition, the dot product always returns a valid number.

Equation 5-9. Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\,a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\,b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

The dot product of the altered feature vectors, which is identical to our 2nd-degree polynomial kernel function, is shown on the left.

The polynomial kernel k(a,b) = (a^T * b)² is used here as the kernel function.

The kernel trick's ultimate benefit is that the objective function we're optimizing to match the higher-dimensional decision boundary only incorporates the modified feature vectors' dot product. As a result, we can simply use the kernel function to replace these dot product terms, and we don't even need to use $\phi(\mathbf{x})$.

- Solution of the dual problem gives us:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \mathbf{x}_i$$

- The decision boundary:

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0$$

- The decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0 \right]$$

- Mapping to a feature space, we have the decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i)) + w_0 \right]$$

The kernel function is used to replace the dot product of the converted vectors in the bottom equation.

Remember, our data is only linearly separable as the vectors $\phi(x)$ in the higher dimensional space, and we are finding the optimal separating hyperplane in this higher dimensional space without having to calculate or in reality even know anything about $\phi(x)$.

11.10 LET US SUM UP

Kernels or kernel techniques are a collection of distinct sorts of pattern analysis algorithms. They are used in conjunction with a linear classifier to tackle a non-linear issue. The SVM employs a "Kernel Trick," in which the data is processed and an optimal boundary for the various outputs is found. Support Vector Machines, or SVMs, are supervised learning models with associated learning algorithms that analyze data for classification. The ideal hyperplane is derived from the function class with the smallest capacity, i.e., the smallest number of independent features/parameters.

The ideal separation hyperplane and its margin for a two-dimensional data set are shown in the graph below. The hyperplane is supported by data points that lie on the hyperplane's margin border. Deleting other data points does not affect the optimal separating hyperplane.

11.11 LIST OF REFERENCES

1. https://www.educba.com/kernel-methods/
2. https://towardsdatascience.com/kernel-function-6f1d2be6091
3. https://www.geeksforgeeks.org/separating-hyperplanes-in-svm/
4. https://mlweb.loria.fr/book/en/optimalhyperplane.html

5. https://deannaschneider.wordpress.com/2018/06/28/understanding-a-maximal-margin-separator/
6. https://bookdown.org/mpfoley1973/data-sci/maximal-margin-classifier.html
7. https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe
8. https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

11.12 UNIT END EXERCISES

1. What is the optimal separating hyperplane with proper example?
2. How can a separating hyperplane with a maximal margin be found?
3. What do know about Hard Margin SVM and Soft Margin SVM?
4. How do you find the maximum margin of a hyperplane?
5. How can we classify a non-linearly separable set of data points using a large margin classifier like SVM?
6. What is the primary motivation for using the kernel trick in machine learning algorithms?
7. How do you define a kernel function?
8. Which classifier identifies optimal hyperplane in the training phase?
9. How does SVM calculate Max margin?
10. What is the working rule for kernel method?
11. What is the purpose of kernel trick?

❄❄❄❄❄❄❄

12

ENSEMBLE METHODS GAUSSIAN MIXTURE MODELS

Unit Structure

12.0 GAUSSIAN MIXTURE MODELS

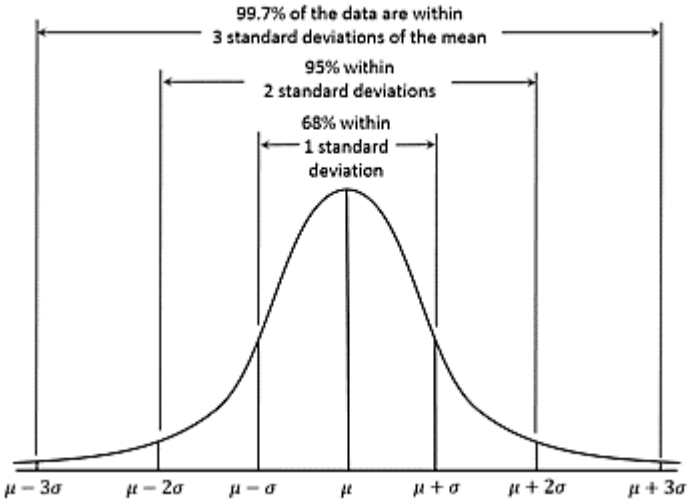
Gaussian Mixture Model is a universally used model for generative unsupervised learning or clustering, called as Expectation-Maximization (EM) Clustering and is based on the optimization strategy. They are also used for representing Normally Distributed subpopulations within an overall population. The significance of these models is they do not require which subpopulation a data point belongs to and allows the model to learn the subpopulations automatically. This constitutes a form of unsupervised learning.

A Gaussian is a type of distribution, and it is a popular and mathematically convenient type of distribution. A distribution is a listing of outcomes of an experiment and the probability associated with each outcome.

Example of cyclist's speeds.

Speed (Km/h)	Frequency
1	4
2	9
3	6
4	7
5	3
6	2

The cyclist reaches the speed of 1 Km/h four times, 2Km/h nine times, 3 Km/h and so on. It looks like it follows a kind of bell curve the frequencies go up as the speed goes up and then it has a peak value and then it goes down again, and we can represent this using a bell curve otherwise known as a Gaussian distribution.



A Gaussian distribution is a type of distribution where half of the data falls on the left of it, and the other half of the data falls on the right of it.

Gaussian distribution would be a great distribution to model the data in those cases where the data reaches a peak and then decreases. Similarly, in Multi Gaussian Distribution, we will have multiple peaks with multiple means and multiple standard deviations.

The formula for Gaussian distribution using the mean and the standard deviation called the **Probability Density Function:**

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

where **μ= Mean**
σ=Standard Deviation

This is a function of a continuous random variable whose integral across an interval gives the probability that the value of the variable lies within the same interval.

Gaussian Mixture Model?

It will look like there are multiple peaks happening here and there. There are two peak points and the data seems to be going up and down twice or maybe three times or four times. But if there are Multiple Gaussian distributions that can represent this data, and can build called a Gaussian Mixture Model.

In other words we can say that, if we have three Gaussian Distribution as GD1, GD2, GD3 having mean as μ1, μ2,μ3 and variance 1,2,3 than for a given set of data points GMM will identify the probability of each data point belonging to each of these distributions.

It is a probability distribution that consists of multiple probability distributions and has Multiple Gaussians.

The probability distribution function of d-dimensions Gaussian Distribution is defined as:

$$N(\mu, \sum) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{|\sum|}} exp(-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu))$$

Where
- μ= Mean
- ∑= Covariance Matrix of the Gaussian
- d= The numbers of features in our dataset
- x=the number of datapoints

Why do we use the Variance-Covariance Matrix?

The Covariance is a measure of how changes in one variable are associated with changes in a second variable. The variance-covariance matrix is a measure of how these variables are related to each other, and in that way it's very similar to the standard deviation except when we have more dimension, the covariance matrix against the standard deviation gives us a better more accurate result.

$$V = \begin{bmatrix} \frac{\sum x_1^2}{N} & \frac{\sum x_1 x_2}{N} & \ldots\ldots\ldots\ldots & \frac{\sum x_1 x_c}{N} & \frac{\sum x_2 x_1}{N} & \frac{\sum x_2^2}{N} & \ldots\ldots\ldots & \frac{\sum x_2 x_c}{N} & \frac{\sum x_c x_1}{N} & \ldots & \frac{\sum x_c x_2}{N} & \ldots\ldots\ldots & \frac{\sum x_c^2}{N} \end{bmatrix}$$

Where, V= c x c variance-covariance matrix
N = the number of scores in each of the c datasets
x_i= is a deviation score from the i^{th} dataset
x_i^2/N= is the variance of element from the i^{th} dataset
$x_i x_j/N$= is the covariance for the elements from the i^{th} and j^{th} datasets
and the probability given in a mixture of K Gaussian where K is a number of distributions:

$$p(x) = \sum_{j=1}^{K} w_j . N(x|\mu_j, \sum_j)$$

Where w_j is the prior probability of the j^{th} Gaussian

$$\sum_{j=1}^{K} w_j = 1 \text{ and } 0 <= w_j <= 1$$

Once we multiply the probability distribution function of d-dimension by W, the prior probability of each of our gaussians, it will give us the probability value X for a given X data point. If we were to plot multiple Gaussian distributions, it would be multiple bell curves.

It is very similar to the k-means algorithm. It uses the same optimization strategy which is the expectation maximization algorithm.

K-Means VS Gaussian Mixture Model

K-means finds k to minimize $(x - \mu_k)^2$

The Gaussian Mixture Model is,

$$\frac{(x-\mu_k)^2}{\sigma^2}$$

The reason that standard deviation is added into this because in the denominator the 2 takes variation into consideration when it calculates its measurement but K means only calculates conventional Euclidean distance. i.e K-means calculates distance and GM calculates weights.

This means that the k-means algorithm gives you a hard assignment: it either says this is going to be this data point is a part of this class or it's a part of this class. Sometimes we want the maximum probability like: This is going to be 70% likely that it's a part of this class but we also want the probability that it's going to be a part of other classes. It is a list of probability values that it could be a part of multiple distributions, it could be in the middle, it could be 60% likely this class and 40% likely of this class. That's why we incorporate the standard deviation.

12.1 EXPECTATION MAXIMIZATION (EM) ALGORITHM

EM can be used for variables that are not directly observable and deduce from the value of other observed variables. It can be used with unlabeled data for its classification. It is one of the popular approaches to maximize the likelihood.

Basic Ideas of EM Algorithm: Given a set of incomplete data and set of starting parameters.

E-Step: Using the given data and the current value of parameters, estimate the value of hidden data.

M-Step: After the E-step, it is used to maximize the hidden variable and joint distribution of the data.

Usage of EM Algorithm

1. Used to fill missing data.
2. Find the values of latent variables.

Disadvantage

Has slow convergence and it converges up to local optima only.

Comparing to Gradient descent

Gradient descent compute the derivative which tells us the direction in which the data wants to move in or in what direction should we move the parameter's data of your model such that the function of our model is optimized to fit our data but what if we can't compute a gradient of a variable.

The Gaussian mixture model (GMM) has a random variable and is a stochastic model.

Applications

- Used in the field of signal processing.
- Provides good results in language Identification.
- Anomaly Detection.
- Track the object in a video frame.
- Classify songs based on genres.

12.2 CLASSIFIER USING MULTIPLE SAMPLES OF THE DATA SET

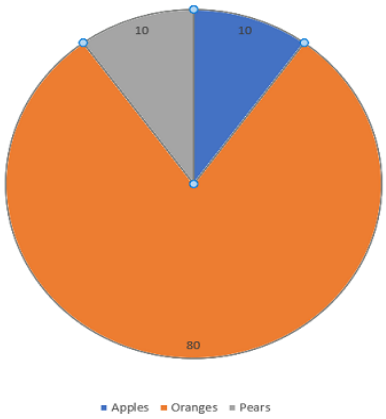
Classification problems having multiple classes with imbalanced dataset present a different challenge than a binary classification problem. The skewed distribution makes many conventional machine learning algorithms less effective, especially in predicting minority class examples.

Multiclass Classification

A classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multi-class classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.

Imbalanced Dataset

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. For example, you may have a 3-class classification problem of set of fruits to classify as oranges, apples or pears with total 100 instances. A total of 80 instances are labeled with Class-1 (Oranges), 10 instances with Class-2 (Apples) and the remaining 10 instances are labeled with Class-3 (Pears). This is an imbalanced dataset and the ratio of 8:1:1. Most classification data sets do not have exactly equal number of instances in each class, but a small difference often does not matter.



Dataset

The data set we will be using for this example is the famous “20 News groups” data set. The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

scikit-learn provides the tools to pre-process the dataset, refer here for more details. The number of articles for each news group given below is roughly uniform.

```
rec.sport.hockey            600
soc.religion.christian      599
rec.motorcycles             598
rec.sport.baseball          597
sci.crypt                   595
rec.autos                   594
sci.med                     594
sci.space                   593
comp.windows.x              593
comp.os.ms-windows.misc     591
sci.electronics             591
comp.sys.ibm.pc.hardware    590
misc.forsale                585
comp.graphics               584
comp.sys.mac.hardware       578
talk.politics.mideast       564
talk.politics.guns          546
alt.atheism                 480
talk.politics.misc          465
talk.religion.misc          377
```

Removing some news articles from some groups to make the overall dataset imbalanced like below.

```
rec.sport.hockey            600
rec.motorcycles             598
rec.sport.baseball          597
rec.autos                   594
talk.politics.guns          546
talk.religion.misc          377
sci.med                     287
sci.electronics             285
sci.space                   197
sci.crypt                   183
misc.forsale                171
comp.os.ms-windows.misc     151
comp.graphics               146
comp.sys.ibm.pc.hardware    137
comp.windows.x              136
comp.sys.mac.hardware       131
soc.religion.christian       86
talk.politics.mideast        67
alt.atheism                  63
talk.politics.misc           55
```

Now our imbalanced dataset with 20 classes is ready for further analysis.

```
In [178]: data_imb.head()
```

Out[178]:

	filename	category	news
0	20news-bydate/20news-bydate-train\rec.sport.ba...	rec.sport.baseball	From: cubbie@garnet.berkeley.edu (...
1	20news-bydate/20news-bydate-train\comp.sys.mac...	comp.sys.mac.hardware	From: gnelson@pion.rutgers.edu (Gregory Nelson...
2	20news-bydate/20news-bydate-train\sci.crypt\15246	sci.crypt	From: crypt-comments@math.ncsu.edu\nSubject: C...
3	20news-bydate/20news-bydate-train\comp.sys.mac...	comp.sys.mac.hardware	From: ()\nSubject: Re: Quadra SCSI Problems??...
4	20news-bydate/20news-bydate-train\alt.atheism\...	alt.atheism	From: keith@cco.caltech.edu (Keith Allan Schne...

Build Model

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=32,  dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(num_labels, activation='softmax'))

# try using different optimizers and different optimizer configs
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())
```

The last layer in the model is Dense(num_labels, activation ='softmax'),with num_labels=20 classes, 'softmax' is used instead of 'sigmoid' . The other change in the model is about changing the loss function to loss = 'categorical_crossentropy', which is suited for multi-class problems.

Train Model

```
num_epochs =10
batch_size = 128
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    validation_split=0.2)
```

Training the model with 20% validation set validation_split=20 and usingverbose=2, we see validation accuracy after each epoch. Just after 10 epochs we reach validation accuracy of 90%.

```
Train on 3357 samples, validate on 840 samples
Epoch 1/10
 - 6s - loss: 2.1579 - acc: 0.4111 - val_loss: 1.1341 - val_acc: 0.7214
Epoch 2/10
 - 2s - loss: 0.6236 - acc: 0.8591 - val_loss: 0.5506 - val_acc: 0.8488
Epoch 3/10
 - 2s - loss: 0.1342 - acc: 0.9827 - val_loss: 0.4166 - val_acc: 0.8750
Epoch 4/10
 - 2s - loss: 0.0324 - acc: 0.9970 - val_loss: 0.3812 - val_acc: 0.8869
Epoch 5/10
 - 2s - loss: 0.0135 - acc: 0.9994 - val_loss: 0.3602 - val_acc: 0.8964
Epoch 6/10
 - 2s - loss: 0.0057 - acc: 1.0000 - val_loss: 0.3707 - val_acc: 0.8845
Epoch 7/10
 - 2s - loss: 0.0046 - acc: 1.0000 - val_loss: 0.3735 - val_acc: 0.8869
Epoch 8/10
 - 2s - loss: 0.0025 - acc: 1.0000 - val_loss: 0.3613 - val_acc: 0.8929
Epoch 9/10
 - 2s - loss: 0.0019 - acc: 1.0000 - val_loss: 0.3616 - val_acc: 0.8952
Epoch 10/10
 - 2s - loss: 0.0016 - acc: 1.0000 - val_loss: 0.3596 - val_acc: 0.9012
```

Evaluate Model

```
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size, verbose=2)

print('Test accuracy:', acc)
```

```
Test accuracy: 0.8780952383223034
```

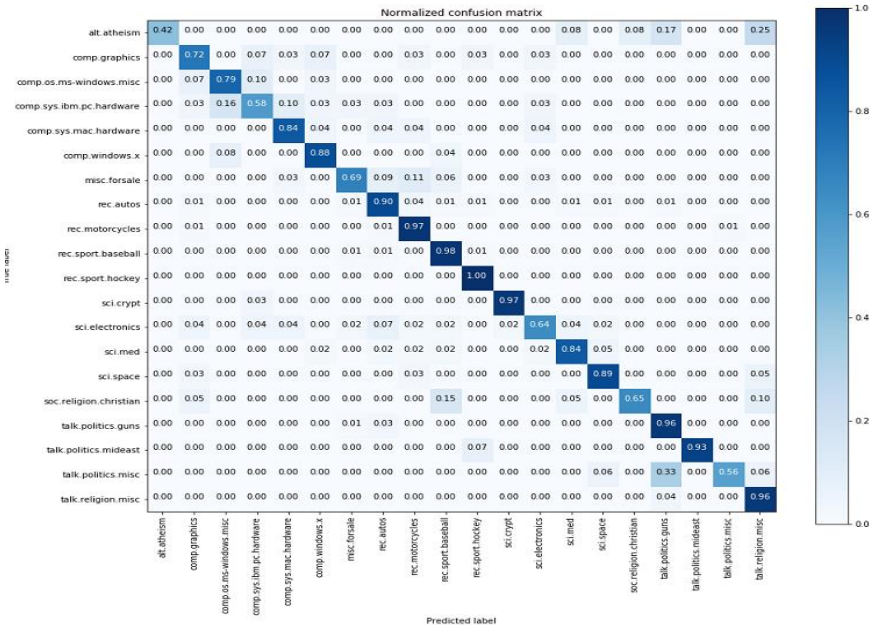
Looks like a very good accuracy ?

Consider that we train our model on imbalanced data of earlier example of fruits and since data is heavily biased towards Class-1 (Oranges), the model over-fits on the Class-1 label and predicts it in most of the cases and we achieve an accuracy of 80% which seems very good at first but looking closely, it may never be able to classify apples or pears correctly.

Confusion-Matrix

It's easy to get a high accuracy without making useful predictions. In case of imbalanced classes confusion-matrix is good technique to summarizing the performance of a classification algorithm.

Confusion Matrix is a performance measurement for a classification algorithm where output can be two or more classes.



x-axis=Predicted label, y-axis, True label

When we closely look at the confusion matrix, we see that the classes [alt.athiesm, talk.politics.misc, soc.religion.christian] which have very less samples [65,53, 86] respectively are indeed having very less scores [0.42, 0.56, 0.65] as compared to the classes with higher number of samples like [rec.sport.hockey, rec.motorcycles]. Looking at the confusion matrix one can clearly see how the model is performing on classifying various classes.

Improve the performance?

There are various techniques involved in improving the performance of imbalanced datasets.

Re-sampling Dataset

To make our dataset balanced there are two ways to do so:

Under-sampling: Remove samples from over-represented classes ; use this if you have huge dataset

Over-sampling: Add more samples from under-represented classes; use this if you have small dataset

Synthetic Minority Over-sampling Technique (SMOTE) is an over-sampling method and creates synthetic samples of the minority class.

```
x_train.shape, y_train.shape
```

```
((4197, 15000), (4197, 20))
```

```
from imblearn.over_sampling import SMOTE
#Over-sampling: SMOTE
#SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class,
#based on those that already exist. It works randomly picking a point from the minority class and computing
#the k-nearest neighbors for this point.The synthetic points are added between the chosen point and its neighbors.
#We'll use ratio='minority' to resample the minority class.
smote = SMOTE('minority')

X_sm, y_sm = smote.fit_sample(x_train, y_train)
print(X_sm.shape, y_sm.shape)
```

```
(4646, 15000) (4646, 20)
```

We have 4197 samples before and 4646 samples after applying SMOTE will check the performance of the model with the new dataset.

```
history = model.fit(X_sm, y_sm,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    class_weight=class_weight,
                    validation_split=0.2)
```

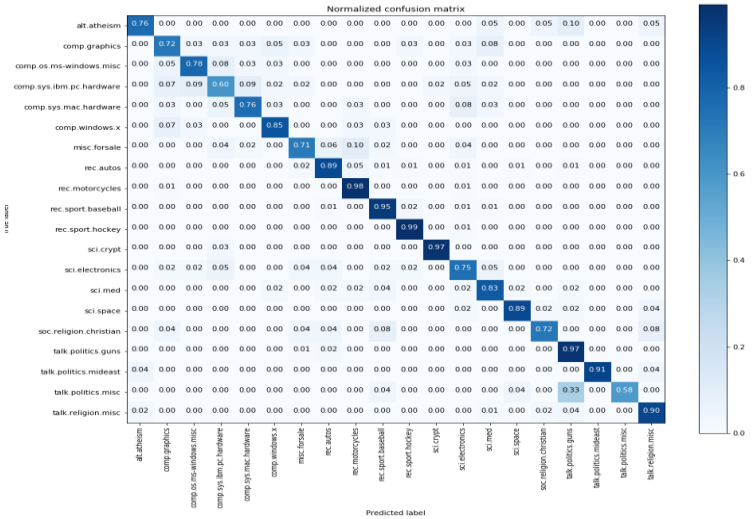
```
Train on 3716 samples, validate on 930 samples
Epoch 1/10
 - 10s - loss: 0.0593 - acc: 0.9839 - val_loss: 0.2841 - val_acc: 0.9075
Epoch 2/10
 - 2s - loss: 0.0138 - acc: 0.9995 - val_loss: 0.1916 - val_acc: 0.9441
Epoch 3/10
 - 3s - loss: 0.0068 - acc: 0.9997 - val_loss: 0.1903 - val_acc: 0.9387
Epoch 4/10
 - 3s - loss: 0.0057 - acc: 0.9997 - val_loss: 0.1924 - val_acc: 0.9376
Epoch 5/10
 - 2s - loss: 0.0054 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9452
Epoch 6/10
 - 2s - loss: 0.0051 - acc: 0.9997 - val_loss: 0.1899 - val_acc: 0.9430
Epoch 7/10
 - 3s - loss: 0.0050 - acc: 0.9997 - val_loss: 0.1897 - val_acc: 0.9419
Epoch 8/10
 - 2s - loss: 0.0048 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9409
Epoch 9/10
 - 2s - loss: 0.0047 - acc: 0.9997 - val_loss: 0.1900 - val_acc: 0.9398
Epoch 10/10
 - 2s - loss: 0.0047 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9409
```

Improved validation accuracy from 90 to 94%. Let us test the model:

```
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size, verbose=2)

print('Test accuracy:', acc)
```

```
Test accuracy: 0.8885714287984939
```

Little improvement in test accuracy than before (from 87 to 88%). Let us have a look at the confusion matrix now.



We see that the classes [alt.athiesm, talk.politics.misc, sci.electronics, soc.religion.christian] having improved scores [0.76, 0.58, 0.75, 0.72] than before. Thus the model is performing better than before while classifying the classes even though accuracy is similar.

Alternate trick

We can estimate class weights in scikit_learn by using compute_class_weight and use the parameter 'class_weight', while training the model and help to provide some bias towards the minority classes while training the model and thus help in improving performance of the model while classifying various classes.

```
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced' ,np.unique(y_train_labels) ,y_train_labels)
num_epochs =10
batch_size = 128
history = model.fit(X_sm, y_sm,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    class_weight=class_weight,
                    validation_split=0.2)
```

Precision-Recall Curves

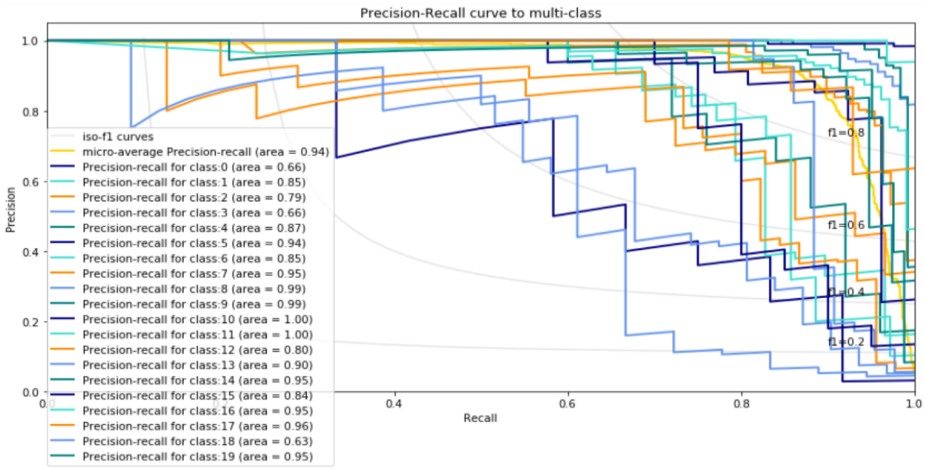
Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. Precision is a measure of the ability of a classification model to identify only the relevant data points, while recall

is a measure of the ability of a model to find all the relevant cases within a dataset.

The precision-recall curve shows the trade-off between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.

High scores for both precision and recall show that the classifier is returning accurate results (precision), as well as returning a majority of all positive results (recall).

Below is a precision-recall plot for 20 News groups dataset using scikit-learn.



We would like to have the area of P-R curve for each class to be close to 1. Except classes 0 , 3 & 18 rest of the classes are having area above .75.

12.3 BAGGING

Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions to form a final prediction. Can be used to reduce the variance of a black-box estimator, by introducing randomization into its construction procedure and then making an ensemble out of it.

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples (or data) from the original training dataset.

N: Size of the original training set

Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

Bagging works on training dataset

Bagging resamples the original training dataset with replacement, some instance(or data) may be present multiple times while others are left out.

Original training dataset: *1, 2, 3, 4, 5, 6, 7, 8, 9, 10*

Resampled training set 1: *2, 3, 3, 5, 6, 1, 8, 10, 9, 1*

Resampled training set 2: *1, 1, 5, 6, 3, 8, 9, 10, 2, 7*

Resampled training set 3: *1, 5, 8, 9, 2, 10, 9, 7, 5, 4*

Algorithm for the Bagging classifier:

Classifier generation:

Let N be the size of the training set.

for each of t iterations:

sample N instances with replacement from the original training set.

apply the learning algorithm to the sample.

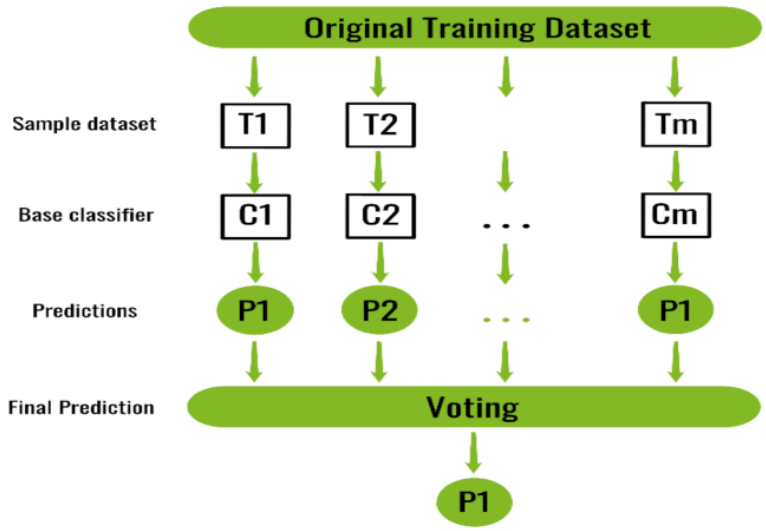
store the resulting classifier.

Classification:

for each of the t classifiers:

predict class of instance using classifier.

return class that was predicted most often.



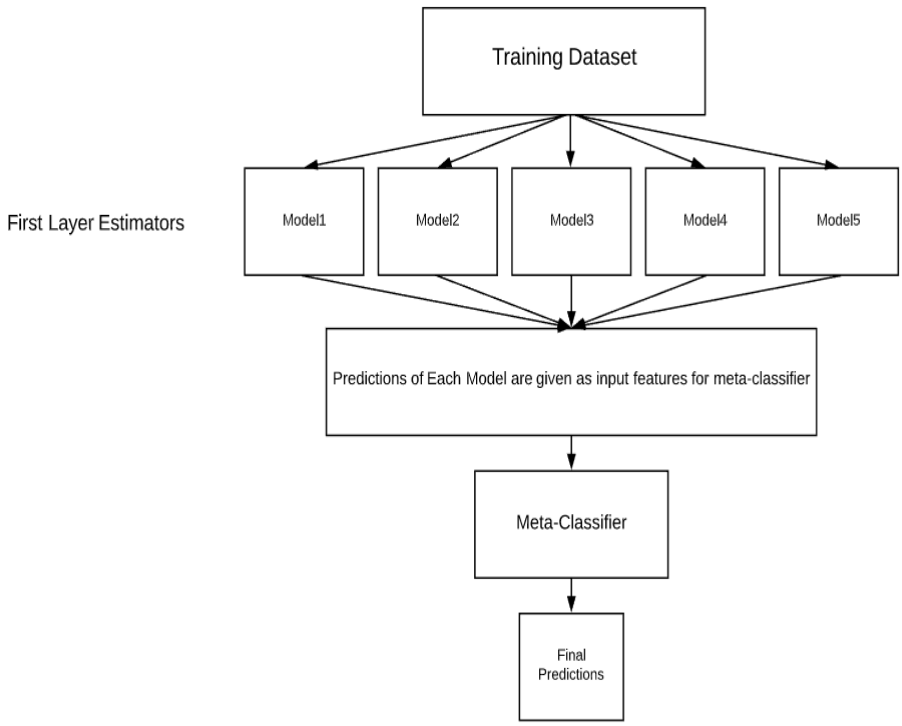
Implementation of the above Algorithm

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

# load the data
url = "/home/debomit/Downloads/wine_data.xlsx"
dataframe = pd.read_excel(url)
arr = dataframe.values
X = arr[:, 1:14]
Y = arr[:, 0]

seed = 8
kfold = model_selection.KFold(n_splits = 3,
                              random_state = seed)

# initialize the base classifier
base_cls = DecisionTreeClassifier()

# no. of base classifier
num_trees = 500

# bagging classifier
model = BaggingClassifier(base_estimator = base_cls,
                          n_estimators = num_trees,
                          random_state = seed)

results = model_selection.cross_val_score(model, X, Y, cv = kfold)
print("accuracy :")
print(results.mean())
```

12.4 STACKING

Stacking is a way of ensembling classification or regression models it consists of two-layer estimators. The first layer consists of all the baseline models that are used to predict the outputs on the test datasets. The second layer consists of Meta-Classifier or Regressor which takes all the predictions of baseline models as an input and generate new predictions.

Architecture



mlxtend:

Mlxtend (machine learning extensions) is a Python library of useful tools for day-to-day data science tasks. It consists of lots of tools that are useful for data science and machine learning tasks for example:

1. Feature Selection
2. Feature Extraction
3. Visualization
4. Ensembling and many more.

Why Stacking?

Most of the Machine-Learning and Data science competitions are won by using stacked models. They can improve the existing accuracy that is shown by individual models. We can get most of the Stacked models by choosing diverse algorithms in the first layer of architecture as different algorithms capture different trends in training data by combining both of the models can give better and accurate results.

Installation of libraries on the system:

```
pip install mlxtend
pip install pandas
pip install -U scikit-learn
```

Code: Import Required Libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_confusion_matrix
from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

Code: Loading the dataset

```
df = pd.read_csv('heart.csv') # loading the dataset
df.head()                     # viewing top 5 rows of dataset
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Code:

```
# Creating X and y for training
X = df.drop('target', axis = 1)
y = df['target']
```

Code: Splitting Data into Train and Test

```
# 20 % training dataset is considered for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

Code: Standardizing Data

```
# initializing sc object
sc = StandardScaler()
# variables that needed to be transformed
var_transform = ['thalach', 'age', 'trestbps', 'oldpeak', 'chol']
X_train[var_transform] = sc.fit_transform(X_train[var_transform]) # standardizing training
data
X_test[var_transform] = sc.transform(X_test[var_transform])          #    standardizing
test data
print(X_train.head())
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
835	-0.585840	1	2	-0.779454	-1.935031	0	0	-1.019094	0	-0.210661	2	3	2
137	1.051477	0	0	2.741732	1.610634	0	1	0.202882	1	-0.912152	2	0	2
534	-0.040068	0	2	-1.347387	0.442176	0	0	0.770228	0	-0.912152	2	0	2
495	0.505705	1	0	0.186033	-0.222636	0	1	0.508376	0	-0.473720	1	0	3
244	-0.367531	1	2	-0.381900	-0.001032	1	0	0.726586	0	1.192321	1	0	2

Code: Building First Layer Estimators

```
KNC = KNeighborsClassifier() # initialising KNeighbors Classifier
NB = GaussianNB()                    # initialising Naive Bayes
```

Let's Train and evaluate with our first layer estimators to observe the difference in the performance of the stacked model and general model

Code: Training KNeighborsClassifier

```
model_kNeighborsClassifier = KNC.fit(X_train, y_train) # fitting Training Set
pred_knc = model_kNeighborsClassifier.predict(X_test) # Predicting on test dataset
```

Code: Evaluation of KNeighborsClassifier

```
acc_knc = accuracy_score(y_test, pred_knc) # evaluating accuracy score
print('accuracy score of KNeighbors Classifier is:', acc_knc * 100)
```

Output

```
accuracy score of KNeighbors Classifier is: 80.48780487804879
```

Code: Training Naive Bayes Classifier

```
model_NaiveBayes = NB.fit(X_train, y_train)
pred_nb = model_NaiveBayes.predict(X_test)
```

Code: Evaluation of Naive Bayes Classifier

```
acc_nb = accuracy_score(y_test, pred_nb)
print('Accuracy of Naive Bayes Classifier:', acc_nb * 100)
```

Output:

```
Accuracy of Naive Bayes Classifier: 80.0
```

Code: Implementing Stacking Classifier

lr = LogisticRegression() # defining meta-classifier

clf_stack = StackingClassifier(classifiers =[KNC, NB], meta_classifier = lr, use_probas = True, use_features_in_secondary = True)

- use_probas=True indicates the Stacking Classifier uses the prediction probabilities as an input instead of using predictions classes.
- use_features_in_secondary=True indicates Stacking Classifier not only take predictions as an input but also uses features in the dataset to predict on new data.

Code: Training Stacking Classifier

```
model_stack = clf_stack.fit(X_train, y_train) # training of stacked model
pred_stack = model_stack.predict(X_test)    # predictions on test data using stacked model
```

Code: Evaluating Stacking Classifier

```
acc_stack = accuracy_score(y_test, pred_stack) # evaluating accuracy
print('accuracy score of Stacked model:', acc_stack * 100)
```

Output:

```
accuray score of Stacked model: 83.90243902439025
```

Our both individual models scores an accuracy of nearly 80% and our Stacked model got an accuracy of nearly 84%.By Combining two individual models we got a significant performance improvement.

Code

```
model_stack = clf_stack.fit(X_train, y_train) # training of stacked model
pred_stack = model_stack.predict(X_test)    # predictions on test data using stacked model
```

Code: Evaluating Stacking Classifier

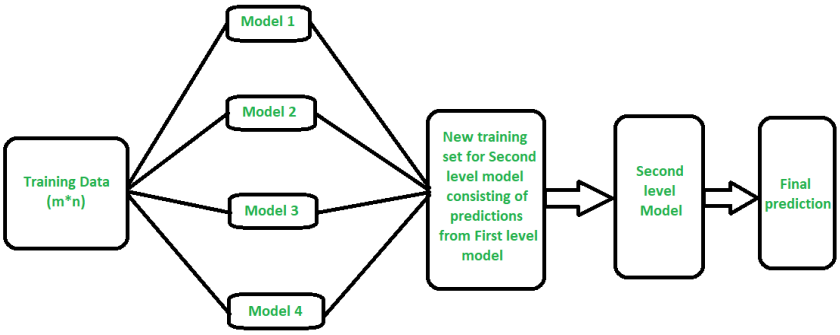
```
acc_stack = accuracy_score(y_test, pred_stack) # evaluating accuracy
print('accuracy score of Stacked model:', acc_stack * 100)
```

```
accuray score of Stacked model: 83.90243902439025
```

Our both individual models scores an accuracy of nearly 80% and our Stacked model got an accuracy of nearly 84%. By Combining two individual models we got a significant performance improvement.

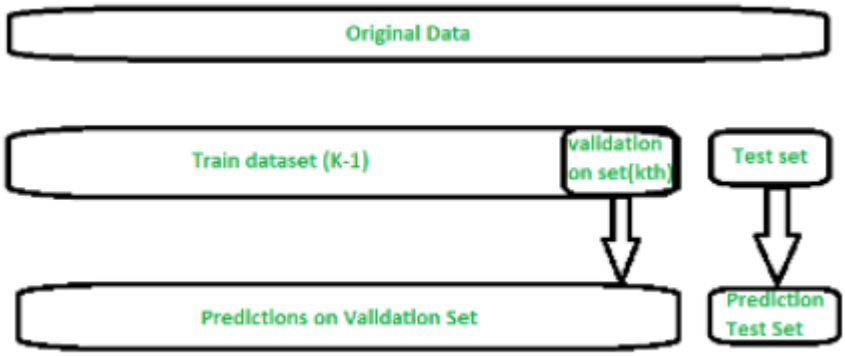
Stacking is a way to ensemble multiple classifications or regression model. There are many ways to ensemble models, the widely known models are ***Bagging*** or ***Boosting***. Bagging allows multiple similar models with high variance are averaged to decrease variance.

The point of stacking is to explore a space of different models for the same problem. The idea is that you can attack a learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem.
This final model is said to be stacked on the top of the others, hence the name.



How stacking works?

1. Splits the training data into K-folds just like K-fold cross-validation.
2. A base model is fitted on the K-1 parts and predictions are made for K^{th} part.
3. We do for each part of the training data.
4. The base model is then fitted on the whole train data set to calculate its performance on the test set.
5. We repeat the 2-4 steps for other base models.
6. Predictions from the train set are used as features for the second level model.
7. Second level model is used to make a prediction on the test set.



Blending

- Blending is a similar approach to stacking.
- The train set is split into training and validation sets.
- We train the base models on the training set.
- We make predictions only on the validation set and the test set.
- The validation predictions are used as features to build a new model.
- This model is used to make final predictions on the test set using the prediction values as features.

❄❄❄❄❄❄❄

13

BOOSTING

Unit Structure

13.0 IMPROVING CLASSIFICATION WITH THE ADABOOST META-ALGORITHM

If you were going to make an important decision, you'd probably get the advice of multiple experts instead of trusting one person. Why should the problems you solve with machine learning be any different? This is the idea behind a meta-algorithm. Meta-algorithms are a way of combining other algorithms. We'll focus on one of the most popular meta-algorithms called AdaBoost. This is a powerful tool to have in your toolbox because AdaBoost is considered by some to be the best-supervised learning algorithm.

In this chapter we're first going to discuss different ensemble methods of classification. We'll next focus on boosting and AdaBoost, an algorithm for boosting. We'll then build a decision stump classifier, which is a single-node decision tree. The AdaBoost algorithm will be applied to our decision stump classifier.

Detecting fraudulent credit card use is a good example of this: we may have 1,000 negative examples for every positive example. How do classifiers work in such a situation?

Classifiers using multiple samples of the dataset

ADABOOST

Pros: Low generalization error, easy to code, works with most classifiers, no parameters to adjust

Cons: Sensitive to outliers

Works with: Numeric values, nominal values

Algorithms have individual strengths and weaknesses. One idea that naturally arises is combining multiple classifiers. Methods that do this are known as ensemble methods or meta-algorithms.

13.1 BUILDING CLASSIFIERS FROM RANDOMLY RESAMPLED DATA: BAGGING

Bootstrap aggregating, which is known as bagging, is a technique where the data is taken from the original dataset S times to make S new datasets. The datasets are the same size as the original. This property allows you to have values in the new dataset that are repeated, and some values from the original won't be present in the new set. After the S datasets are built, a learning algorithm is applied to each one individually. When you'd like to classify a new piece of data, you'd apply our S classifiers to the new piece of data and take a majority vote.

13.2 BOOSTING

Boosting is a technique similar to bagging. In boosting and bagging, you always use the same type of classifier. But in boosting, the different classifiers are trained sequentially. Each new classifier is trained based on the performance of those already trained. Boosting makes new classifiers focus on data that was previously misclassified by previous classifiers. Boosting is different from bagging because the output is calculated from a weighted sum of all classifiers. The weights aren't equal as in bagging but are based on how successful the classifier was in the previous iteration.

There are many versions of boosting, but we will focus on the most popular version, AdaBoost.

GENERAL APPROACH TO ADABOOST

1. Collect: Any method.
2. Prepare: It depends on which type of weak learner you're going to use.
3. Analyze: Any method.
4. Train: The classifier will train the weak learner multiple times over the same dataset.
5. Test: Calculate the error rate.
6. Use: AdaBoost predicts one of two classes. If you want to use it for classification involving more than two classes, then you'll need to apply some of the same methods as for support vector machines

Train: improving the classifier by focusing on errors

An interesting theoretical question is can we take a weak classifier and use multiple instances of it to create a strong classifier? That is to say, its error rate is greater than 50% in the two-class case. The "strong" classifier will have a much lower error rate.

AdaBoost is short for adaptive boosting. AdaBoost works this way: A weight is applied to every example in the training data. We'll call the weight vector D. Initially, these weights are all equal. A weak classifier is first trained on the training data. The errors from the weak classifier are calculated, and the weak classifier is trained a second time with the same dataset. This second time the weak classifier is trained, the weights of the training set are adjusted so the examples properly classified the first time are weighted less and the examples incorrectly classified in the first iteration are weighted more. To get one answer from all of these weak classifiers, AdaBoost assigns α values to each of the classifiers. The α values are based on the error of each weak classifier. The error ε is given by

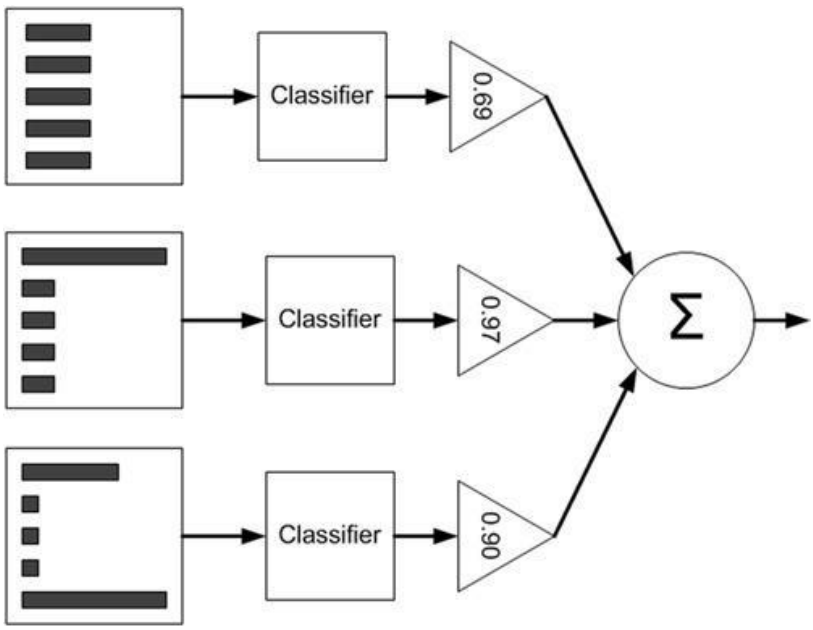
$$\varepsilon = \frac{number\ of\ incorrectly\ classified\ examples}{total\ number\ of\ examples}$$

and α is given by

$$\alpha = \frac{1}{2} ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

The AdaBoost algorithm can be seen schematically in the following figure.

AdaBoost with the dataset on the left side, the different widths of the bars represent weights applied to each instance. The weighted predictions pass through a classifier, which is then weighted by the triangles (α values). The weighted output of each triangle is summed up in the circle, which produces the final output.



After you calculate α, you can update the weight vector D so that the examples that are correctly classified will decrease in weight and the misclassified examples will increase in weight. D is given by

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha}}{Sum(D)}$$

if correctly predicted and

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{\alpha}}{Sum(D)}$$

if incorrectly predicted.

After D is calculated, AdaBoost starts on the next iteration. The AdaBoost algorithm repeats the training and weight-adjusting iterations until the training error is 0 or until the number of weak classifiers reaches a user-defined value.

Creating a weak learner with a decision stump

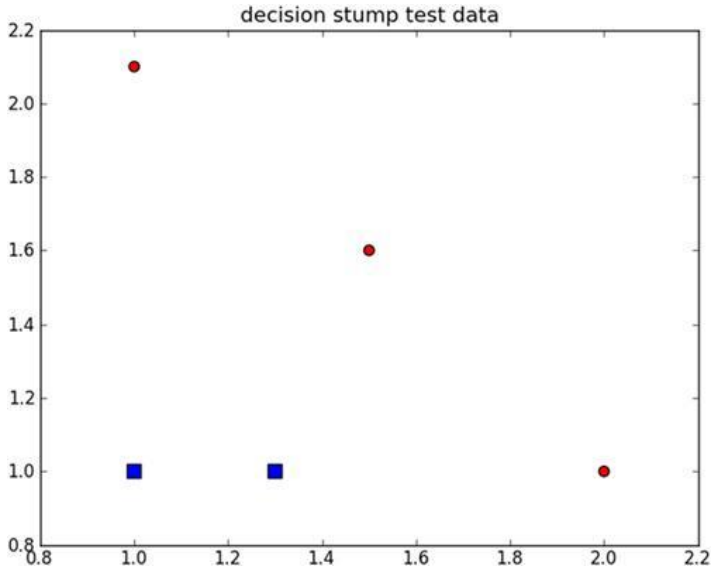
A decision stump is a simple decision tree that makes a decision on one feature only. It's a tree with only one split, so it's a stump.

We create a new file called adaboost.py and add the following code:

```
1
2
3
4
5
6
7
8
def loadSimpData():
    datMat = matrix([[ 1. ,  2.1],
        [ 2. ,  1.1],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
  return datMat,classLabels
```

By using multiple decision stumps, we'll be able to build a classifier to completely classify the data.

It’s not possible to choose one threshold on one axis that separates the squares from the circles. AdaBoost will need to combine multiple decision stumps to classify this set without error.



1
2

```
>>> import adaboost
```

```
>>> datMat,classLabels=adaboost.loadSimpData()
```

As we have the dataset loaded, we can create a few functions to build our decision stump.

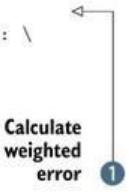
The first one will be used to test if any of values are less than or greater than the threshold value we’re testing. The second, more involved function will loop over a weighted version of the dataset and find the stump that yields the lowest error.

The pseudo-code will look like this:
Set the minError to + ∞
For every feature in the dataset:
For every step:
For each inequality:
Build a decision stump and test it with the weighted dataset
If the error is less than minError: set this stump as the best stump Return the best stump

Enter the code from the following listing into adaboost.py and save the file.

Decision stump–generating functions

```
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = ones((shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray

def buildStump(dataArr,classLabels,D):
    dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
    m,n = shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))
    minError = inf
    for i in range(n):
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps

        for j in range(-1,int(numSteps)+1):
            for inequal in ['lt', 'gt']:
                threshVal = (rangeMin + float(j) * stepSize)
                predictedVals = \
                        stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr = mat(ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                weightedError = D.T*errArr
                #print "split: dim %d, thresh %.2f, thresh ineqal: \
                      %s, the weighted error is %.3f" %\
                      (i, threshVal, inequal, weightedError)
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst
```



The above code contains two functions. The first function, stumpClassify(), performs a threshold comparison to classify data. Everything on one side of the threshold is thrown into class -1, and everything on the other side is thrown into class +1.. You can make this comparison on any feature in the dataset, and you can also switch the inequality from greater than to less than.

The next function, buildStump(), will iterate over all of the possible inputs to stumpClassify() and find the best decision stump for our dataset. Best here will be with respect to the data weight vector D. You'll see how this is done in a bit. The function starts out by making sure the input data is in the proper format for matrix math. Then, it creates an empty dictionary called bestStump, which you'll use to store the classifier information corresponding to the best choice of a decision stump given this weight vector D. The variable numSteps will be used to iterate over the possible values of the features. You also initialize the variable minError to positive infinity; this variable is used in finding the minimum possible error later.

Inside the nested three for loops, you call stumpClassify() with the dataset and your three loop variables. stumpClassify() returns its class prediction based on these loop variables. You next create the column vector errArr, which contains a 1 for any value in predictedVals that isn't equal to the actual class in labelMat.

You multiply these errors by the weights in D and sum the results to give you a single number: weightedError. This is the line where AdaBoost interacts with the classifier. You're evaluating your classifier based on the weights D, not on another error measure. If you want to use another classifier, you'd need to include this calculation to define the best classifier for D.

You next print out all the values. This line can be commented out later, but it's helpful in understanding how this function works. Last, you compare the error to your known minimum error, and if it's below it, you save this decision stump in your dictionary bestStump. The dictionary, the error, and the class estimates are all returned to the AdaBoost algorithm.

To see this in action, enter the following in the Python shell:

```
>>> D = mat(ones((5,1))/5)
>>> adaboost.buildStump(datMat,classLabels,D)
split: dim 0, thresh 0.90, thresh ineqal: lt, the weighted error is 0.400
split: dim 0, thresh 0.90, thresh ineqal: gt, the weighted error is 0.600
split: dim 0, thresh 1.00, thresh ineqal: lt, the weighted error is 0.400
split: dim 0, thresh 1.00, thresh ineqal: gt, the weighted error is 0.600
                                .
                                .
split: dim 1, thresh 2.10, thresh ineqal: lt, the weighted error is 0.600
split: dim 1, thresh 2.10, thresh ineqal: gt, the weighted error is 0.400
({'dim': 0, 'ineq': 'lt', 'thresh': 1.3}, matrix([[ 0.2]]), array([[-1.],
       [ 1.],
       [-1.],
       [-1.],
       [ 1.]]))
```

As buildStump iterates over all of the possible values, you can see the output, and finally you can see the dictionary returned..

13.3 Implementing the full AdaBoost algorithm

In the last section, we built a classifier that could make decisions based on weighted input values. We now have all we need to implement the full AdaBoost algorithm. We'll implement the algorithm outlined with the decision stump.

Pseudo-code for this will look like this:

For each iteration:

Find the best stump using buildStump()

Add the best stump to the stump array

Calculate alpha

Calculate the new weight vector – D

Update the aggregate class estimate

If the error rate ==0.0: break out of the for loop

To put this function into Python, open adaboost.py and add the code from the following listing.

AdaBoost training with decision stumps

```
def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    weakClassArr = []
    m = shape(dataArr)[0]
    D = mat(ones((m,1))/m)
    aggClassEst = mat(zeros((m,1)))
    for i in range(numIt):
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)
        print "D:",D.T
        alpha = float(0.5*log((1.0-error)/max(error,1e-16)))
        bestStump['alpha'] = alpha
        weakClassArr.append(bestStump)
        print "classEst: ",classEst.T
        expon = multiply(-1*alpha*mat(classLabels).T,classEst)
        D = multiply(D,exp(expon))
        D = D/D.sum()
        aggClassEst += alpha*classEst
        print "aggClassEst: ",aggClassEst.T
        aggErrors = multiply(sign(aggClassEst) !=
                    mat(classLabels).T,ones((m,1)))

        errorRate = aggErrors.sum()/m
        print "total error: ",errorRate,"\n"
        if errorRate == 0.0: break
    return weakClassArr
```

1 Calculate D for next iteration

2 Aggregate error calculation

```
>>> classifierArray = adaboost.adaBoostTrainDS(datMat,classLabels,9)
D: [[ 0.2  0.2  0.2  0.2  0.2]]
classEst:  [[-1.  1. -1. -1.  1.]]
aggClassEst:  [[-0.69314718  0.69314718 -0.69314718 -0.69314718
  0.69314718]]
total error:  0.2

D: [[ 0.5    0.125  0.125  0.125  0.125]]
classEst:  [[ 1.  1. -1. -1. -1.]]
aggClassEst:  [[ 0.27980789  1.66610226 -1.66610226 -1.66610226
  -0.27980789]]
total error:  0.2
D: [[ 0.28571429  0.07142857  0.07142857  0.07142857  0.5       ]]
classEst:  [[ 1.  1.  1.  1.  1.]]
aggClassEst:  [[ 1.17568763  2.56198199 -0.77022252 -0.77022252
  0.61607184]]
total error:  0.0
```

The AdaBoost algorithm takes the input dataset, the class labels, and one parameter, numIt, which is the number of iterations. This is the only parameter you specify for the whole AdaBoost algorithm.

You set the number of iterations to 9. But the algorithm reached a total error of 0 after the third iteration and quit, so you didn't get to see all nine iterations. Intermediate output from each of the iterations comes from the print statements. You'll comment these out later, but for now let's look at the output to see what's going on under the hood of the AdaBoost algorithm.

The DS at the end of the function names stands for decision stump. Decision stumps are the most popular weak learner in AdaBoost.

```
1
2
3
4
5
>>> classifierArray
[{'dim': 0, 'ineq': 'lt', 'thresh': 1.3, 'alpha': 0.69314718055994529},
   {'dim': 1, 'ineq': 'lt', 'thresh': 1.0, 'alpha': 0.9729550745276565},
   {'dim': 0,'ineq': 'lt', 'thresh': 0.90000000000000002, 'alpha':
    0.895
```

13.4 TEST: CLASSIFYING WITH ADABOOST

We need to do is take the train of weak classifiers from training function and apply these to an instance. The result of each weak classifier is weighted by its alpha. The weighted results from all of these weak classifiers are added together, and you take the sign of the final weighted sum to get final answer.

AdaBoost classification function

```
1
2
3
4
5
6
7
8
9
10
11
def adaClassify(datToClass,classifierArr):
   dataMatrix = mat(datToClass)
   m = shape(dataMatrix)[0]
```

```
    aggClassEst = mat(zeros((m,1)))
    for i in range(len(classifierArr)):
        classEst = stumpClassify(dataMatrix,classifierArr[i]['dim'],\
                                 classifierArr[i]['thresh'],\
                                 classifierArr[i]['ineq'])
        aggClassEst += classifierArr[i]['alpha']*classEst
        print aggClassEst
    return sign(aggClassEst)
```

EXAMPLE: USING ADABOOST ON A DIFFICULT DATASET

1. Collect: Text file provided.
2. Prepare: We need to make sure the class labels are +1 and -1, not 1 and 0.
3. Analyze: Manually inspect the data.
4. Train: We'll train a series of classifiers on the data using the adaBoostTrainDS() function.
5. Test: We have two datasets. With no randomization, we can have an apples-to-apples comparison of the AdaBoost results versus the logistic regression results.
6. Use: We'll look at the error rates in this example. But you could create a website that asks a trainer for the horse's symptoms and then predicts whether the horse will live or die.

Before you use the functions from the previous code listings in this chapter, you need to have a way to load data from a file. The familiar loadDataSet() is given in the following listing.

Adaptive load data function

```
1
2
3
4
5
6
7
8
9
10
11
12
def loadDataSet(fileName):
    numFeat = len(open(fileName).readline().split('\t'))
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
```

```
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat-1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat
```

13.5 IMPLEMENTING THE ADABOOST ALGORITHM

Iris dataset is used as an example in building the algorithm and also considered only two classes (Versicolor and Virginica).

```
#considering only two classes
example = iris[(iris['Species'] == 'versicolor') | (iris['Species'] == 'virginica')]
```

Step 1: Assign Equal Weights to all the observations

Initially assign same weights to each record in the dataset.

Sample weight = 1/N

Where N = Number of records

```
#Initially assign same weights to each records in the dataset
example['probR1'] = 1/(example.shape[0])
```

Step 2: Classify random samples using stumps

Draw random samples with replacement from original data with the probabilities equal to the sample weights and fit the model. Decision trees are created with one depth which has one node and two leaves also referred to as stumps. Fit the model to the random samples and predict the classes for the original data.

```
#simple random sample with replacement
random.seed(10)
example1 = example.sample(len(example), replace = True, weights = example['probR1'])
```

```
#fitting the DT model with depth one
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)
```

SepalLength	SepalWidth	PetalLength	PetalWidth	Label	probR1	pred1
7	3.2	4.7	1.4	1	0.01	1
5.9	3.2	4.8	1.8	1	0.01	-1

'pred1' is the newly predicted class.

Step 3: Calculate Total Error

Total error is nothing but the sum of weights of misclassified record.

Total Error = **Weights of misclassified records**

Total error will be always between 0 and 1.

0 represents perfect stump (correct classification)

1 represents weak stump (misclassification)

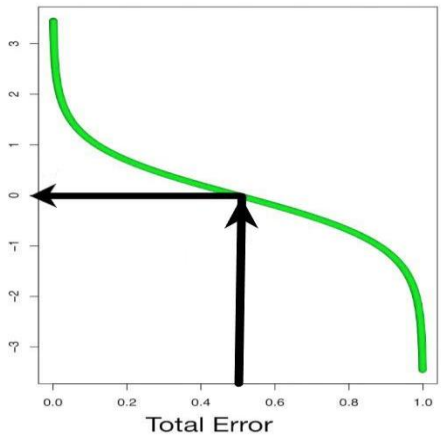
```
#error calculation
e1 = sum(example['misclassified'] * example['probR1'])
```

Step 4: Calculate Performance of the Stump

Using the Total Error, determine the performance of the base learner. The calculated performance of stump(α) value is used to update the weights in consecutive iteration and also used for final prediction calculation.

Performance of the stump(α) = ½ ln (1 – Total error/Total error)

```
#calculation of alpha (performance)
alpha1 = 0.5*log((1-e1)/e1)
```



Cases:

1. If the total error is 0.5, then the performance of the stump will be zero.
2. If the total error is 0 or 1, then the performance will become infinity or -infinity respectively.

When the total errors are equal to 1 or 0, the above equation will behave in a weird manner. So, in practice a small error term is added to prevent this from happening.

When the performance(α) is relatively large, the stump did a good job in classifying the records. When the performance(α) is relatively low, the stump did not do a good job in classifying the records. Using the performance parameter(α), we can increase the weights of the wrongly classified records and decrease the weights of the correctly classified records.

Step 5: Update Weights

Based on the performance of the stump(α) update the weights. We need the next stump to correctly classify the misclassified record by increasing the corresponding sample weight and decreasing the sample weights of the correctly classified records.

New weight = Weight * $e^{(performance)}$ → misclassified records

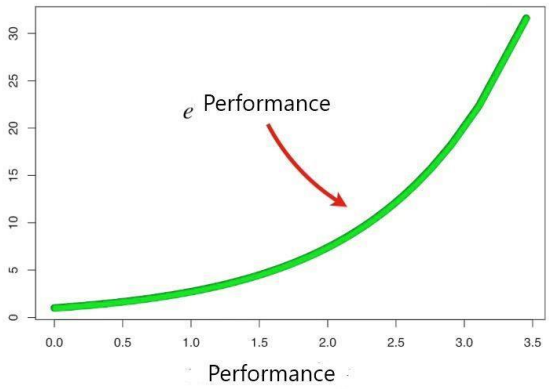
New weight = Weight * $e^{-(performance)}$ → correctly classified records

```
#update weight
new_weight = example['probR1']*np.exp(-1*alpha1*example['Label']*example['pred1'])
```

If the 'Label' and 'pred1' are same (i.e. 1 or -1) then substituting the values in the above equation will give the equation corresponding to correctly classified record.

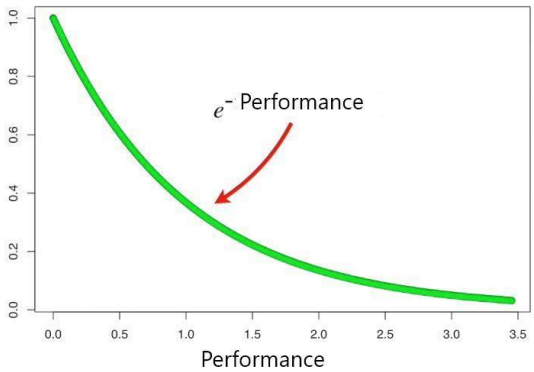
Short note on e^(performance) i.e. for misclassification

When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be much larger than the old one. When the performance is relatively low the last stump did not do a good job in classifying the records now the new sample weight will only be little larger than the old one.



Short note on e^-(performance) i.e. for no misclassification

When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be very small than the old one. When the performance is relatively small the last stump did not do a good job in classifying the records now the new sample weight will only be little smaller than the old one.



Here the sum of the updated weights is not equal to 1. whereas in case of initial sample weight the sum of total weights is equal to 1. So, to achieve this we will be dividing it by a number which is nothing but the sum of the updated weights (normalizing constant).

Normalizing constant = ∑ New weight

Normalized weight = New weight / Normalizing constant

Now the sum of normalized weight is equal to 1.

```
#normalized weight
z = sum(new_weight)
normalized_weight = new_weight/sum(new_weight)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Label	probR1	pred1	misclassified	prob2
7	3.2	4.7	1.4	1	0.01	1	0	0.0053
6.4	3.2	4.5	1.5	1	0.01	1	0	0.0053
5.9	3.2	4.8	1.8	1	0.01	-1	1	0.0833

'prob2' is the newly updated weights.

Step 6: Update weights in iteration

Use the normalized weight and make the second stump in the forest. Create a new dataset of same size of the original dataset with repetition based on the newly updated sample weight. So that the misclassified records get higher probability of getting selected. Repeat step 2 to 5 again by updating the weights for a particular number of iterations.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Label	probR1	pred1	misclassified	prob2	pred2	misclassified2	prob3	pred3	misclassified3	prob4
7	3.2	4.7	1.4	1	0.01	1	0	0.0053	1	0	0.003	-1	1	0.0055
6.4	3.2	4.5	1.5	1	0.01	1	0	0.0053	1	0	0.003	-1	1	0.0055
6.9	3.1	4.9	1.5	1	0.01	1	0	0.0053	-1	1	0.023	-1	1	0.042

'prob4' is the final weights of each observation.

Step 7: Final Predictions

Final prediction is done by obtaining the sign of the weighted sum of final predicted value.

Final prediction/sign (weighted sum) = $\sum$ (α_i* (predicted value at each iteration))

Calculation:

t = 1.0*0.2 + 1.0*0.5 - 1.0*0.8 + 1.0*0.2 - 1.0*0.9 = -0.8

Taking the sign alone into consideration, the final prediction will be -1.0 or the second class.

```
#final prediction
t = alpha1 * example['pred1'] + alpha2 * example['pred2'] + alpha3 * example['pred3'] + alpha4 * example['pred4']

#sign of the final prediction
np.sign(list(t))
```

Advantages of AdaBoost Algorithm:

- Fast, simple and easy to program.
- Robust to overfitting.
- Extended to learning problems beyond binary classification (i.e.) can be used with text or numeric data.

Drawbacks:

- Sensitive to noisy data and outliers.
- Weak classifiers can lead to overfitting.

CONCLUSION:

AdaBoost helps in choosing the training set for each new classifier that is trained based on the results of the previous classifier. Also, while combining the results; it determines how much weight should be given to each classifier's proposed answer. It combines the weak learners to create a strong one to correct classification errors which is also the first successful boosting algorithm for binary classification problems.

13.6 SMO ALGORITHM

Overview of SMO

Sequential Minimal Optimization (SMO) contains many optimizations designed to speed up the algorithm on large datasets and ensure that the algorithm converges even under degenerate conditions.

Support vector machine (SVM) computes a linear classifier of the form

$$f(x) = w^T x + b.$$

Since we want to apply this to a binary classification problem, we will ultimately predict y = 1 if f(x) ≥ 0 and y = −1 if f(x) < 0, but for now we simply consider the function f(x). By looking at the dual problem, we see that it can also be expressed using inner products as

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

where we can substitute a kernel

$$K(x^{(i)}, x)$$

in place of the inner product if we so desire. The SMO algorithm gives an efficient way of solving the dual problem of the (regularized) support vector machine optimization problem.

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

The KKT conditions can be used to check for convergence to the optimal point. For this problem the KKT conditions are

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \ge 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \le 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1. \end{aligned}$$

In other words, any α_i 's that satisfy these properties for all i will be an optimal solution to the optimization problem given above. The SMO algorithm iterates until all these conditions are satisfied ensuring convergence.

THE SIMPLIFIED SMO ALGORITHM

SMO algorithm selects two α parameters, α_i and α_j and optimizes the objective value jointly for both these α's and adjusts the b parameter based on the new α's. This process is repeated until the α's converge.

Selecting α Parameters

SMO algorithm is dedicated to heuristics for choosing which α_i and α_j to optimize so as to maximize the objective function as much as possible. For large data sets, this is critical for the speed of the algorithm, since there are m(m − 1) possible choices for α_i and α_j, and some will result in much less improvement than others. We simply iterate over all αi , i = 1, . . . m. If α_i does not fulfill the KKT conditions to within some numerical tolerance, we select α_j at random from the remaining m − 1 α's and attempt to jointly optimize α_i and α_j . If none of the α's are changed after a few iteration over all the α_i 's, then the algorithm terminates. It is important to realize that by employing this simplification, the algorithm is no longer guaranteed to converge to the global optimum.

Optimizing α_i and α_j

Having chosen the Lagrange multipliers α_i and α_j to optimize, we first compute constraints on the values of these parameters, then we solve the constrained maximization problem. First we want to find bounds L and H such that $L \leq \alpha_j \leq H$ must hold in order for α_j to satisfy the constraint that $0 \leq \alpha_j \leq C$. It can be shown that these are given by the following:

$$\begin{aligned} &\text{If } y^{(i)} \neq y^{(j)}, \quad L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i) \\ &\text{If } y^{(i)} = y^{(j)}, \quad L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j) \end{aligned}$$

Now we want to find α_j so as to maximize the objective function. If this value ends up lying outside the bounds L and H, we simply clip the value of α_j to lie within this range. It can be shown that the optimal α_j is given by:

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta}$$

$$\begin{aligned} E_k &= f(x^{(k)}) - y^{(k)} \\ \eta &= 2\langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(j)} \rangle. \end{aligned}$$

E_k the error between the SVM output on the k^{th} example and the true label $y^{(k)}$. When calculating the η parameter you can use a kernel function K in

place of the inner product if desired. Next we clip α_j to lie within the range [L, H]

$$\alpha_j := \begin{cases} H & \text{if } \alpha_j > H \\ \alpha_j & \text{if } L \leq \alpha_j \leq H \\ L & \text{if } \alpha_j < L. \end{cases}$$

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(\text{old})} - \alpha_j)$$

where $\alpha_j^{(\text{old})}$ is the value of α_j before optimization.

The full SMO algorithm handles the rare case that $\eta = 0$. For our purposes, if $\eta = 0$, you can treat this as a case where we cannot make progress on this pair of α's.

Pseudo-Code for Simplified SMO

Algorithm: Simplified SMO

Input:

C: regularization parameter
tol: numerical tolerance
max_passes: max # of times to iterate over α's without changing
$(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$: training data

Output:

$\alpha \in \mathbb{R}^m$: Lagrange multipliers for solution
$b \in \mathbb{R}$: threshold for solution

- Initialize $\alpha_i = 0, \forall i, \quad b = 0$.
- Initialize $passes = 0$.
- **while** $(passes < max\_passes)$
  - $num\_changed\_alphas = 0$.
  - **for** $i = 1, \ldots m$,
    - Calculate $E_i = f(x^{(i)}) - y^{(i)}$ using (2).
    - **if** $((y^{(i)} E_i < -tol \quad \&\& \quad \alpha_i < C) \;||\; (y^{(i)} E_i > tol \quad \&\& \quad \alpha_i > 0))$
      - Select $j \neq i$ randomly.
      - Calculate $E_j = f(x^{(j)}) - y^{(j)}$ using (2).
      - Save old α's: $\alpha_i^{(\text{old})} = \alpha_i$, $\alpha_j^{(\text{old})} = \alpha_j$.
      - Compute L and H by (10) or (11).
      - **if** $(L == H)$
        continue to next i.
      - Compute η by (14).
      - **if** $(\eta >= 0)$
        continue to next i.

- Compute and clip new value for α_j using (12) and (15).
- **if** $(|\alpha_j - \alpha_j^{(\text{old})}| < 10^{-5})$
  continue to next i.
- Determine value for α_i using (16).
- Compute b_1 and b_2 using (17) and (18) respectively.
- Compute b by (19).
- $num\_changed\_alphas := num\_changed\_alphas + 1$.
- **end if**
- **end for**
- **if** $(num\_changed\_alphas == 0)$
  $passes := passes + 1$
- **else**
  $passes := 0$
- **end while**

REFERENCES

1. Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction, Second Edition.*
2. https://github.com/jamesajeeth/Data-Science/tree/master/Adaboost%20from%20scratch [Last Accessed on 10.03.2022)
3. https://www.kdnuggets.com/2020/12/implementing-adaboost-algorithm-from-scratch.html
4. https://cs229.stanford.edu/materials/smo.pdf
5. https://livebook.manning.com/book/machine-learning-in-action/chapter-7/152
6. https://livebook.manning.com/book/machine-learning-in-action/chapter-7/
7. https://www.geeksforgeeks.org/stacking-in-machine-learning/
8. https://www.geeksforgeeks.org/ml-bagging-classifier/
9. https://vitalflux.com/gaussian-mixture-models-what-are-they-when-to-use/
10. https://www.mygreatlearning.com/blog/gaussian-mixture-model/
11. https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a
12. Shevade SK, Keerthi SS, Bhattacharyya C, Murthy KR. Improvements to the SMO algorithm for SVM regression. IEEE transactions on neural networks. 2000 Sep;11(5):1188-93.
13. Sun Z, Ampornpunt N, Varma M, Vishwanathan S. Multiple kernel learning and the SMO algorithm. Advances in neural information processing systems. 2010;23.
14. Dellaert F. The expectation maximization algorithm. Georgia Institute of Technology; 2002.
15. Moon TK. The expectation-maximization algorithm. IEEE Signal processing magazine. 1996 Nov;13(6):47-60.

16. Ng SK, Krishnan T, McLachlan GJ. The EM algorithm. InHandbook of computational statistics 2012 (pp. 139-172). Springer, Berlin, Heidelberg.
17. Moon TK. The expectation-maximization algorithm. IEEE Signal processing magazine. 1996 Nov;13(6):47-60.
18. Dudoit S, Fridlyand J. Bagging to improve the accuracy of a clustering procedure. Bioinformatics. 2003 Jun 12;19(9):1090-9.
19. Grandvalet Y. Bagging equalizes influence. Machine Learning. 2004 Jun;55(3):251-70.
20. Quinlan JR. Bagging, boosting, and C4. 5. InAaai/Iaai, vol. 1 1996 Aug 4 (pp. 725-730).
21. Oza NC, Russell SJ. Online bagging and boosting. InInternational Workshop on Artificial Intelligence and Statistics 2001 Jan 4 (pp. 229-236). PMLR.
22. Ting KM, Witten IH. Stacking bagged and dagged models.
23. Džeroski S, Ženko B. Is combining classifiers with stacking better than selecting the best one?. Machine learning. 2004 Mar;54(3):255-73.
24. Canovas O, Lopez-de-Teruel PE, Ruiz A. Detecting indoor/outdoor places using WiFi signals and AdaBoost. IEEE sensors journal. 2016 Dec 15;17(5):1443-53.
25. Gosztolya G, Grósz T, Busa-Fekete R, Tóth L. Detecting the intensity of cognitive and physical load using AdaBoost and Deep Rectifier Neural Networks. InFifteenth Annual Conference of the International Speech Communication Association 2014.

MOOCS

1. Decision Trees, Random Forests, AdaBoost & XGBoost in Python. Udemy. https://www.udemy.com/course/machine-learning-advanced-decision-trees-in-python/

2. The Ultimate Guide to AdaBoost Algorithm | What is AdaBoost Algorithm?. https://www.mygreatlearning.com/blog/adaboost-algorithm/

3. AdaBoost.https://www.mathworks.com/discovery/adaboost.html

4. Decision Trees, Random Forests, AdaBoost & XGBoost in Python.https://alison.com/course/decision-trees-random-forests-adaboost-and-xgboost-in-python

5. AdaBoost Algorithm.https://www.educba.com/adaboost-algorithm/

6. Machine Learning: Classification. University of Washington, Coursera. https://www.coursera.org/lecture/ml-classification/adaboost-overview-oyMBd

VIDEO LECTURES

1. Gaussian Mixture Models for Clustering. https://www.youtube.com/watch?v=DODphRRL79c
2. Gaussian Mixture Models. https://www.youtube.com/watch?v=q71Niz856KE
3. Gaussian mixture model (Mixture of Gaussians). https://www.youtube.com/watch?v=Rkl30Fr2S38
4. Gaussian Mixture Model. https://www.youtube.com/watch?v=EWd1xRkyEog
5. Unsupervised Learning: Gaussian Mixture Model. https://www.youtube.com/watch?v=fVsmnZqrBUs
6. Gaussian Mixture Models - The Math of Intelligence (Week 7). https://www.youtube.com/watch?v=JNlEIEwe-Cg
7. Gaussian Mixture Model | Object Tracking. https://www.youtube.com/watch?v=0nz8JMyFF14
8. EM algorithm: how it works. https://www.youtube.com/watch?v=REypj2sy_5U
9. Implementing the EM for the Gaussian Mixture in Python | NumPy & TensorFlow Probability. https://www.youtube.com/watch?v=vl_5evp-CYo
10. Matlab video 31: Gaussian mixtures and clustering (1/2). https://www.youtube.com/watch?v=DBVjKZOzw-g
11. EM.1: Introduction to mixture models. https://www.youtube.com/watch?v=3JYcCbO5s6M
12. Gaussian Mixture Models. https://www.youtube.com/watch?v=I9dfOMAhsug
13. SMO and Stochastic SVM. https://www.youtube.com/watch?v=vqoVIchkM7I
14. Sequential minimal optimization (SMO). https://www.youtube.com/watch?v=I73oALP7iWA
15. Support Vector Machines w/ Python & SMO (Sequential Minimal Optimization). https://www.youtube.com/watch?v=ZwGaLJbKHiQ
16. How SVM (Support Vector Machine) algorithm works. https://www.youtube.com/watch?v=1NxnPkZM9bc
17. SVM - Sequential Minimum Optimization Algorithm. https://www.youtube.com/watch?v=xsuyqXwCXRk
18. SMO Coordinate Ascent Algorithm in SVMs. https://www.youtube.com/watch?v=qatOnbAEfjs
19. SMOTE (Synthetic Minority Oversampling Technique) for Handling Imbalanced Datasets. https://www.youtube.com/watch?v=U3X98xZ4_no
20. Sequential Minimal Optimization (SMO) (1/2). https://www.youtube.com/watch?v=vywmP6Ud1HA
21. SMO, Random forest and Bayes net algorithms: why does Random forest perform better?. https://www.youtube.com/watch?v=MXQb2Trvy3s

❄❄❄❄❄❄❄

14

DIMENSIONALITY REDUCTION

Unit Structure

14.0 OBJECTIVES

Dimensionality reduction has several benefits for machine learning data, including:

- Less data means less complexity
- Less data means less computation time
- Model accuracy improves due to less misleading data
- Algorithms train faster thanks to fewer data
- Reducing the data set's feature dimensions helps visualize the data faster
- It removes noise and redundant features.

14.1 INTRODUCTION

Machine learning is a field of study that makes computers enables to "learn" in the same way that humans do without the need for explicit programming.

What are Predictive Modeling and How Does It Work? Predictive modeling is a probabilistic method for forecasting outcomes based on a set of predictors. These predictors are essential characteristics that are considered while determining the ultimate result, the model's outcome.

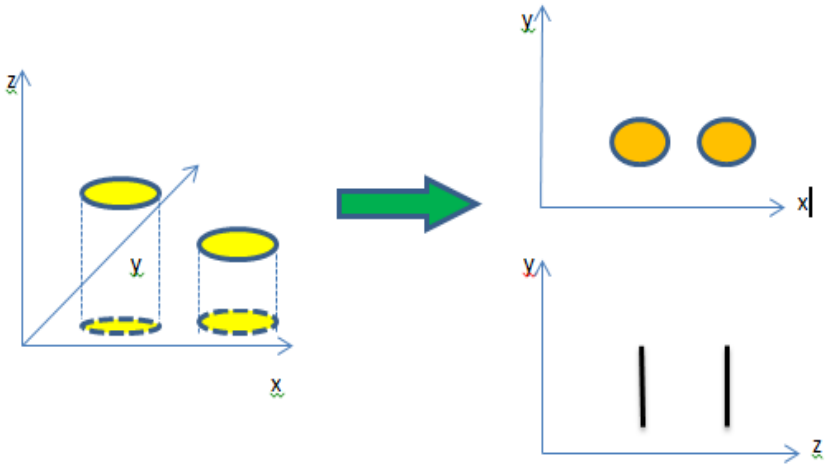
What is Dimensionality Reduction?

There are frequently too many factors on which the final categorization is made in machine learning classification issues. These elements are essentially variables referred to as features. The more features there are,

the more difficult it is to envision the training set and subsequently work on it. Most of these characteristics are sometimes connected and hence redundant. Dimensionality reduction methods are useful in this situation. The technique of lowering the number of random variables under consideration by generating a set of primary variables is known as dimensionality reduction. It is split into two parts: feature selection and feature extraction.

What is the significance of dimensionality reduction in machine learning and predictive modelling?

A simple e-mail classification problem, in which we must determine if the e-mail is spam or not, provides an intuitive illustration of dimensionality reduction. This can include a variety of factors, such as whether or not the email has a generic subject, the email's content, whether or not the email employs a template, and so on. Some of these characteristics, however, may overlap. In another situation, a classification problem that relies on both humidity and rainfall can be reduced into just one underlying feature, because the two are highly associated. As a result, the number of features in such situations can be reduced. A 3-D classification problem can be difficult to visualize, but a 2-D problem can be converted to a basic two-dimensional space and a 1-D problem to a simple line. This notion is explained in the figure below, where a 3-D feature space is split into two 1-D feature spaces, and then the number of features can be decreased even lower if they are discovered to be associated.



14.2 AN OVERVIEW

Dimensionality Reduction

Dimensionality refers to the number of input characteristics, variables, or columns in a dataset, and dimensionality reduction refers to the process of reducing these features.

In some circumstances, a dataset has a large number of input features, making predictive modelling more difficult. Because it is difficult to visualize or forecast a training dataset with a large number of characteristics, dimensionality reduction techniques must be used in such circumstances.

"It is a strategy of turning the higher dimensions dataset into lower dimension dataset while guaranteeing that it gives similar information," says one definition. These methods are commonly applied in machine learning to develop a more accurate predictive model while solving classification and regression challenges.

Speech recognition, signal processing, bioinformatics, and other fields that deal with high dimensional data use it frequently. It can also be used to visualize data, reduce noise, and do cluster analysis, among other things.

We must first grasp dimensionality before we can give a precise explanation of dimensionality reduction. Machine learning algorithm performance may suffer if there are too many input variables. Assume you're representing your ML data using rows and columns similar to those seen on a spread sheet. The columns then become input variables (also known as features) for a model that predicts the target variable.

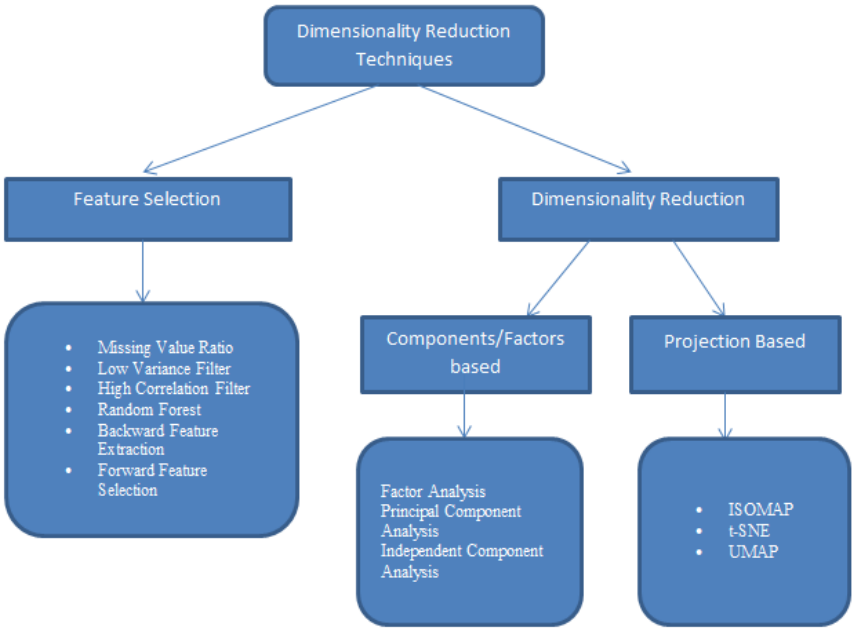
Furthermore, the data columns can be viewed as dimensions on an n-dimensional feature space, whilst the data rows can be viewed as points on the space. Geometrically interpreting a data set is the term for this technique.

Unfortunately, having a lot of dimensions in the feature area leads to a lot of wasted space. As a result, the data points and rows may only represent a small, non-representative sample. This imbalance can harm the performance of machine learning algorithms. The "curse of dimensionality" is the name given to this condition. In the end, a data collection with a large number of input attributes makes predictive modelling more difficult, putting performance and accuracy at risk.

Here's an illustration to help you visualize the issue. Assume you travelled 50 yards in a straight line and dropped a quarter somewhere along the way. You'll most likely discover it quickly. Now, imagine your search area is 50 yards by 50 yards. Your search will now take several days! But we aren't finished yet. Make that search area a 50-by-50-by-50-yard cube now. You might wish to bid that quarter farewell! The more dimensions involved, the more difficult and time-consuming the search becomes.

How can we free ourselves from the dimensionality curse? By lowering the number of input features, the number of dimensions in the feature space is reduced. As a result, "dimensionality reduction" was coined.

14.3 SUBSET SELECTION



The Dimensionality Curse

The curse of dimensionality refers to the difficulty of dealing with high-dimensional data in practice. Any machine learning technique and model becomes increasingly sophisticated as the dimensionality of the input dataset grows. As the number of features grows, the number of samples grows proportionally as well, increasing the risk of over fitting. When a machine learning model is trained on large amounts of data, it becomes over fitted and performs poorly.

As a result, it is frequently necessary to reduce the number of features, which can be accomplished by dimensionality reduction.

Dimension Reduction Methodologies

The dimension reduction approach can be used in two ways, as shown below:

Selection of Features

To develop a high-accuracy model, feature selection is the process of picking a subset of important characteristics and excluding irrelevant features from a dataset. In other words, it's a method for choosing the best characteristics from a dataset.

The feature selection is done in three ways:

1. Filtering Techniques

The dataset is filtered in this manner, and only the relevant features are taken as a subset. The following are some examples of filtering techniques:

- Correlation
- Chi-Square Test
- ANOVA
- Information Gain, etc.

2. Wrappers Methods

The wrapper method achieves the same aim as the filter function, but it analyses it using a machine learning model. In this procedure, various features are fed into the machine learning model, and the performance is evaluated. To enhance the model's accuracy, the performance selects whether to include or eliminate those features. This method is more accurate than filtering, but it is hard to implement. The following are some examples of wrapper methods:

- Forward Selection
- Backward Selection
- Bi-directional Elimination

3. Embedded Methods:

Methods that are built-in examine the machine learning model's various training iterations and score each feature's relevance. Embedded approaches are used in a variety of ways.

- LASSO
- Elastic Net
- Ridge Regression, etc.

Feature Extraction:

The process of changing space with many dimensions into a space with fewer dimensions is known as feature extraction. This method is handy when we want to keep all of the information while processing it with fewer resources.

The following are some examples of common feature extraction techniques:

Principal Component Analysis

- Principal Component Analysis
- Linear Discriminant Analysis
- PCA in the Kernel
- Discriminant Analysis Quadratic

14.4 PRINCIPAL COMPONENTS ANALYSIS

With the use of orthogonal transformation, Principal Component Analysis turns the observations of correlated characteristics into a set of linearly uncorrelated features. The Principal Components are the newly altered features. It's one of the most widely used programs for exploratory data analysis and predictive modeling.

The variance of each characteristic is taken into account by PCA since the high attribute indicates a good separation between the classes and so minimizes dimensionality. Image processing, movie recommendation systems, and optimizing power allocation in multiple communication channels are some of the real-world uses of PCA.

Backward Feature Elimination

When creating a Linear Regression or Logistic Regression model, the backward feature removal strategy is commonly utilised. This technique reduces dimensionality or selects features by doing the following steps:

- In this method, the model is trained using all n variables from the given dataset.
- The model's functionality is examined.
- We'll now delete one feature at a time and train the model on n-1 features for n times before calculating the model's performance.
- We'll look for the variable or features that have had the smallest or no effect on the model's performance, and then remove them, leaving us with n-1 features.
- Continue until no more features may be dropped.

We can specify the ideal number of features required for machine learning algorithms using this strategy, which involves picking the best model performance and the lowest acceptable error rate.

Forward Feature Selection

The backward elimination method is reversed in the forward feature selection phase. It indicates that in this strategy, we won't remove a feature; instead, we'll locate the greatest characteristics that will boost the model's performance the most. This approach involves the following steps:

- We will begin with a single feature and gradually add each feature one at a time.
- We'll train the model on each feature separately at this point.
- The feature that performs the best is chosen.
- The process will be continued until the model's performance has significantly improved.

Missing Value Ratio

If there are too many missing values in a dataset, we remove those variables because they don't provide much information. To do this, we may specify a threshold level, and if a variable has more missing values than that threshold, the variable will be dropped. The more efficient the reduction, the higher the threshold value.

Low Variance Filter

Data columns with certain changes in the data have less information, just like the missing value ratio technique. As a result, we must calculate the variance of each variable, and all data columns with variance less than a certain threshold must be removed, as low variance characteristics will have no effect on the target variable.

High Correlation Filter

When two variables provide essentially similar information, this is referred to be a high correlation. The model's performance may be harmed as a result of this factor. The estimated value of the correlation coefficient is based on the correlation between the independent numerical variables. We can eliminate one of the variables from the dataset if this value is higher than the threshold value. Those factors or traits with a high correlation with the target variable can be considered.

Random Forest

In machine learning, Random Forest is a well-known and helpful feature selection approach. We don't need to program the feature importance package separately because this method already has one. We need to construct a huge number of trees against the target variable in this technique, and then locate the subset of features using usage statistics for each attribute.

Because the random forest algorithm only accepts numerical variables, we must use hot encoding to convert the input data to numeric data.

Factor Analysis

Factor analysis is a technique in which each variable is grouped based on its relationships with other variables. This means that variables within a group may have a strong connection with one another, but have a weak association with variables from other groups.

We can grasp two variables, for example, if they are Income and Spending. These two variables are strongly linked, meaning that individuals with more income spend more and vice versa. As a result, comparable variables are grouped, and this group is called the factor. The number of these components will be lowered in comparison to the dataset's original dimension.

Auto-encoders

The auto-encoder, a type of ANN or artificial neural network to copy inputs to outputs, is one of the most popular methods of dimensionality

reduction. This compresses the input into a latent-space representation, which is then used to produce the output. There are two key elements to it:

o Encoder: The encoder's job is to compress the data so that it may be represented in latent space.

o Decoder: The decoder's job is to recreate the latent-space representation's output.

Principal Component Analysis is an unsupervised learning approach used in machine learning to reduce dimensionality. With the help of orthogonal transformation, it is a statistical technique that turns observations of correlated features into a set of linearly uncorrelated data. The Principal Components are the newly altered features. It's one of the most widely used programs for exploratory data analysis and predictive modeling. It's a method for extracting strong patterns from a dataset by lowering variances.

PCA seeks out the lowest-dimensional surface on which to project the high-dimensional data.

The variance of each characteristic is taken into account by PCA since the high attribute indicates a good separation between the classes and so minimises dimensionality. Image processing, movie recommendation systems, and optimising power allocation in multiple communication channels are some of the real-world uses of PCA. Because it is a feature extraction technique, it keeps the important variables while discarding the less important ones.

The PCA technique is based on a number of mathematical ideas, including: SQL Triggers (Hindi)

- Variance and Covariance
- Eigenvalues and Eigen factors

The following are some terms that are commonly used in the PCA algorithm:

- Dimensionality: This refers to the number of characteristics or variables in a dataset. It's the number of columns in the dataset, to put it simply.
- Correlation: This term refers to the degree to which two variables are related to one another. For example, if one variable changes, the other variable changes as well. The correlation value can be anywhere between -1 and +1. If the variables are inversely proportional to each other, the result is -1, and if the variables are directly proportional to each other, the result is +1.
- Orthogonal: It denotes that the variables are unrelated to one another, and thus the correlation between them is 0.
- Eigenvectors: If a square matrix M is given with a non-zero vector v. If Av is the scalar multiple of v, then v is an eigenvector.

- Covariance Matrix: The Covariance Matrix is a matrix that contains the covariance between two variables.

Principal Components in PCA

The Principal Components are the converted new features or the result of PCA, as stated above. The number of PCs in this dataset is either equal to or less than the number of original features in the dataset. The following are some of the properties of these main components:

- The linear combination of the original features must be the main component.
- These components are orthogonal, which means there is no link between two variables.
- As the number of components increases from 1 to n, the importance of each component diminishes, indicating that the 1 PC is the most important and the n PC is the least important.

Steps for PCA algorithm

1. Obtaining the dataset:
   To begin, we must divide the input dataset into two halves, X and Y, with X being the training set and Y being the validation set.

2. Organizing information into a structure
   Now we'll use a structure to represent our data. As an example, the two-dimensional matrix of independent variable X will be represented. Each column correlates to the Features, and each row corresponds to the data elements. The dataset's dimensions are determined by the number of columns.

3. Data standardization
   We'll normalize our data in this stage. In a given column, for example, features with large variance are more essential than features with a lower variance.

   If the importance of features is unaffected by the variance of the feature, we shall divide each data item in a column by the column's standard deviation. The matrix will be referred to as Z in this case.

4. Determining Z's Covariance
   We will take the matrix Z and transpose it to get the covariance of Z. We'll multiply it by Z after it's been transposed. The Covariance matrix of Z will be the output matrix.

5. Determining Z's Covariance
   We will take the matrix Z and transpose it to get the covariance of Z. We'll multiply it by Z after it's been transposed. The Covariance matrix of Z will be the output matrix.

6. Eigen Values and Eigen Vectors Calculation
   The eigenvalues and eigenvectors for the resulting covariance matrix Z must now be calculated. The directions of the axes with high information are called eigenvectors or the covariance matrix. The eigenvalues are defined as the coefficients of these eigenvectors.

7. Eigen Values and Eigen Vectors Calculation
   The eigenvalues and eigenvectors for the resulting covariance matrix Z must now be calculated. The directions of the axes with high information are called eigenvectors or the covariance matrix. The eigenvalues are defined as the coefficients of these eigenvectors.

8. Sorting the Eigen Vectors is the eighth step.
   We'll take all of the eigenvalues and arrange them in decreasing order, from largest to smallest, in this phase. In the eigenvalues matrix P, sort the eigenvectors in the same order. P* will be the name of the resulting matrix.

9. figuring out the new features Alternatively, Principal Components
   We'll calculate the new features here. We'll do this by multiplying the P* matrix by the Z. Each observation in the resulting matrix Z* is a linear combination of the original features. The Z* matrix's columns are all independent of one another.

10. Remove features from the new dataset that are less significant or irrelevant.
    We'll determine what to keep and what to discard now that the new feature set has arrived. It indicates that only relevant or crucial features will be kept in the new dataset, while unimportant ones will be deleted.

Applications of Principal Component Analysis

- PCA is mostly employed as a dimensionality reduction approach in AI applications like computer vision, picture compression, and so on.
- If the data includes a lot of dimensions, it can also be utilized to find hidden patterns. Finance, data mining, psychology, and other fields employ PCA.

14.5 MULTIDIMENSIONAL SCALING

The Multidimensional Scaling (MDS) algorithm for dimensionality reduction

- **Introduction**
- **With bigger data comes the need for a lower representation**

Massive amounts of very high-dimensional (with a large number of characteristics) or unstructured data are continuously produced and stored at significantly lower costs than they were previously. The need for data

interpretation and analysis is growing as people, industries, and research projects become increasingly data-driven. As a result, a growing number of businesses are investing in data-driven products to satisfy their monitoring, experimentation, data analysis, simulations, and other knowledge and business requirements.

The issue with the rise of real-world data is the way of representing them. We mean the selection of top, descriptive features as representations of the data itself when we say "represent them." Because real-world data is often very unstructured and collected in enormous amounts, extracting features or attributes of the data for analysis and research is a difficult and time-consuming task for all researchers and enterprises that rely on data as a key source of energy. As a result, when evaluating high-dimensional data, dimension reduction and variable selection are critical in terms of statistical correctness.

- **Dimensionality reduction in a nutshell**

The technique of lowering the number of random variables under consideration by generating a collection of primary variables is known as dimensionality reduction.

Dimension reduction strategies are used to condense the original p-dimensional data space into a reduced k-dimensional components subspace. Many statistical and mathematical methodologies, such as Principal component analysis (PCA), Linear discriminant analysis (LDA), Factor Analysis, and others, were developed to achieve this purpose.

The MDS algorithm

Multidimensional scaling sometimes referred to as Principal Coordinates Analysis (PCoA), Torgerson Scaling, or Torgerson–Gower scaling, is a psychometrics-based statistical technique. Dissimilarities between pairs of objects are the data utilized in multidimensional scaling (MDS).

The basic goal of MDS is to express these dissimilarities as distances between points in a low-dimensional space that areas near to the dissimilarities as possible.

The vanilla or classical MDS

Young and Householder demonstrated how, starting with a matrix of distances between points in a Euclidean space, coordinates for the points may be obtained while distances are kept in the classic, non-revised version of MDS in the 1930s. However, there are a few non-Euclidean distance measurements that are only useful for extremely specific research issues. Torgerson eventually popularised the subject by employing the scaling technique. As a result, multidimensional scaling emerged as a strategy for reducing dimensionality and visualizing high-dimensional data. The following is the algorithm for the classic version of multidimensional scaling:

Step1: Given a matrix D with dimensionality of m, calculate for matrix X with the reduced dimension p.

Step 2: From D, compute matrix B by applying a centering matrix to D.

Step 3: Determine the largest eigenvalues (λ_1,λ_2,……,λ_{n-1}) and corresponding eigenvectors (V_1,V_2……V_{n-1}) of matrix B with respect to p.

Step 4: Get the square root of the dot product of the matrix of eigenvectors and dot product of the matrix of eigenvectors and the diagonal matrix of eigenvalues of B.

According to the procedure, Euclidean space with at least n-1 dimensions may be found whose distances equal the initial dissimilarities. The entire n-1 dimensions are required in the space since the matrix B utilized in the technique is usually of rank n-1, and therefore little has been achieved in data dimension reduction.

Furthermore, Gower was the first to fully express the formulation and relevance of the classical scaling technique, and he invented the term "principal coordinates analysis" from his selection of the first p "principal coordinates" for the configuration (PCA). Principal coordinates analysis, like the phrase metric scaling, has become synonymous with traditional multidimensional scaling. Metric scaling, on the other hand, is more than just this one technique.

The distances between the points in the n-1 dimensional Euclidean space are given by the equation below, which is the spectral decomposition of matrix B..

$$d_{rs}^2 = \sum_{i=1}^{n-1} \lambda_i (X_{ri} - X_{si})^2$$

Matrix B decomposition equation

As a result, if a large number of eigenvalues are "small," their contribution to the squared distance d2rs can be ignored. If only p eigenvalues are retained as being significantly large (about the target number of projected dimensions), then the p dimensional Euclidean space formed for the first p eigenvalues and with xr truncated to the first p elements can be used to represent and interpret the objects in the lower dimension.

Furthermore, negative eigenvalues (and their eigenvectors) are ignored as errors in classical scaling. For the convenience of graphic representation, p should be small, preferably 2 or 3.

Data for MDS

'Proximities' is a term used to describe the data utilized in MDS analysis. The general similarity (or dissimilarity) of the pieces in the data is indicated by proximity. MDS will strive for a spatial configuration of the elements in which the distances between them are as close as possible to

their proximities. The data is frequently organized in a square matrix called a proximity matrix. There are two types of methods for calculating distances: direct and indirect approaches.

Subjects might use direct methods to provide a numerical similarity or dissimilarity value to each pair of objects, or they could rank the pairs according to their similarity or dissimilarity. Both strategies are direct approaches to gathering proximity data.

Indirect proximity data approaches do not require a subject to explicitly assign a numerical value to the elements of the proximity matrix. Rather, different measurements are used to create the closeness matrix. Data from confusion matrices or correlation matrices are examples of this.

Unlike univariate statistical methods, however, the outcome of an MDS analysis is more reliant on the judgments made in advance. According to Wickelmaier, during the data collecting stage, it is important to remember that asking for similarity ratings rather than dissimilarity ratings may have an impact on the results. A similarity judgment, for example, cannot simply be thought of as the "opposite" of a dissimilarity judgment. Furthermore, a choice must be made between direct and indirect data collection methods, as well as symmetric versus asymmetric data collection methods.

Example using MDS

An example or two will best illustrate the goal of a multidimensional scaling analysis. In one case, Ekman used data from the discipline of psychology to investigate the perception of fourteen different colors. A respondent rated each pair of colors on a scale of 'no similarity' to 'identical.' The results can be adjusted so that identical colors are represented by 0 and completely different colors are represented by 1. The table below shows the averages of these dissimilarity scores across the 31 respondents.

nm	434	445	465	472	490	504	537	555	584	600	610	628	651	674
434	–													
445	.14	–												
465	.58	.50	–											
472	.58	.56	.19	–										
490	.82	.78	.53	.46	–									
504	.94	.91	.83	.75	.39	–								
537	.93	.93	.90	.90	.69	.38	–							
555	.96	.93	.92	.91	.74	.55	.27	–						
584	.98	.98	.98	.98	.93	.86	.78	.67	–					
600	.93	.96	.99	.99	.98	.92	.86	.81	.42	–				
610	.91	.93	.98	1.00	.98	.98	.95	.96	.63	.26	–			
628	.88	.89	.99	.99	.99	.98	.98	.97	.73	.50	.24	–		
651	.87	.87	.95	.98	.98	.98	.98	.98	.80	.59	.38	.15	–	
674	.84	.86	.97	.96	1.00	.99	1.00	.98	.77	.72	.45	.32	.24	–

Table I. Dissimilarities of colors with wavelengths

The colors span from bluish-purple through blue, green, yellow, and red starting at wavelength 434. The dissimilarities in this scenario are

symmetric. Similarly, the degree to which colors with wavelengths 490 and 584 are the same is the same as the degree to which colors 584 and 490 are the same. As a result, just the lower triangular portion of the data has to be presented in the table. In MDS, the diagonal is also irrelevant because an object's distance from itself is always 0.

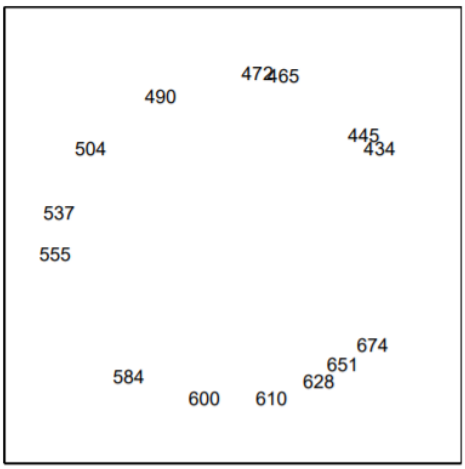


Figure 1. Transformed two-dimensional matrix from Table I data

MDS seeks to express the differences in Table I in a map, as indicated in the definition, to make it more interpretable and easier to analyze correlations and such. Figure 1 depicts a two-dimensional MDS map. The wavelengths of the colors are represented in the shape of a circle. The inter-point distances indicated on this flattened map should be used to interpret it.

Because distances do not vary with rotation, a map rotation does not affect the interpretation.

Similarly, neither a translation of the solution (that is, a shift of all coordinates by a fixed value per dimension) nor a reflection of one or both axes changes the distances.

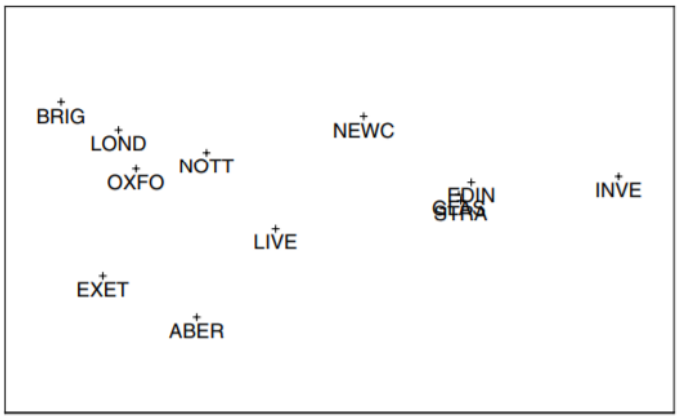


Figure 2. Generated map of British cities using MDS

Road distances between certain British cities were presented as another illustration by Cox & Cox. For example, the road travel times between twelve British cities were subjected to multidimensional scaling. The points obtained by the procedure are shown in Figure 2 above. The positions of the points representing the cities in Figure 2 are strikingly similar to the positions of the same cities in a geographical map of Great Britain, with the exception that the cities in Figure 2 appear to be reflected around a line and rotated from the geographical map typically presented in an atlas.

MDS as a visualization tool

Many of the various learning approaches (such as MDS) are said to be bad for data visualization, which is easy to detect empirically. The reason for this is that they were created to find a d-dimensional manifold if the data's inherent dimensionality is d. The display must have d = 2 or d = 3 for viewing; that is, the dimensionality of the data may need to be lowered beyond its intrinsic dimensionality.

It is well-known that a high-dimensional data set cannot be correctly represented in a lower-dimensional space, such as the plane with d = 2, according to Kaski and Jaakko. As a result, a visualisation method must decide which kind of errors to make. The decision should, of course, be based on the visualisation aim; but, it turns out that under a specified but broad goal, the decision can be described as an intriguing tradeoff, as illustrated below.

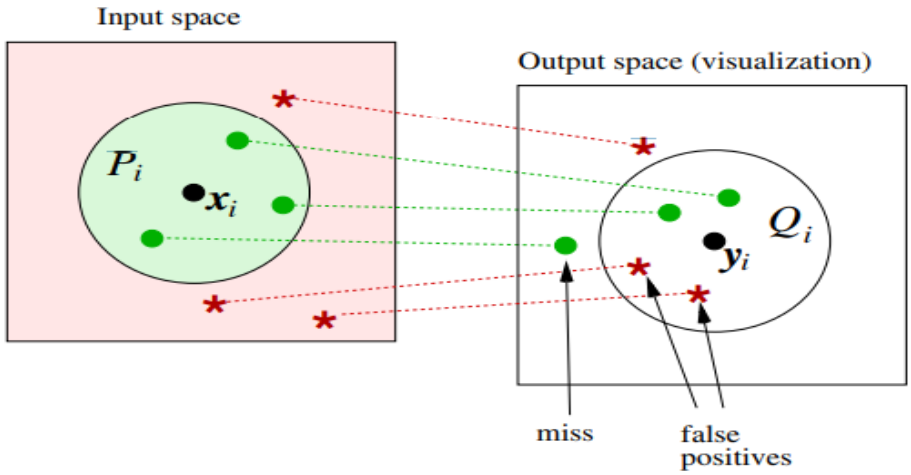


Figure 3. Two kinds of errors when projecting to a lower-dimensional space

when the task is to visualize which data points are similar, the visualization can make two types of errors, as shown in the diagram above: it can miss some similarities (by placing similar data points far apart as false negatives) or it can bring dissimilar data points too close together as false positives.

When and when should MDS be used?

Multidimensional scaling, according to Young, can be used (and misapplied) to a wide range of data types. Technically, any matrix of raw or converted data is a candidate for multidimensional scaling analysis if the elements of the data matrix show the strength or degree of relationship between the objects or events represented by the data matrix's rows and columns. Correlations, distances, proximities, similarities, multiple rating scales, preference matrices, and other forms of such data are referred to as relational data.

For all types of relational data matrices, including symmetric and asymmetric matrices, rectangular and square matrices, matrices with or without missing elements, equally and unequally replicated data matrices, two-way and multi-way matrices, and other types of matrices, multidimensional scaling methods have been developed.

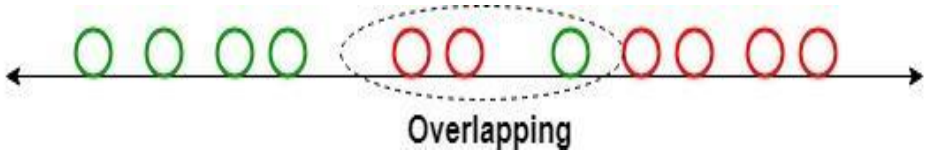
14.6 LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis

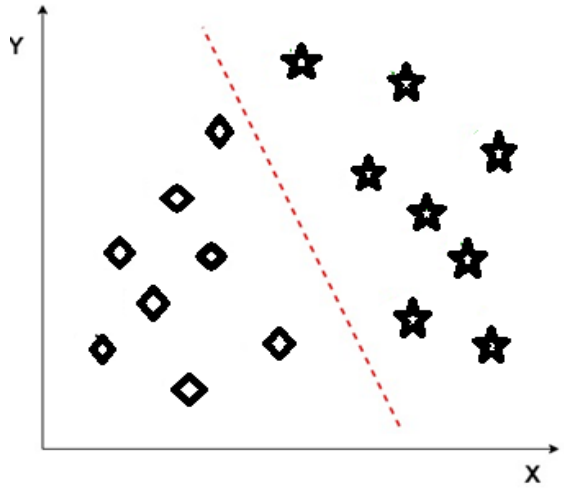
(LDA) is a type of discriminant analysis that

A dimensionality reduction technique is known as Linear Discriminant Analysis, Normal Discriminant Analysis, or Discriminant Function Analysis is often employed for supervised classification problems. It's used to represent group differences, such as separating two or more classes. It is used to project higher-dimensional features onto a lower-dimensional space more

We have two classes, for example, and we need to effectively divide them. Classes can have a variety of characteristics. Using only one feature to classify them can lead to some overlap, as seen in the diagram below. As a result, we will continue to increase the number of features required for proper classification.



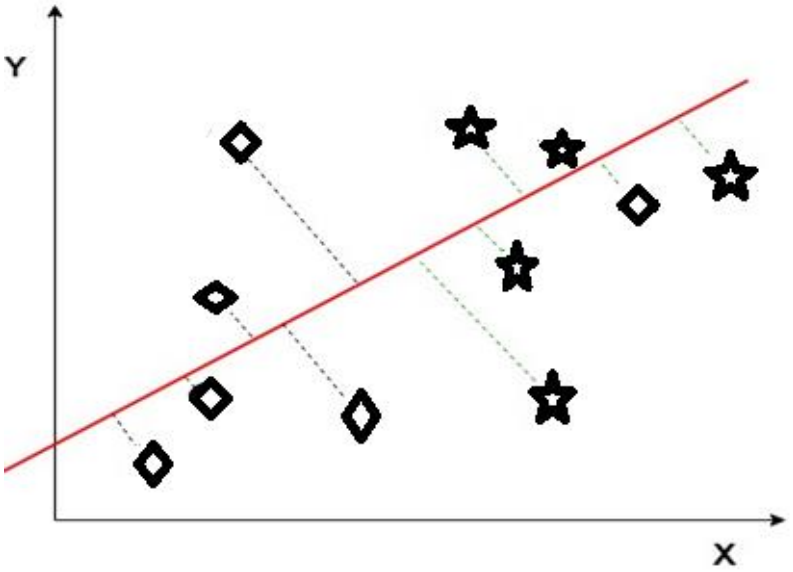
Example:
Assume we have two sets of data points to categorize, each of which belongs to a distinct class. When the data points are displayed on the 2D plane, there is no straight line that can entirely divide the two classes of data points, as seen in the 2D graph. As a result, LDA (Linear Discriminant Analysis) is utilized in this scenario, which lowers the 2D graph to a 1D graph to optimize the separability between the two classes.



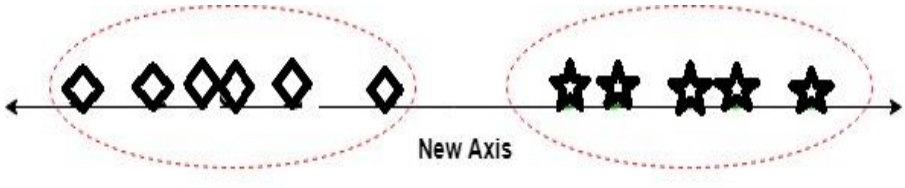
Linear Discriminant Analysis, in this case, uses both axes (X and Y) to establish a new axis and projects data onto it to maximize the separation of the two categories and so reduce the 2D graph to a 1D graph.

LDA uses two criteria to construct a new axis:

1. Increase the distance between the two classes' means.
2. Keep the diversity within each class to a minimum.



In the above graph, a new axis (in red) is constructed and plotted in the 2D graph in such a way that it optimizes the distance between the two classes' means while minimizing variance within each class. To put it another way, this newly created axis widens the gap between the data points of the two classes. All of the data points from the classes are plotted on this new axis when the above-mentioned criteria are applied, as illustrated in the image below.



In the above graph, a new axis (in red) is constructed and plotted in the 2D graph in such a way that it optimizes the distance between the two classes' means while minimizing variance within each class. To put it another way, this newly created axis widens the gap between the data points of the two classes. All of the data points from the classes are plotted on this new axis when the above-mentioned criteria are applied, as illustrated in the image below.

Assume we have two classes and a collection of d-dimensional samples, such as x1, x2,... x-n, with

- n1 samples from class (c1) and n2 samples from class (c2) (c2).
- vTxi is the projection of xi on the line represented by unit vector v if xi is the data point.

Consider u1 and u2 to be the means of samples from classes c1 and c2 before projection, and u1hat to be the mean of samples from class c2.

to be the mean of samples from class after projection, which may be calculated as follows:

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

Similarly,

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

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 c1 is:

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

Similarly:

$$\widetilde{s_2^2} = \sum_{y_i \in c_1} (y_i - \mu_2)^2$$

Now, we need to project our data on the line having direction v which maximizes

$$J(v) = \frac{\widetilde{\mu_1} - \widetilde{\mu_2}}{s_1^2 + s_2^2}$$

To optimize the above equation, we must construct a projection vector that maximizes the difference in means while reducing both classes' scatters. The scatter matrices of classes c1 and c2 are now:

$$s_1 = \sum_{x_i \in c_1} (x_i - \mu_1)(x_i - \mu_1)^T$$

and s2

$$s_2 = \sum_{x_i \in c_2} (x_i - \mu_2)(x_i - \mu_2)^T$$

After simplifying the above equation, we get:

Now, we define, scatter within the classes(s_w) and scatter b/w the classes(s_b):

$$s_w = s_1 + s_2$$

$$s_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

Now, we try to simplify the numerator part of J(v)

$$J(v) = \frac{|\widetilde{\mu_1} - \widetilde{\mu_2}|}{s_1^2 + s_2^2} = \frac{v^T s_b v}{v^T s_w v}$$

Now, To maximize the above equation we need to calculate differentiation with respect to v

$$\frac{dJ(v)}{dv} = s_b v - \frac{v^T s_b v (s_w v)}{v^T s_w v}$$

$$= s_b v - \lambda s_w v = 0$$

$$s_b v = \lambda s_w v$$

$$s_w^{-1} s_b v = \lambda v$$

$$Mv = \lambda v$$

$$where,$$

$$\lambda = \frac{v^T s_b v}{v^T s_w v} and$$

$$M = s_w^{-1} s_b$$

We'll use the value that corresponds to the highest eigenvalue for the largest value of J(v). This will provide us with the finest LDA solution.

LDA's Expansions:

1. Quadratic Discriminant Analysis (QDA): Each class calculates its variance estimate (or covariance when there are multiple input variables).

2. Flexible Discriminant Analysis (FDA): When non-linear input combinations, such as splines, are used.

3. Regularized Discriminant Analysis (RDA): RDA modifies the influence of different variables on LDA by introducing regularisation into the estimate of variance (really covariance).

14.7 LET US SUM UP

Reducing the data set's feature dimensions helps visualize the data faster and removes noise and redundant features. The technique of lowering the number of random variables under consideration by generating a set of primary variables is known as dimensionality reduction. It is split into two parts: feature selection and feature extraction. Dimensionality refers to the number of input characteristics, variables, or columns in a dataset. It is commonly applied in machine learning to develop a more accurate predictive model.

It can also be used to visualize data, reduce noise, and do cluster analysis, among other things. This imbalance can harm the performance of machine learning algorithms. The "curse of dimensionality" is the name given to this condition. How can we free ourselves from the dimensionality curse? By lowering the number of input features.

Any machine learning technique and model becomes increasingly sophisticated as the dimensionality of the input dataset grows. To develop a high-accuracy model, feature selection is the process of picking a subset of important characteristics and excluding irrelevant features from a dataset.

14.8 LIST OF REFERENCES

1. https://www.geeksforgeeks.org/dimensionality-reduction/

2. https://www.javatpoint.com/dimensionality-reduction-technique

3. https://medium.datadriveninvestor.com/the-multidimensional-scaling-mds-algorithm-for-dimensionality-reduction-9211f7fa5345

4. https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/

14.9 BIBLIOGRAPHY

1. https://www.geeksforgeeks.org/dimensionality-reduction/
2. https://www.javatpoint.com/dimensionality-reduction-technique
3. https://medium.datadriveninvestor.com/the-multidimensional-scaling-mds-algorithm-for-dimensionality-reduction-9211f7fa5345
4. https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/

14.10 UNIT END EXERCISES

1. What is Dimensionality Reduction?
2. Explain the significance of Dimensionality Reduction.
3. What is PCA? What does a PCA do?
4. List down the steps of a PCA algorithm.
5. Is it important to standardize the data before applying PCA?
6. What are the assumptions taken into consideration while applying PCA?
7. What are the properties of Principal Components in PCA?
8. What does the coefficient of Principal Component signify?
9. What are the Advantages of Dimensionality Reduction?
10. What does linear discriminant analysis do?

❄❄❄❄❄❄❄