ST.MARTIN'S ENGINEERING COLLEGE

(UGC Autonomous)

Dhulapally, Near Kompally,Medchal–Malkajgiri district

Secunderabad-500100,Telangana, India

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

III B.Tech – II SEMESTER (R22)

CS602PC-MACHINE LEARNING

Study Material

SYLLABUS

UNIT-I

Learning Types of Machine Learning - Supervised Learning The Brain and the Neuron Design a Learning System - Perspectives and Issues in Machine Learning - Concept Learning Task Concept Learning as Search Finding a Maximally Specific Hypothesis - Version Spaces and the Candidate Elimination Algorithm Perceptron-Linear Separability- Linear Regression.

UNIT-II

Multi-layer Perceptron-Going Forwards - Going Backwards: Back Propagation Error - Multi-layer Perceptron in Practice Examples of using the MLP Overview Deriving Back-Propagation Radial Basis Functions and Splines Concepts RBF Network Curse of Dimensionality Interpolations and Basis Functions Support Vector Machines

UNIT-III

Learning with Trees Decision Trees Constructing Decision Trees Classification and Regression Trees Ensemble Learning Boosting Bagging

TEXT BOOKS:

1. Stephen Marsland, —Machine Learning – An Algorithmic Perspective||, Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014.

2. Tom M Mitchell, —Machine Learning||, First Edition, McGraw Hill Education, 2013.

REFERENCES:

1. Peter Flach, —Machine Learning: The Art and Science of Algorithms that Make Sense of Data||, First Edition, Cambridge University Press, 2012.

2. Jason Bell, —Machine learning – Hands on for Developers and Technical Professionals||, First Edition, Wiley, 2014

3. Ethem Alpaydin, —Introduction to Machine Learning 3e (Adaptive Computation and Machine Learning Series)||, Third Edition, MIT Press, 2014

Introduction:-Data science is an inter-disciplinary is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from many structural and unstructured data is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from many structural and unstructured data. Data science is related to data mining(Data mining is a process of discovering patterns in large data sets is a process of discovering patterns in large data sets), machine learning is a process of discovering patterns in large data sets), machine learning and big data.

Unstructured data (or unstructured information) is information thateither does not have a pre-defined data model) is information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Unstructured information is typically text) is information that either does not have a pre-defined data model or is not organized in a 3

What is Data?

- Data is often viewed as the lowest level of abstraction from which information and knowledge are derived.
- Data can be numbers, words,measurements, observations or even justdescriptions of things. Also, data is a representation of a fact, figure and idea.
- Data on its own carries no meaning. If data to be an information, it must beinterpreted and take on a meaning.

An example of raw data table. It is just a collection of random info and data.

Machine learning (ML) is the study of computer algorithms that improveautomatically through experience. It is seen as a subset of artificialintelligence)is the study of computer algorithms that improve automatically through experience .It is seen as a subset of artificialintelligence. Machine learning algorithms build a mathematical model) isthe study of computer algorithms that improve automatically throughexperience. It is seen as a subset of artificial intelligence. Machine learningalgorithms build a mathematical model based on sample data, known as training data)is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data known as "training data" in order to make predictions or decisions without being explicitly programmed to do so.Machine learning algorithms are used in a wide variety of applications,such as email filtering) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial6

Types of Data / Variables

Categorical Data is the data that is non numeric.

e.g.. Favorite color, Place of Birth, Types of Car

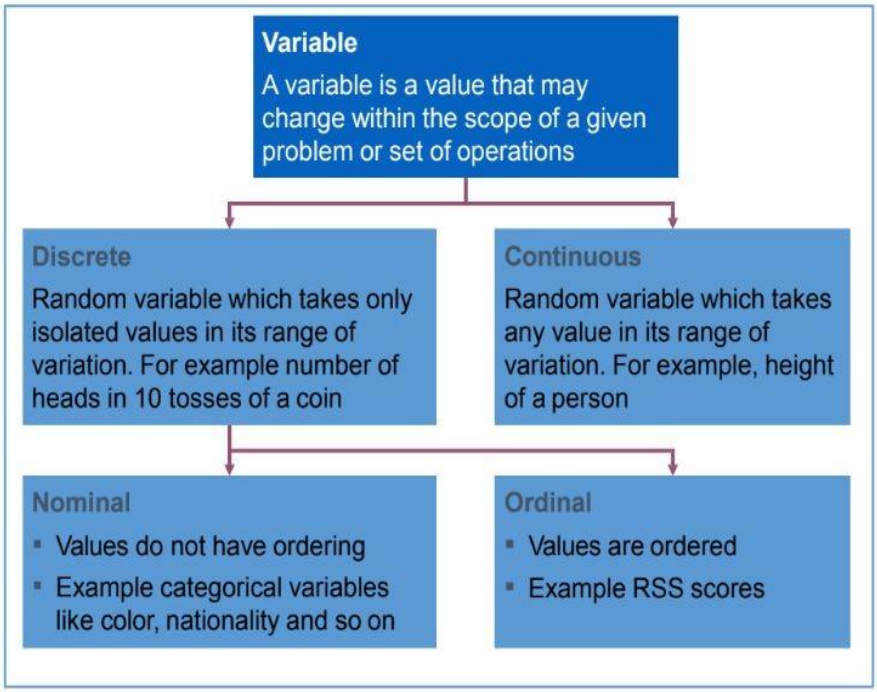
Quantitative Data is numerical. There are 2 types of quantitative data.

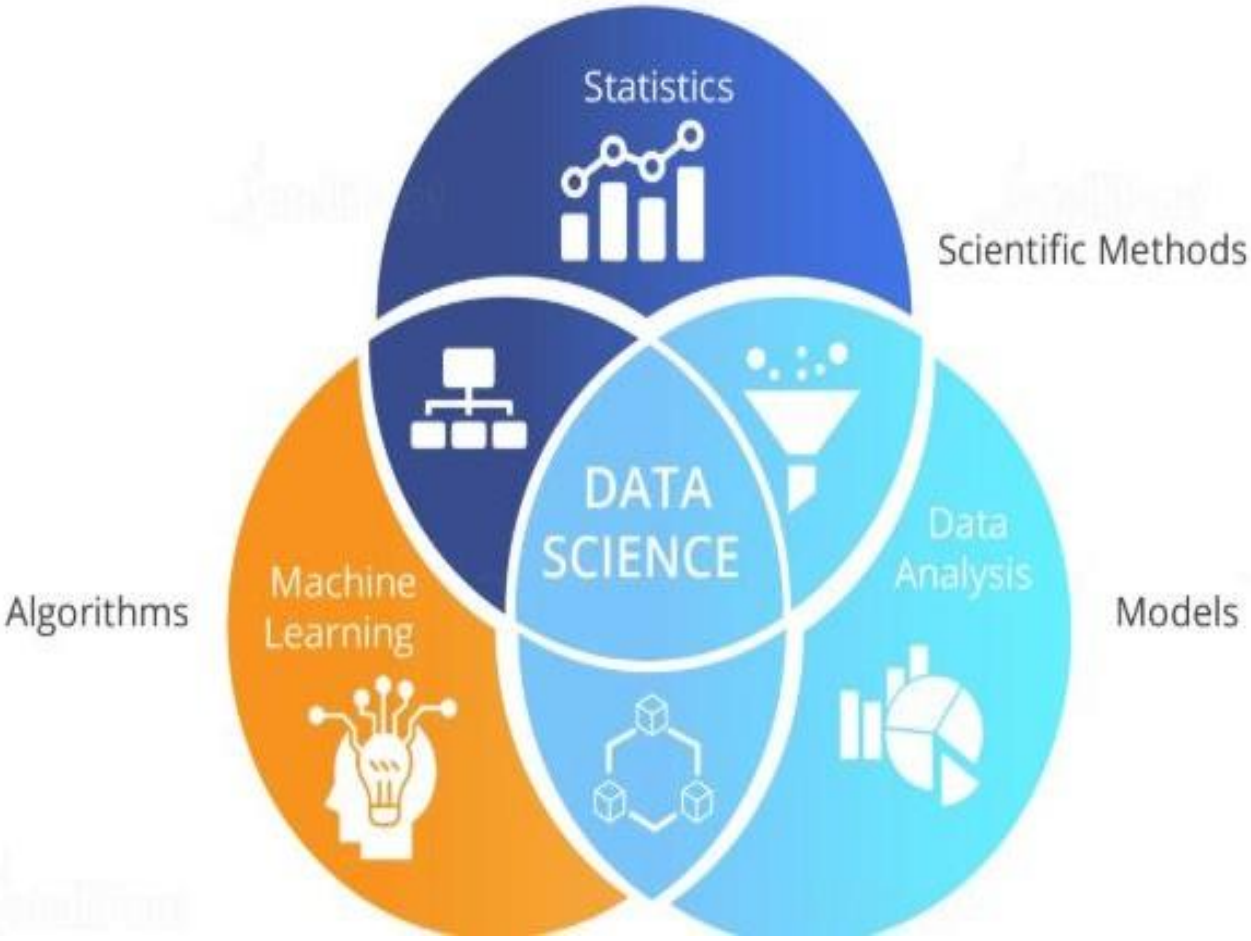
1. **Discrete data** can only take specific values;

   e.g. shoe size, number of brothers, number of cars in a car park.

2. **Continuous data** can take any numerical value;

   e.g. height, mass, length.





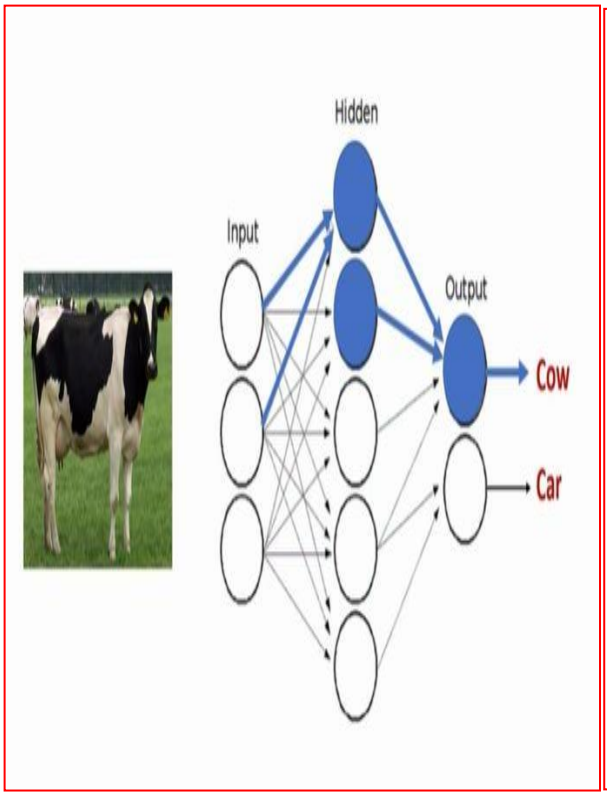
What is Machine Learning?

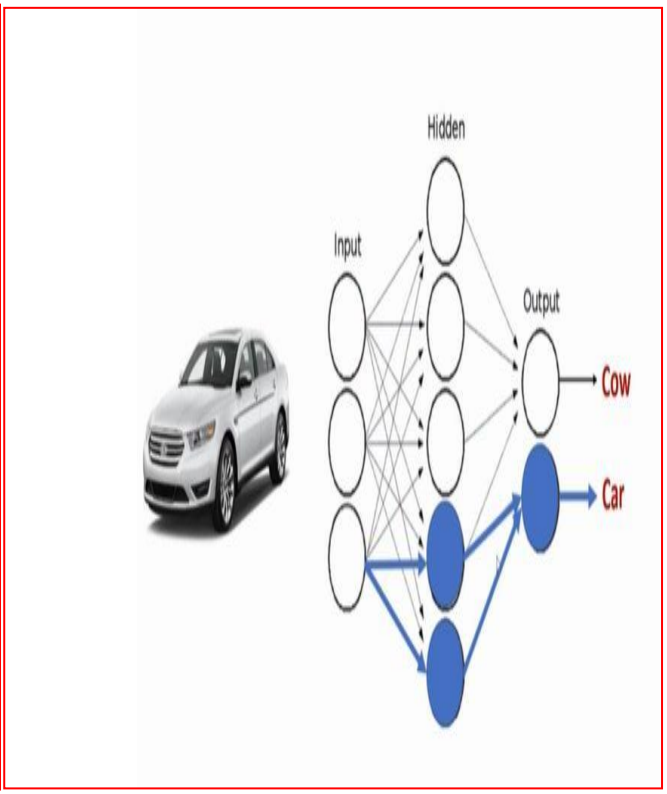
The subfield of computer science that "gives computers the ability to learn without being explicitly programmed".
(Arthur Samuel, 1959)

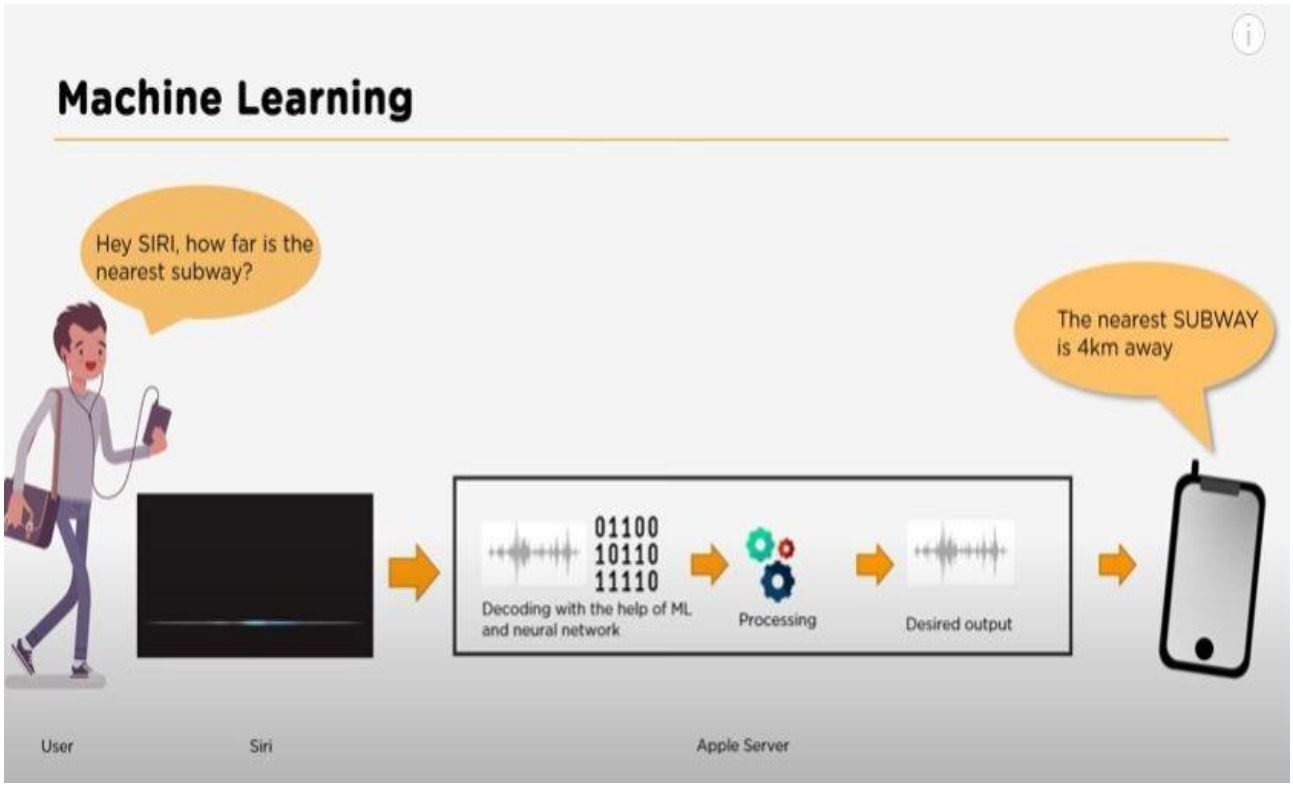
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."
(Tom Mitchell, 1997)

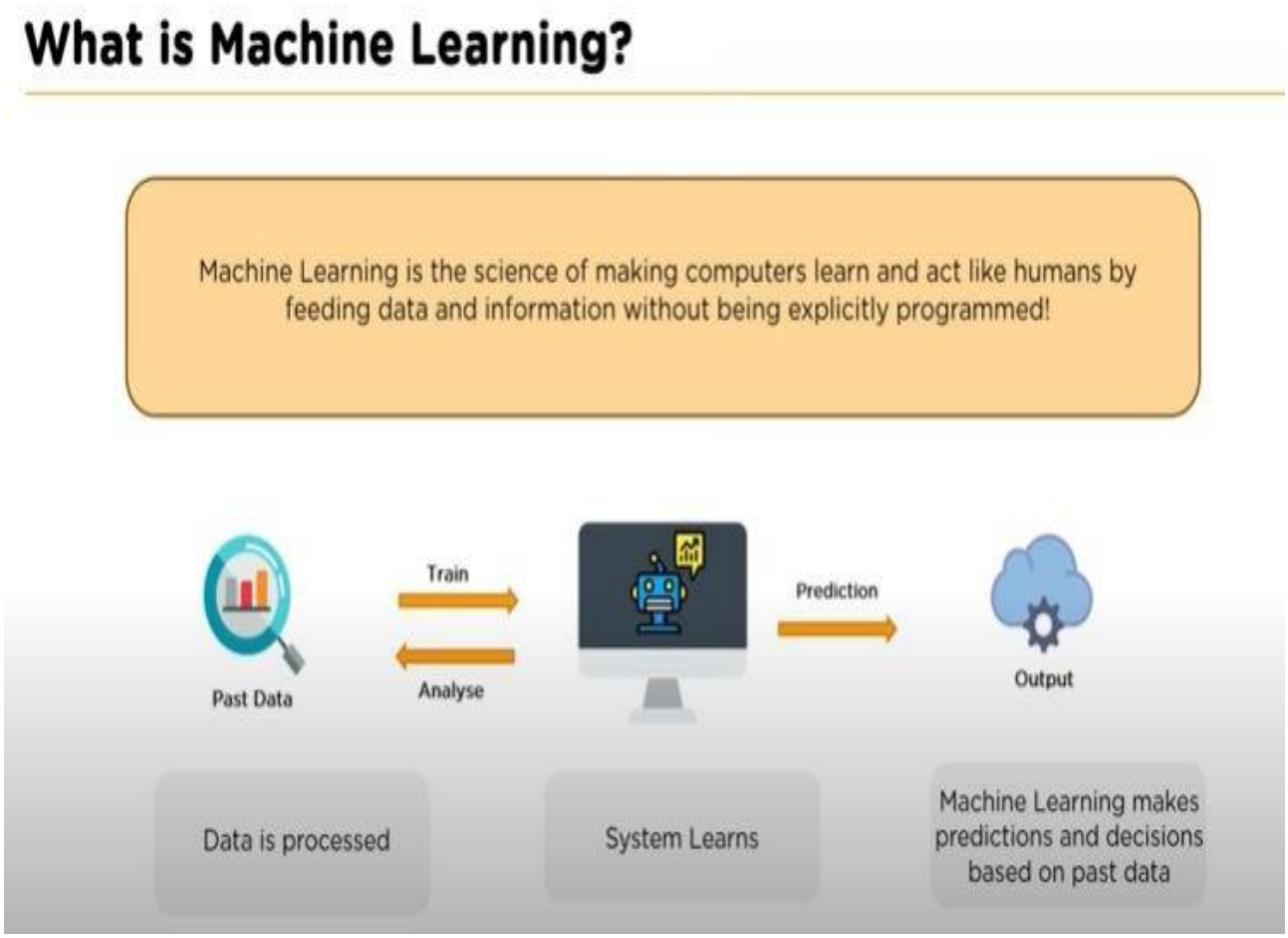
Using data for **answering questions**

Training Predicting



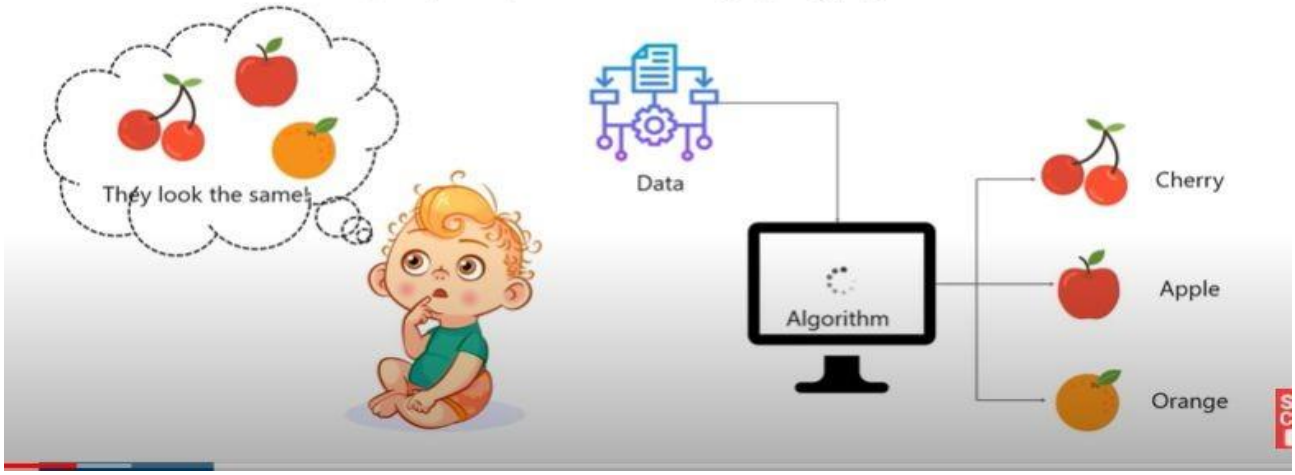



Machine Learning
Hey SIRI, how far is the nearest subway?
The nearest SUBWAY is 4km away
01100
10110
11110
Decoding with the help of ML and neural network
Processing
Desired output
User
Siri
Apple Server

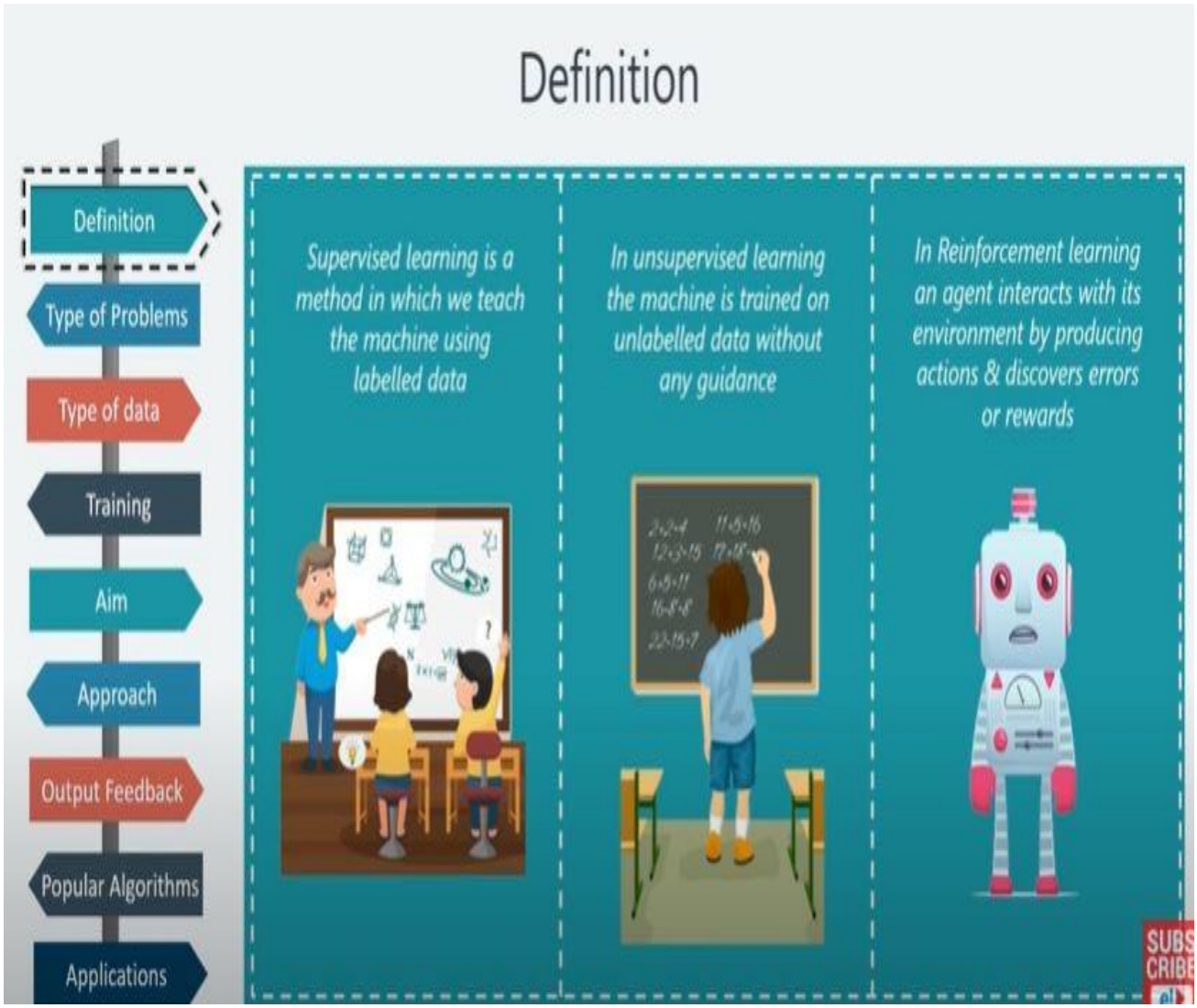

What is Machine Learning?
Machine Learning is the science of making computers learn and act like humans by feeding data and information without being explicitly programmed!
Past Data
Train
Analyse
Prediction
Output
Data is processed
System Learns
Machine Learning makes predictions and decisions based on past data

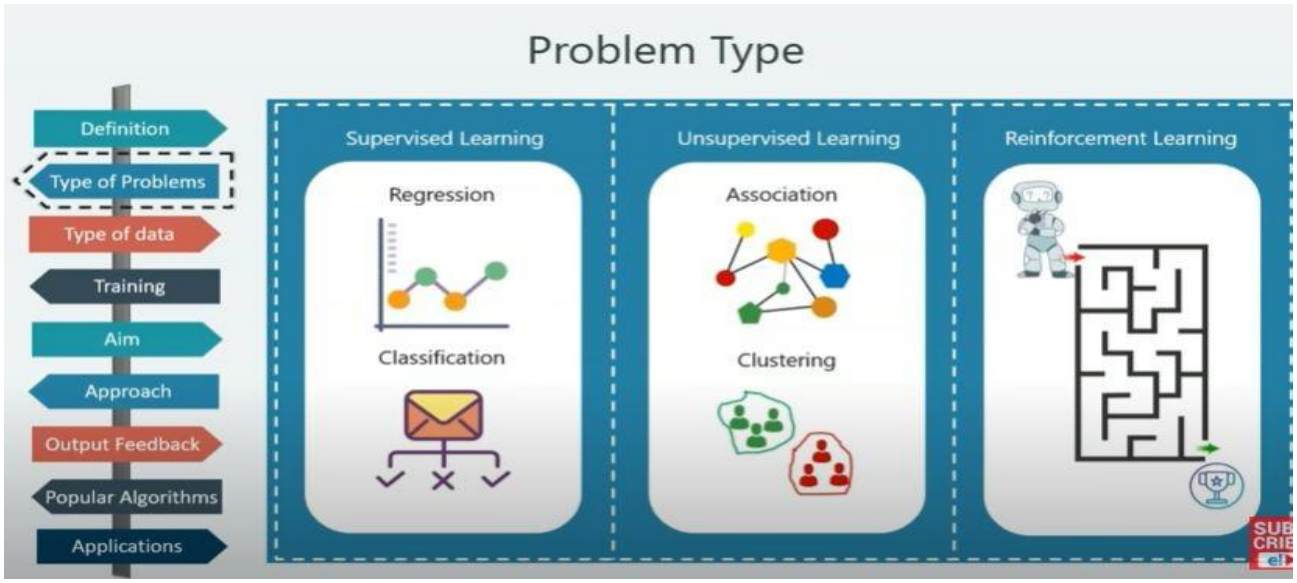
What Is Machine Learning?

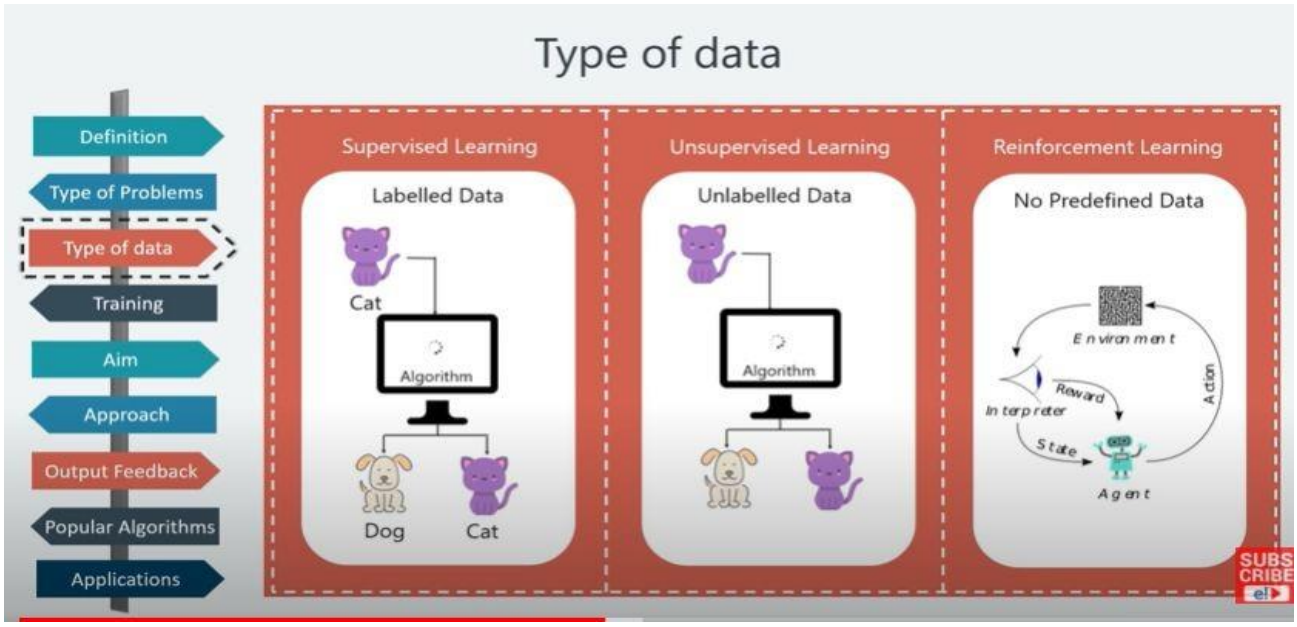
Machine learning is a subset of artificial intelligence (AI) which provides machines the ability to learn automatically & improve from experience without being explicitly programmed.



Definition

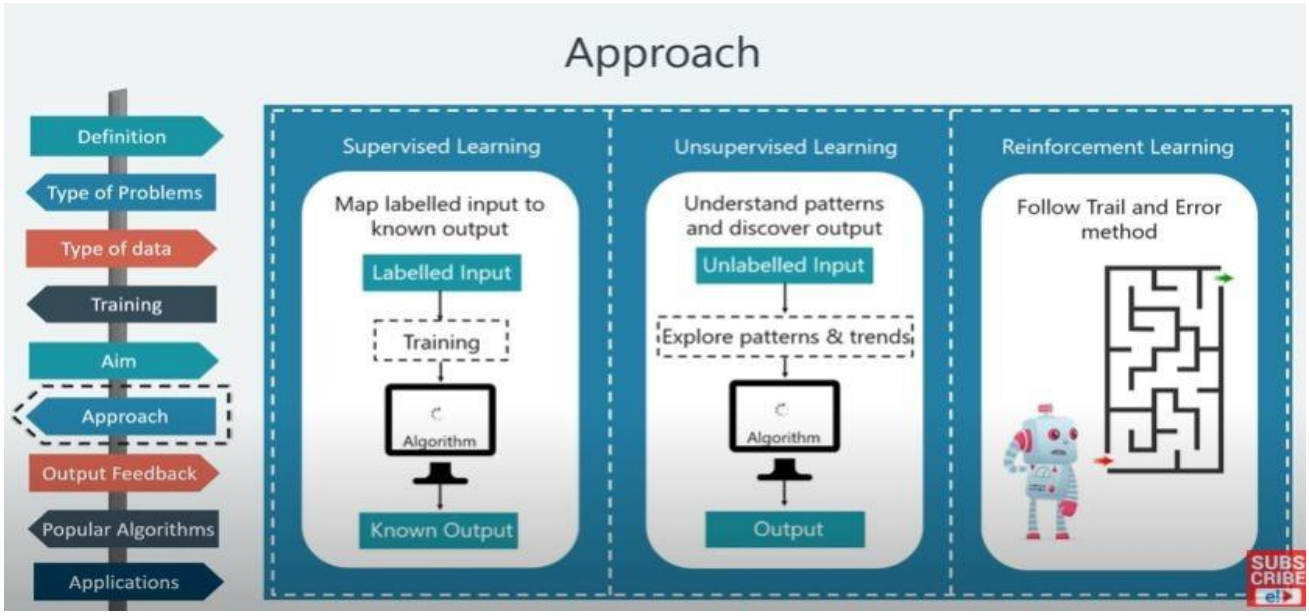


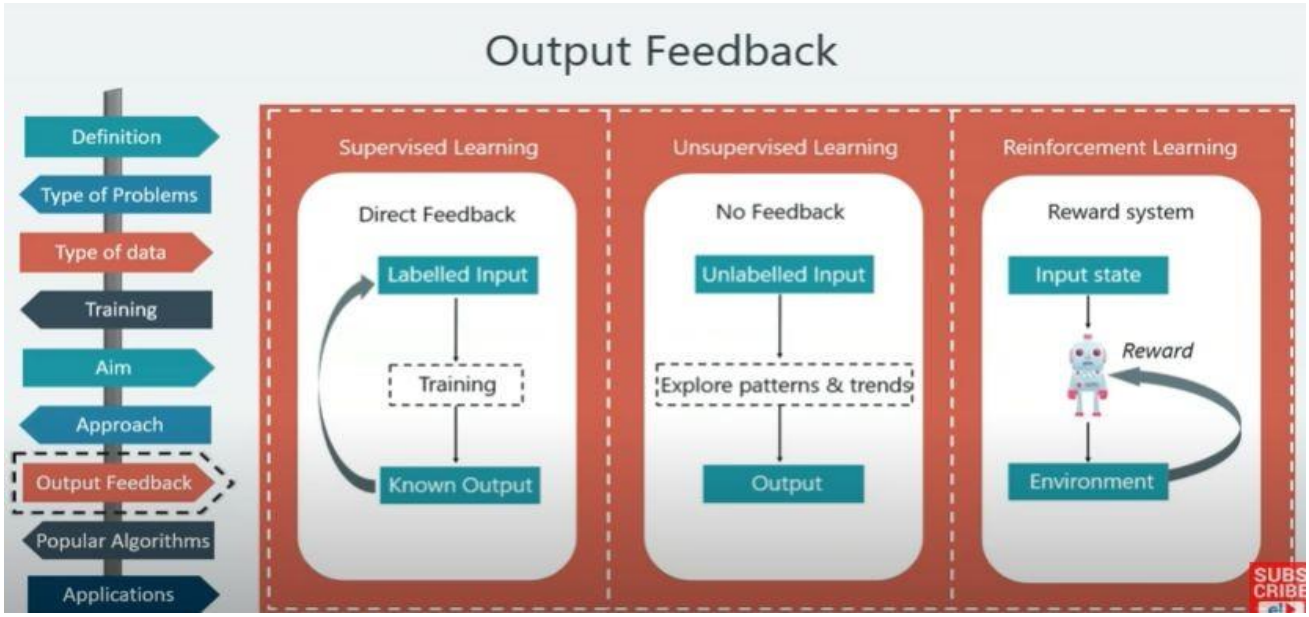

Problem Type
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Regression
Classification
Unsupervised Learning
Association
Clustering
Reinforcement Learning

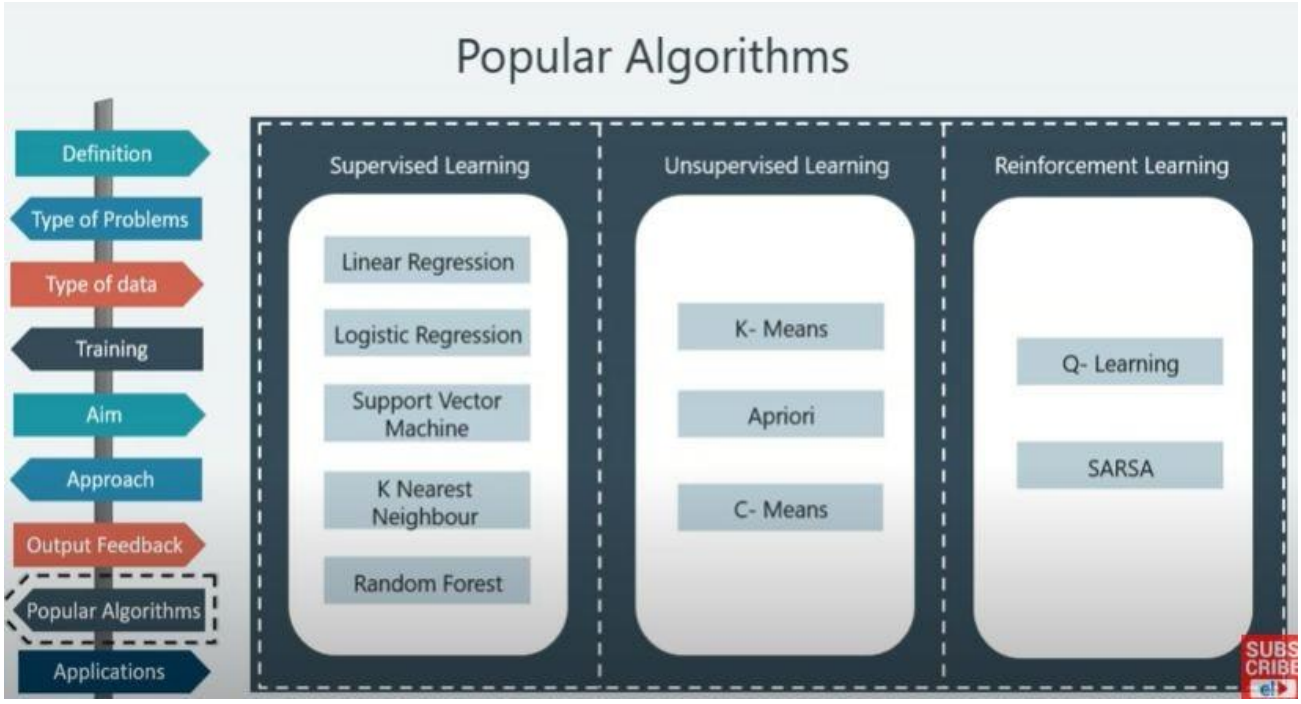

Type of data
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Labelled Data
Cat
Algorithm
Dog
Cat
Unsupervised Learning
Unlabelled Data
Algorithm
Reinforcement Learning
No Predefined Data
Environment
Reward
Interpreter
State
Action
Agent


Training
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
External supervision
Unsupervised Learning
No supervision
Reinforcement Learning
No supervision


Aim
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Forecast outcomes
Unsupervised Learning
Discover underlying patterns
Reinforcement Learning
Learn series of action
SUBSCRIBE

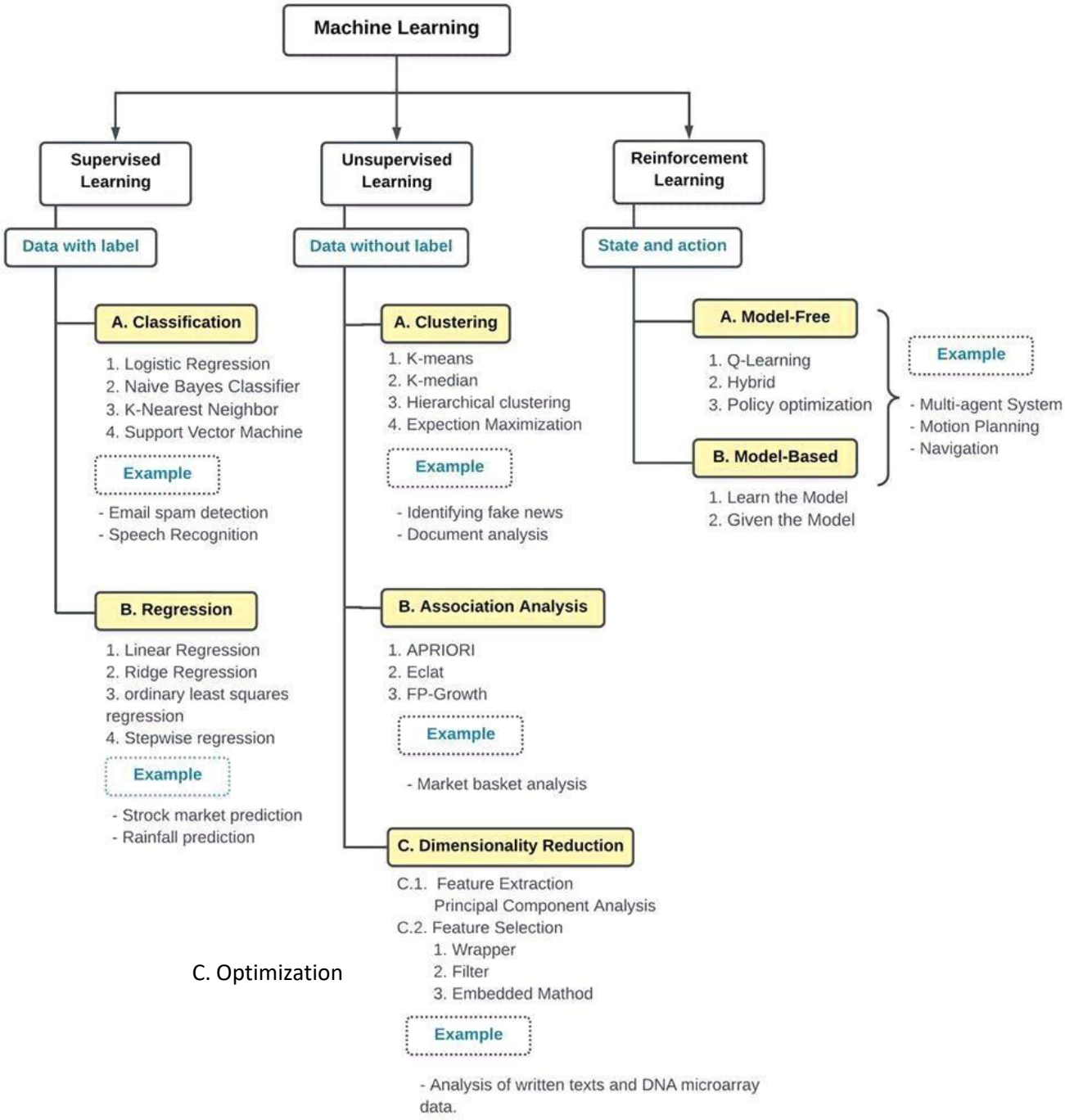

Approach
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Map labelled input to known output
Labelled Input
Training
Algorithm
Known Output
Unsupervised Learning
Understand patterns and discover output
Unlabelled Input
Explore patterns & trends
Algorithm
Output
Reinforcement Learning
Follow Trail and Error method
SUBSCRIBE


Output Feedback
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Direct Feedback
Labelled Input
Training
Known Output
Unsupervised Learning
No Feedback
Unlabelled Input
Explore patterns & trends
Output
Reinforcement Learning
Reward system
Input state
Reward
Environment
SUBSCRIBE


Popular Algorithms
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Linear Regression
Logistic Regression
Support Vector Machine
K Nearest Neighbour
Random Forest
Unsupervised Learning
K- Means
Apriori
C- Means
Reinforcement Learning
Q- Learning
SARSA


Applications
Definition
Type of Problems
Type of data
Training
Aim
Approach
Output Feedback
Popular Algorithms
Applications
Supervised Learning
Risk Evaluation
Forecast Sales
Unsupervised Learning
Recommendation Systems
Anomaly Detection
Reinforcement Learning
Self driving cars
Gaming

Types of Machine Learning

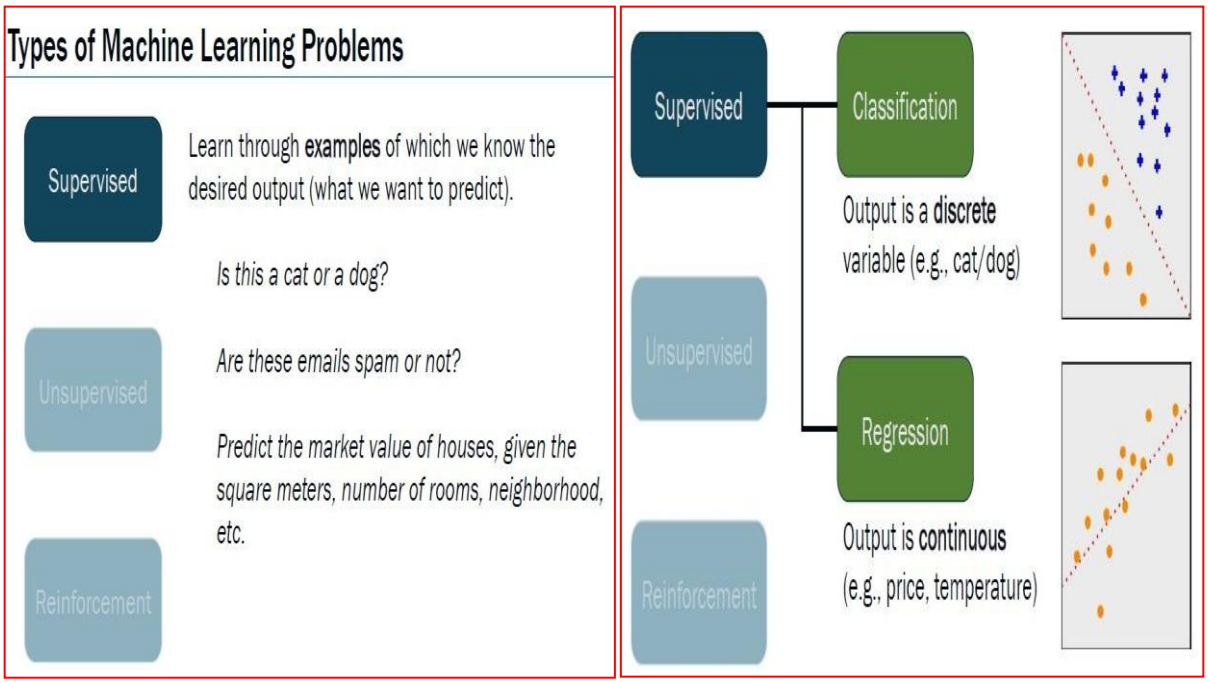


C. Optimization

D. Recommender System

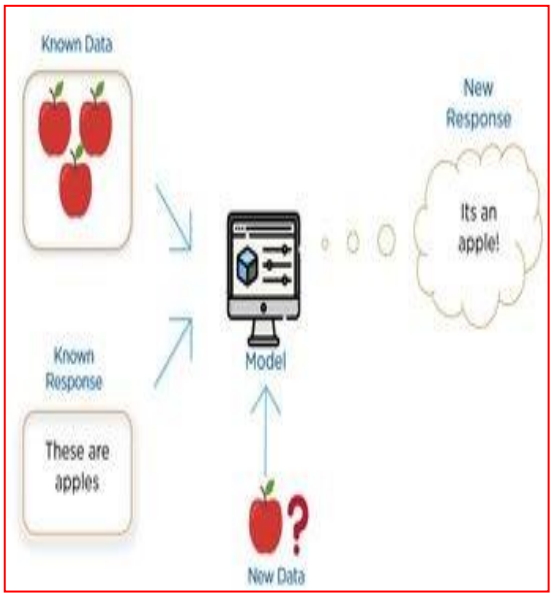
E. Feature Analysis

F. Sentiment Analysis

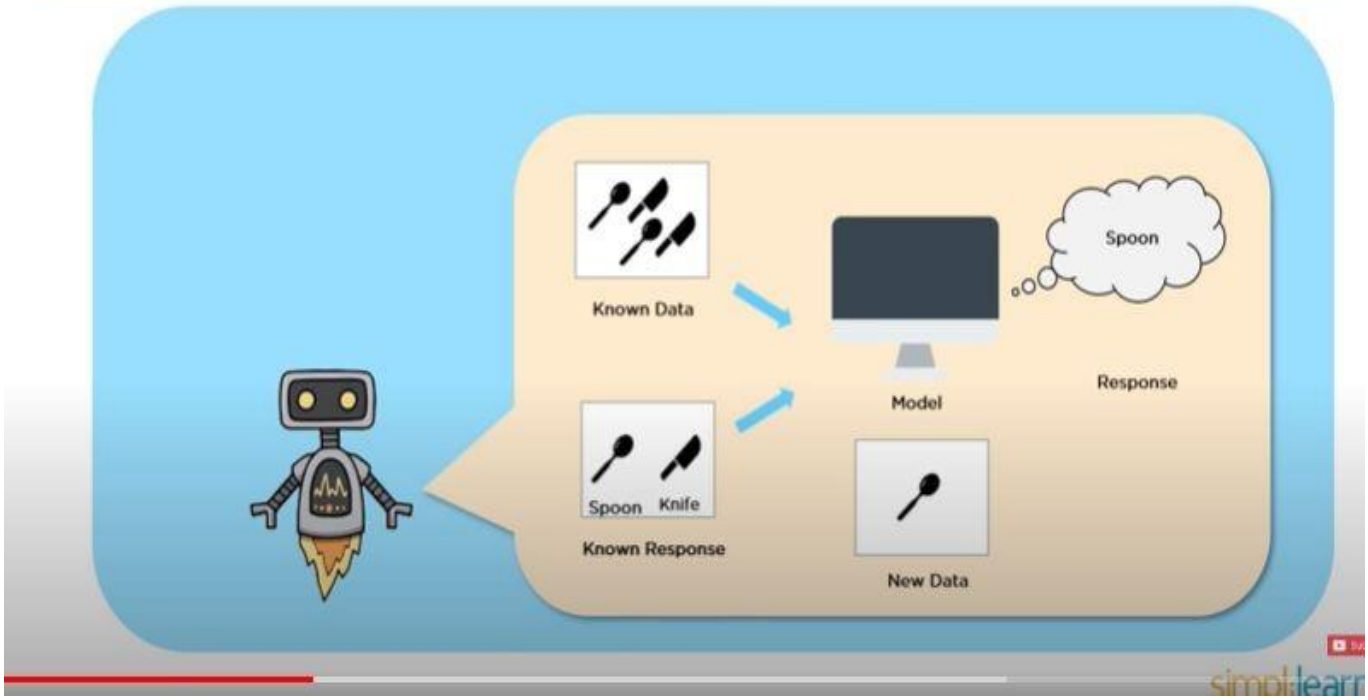


Supervised learning

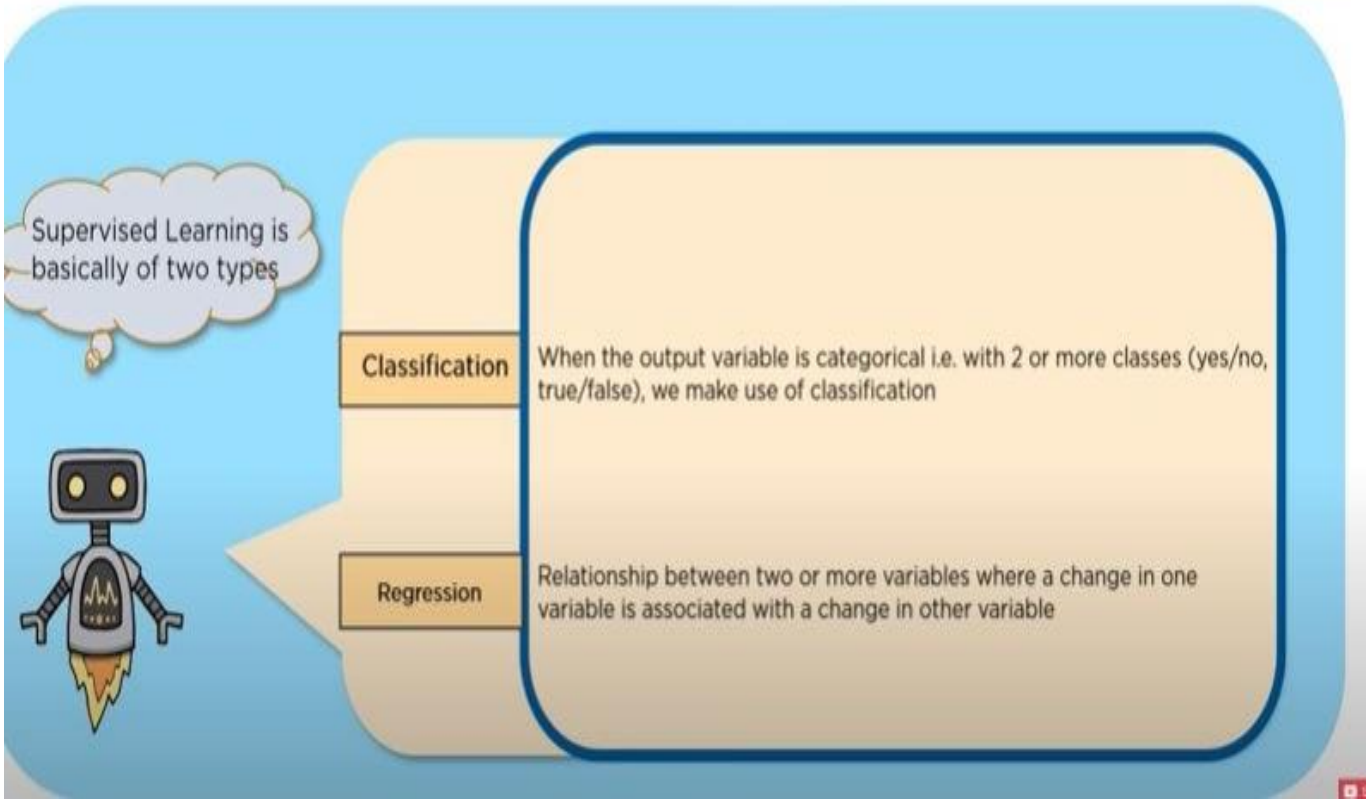
It is very similar to teaching a child with the use of flash cards



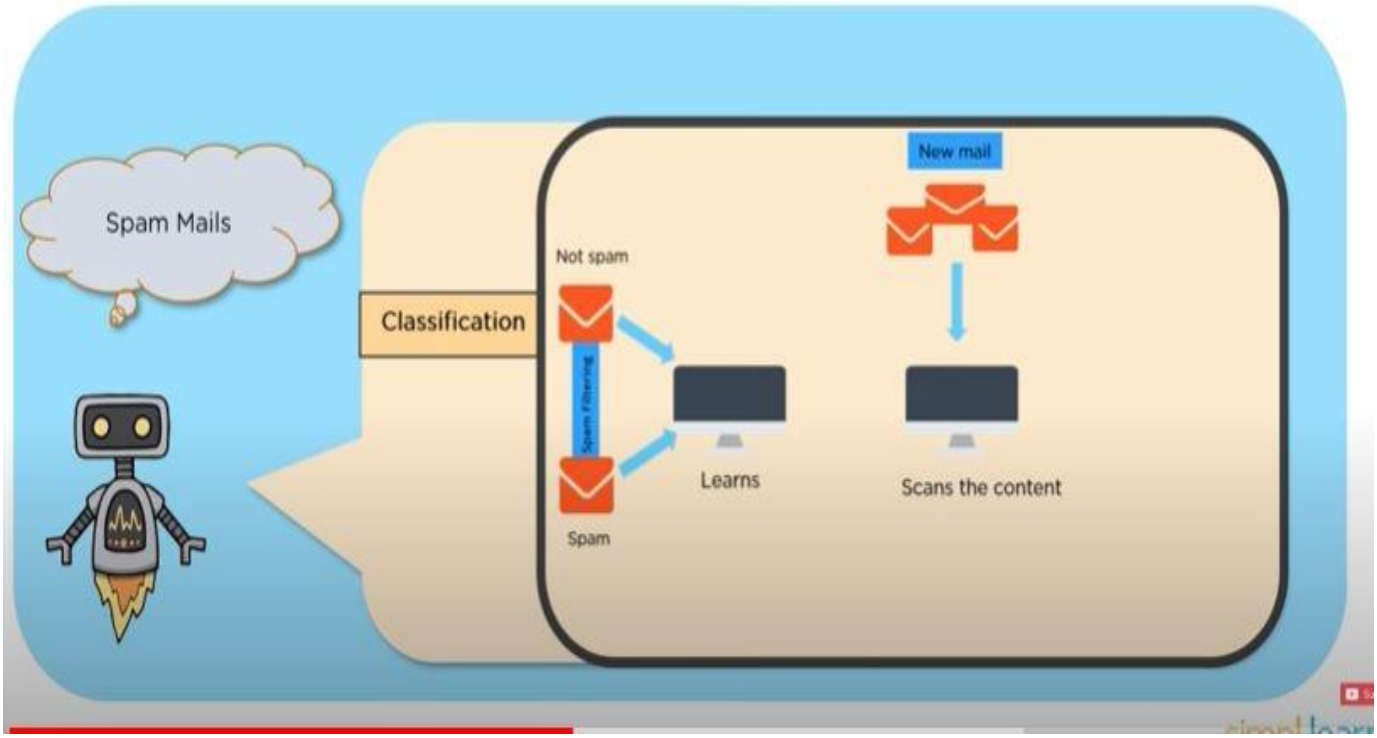
Supervised Learning



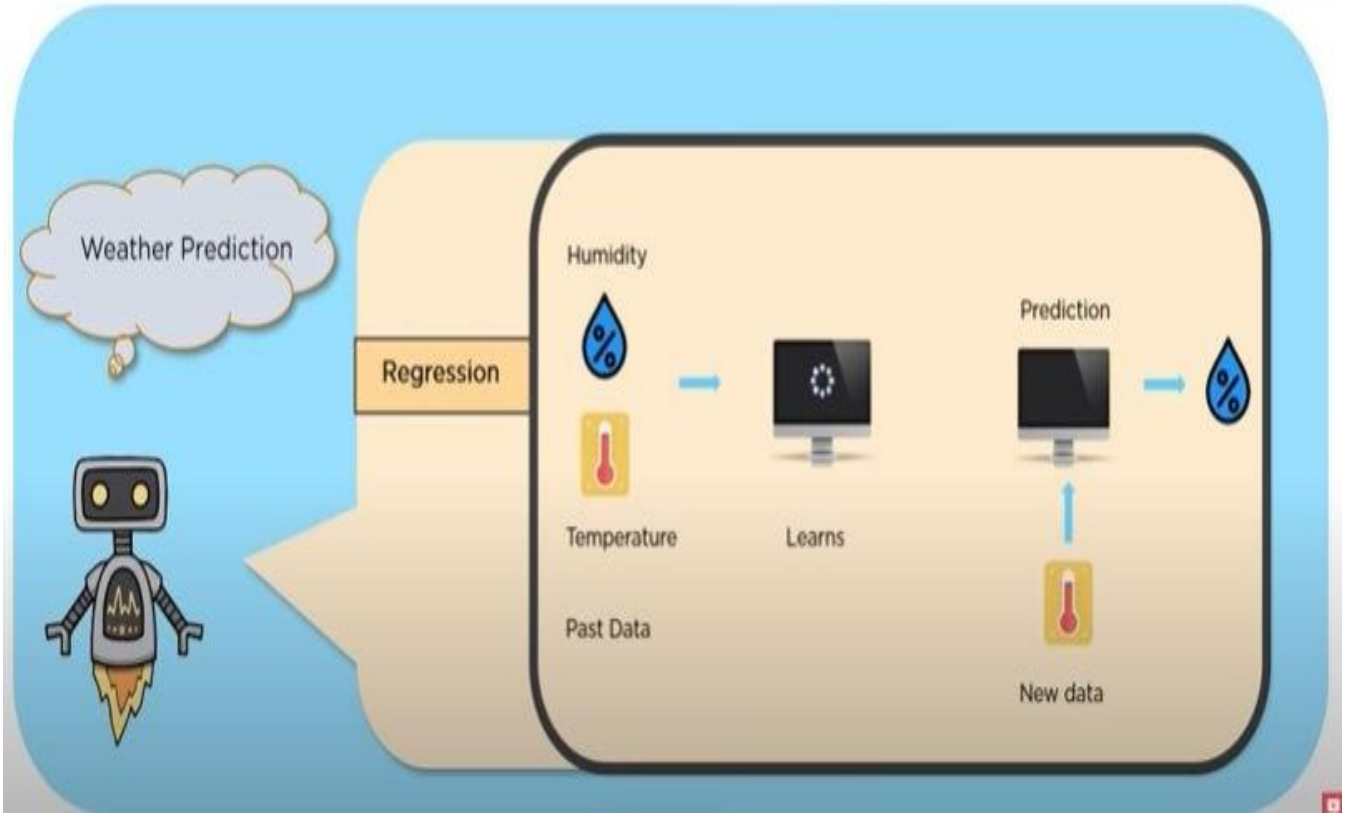
Types of Supervised Learning



Types of Supervised Learning

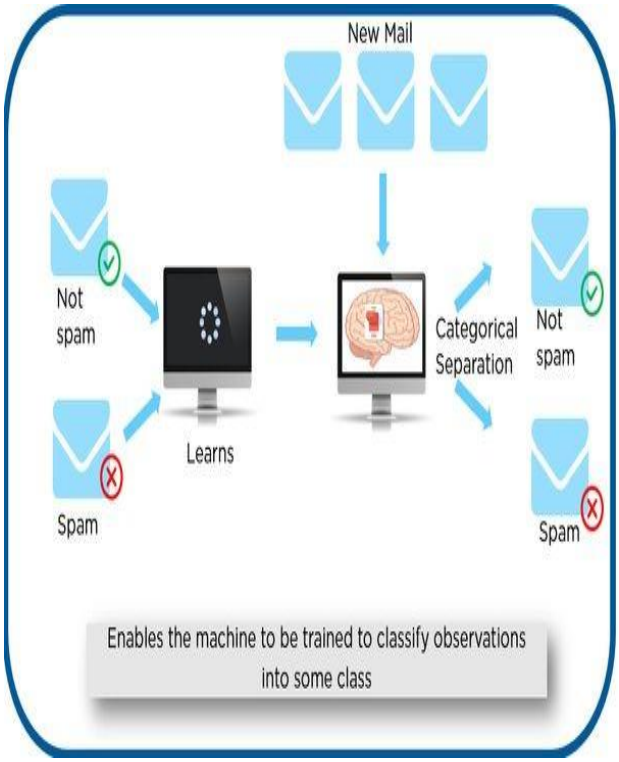


Types of Supervised Learning



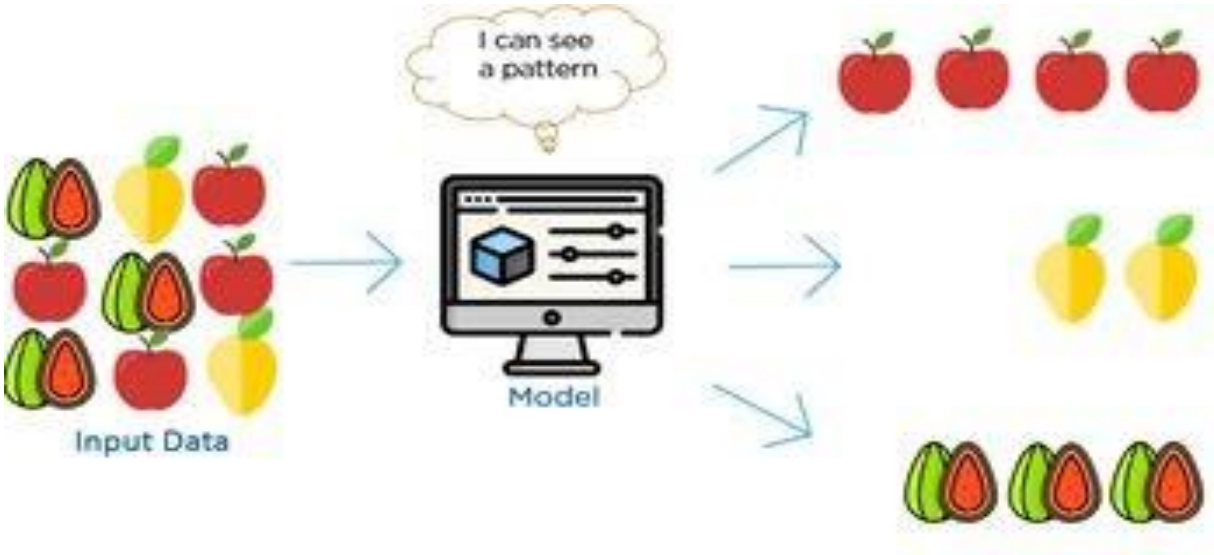
Application of supervised learning

- Advertisement Popularity: Selecting advertisements - Many of the ads - internet are placed because a learning algorithm.
- Spam Classification: If you use a modern email system, chances are you've encountered a spam filter. That spam filter is a supervised learning system. Fed email examples and labels (spam/not spam), these systems learn how to preemptively filter out malicious emails so that their user is not harassed by them. Many of these also behave in such a way that a user can provide new labels to the system and it can learn user preference.
- Face Recognition: Do you use Facebook? Most likely your face has been used in a supervised learning algorithm that is trained to recognize your face. Having a system that takes a photo, findsfaces, and guesses who that is in the photo (suggesting a tag) is a supervised process. It has multiple layers to it, finding faces and then identifying them, but is still supervised nonetheless.



unsupervised learning

- Unsupervised learning is very much the opposite of supervised learning. It featuresno labels. Instead, our algorithm would be fed a lot of data and given the tools to understand the properties of the data. Fromthere, it can learn to group, cluster, and/or organize the data in a way such that a human (or other intelligent algorithm) can come in and make sense of the newly organized data.



Supervised

Unsupervised

Reinforcement

There is **no *desired* output**. Learn something about the data. *Latent* relationships.

I have photos and want to put them in 20 groups.

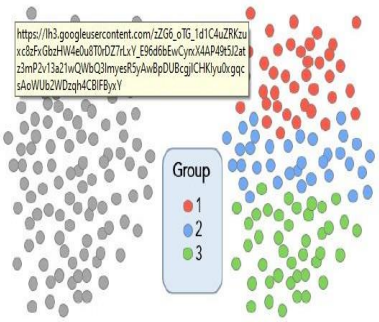
I want to find anomalies in the credit card usage patterns of my customers.

Supervised

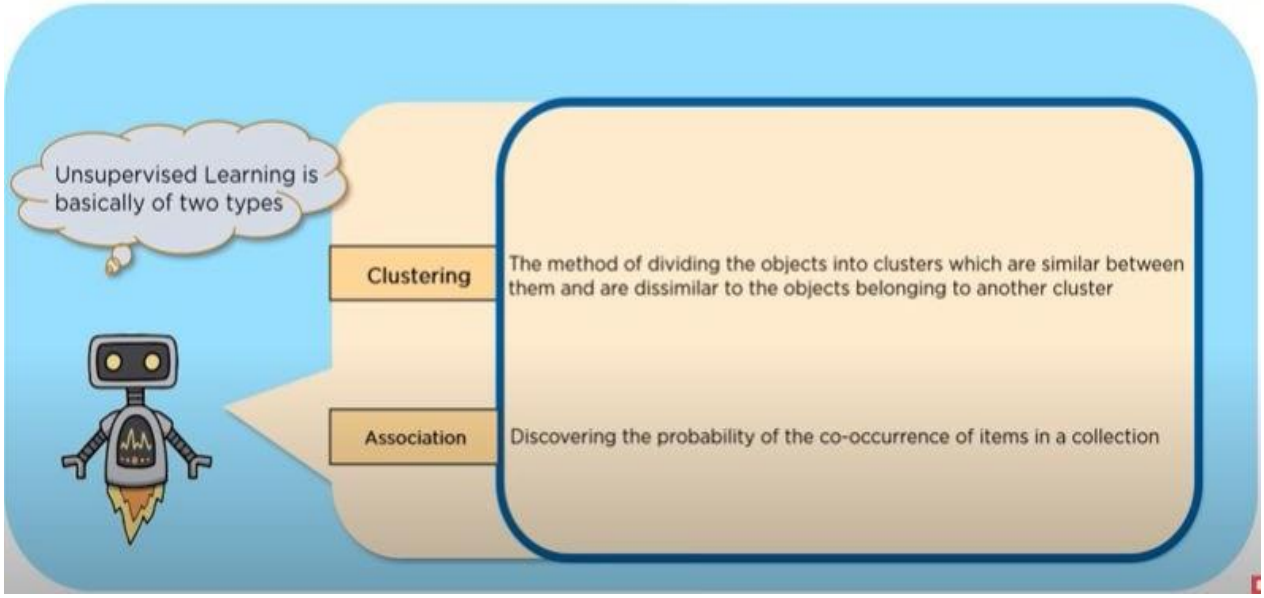
Unsupervised

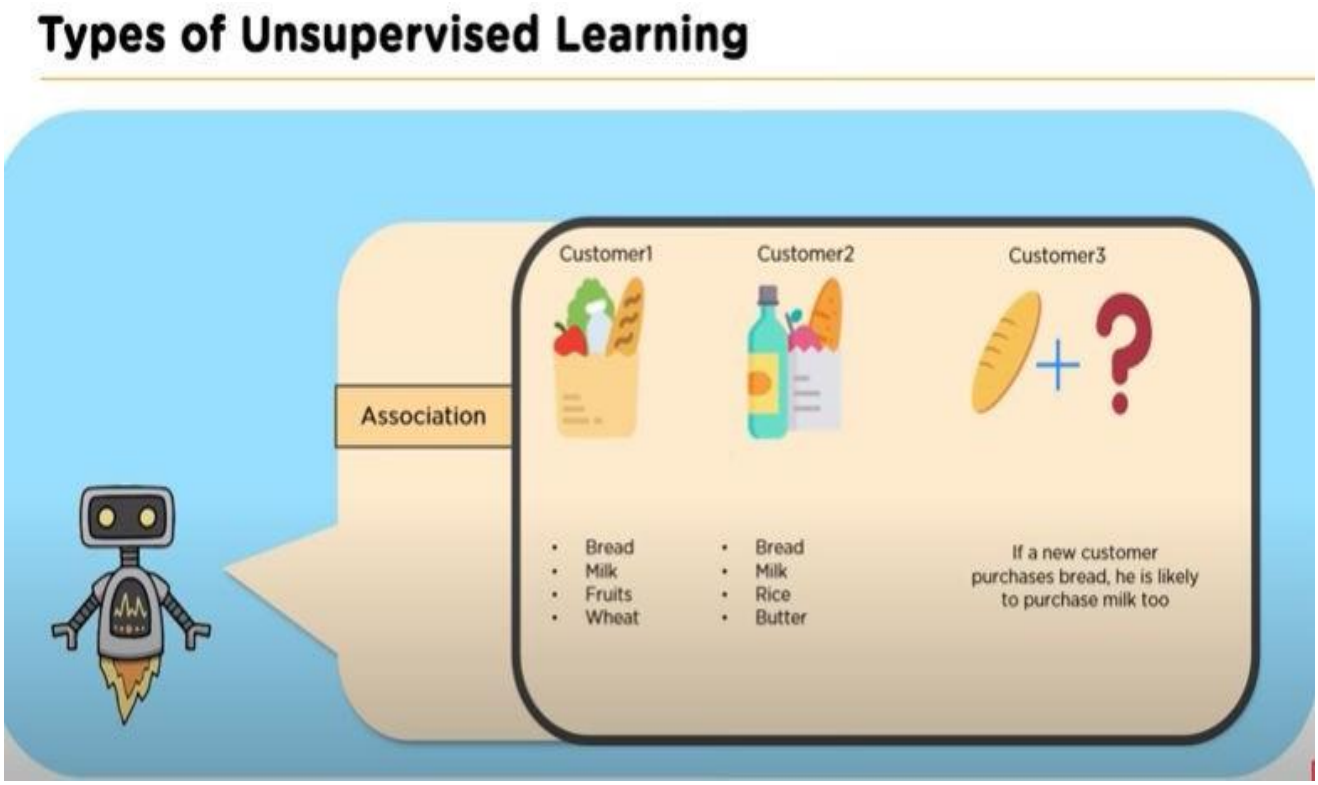
Reinforcement

Useful for learning structure in the data (**clustering**), hidden correlations, reduce dimensionality, etc.



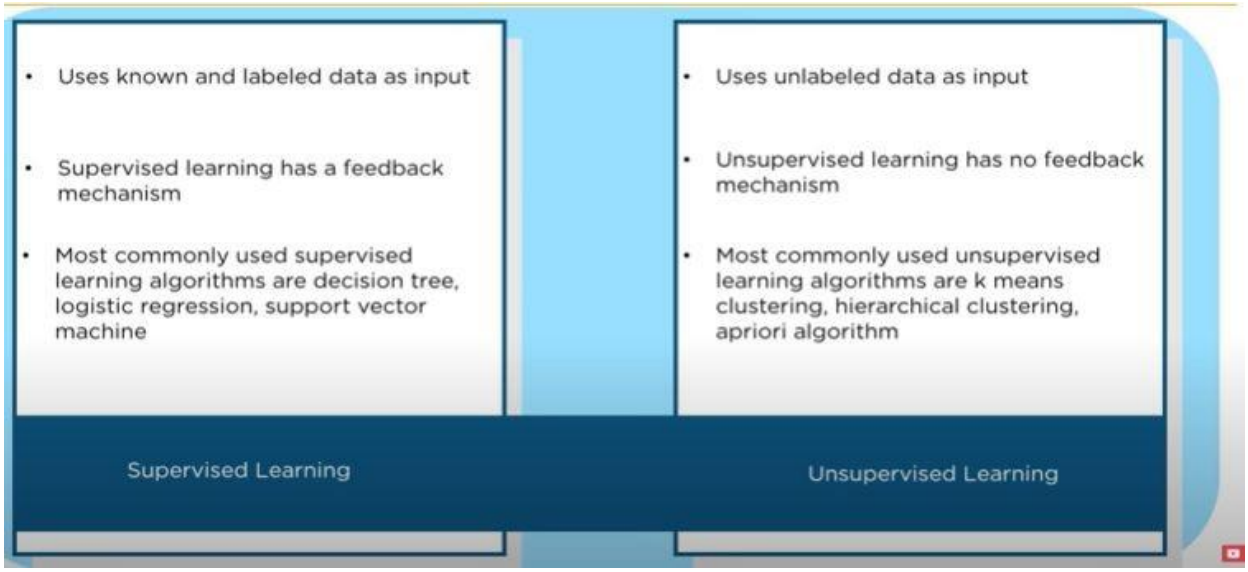
Types of Unsupervised Learning





Application of unsupervised learning

- Recommender Systems: If you've ever used YouTube or Netflix, you've most likely encountered a video recommendation system. These systems are often times placed in the unsupervised domain. We know things about videos, maybe their length, their genre, etc. We also know the watch history of many users. Considering usersthat have watched similar videos as you and then enjoyed other videos that you have yet to see, a recommender system can see this relationship in the data and prompt you with such a suggestion.
- Buying Habits-Market based analysis: Buying habits -contained in a database somewhere and that data is being bought and sold actively at this time. These buying habits can be used in unsupervised learning algorithms to group customers into similar purchasing segments. This helps companies' market to these grouped segments and can even resemble recommender systems.
- Grouping User Logs-semantic clustering: Less user facing, but still very relevant, we can use unsupervised learningto group user logs and issues. This can help companies identify central themes to issues their customers face and rectify these issues, through improving a product or designing an FAQ to handle common issues.



Reinforcement of learning

- Reinforcement learning as learning from mistakes. Placea reinforcement learning algorithm into any environmentand it will make a lotof mistakes in the beginning.

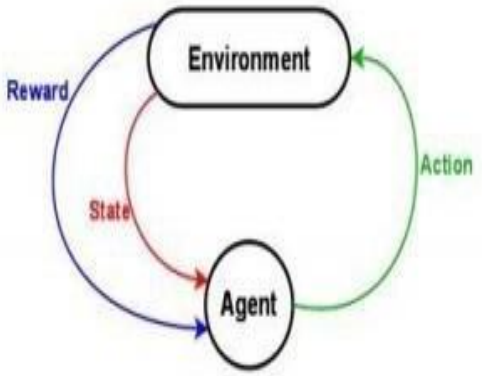
Supervised

Unsupervised

Reinforcement

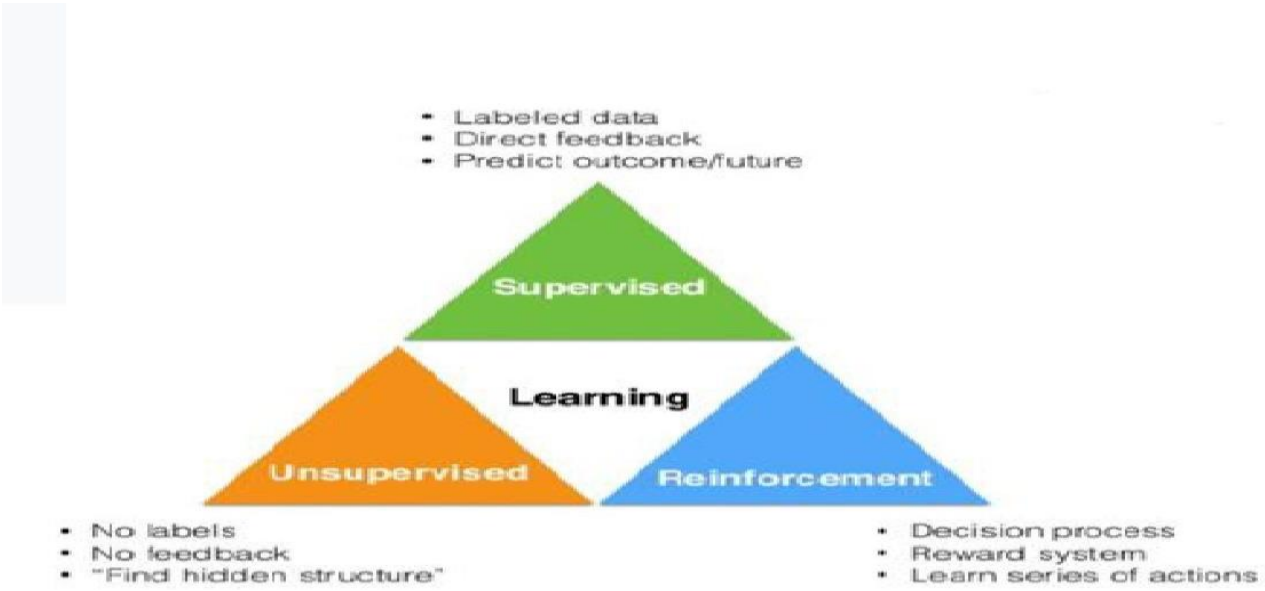
An agent **interacts** with an **environment** and watches the result of the interaction.

Environment gives feedback via a positive or negative **reward signal**.

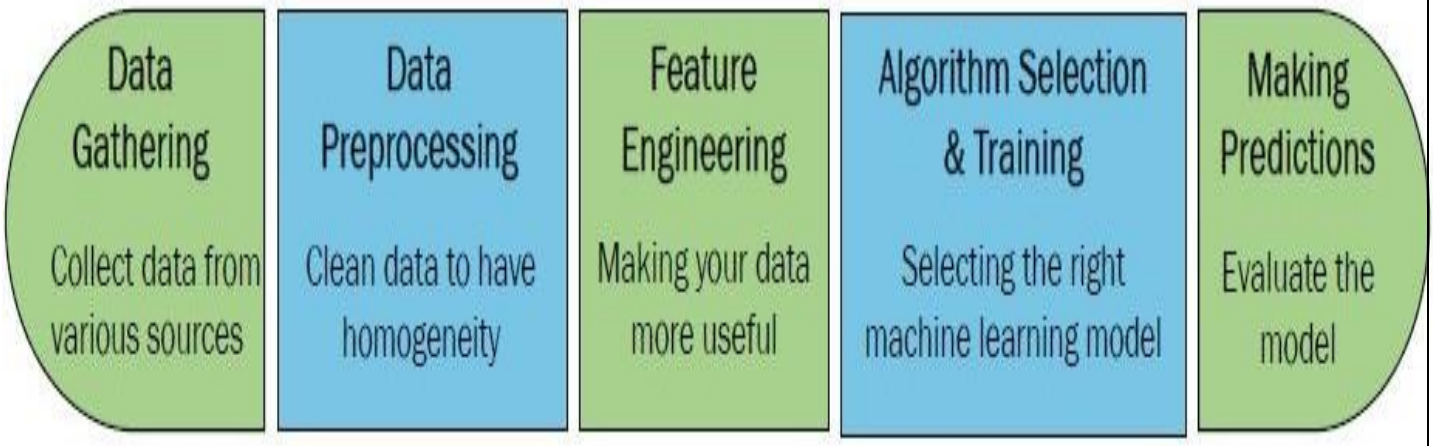


Application of Reinforcement learning

- **Video Games**: Google's reinforcement learning application, AlphaZero and AlphaGo which learned to play the game Go. Our Mario example is also a common example
- **Industrial Simulation**: For many robotic applications (think assemblylines), it is useful to have our machines learn to complete their tasks without having to hardcode their processes.
- **Resource Management**: Reinforcement learning is good for navigating complex environments. It can handle the need to balance certain requirements. For example, Google's data centers.
-



Steps to solve a Machine Learning Problem



Data Gathering

Might depend on human work

- Manual labeling for supervised learning.
- Domain knowledge. Maybe even experts.

May come for free, or "sort of"

- E.g., Machine Translation.

The more the better: Some algorithms need large amounts of data to be useful (e.g., neural networks).

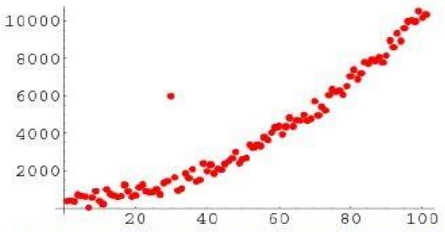
The **quantity** and **quality** of data dictate the model **accuracy**

Data Preprocessing

Is there anything **wrong** with the data?

- Missing values
- Outliers
- Bad encoding (for text)
- Wrongly-labeled examples
- Biased data
  - Do I have many more samples of one class than the rest?

Need to fix/remove data?



Feature Engineering

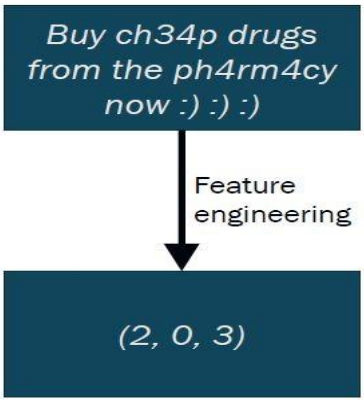
What is a feature?

A feature is an individual measurable property of a phenomenon being observed

Our inputs are represented by a **set of features**.

To classify spam email, features could be:

- Number of words that have been *ch4ng3d* like this.
- Language of the email (0=English, 1=Spanish)
- Number of emojis



Feature Engineering

Extract **more** information from **existing** data, not adding "new" data per-se

- Making it more **useful**
- With good features, most algorithms can learn **faster**

It can be an art

- Requires thought and knowledge of the data

Two steps:

- Variable transformation (e.g., dates into weekdays, normalizing)
- Feature creation (e.g., n-grams for texts, if word is capitalized to detect names, etc.)

Algorithm Selection & Training

Supervised

- Linear classifier
- Naive Bayes
- Support Vector Machines (SVM)
- Decision Tree
- Random Forests
- k-Nearest Neighbors
- **Neural Networks (Deep learning)**

Unsupervised

- PCA
- t-SNE
- k-means
- DBSCAN

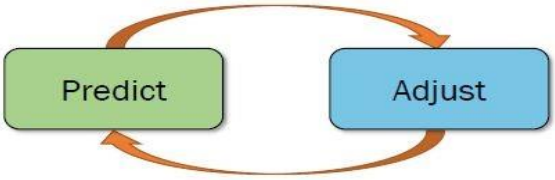
Reinforcement

- SARSA–λ
- Q-Learning

Algorithm Selection & Training

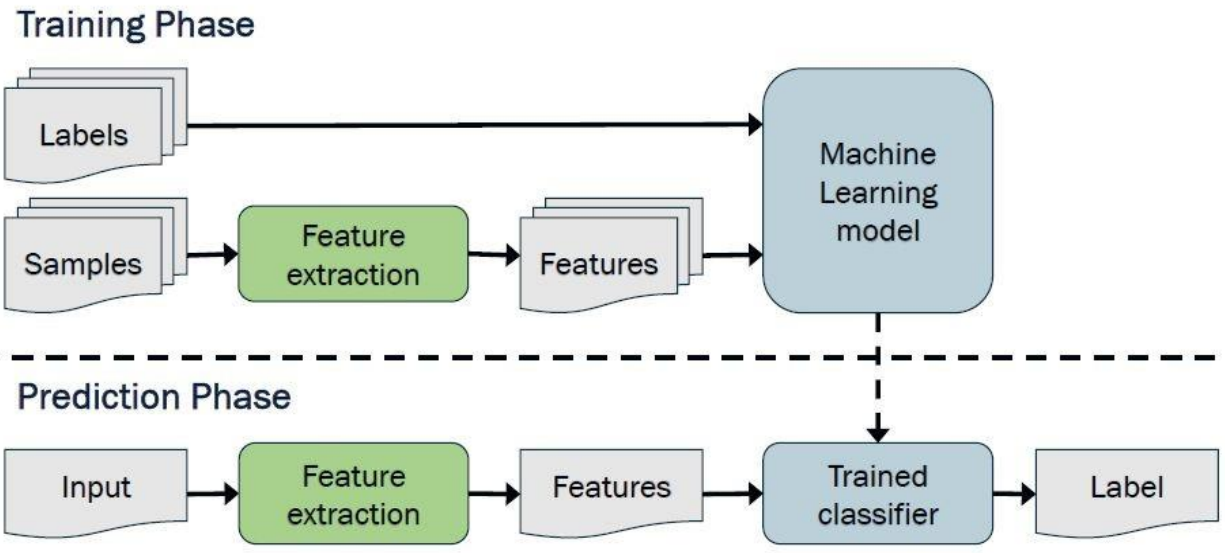
Goal of training: making the correct prediction as often as possible

- Incremental improvement:



- Use of metrics for **evaluating** performance and comparing solutions
- **Hyperparameter tuning**: more an art than a science

Making Predictions



Basic design issues and approaches to machine learning, let us consider designing a program to learn to playcheckers, with the goal of entering it in the world checkers tournament.

Performance measure: the percent of games it wins in this world tournament.

1.2.1 Choosing the Training Experience

The first design choice is to choose the type of training experience from which our system will learn. The type oftraining experience available can have a significant impact on success or failure of the learner. (driving class to drive a car)

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

Whether the training experience provides direct or indirect feedback regarding the choices made by theperformance system.

The degree to which the learner controls the sequence of training examples

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

Example : Chess problem

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

For example in checkers game

- In learning to play checkers, the system might learn from *direct* training examples consisting of individualcheckers board states and the correct move for each.
- *Indirect* training examples consisting of the move sequences and final outcomes of various games played.
- The information about the correctness of specific moves early in the game must be inferred indirectly from thefact that the game was eventually won or lost.
- Here the learner faces an additional problem of *credit assignment,* or determining the degree to which eachmove in the sequence deserves credit or blame for the final outcome.
- Credit assignment can be a particularly difficult problem because the game can be lost even when early movesare optimal, if these are followed later by poor moves
- Hence learning from direct training feedback is typically easier than learning from indirect feedback

2. A second important attribute of the training experience ***is the degree to which the learner controls the sequence of training examples***

For example, in checkers game:

- The learner might depends on the teacher to select informative board states and to provide the correct move for each.
- Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
- The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.
- Notice in this last case the learner may choose between experimenting with novel board states that it has not yet considered, or honing its skill by playing minor variations of lines of play it currently finds most promising.

3. A third attribute of the training experience is how well it represents ***the distribution of examples*** over which the final system performance P must be measured.

Learning is most reliable when the training examples follow a distribution similar to that of future test examples.

For example, in checkers game:

- In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
- If its training experience E consists only of games played against itself, there is an danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
- It is necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated. Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

A checkers learning problem:

Task T: playing checkers

Performance measure ***P:*** percent of games won in the world tournament

Training experience E: games played against itself

In order to complete the design of the learning system, we must now choose

1. the exact type of knowledge to be, learned
2. a representation for this target knowledge
3. a learning mechanism

Choosing the Target Function

- The next design choice is to determine exactly what type of knowledge will belearned and how this will be used by the performance program. Let us begin with a checkers-playing program that can generate the *legal* moves fromany board state.
- The program needs only to learn how to choose the *best* move from among theselegal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.
- Many optimization problems fall into this class, such as the problems of scheduling and controlling manufacturing processes where the available manufacturing steps are well understood, but the best strategy for sequencing them isnot.

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

Let the target function V and the notation

$$V : B \rightarrow R$$

which denote that V maps any legal board state from the set B to some real value

We intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

Let us therefore define the target value ***V(b)*** for an arbitrary board state ***b*** in

B, ***as*** follows:

1. if ***b*** is a final board state that is won, then ***V(b)*** = 100
2. if b is a final board state that is lost, then ***V(b)*** = -100
3. if b is a final board state that is drawn, then ***V(b)*** = ***0***
4. if b is a not a final state in the game, then V(b) = V(b'), where b' is the bestfinal board state that can be achieved starting from b and playing **optimally** until the end of the game

1.2.2 Choosing a Representation for the Target Function

let us choose a simple representation: for any given board state, the function c will be calculated as **alinear combination** of the following board features:

- ***xl:*** the number of black pieces on the board
- ***x2:*** the number of red pieces on the board
- ***x3:*** the number of black kings on the board
- **x4:** the number of red kings on the board
- ***x5:*** the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- ***X6:*** the number of red pieces threatened by black

Thus, our learning program will represent v(b) as a linear function of the form

where ***wo*** through w6 are numerical coefficients, or weights, to be chosen by the learning algorithm. Learned values for the weights ***wl*** through ***W6*** will determine the relative importance of the various boardfeatures in determining the value of the board, whereas the weight ***wo*** will provide an additive constant tothe board value

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

1.2.3 Choosing a Function Approximation Algorithm

- In order to learn the target function **v** we require a set of training examples, eachdescribing a specific board state b and the training value Vtrain(b) for b.
- In other words, each training example is an ordered pair of the form (b, V',,,i,(b)).
- For instance, the following training example describes a board state b in which black
- Function Approximation Procedure

1 Derive training examples from the indirect training experience available to thelearner

2 Adjust the weights w_i to best fit these training examples- reduce error

- In order to learn the target function *f* we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.
- Each training example is an ordered pair of the form $(b, V_{train}(b))$.
- For instance, the following training example d escribes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

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

1.2.4.1 ESTIMATING TRAINING VALUES

□ assign the training value of Vtrain(b) for any intermediate board state b to be V(successor(b), where v is the learner's current approximation to Vand where Successor(b) denotes the next board state following b

(what will be opponent move)

Rule for estimating training values.

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

Where ,

$\hat{v}$ is the learner's current approximation to V

Successor(b) denotes the next board state following b for which it is again the program's turn to move

1.2.4.2 ADJUSTING THE WEIGHTS

- All that remains is to specify the learning algorithm for choosing theweights wi to best fit the set of training examples {(b, Vtrain(b))}.
- As a first step we must define what we mean by the best fit to the trainingdata.
- One common approach is to define the best hypothesis, or set of weights, asthat which minimizes the squared error E between the training values and the values predicted by the hypothesis V. If error =0.2 then squared 0.04.it reduces error

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

• One such algorithm is called the least mean squares, or LMS training rule. For each observed training example, it adjusts the weights a small amount inthe direction that reduces the error on this training example.

LMS weight update rule.

For each training example $\langle b, V_{train}(b) \rangle$

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

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

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

Working of weight update rule

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

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update. To get an intuitive understanding for why this weight update rule works, notice that when the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed. When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.

1.2.4 The Final Design

- The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.
- These four modules, summarized in Figure 1.1, are as follows:
- The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned targetfunction(s). It takes an instance of a new problem (new game) as input andproduces a trace of its solution (game history) as output
- The Critic takes as input the history or trace of the game and produces asoutput a set of training examples of the target function
- The Generalizer takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples. In our example, the Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function f described by the learned weights wo, . . . , W6.
- The Experiment Generator takes as input the current hypothesis (currently learnedfunction) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize thelearning rate of the overall system. In our example, the Experiment Generator follows a very simple strategy: It always proposes the same initial game board to begin a new game

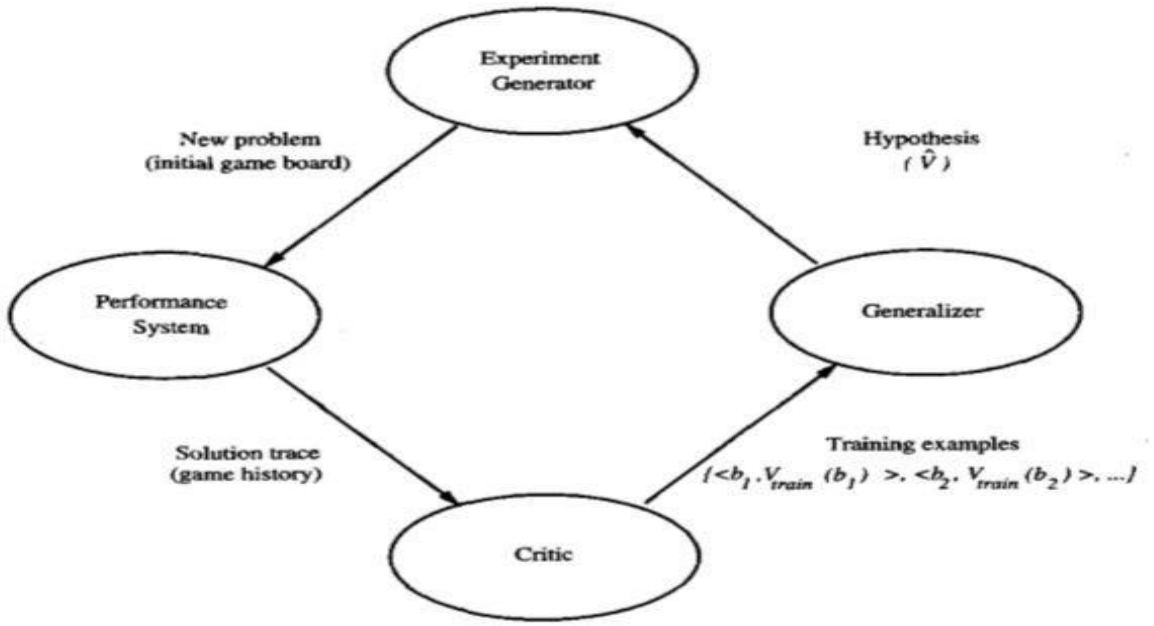


FIGURE 1.1
Final design of the checkers learning program.

Perspective &Issues in Machine Learning Perspective:

It involves searching a very large space of possible hypothesis to determine the onethat best fits the observed data.

Issues:

- Which algorithm performs best for which types of problems & representation?
- How much training data is sufficient?
- Can prior knowledge be helpful even when it is only approximately correct?
- The best strategy for choosing a useful next training experience.
- What specific function should the system attempt to learn?
- How can learner automatically alter it's representation to improve it's ability torepresent and learn the target function?

Concept Learning

- Inducing general functions from specific training examples is a main issue of machine learning.
- Concept Learning: Acquiring the definition of a general category from given sample positive and negative training examples of the category.
- Concept Learning can see as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.
- The hypothesis space has a general-to-specific ordering of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space.

- A Formal Definition for Concept Learning:

Inferring a Boolean-valued function from training examples of its input and output.

- An example for concept-learning is the learning of bird-concept from the given examples of birds (positive examples) and non-birds (negative examples).
- We are trying to learn the definition of a concept from given examples.

A Concept Learning Task – Enjoy SportTraining Examples

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
1	Sunny	Warm	Normal	Strong	Warm	Same	**YES**
2	Sunny	Warm	High	Strong	Warm	Same	**YES**
3	Rainy	Cold	High	Strong	Warm	Change	**NO**
4	Sunny	Warm	High	Strong	Warm	Change	**YES**

ATTRIBUTES **CONCEPT**

- A set of example days, and each is described by six attributes.
- The task is to learn to predict the value of Enjoy-Sport for arbitrary day, based on the values of its attribute values.

Enjoy-Sport – Hypothesis Representation

- Each hypothesis consists of a conjunction of constraints on the instance attributes.
- Each hypothesis will be a vector of six constraints, specifying the values of the six attributes

(Sky, Air-Temp, Humidity, Wind, Water, and Forecast).

- Each attribute will be:

? - indicating any value is acceptable for the attribute (don't care) single value – specifying a single required value (ex. Warm) (specific)

0 - indicating no value is acceptable for the attribute (no value)

Hypothesis Representation

- A hypothesis:

Sky AirTemp Humidity Wind Water Forecast

< Sunny, ? , ? , Strong , ? , Same >

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

<?, ?, ?, ?, ?, ?>

- The most specific hypothesis – that no day is a positive example

<0, 0, 0, 0, 0, 0>

- EnjoySport concept learning task requires learning the sets of days for which EnjoySport=yes, describing this set by a conjunction of constraints over the instance attributes.

EnjoySport Concept Learning Task

Given

Instances X : set of all possible days, each described by the attributes

- Sky – (values: Sunny, Cloudy, Rainy)

- Air-Temp – (values: Warm, Cold)
- Humidity – (values: Normal, High)
- Wind – (values: Strong, Weak)
- Water – (values: Warm, Cold)
- Forecast – (values: Same, Change)

Target Concept (Function) c : Enjoy-Sport : X □ {0,1}

Hypotheses H : Each hypothesis is described by a conjunction of constraints on the attributes.

Training Examples D : positive and negative examples of the target function

Determine

A hypothesis h in H such that h(x) = c(x) for all x in D.

The Inductive Learning Hypothesis

•Although the learning task is to determine a hypothesis h identical to the target concept cover the entire set of instances X, the only information available about c is its value over the training examples.

Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.

Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. This is the fundamental assumption of inductive learning.

•The Inductive Learning Hypothesis - Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

•Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.

•The goal of this search is to find the hypothesis that best fits the training examples.

•By selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.

•Sky has 3 possible values, and other 5 attributes have 2 possible values.

•There are 96 (= 3.2.2.2.2.2) distinct instances in X.

•There are 5120 (=5.4.4.4.4.4) syntactically distinct hypotheses in H.

Two more values for attributes: ? and 0

•Every hypothesis containing one or more 0 symbols represents the empty set of instances; that is, it classifies every instance as negative.

•There are 973 (= 1 + 4.3.3.3.3.3) semantically distinct hypotheses in H.

Only one more value for attributes: ?, and one hypothesis representing empty set of instances.

•Although Enjoy-Sport has small, finite hypothesis space, most learning tasks have much larger (even infinite) hypothesis spaces.

We need efficient search algorithms on the hypothesis spaces.

General-to-Specific Ordering of Hypotheses

Many algorithms for concept learning organize the search through the hypothesis space by relying on a general-to-specific ordering of hypotheses.

By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis. Consider two hypotheses

h1 = (Sunny, ?, ?, Strong, ?, ?)

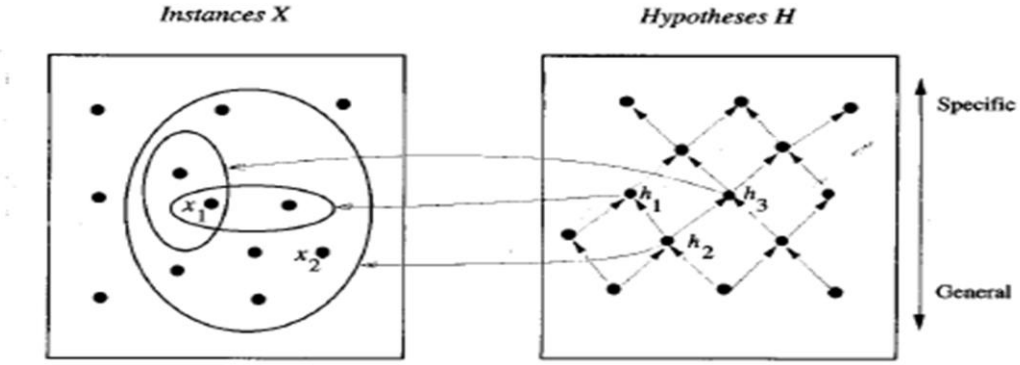
h2 = (Sunny, ?, ?, ?, ?, ?)

Now consider the sets of instances that are classified positive by hl and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is more general than hl.

More-General-Than Relation

For any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1.

More-General-Than-Or-Equal Relation: Let h1 and h2 be two Boolean-valued functions defined over X. Then h1 is more-general-than-or-equal-to h2 (written h1 ≥ h2) if and only if any instance that satisfies h2 also satisfies h1. h1 is more-general-than h2 (h1 > h2) if and only if h1≥h2 is true and h2≥h1 is false. We also say h2 is more-specific-than h1.



h2 > h1 and h2 > h3

But there is no more-general relation between h1 and h3

FIND-S Algorithm

FIND-S Algorithm starts from the most specific hypothesis and generalize it by considering only positive examples. FIND-S algorithm ignores negative examples. As long as the hypothesis space contains a hypothesis that describes the true target concept, and the training data contains no errors, ignoring negative examples does not cause to any problem. FIND-S algorithm finds the most specific hypothesis within H that is consistent with the positive training examples. The final hypothesis will also

be consistent with negative examples if the correct target concept is in H, and the training examples are correct.

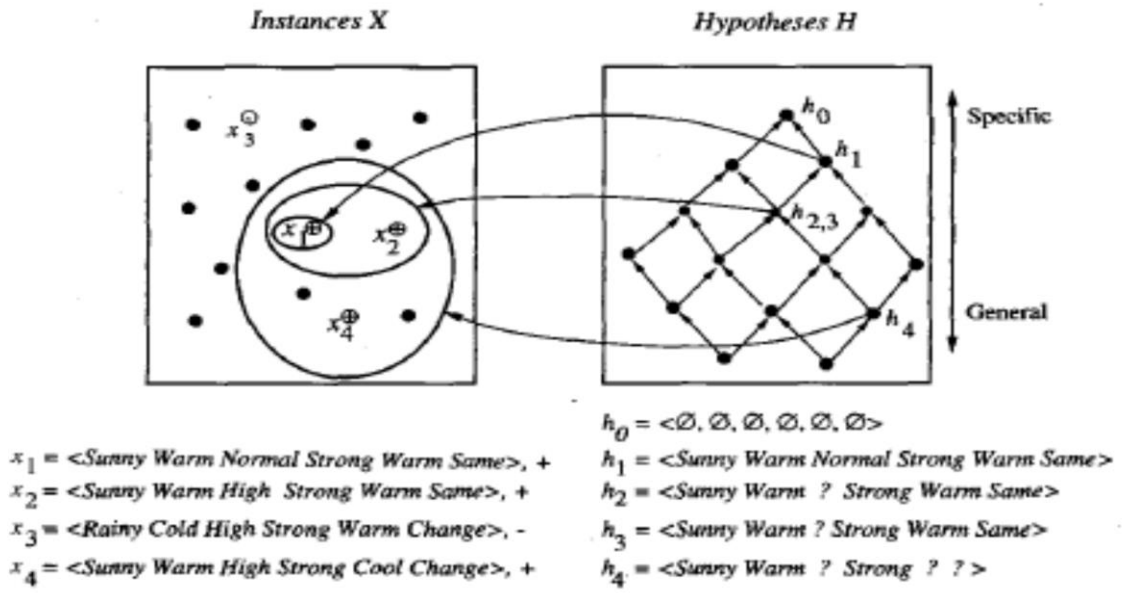
1.Initialize h to the most specific hypothesis in H

2.For each positive training instance x for each attribute constraint a, in h

3.If the constraint a, is satisfied by x Then do nothing

Else replace a, in h by the next more general constraint that is satisfied by x

4. Output hypothesis h



Unanswered Questions by FIND-S Algorithm

Has FIND-S converged to the correct target concept?

Although FIND-S will find a hypothesis consistent with the training data, it has no way to determine whether it has found the only hypothesis in H consistent with the data (i.e., the correct target concept), or whether there are many other consistent hypotheses as well.

We would prefer a learning algorithm that could determine whether it had converged and, if not, at least characterize its uncertainty regarding the true identity of the target concept.

Why prefer the most specific hypothesis?

In case there are multiple hypotheses consistent with the training examples, FIND-S will find the most specific.

It is unclear whether we should prefer this hypothesis over, say, the most general, or some other hypothesis of intermediate generality.

•Are the training examples consistent?

In most practical learning problems there is some chance that the training examples will contain at least some errors or noise.

Such inconsistent sets of training examples can severely mislead FIND-S, given the fact that it ignores negative examples.

We would prefer an algorithm that could at least detect when the training data is inconsistent and, preferably, accommodate such errors.

•What if there are several maximally specific consistent hypotheses?

In the hypothesis language H for the Enjoy-Sport task, there is always a unique, most specific hypothesis consistent with any set of positive examples.

However, for other hypothesis spaces there can be several maximally specific hypotheses consistent with the data.

In this case, FIND-S must be extended to allow it to backtrack on its choices of how to generalize the hypothesis, to accommodate the possibility that the target concept lies along a different branch of the partial ordering than the branch it has selected.

Candidate-Elimination Algorithm

•FIND-S outputs a hypothesis from H, that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well.

•The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.

Candidate-Elimination algorithm computes the description of this set without explicitly enumerating all of its members.

This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.

Consistent hypothesis:

•The key difference between this definition of consistent and satisfies.

•An example x is said to satisfy hypothesis h when h(x) = 1, regardless of whether x is a positive or negative example of the target concept.

•However, whether such an example is consistent with h depends on the target concept, and in particular, whether h(x) = c(x).

A hypothesis h is **consistent** with a set of training examples D of target concept c if and only if $h(x) = c(x)$ for each training example $\langle x, c(x) \rangle$ in D.

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

Version Spaces:

- The Candidate-Elimination algorithm represents the set of

all hypotheses consistent with the observed training examples.

- This subset of all hypotheses is called the version space with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.

List-Then-Eliminate Algorithm

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

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

•List-Then-Eliminate algorithm initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example.

•The version space of candidate hypotheses thus shrinks as more examples are observed, until ideally just one hypothesis remains that is consistent with all the observed examples.

Presumably, this is the desired target concept.

If insufficient data is available to narrow the version space to a single hypothesis, then the algorithm can output the entire set of hypotheses consistent with the observed data.

•List-Then-Eliminate algorithm can be applied whenever the hypothesis space H is finite.

It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data.

Unfortunately, it requires exhaustively enumerating all hypotheses in H - an unrealistic requirement for all but the most trivial hypothesis spaces.

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$

   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$

Compact Representation of Version Spaces

•A version space can be represented with its general and specific boundary sets.

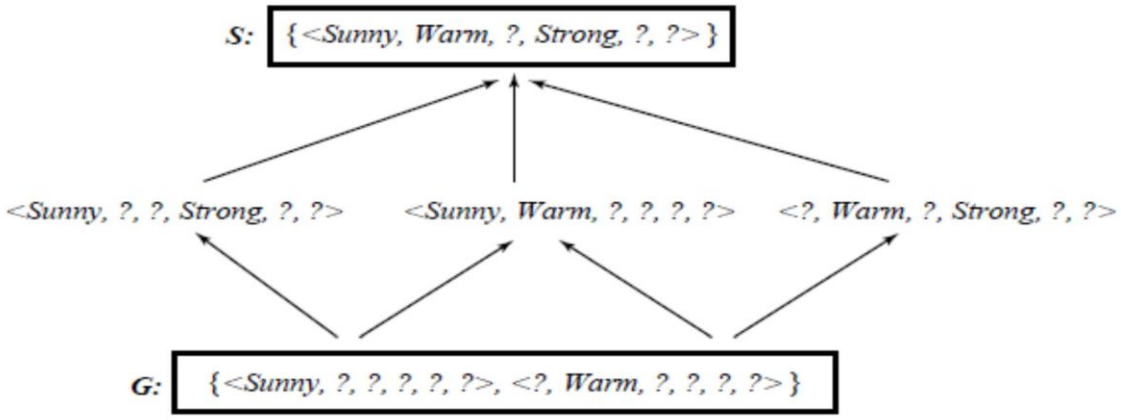
•The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.

•Given only these two sets S and G, it is possible to enumerate all members of a version space by generating hypotheses that lie between these two sets in general-to-specific partial ordering over hypotheses.

•Every member of the version space lies between these boundaries

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq h \geq s)\}$$

where x ≥y means x is more general or equal to y.



- A version space with its general and specific boundary sets.
- The version space includes all six hypotheses shown here, but can be represented more simply by S and G.

Candidate-Elimination Algorithm

- The Candidate-Elimination algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.
- It begins by initializing the version space to the set of all hypotheses in H; that is, by initializing the G boundary set to contain the most general hypothesis in H

G0 ← { <?, ?, ?, ?, ?, ?> }

and initializing the S boundary set to contain the most specific hypothesis S0 ← { <0, 0, 0, 0, 0, 0> }

- These two boundary sets delimit the entire hypothesis space, because every other hypothesis in H is both more general than S0 and more specific than G0.
- As each training example is considered, the S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypotheses found inconsistent with the new training example.
- After all examples have been processed, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses.
- Initialize G to the set of maximally general hypotheses in H
- Initialize S to the set of maximally specific hypotheses in H
- For each training example d, do

– If d is a positive example

- Remove from G any hypothesis inconsistent with d ,
- For each hypothesis s in S that is not consistent with d ,-

Remove s from S

Add to S all minimal generalizations h of s such that

h is consistent with d, and some member of G is more general than h

Remove from S any hypothesis that is more general than another hypothesis in S

If d is a negative example

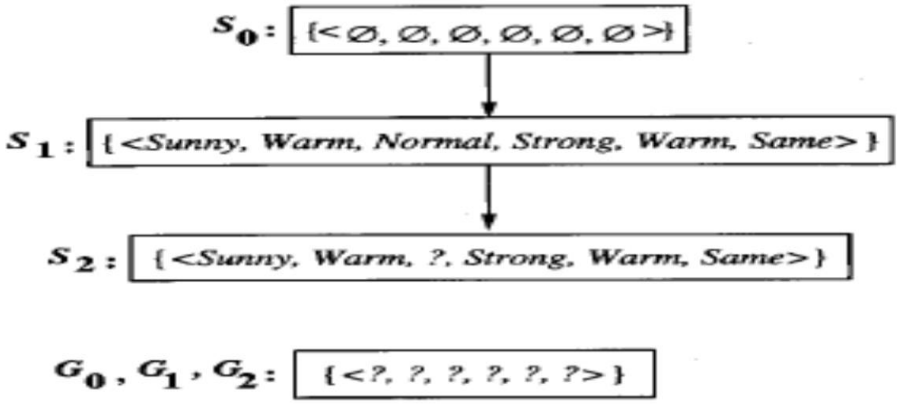
- Remove from S any hypothesis inconsistent with d
- For each hypothesis g in G that is not consistent with d

Remove g from G

Add to G all minimal specializations h of g such that

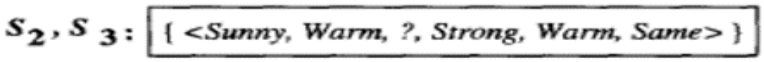
h is consistent with d, and some member of S is more specific than h

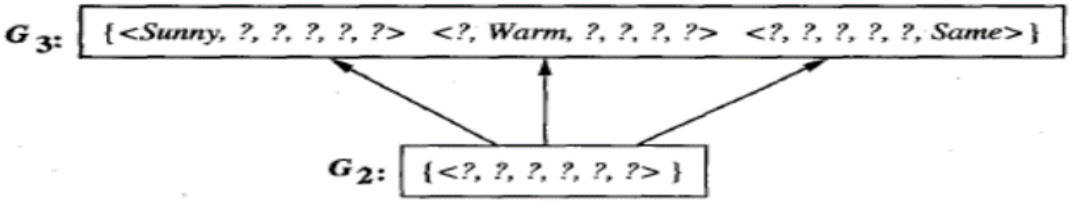
Remove from G any hypothesis that is less general than another hypothesis in G



Training examples:

1. *<Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes*
2. *<Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes*



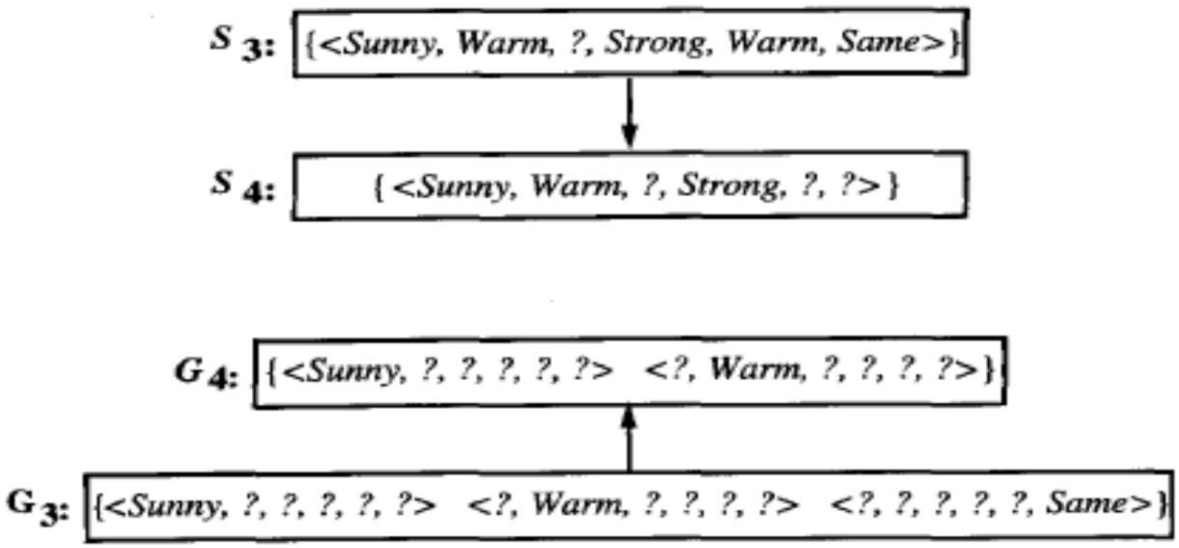


Training Example:

3. *<Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No*

•Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?

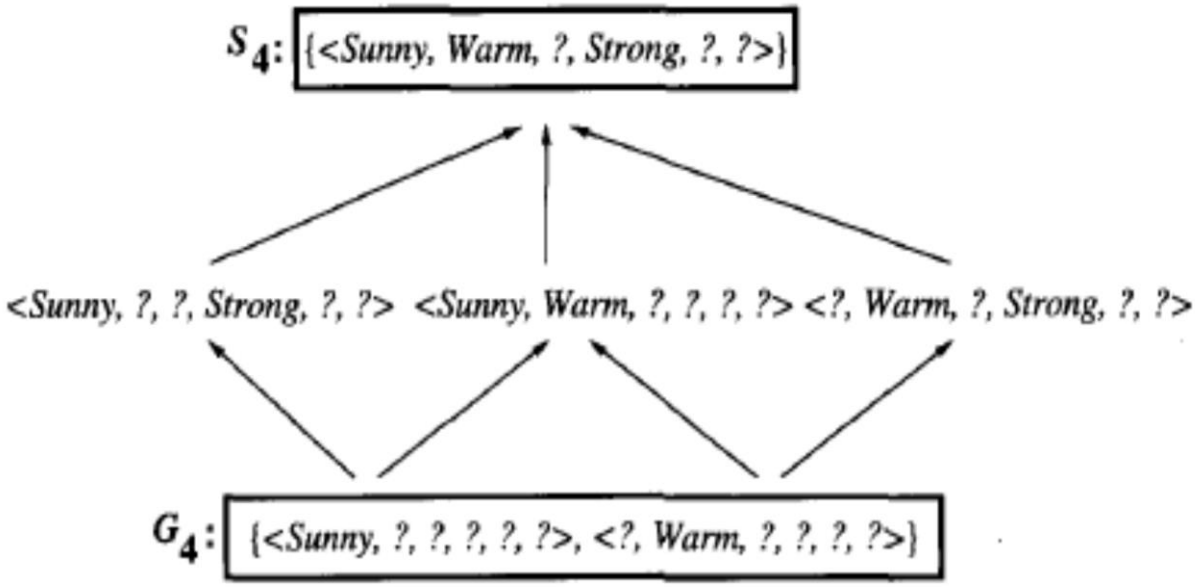
•For example, the hypothesis h = <?, ?, Normal, ?, ?, ?> is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with S2. The algorithm determines this simply by noting that h is not more general than the current specific boundary, S2. In fact, the S boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples. The G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples



Training Example:

4. <Sunny, Warm, High, Strong, Cool, Change>, EnjoySport = Yes

•The fourth training example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example. To understand the rationale for this step, it is useful to consider why the offending hypothesis must be removed from G. Notice it cannot be specialized, because specializing it would not make it cover the new example. It also cannot be generalized, because by the definition of G, any more general hypothesis will cover at least one negative training example. Therefore, the hypothesis must be dropped from the G boundary, thereby removing an entire branch of the partial ordering from the version space of hypotheses remaining under consideration



- After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.
- This learned version space is independent of the sequence in which the training examples are presented (because in the end it contains all hypotheses consistent with the set of examples).
- As further training data is encountered, the S and G boundaries will move monotonically closer to each other, delimiting a smaller and smaller version space of candidate hypotheses.

Will Candidate-Elimination Algorithm Converge to Correct Hypothesis?

•The version space learned by the Candidate-Elimination Algorithm will converge toward the hypothesis that correctly describes the target concept, provided There are no errors in the training examples, and there is some hypothesis in H that correctly describes the target concept.

•What will happen if the training data contains errors?

The algorithm removes the correct target concept from the version space. S and G boundary sets eventually converge to an empty version space if sufficient additional training data is available. Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples. A similar symptom will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation. e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions

•We have assumed that training examples are provided to the learner by some external teacher.

•Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).

This scenario covers situations in which the learner may conduct experiments in nature or in which a teacher is available to provide the correct classification. We use the term query to refer to such instances constructed by the learner, which are then classified by an external oracle. Considering the version space learned from the four training examples of the Enjoy-Sport concept.

What would be a good query for the learner to pose at this point?

What is a good query strategy in general?

The learner should attempt to discriminate among the alternative competing hypotheses in its current version space.

Therefore, it should choose an instance that would be classified positive by some of these hypotheses, but negative by others.

One such instance is <Sunny, Warm, Normal, Light, Warm, Same>

This instance satisfies three of the six hypotheses in the current version space.

If the trainer classifies this instance as a positive example, the S boundary of the version space can then be generalized.

Alternatively, if the trainer indicates that this is a negative example, the G boundary can then be specialized.

In general, the optimal query strategy for a concept learner is to generate instances that satisfy exactly half the hypotheses in the current version space.

When this is possible, the size of the version space is reduced by half with each new example, and the correct target concept can therefore be found with only log2 |VS| experiments.

How Can Partially Learned Concepts Be Used?

•The version space learned by the Candidate-Elimination Algorithm will converge toward the hypothesis that correctly describes the target concept provided. There are no errors in the training examples, and there is some hypothesis in H that correctly describes the target concept.

•What will happen if the training data contains errors?

The algorithm removes the correct target concept from the version space. S and G boundary sets eventually converge to an empty version space if sufficient additional training data is available. Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples.

•A similar symptom will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation. e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions

We have assumed that training examples are provided to the learner by some external teacher.

Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).

This scenario covers situations in which the learner may conduct experiments in nature or in which a teacher is available to provide the correct classification.

We use the term query to refer to such instances constructed by the learner, which are then classified by an external oracle.

•Considering the version space learned from the four training examples of the Enjoy-Sport concept.

What would be a good query for the learner to pose at this point?

What is a good query strategy in general?

•The learner should attempt to discriminate among the alternative competing hypotheses in its current version space. Therefore, it should choose an instance that would be classified positive by some of these hypotheses, but negative by others.

One such instance is <Sunny, Warm, Normal, Light, Warm, Same>

This instance satisfies three of the six hypotheses in the current version space. If the trainer classifies this instance as a positive example, the S boundary of the version space can then be generalized.

Alternatively, if the trainer indicates that this is a negative example, the G boundary can then be specialized. In general, the optimal query strategy for a concept learner is to generate instances that satisfy exactly half the hypotheses in the current version space. When this is possible, the size of the version space is reduced by half with each new example, and the correct target concept can therefore be found with only ⌈log2 |VS|⌉ experiments.

How Can Partially Learned Concepts Be Used?

•Even though the learned version space still contains multiple hypotheses, indicating that the target concept has not yet been fully learned, it is possible to classify certain examples with the same degree of confidence as if the target concept had been uniquely identified.

•Let us assume that the followings are new instances to be classified:

Instance	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Light	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

Instance A was is classified as a positive instance by every hypothesis in the current version space.

Because the hypotheses in the version space unanimously agree that this is a positive instance, the learner can classify instance A as positive with the same confidence it would have if it had already converged to the single, correct target concept. Regardless of which hypothesis in the version space is eventually found to be the correct target concept, it is already clear that it will classify instance A as

a positive example. Notice furthermore that we need not enumerate every hypothesis in the version space in order to test whether each classifies the instance as positive.

This condition will be met if and only if the instance satisfies every member of S. The reason is that every other hypothesis in the version space is at least as general as some member of S. By our definition of more-general-than, if the new instance satisfies all members of S it must also satisfy each of these more general hypotheses.

Instance B is classified as a negative instance by every hypothesis in the version space. This instance can therefore be safely classified as negative, given the partially learned concept. An efficient test for this condition is that the instance satisfies none of the members of G. Half of the version space hypotheses classify instance C as positive and half classify it as negative. Thus, the learner cannot classify this example with confidence until further training examples are available. Instance D is classified as positive by two of the version space hypotheses and negative by the other four hypotheses. In this case we have less confidence in the classification than in the unambiguous cases of instances A and B.

Still, the vote is in favour of a negative classification, and one approach we could take would be to output the majority vote, perhaps with a confidence rating indicating how close the vote was.

•The Candidate-Elimination Algorithm will converge toward the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.

•What if the target concept is not contained in the hypothesis space?

•Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?

•How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?

•How does the size of the hypothesis space influence the number of training examples that must be observed?

Inductive Bias - A Biased Hypothesis Space

In Enjoy-Sport example, we restricted the hypothesis space to include only conjunctions of attribute values. Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as "Sky = Sunny or Sky = Cloudy."

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Cool	Change	Yes
2	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
3	Rainy	Warm	Normal	Strong	Cool	Change	No

- From first two examples S2 : <?, Warm, Normal, Strong, Cool, Change>

• This is inconsistent with third examples, and there are no hypotheses consistent with these three examples

PROBLEM: We have biased the learner to consider only conjunctive hypotheses.

 We require a more expressive hypothesis space.

Inductive Bias - An Unbiased Learner

The obvious solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space capable of representing every teachable concept.

Every possible subset of the instances X the power set of X.

What is the size of the hypothesis space H (the power set of X) ?

In EnjoySport, the size of the instance space X is 96.

The size of the power set of X is 2|X| The size of H is 296

Our conjunctive hypothesis space is able to represent only 973of these hypotheses. A very biased hypothesis space

Inductive Bias - An Unbiased Learner : Problem

•Let the hypothesis space H to be the power set of X. A hypothesis can be represented with disjunctions, conjunctions, and negations of our earlier hypotheses.

The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

<Sunny, ?, ?, ?, ?, ?> <Cloudy, ?, ?, ?, ?, ?>

NEW PROBLEM: our concept learning algorithm is now completely unable to generalize beyond the observed examples.

three positive examples (xl,x2,x3) and two negative examples (x4,x5) to the learner.

S : { x1 x2 x3 } and G : { (x4 x5) } NO GENERALIZATION

Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.

Inductive Bias –

Fundamental Property of Inductive Inference

•A learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.

•Inductive Leap: A learner should be able to generalize training data using prior assumptions in order to classify unseen instances.

•The generalization is known as inductive leap and our prior assumptions are the inductive bias of the learner.

•Inductive Bias (prior assumptions) of Candidate-Elimination Algorithm is that the target concept can be represented by a conjunction of attribute values, the target concept is contained in the hypothesis space and training examples are correct.

Inductive Bias – Formal Definition

Inductive Bias:

Consider a concept learning algorithm L for the set of instances X.

Let c be an arbitrary concept defined over X, and

let Dc = {<x , c(x)>} be an arbitrary set of training examples of c.

Let L(xi, Dc) denote the classification assigned to the instance xi by L

after training on the data Dc.

The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples Dc the following formula holds.

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

Inductive Bias – Three Learning Algorithms

ROTE-LEARNER: Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to classify the new instance.

Inductive Bias: No inductive bias

CANDIDATE-ELIMINATION: New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance.

Inductive Bias: the target concept can be represented in its hypothesis space.

FIND-S: This algorithm, described earlier, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.

Inductive Bias: the target concept can be represented in its hypothesis space, and all instances are negative instances unless the opposite is entailed by its other know1edge.

Concept Learning – Summary

•Concept learning can be seen as a problem of searching through a large predefined space of potential hypotheses.

•The general-to-specific partial ordering of hypotheses provides a useful structure for organizing the search through the hypothesis space.

•The FIND-S algorithm utilizes this general-to-specific ordering, performing a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples.

•The CANDIDATE-ELIMINATION algorithm utilizes this general-to- specific ordering to compute the version space (the set of all hypotheses consistent with the training data) by incrementally computing the sets of maximally specific (S) and maximally general (G) hypotheses.

• Because the S and G sets delimit the entire set of hypotheses consistent with the data, they provide the learner with a description of its uncertainty regarding the exact identity of the target concept. This version space of alternative hypotheses can be examined

To determine whether the learner has converged to the target concept,

To determine when the training data are inconsistent,

To generate informative queries to further refine the version space, and

To determine which unseen instances can be unambiguously classified based on the partially learned concept.

The CANDIDATE-ELIMINATION algorithm is not robust to noisy data or to situations in which the unknown target concept is not expressible in the provided hypothesis space.

Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another.

If the hypothesis space is enriched to the point where there is a hypothesis corresponding to every possible subset of instances (the power set of the instances), this will remove any inductive bias from the CANDIDATE-ELIMINATION algorithm .

Unfortunately, this also removes the ability to classify any instance beyond the observed training examples. An unbiased learner cannot make inductive leaps to classify unseen examples.

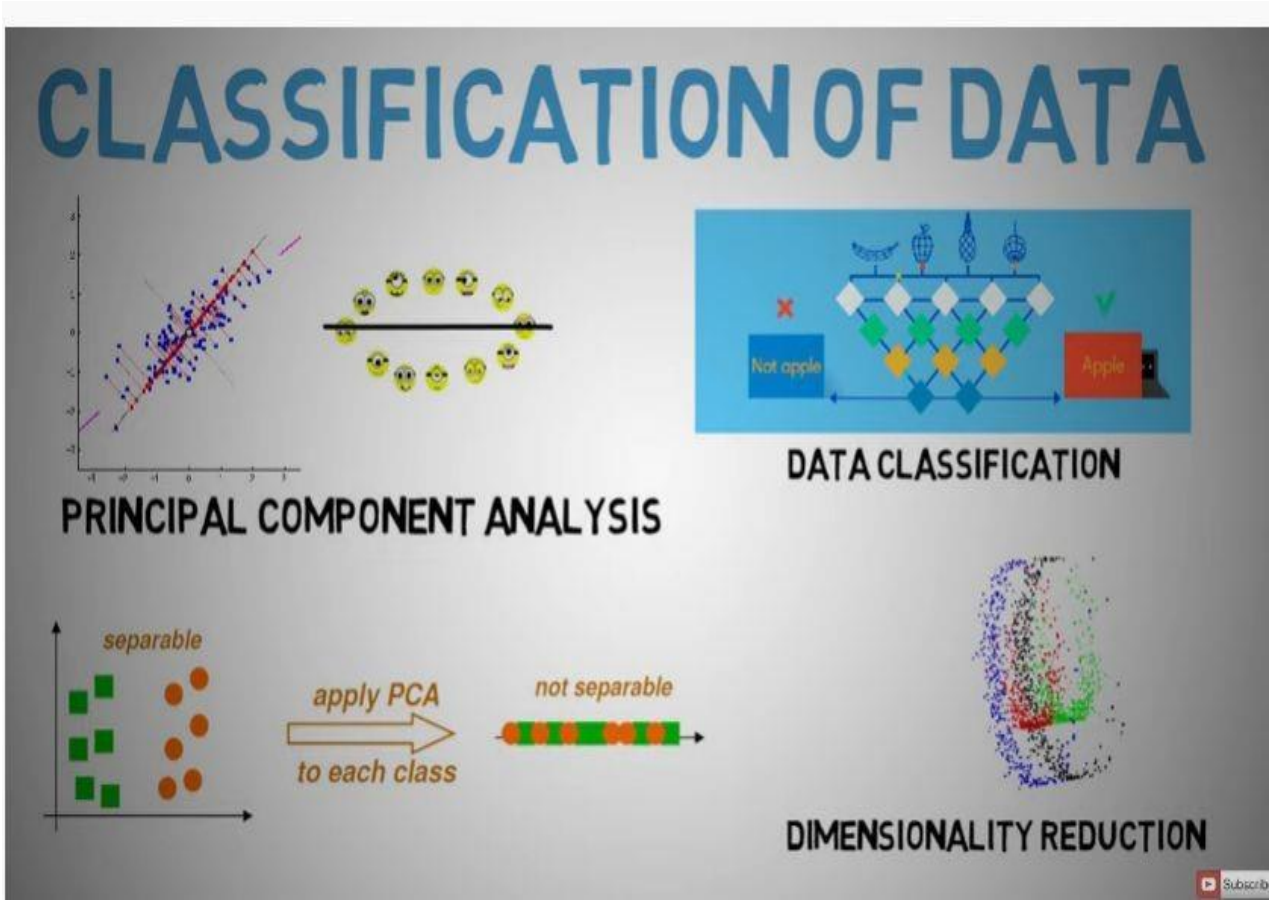
Linear Discriminant analysis

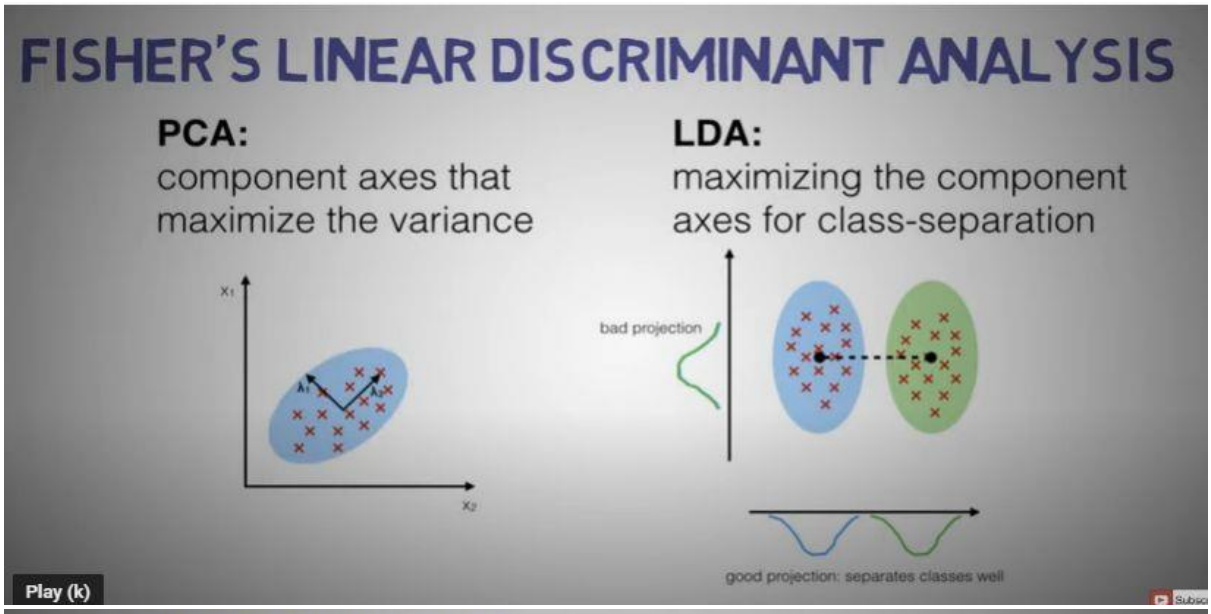
Example

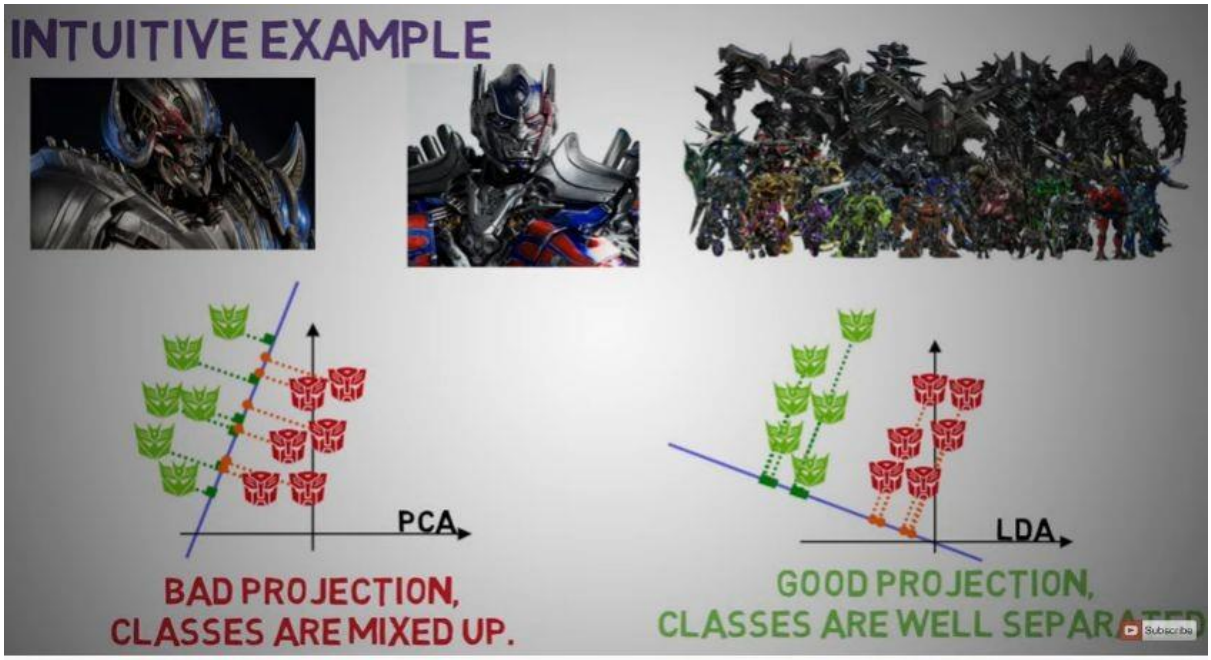
- Discriminating Students in high school –
  - Will go to College
  - Will go to trade school
  - Discontinue education

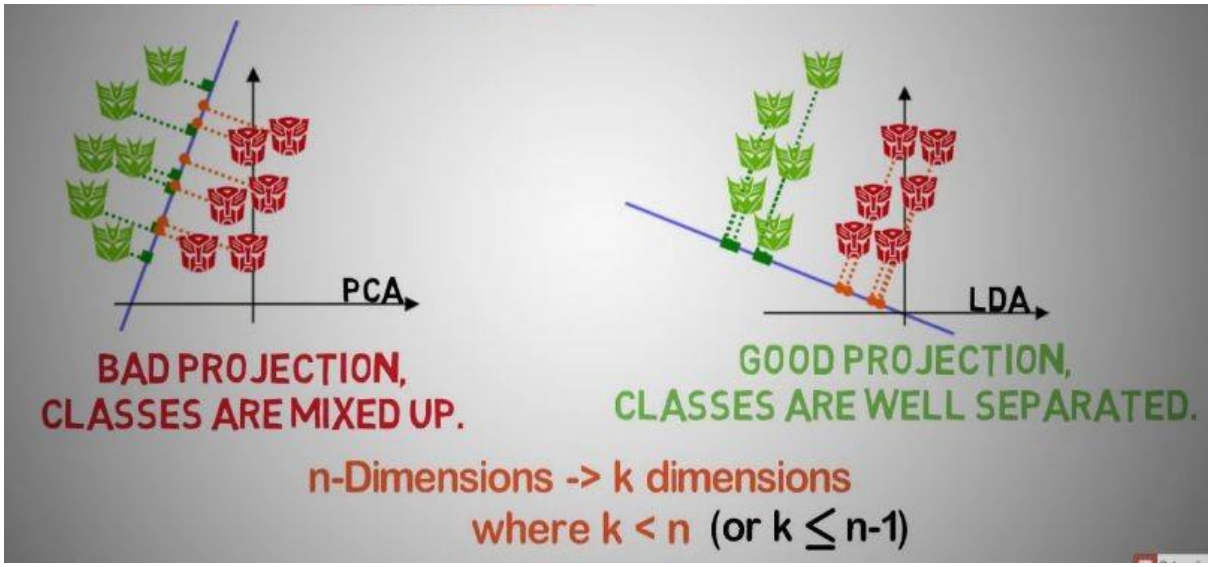
Some pattern must be there that decided where a student belong to after school. We collect family background, academic information

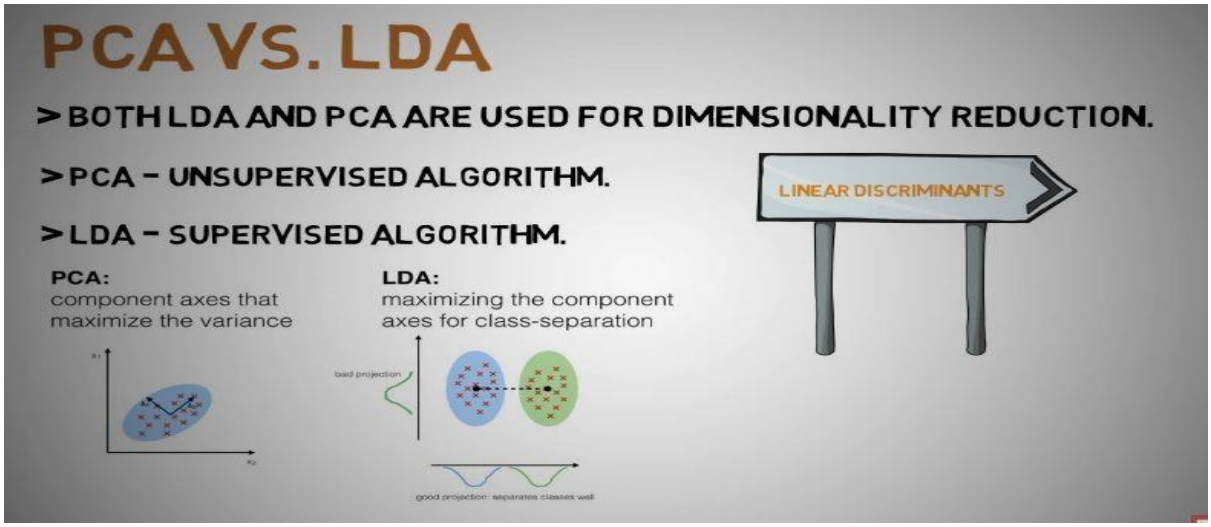
- Discriminate a person between male or female based on height

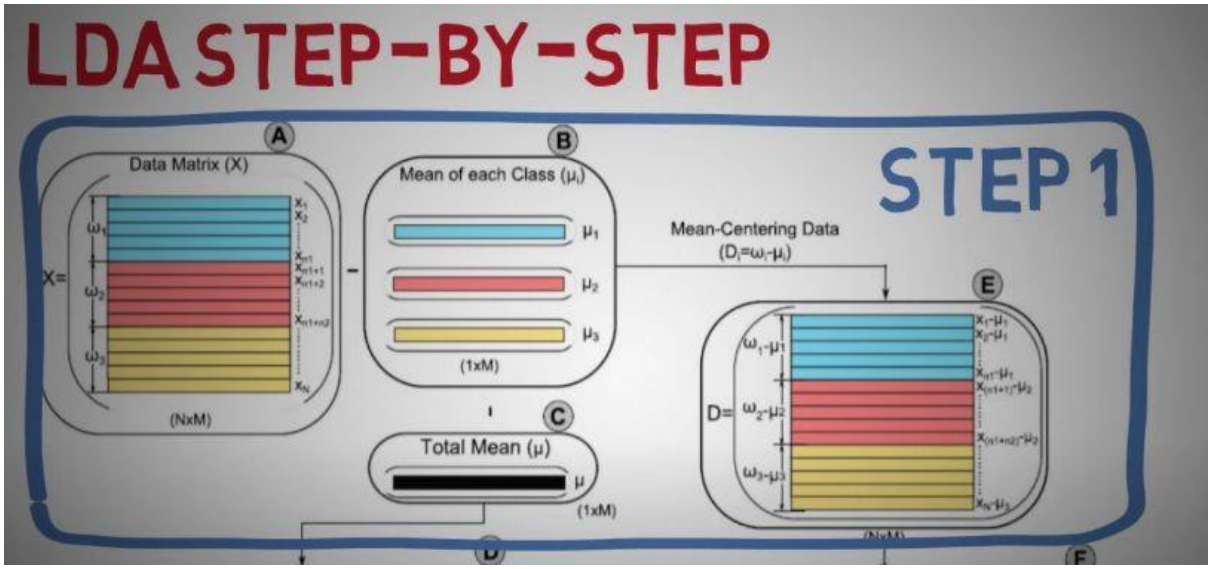


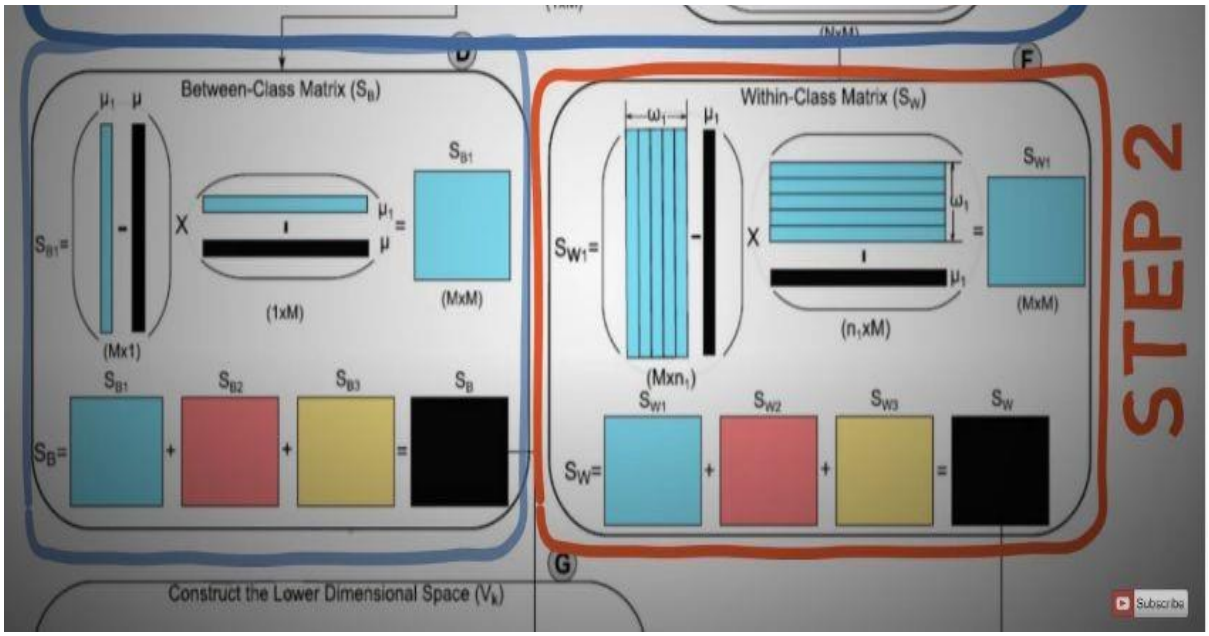

FISHER'S LINEAR DISCRIMINANT ANALYSIS
PCA:
component axes that maximize the variance
LDA:
maximizing the component axes for class-separation
bad projection
good projection: separates classes well
Play (k)

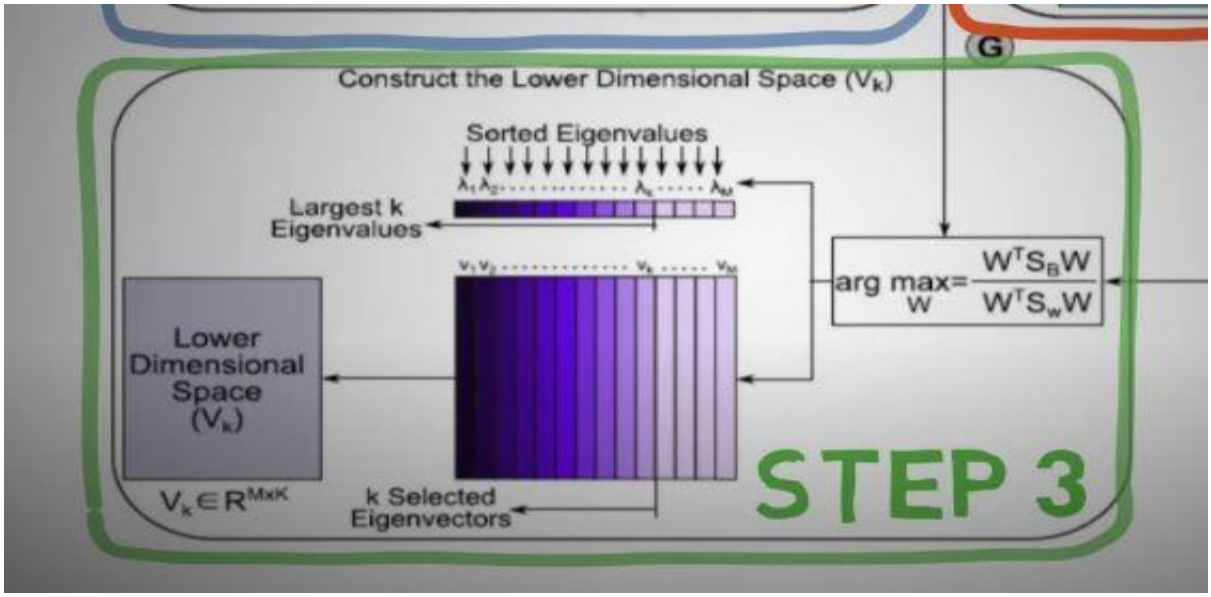

INTUITIVE EXAMPLE
PCA
LDA
BAD PROJECTION, CLASSES ARE MIXED UP.
GOOD PROJECTION, CLASSES ARE WELL SEPARATED

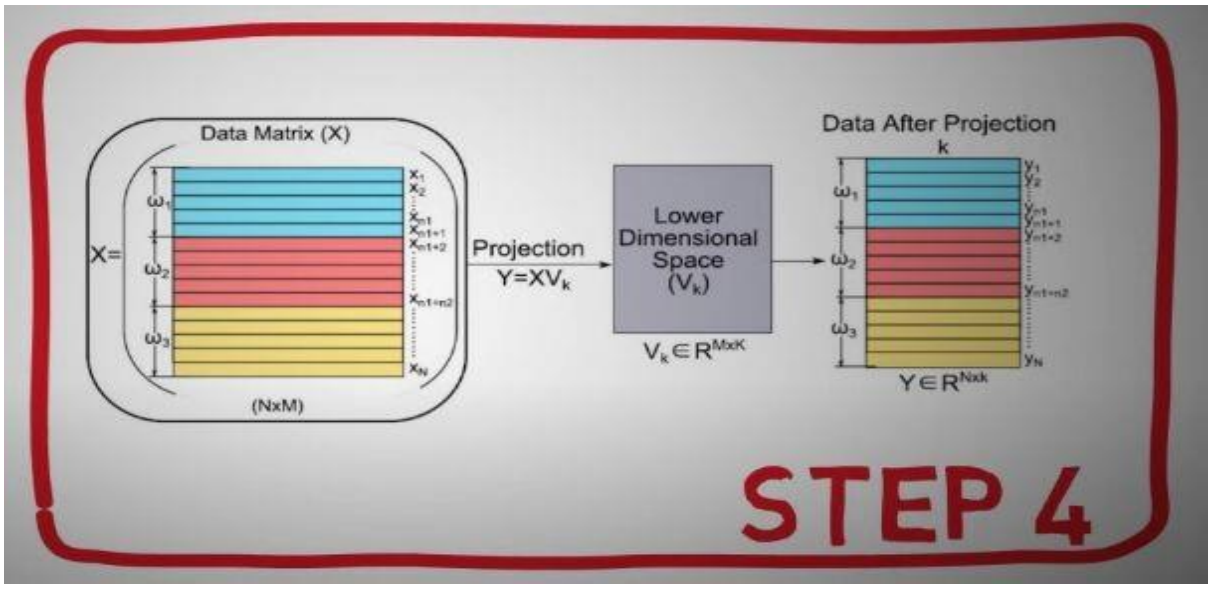

PCA
LDA
BAD PROJECTION, CLASSES ARE MIXED UP.
GOOD PROJECTION, CLASSES ARE WELL SEPARATED.
n-Dimensions -> k dimensions
where k < n (or k ≤ n-1)


PCA VS. LDA
> BOTH LDA AND PCA ARE USED FOR DIMENSIONALITY REDUCTION.
> PCA – UNSUPERVISED ALGORITHM.
> LDA – SUPERVISED ALGORITHM.
PCA:
component axes that maximize the variance
LDA:
maximizing the component axes for class-separation
LINEAR DISCRIMINANTS
bad projection
good projection: separates classes well


LDA STEP-BY-STEP
STEP 1
A
Data Matrix (X)
(NxM)
B
Mean of each Class (μi)
(1xM)
C
Total Mean (μ)
(1xM)
Mean-Centering Data
(Di=ωi-μi)
E
D
F


D
Between-Class Matrix (SB)
(Mx1)
(1xM)
(MxM)
SB = SB1 + SB2 + SB3 = SB
F
Within-Class Matrix (SW)
(Mxn1)
(n1xM)
(MxM)
SW = SW1 + SW2 + SW3 = SW
STEP 2
G
Construct the Lower Dimensional Space (Vk)
Subscribe


G
Construct the Lower Dimensional Space (V_k)
Sorted Eigenvalues
Largest k Eigenvalues
$\arg\max_W \frac{W^T S_B W}{W^T S_w W}$
Lower Dimensional Space (V_k)
$V_k \in R^{MxK}$
k Selected Eigenvectors
STEP 3

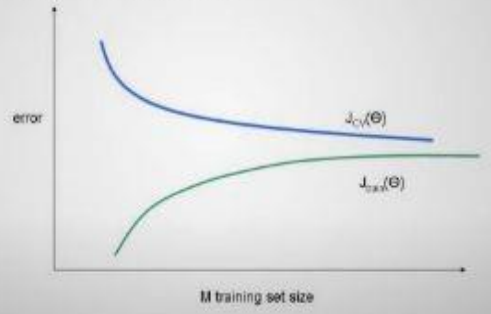

Data Matrix (X)
X=
(NxM)
Projection
$Y=XV_k$
Lower Dimensional Space (V_k)
$V_k \in R^{MxK}$
Data After Projection
k
$Y \in R^{Nxk}$
STEP 4

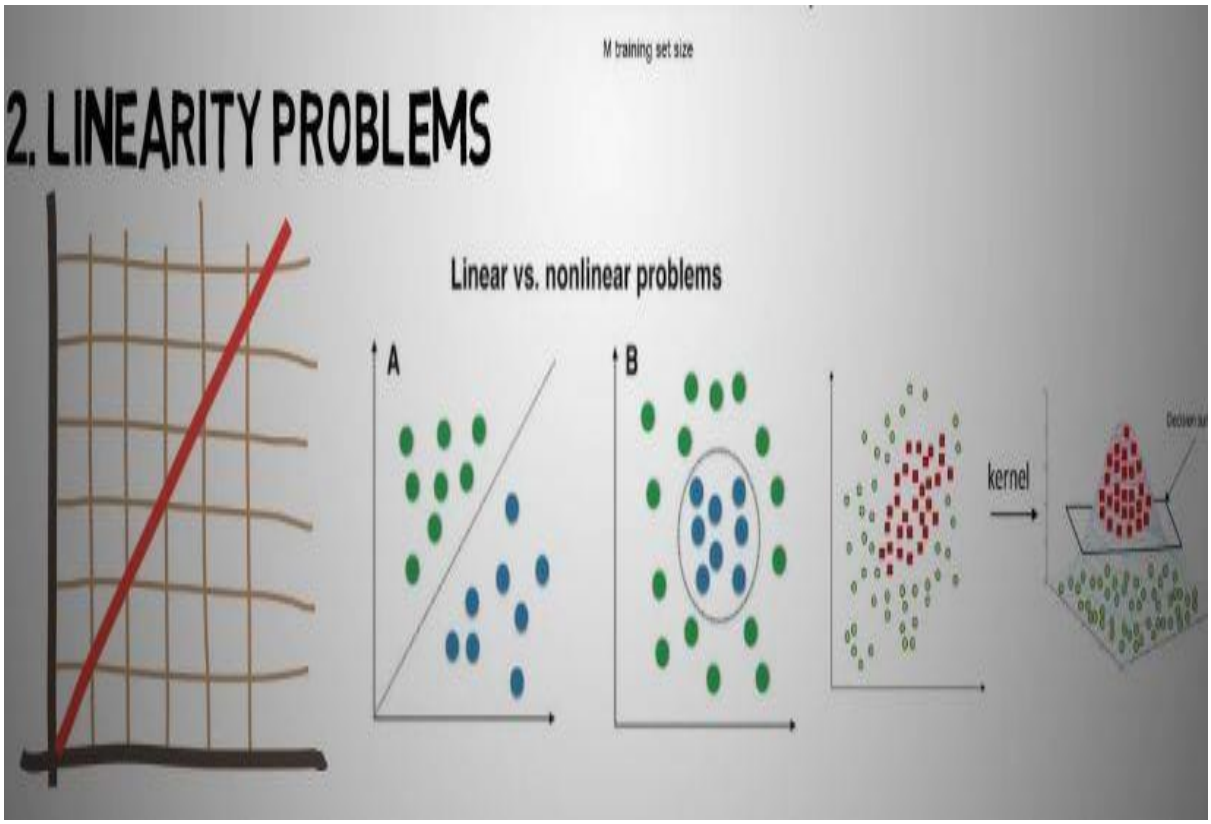
LDA DISADVANTAGES

1.SMALL SAMPLE SIZE (SSS)

> SOLVED WITH REGULARIZATION (RLDA)

SUBSPACE AND NULL SPACE METHODS



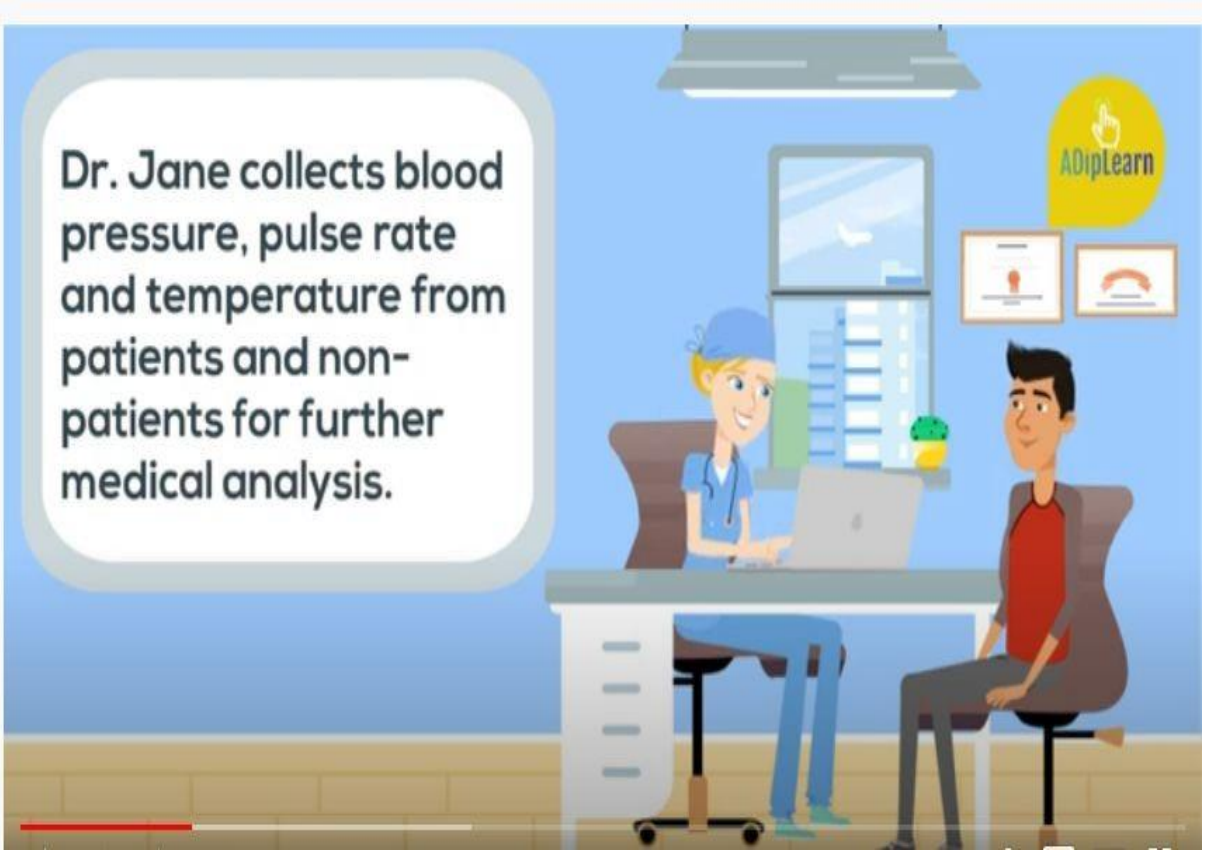


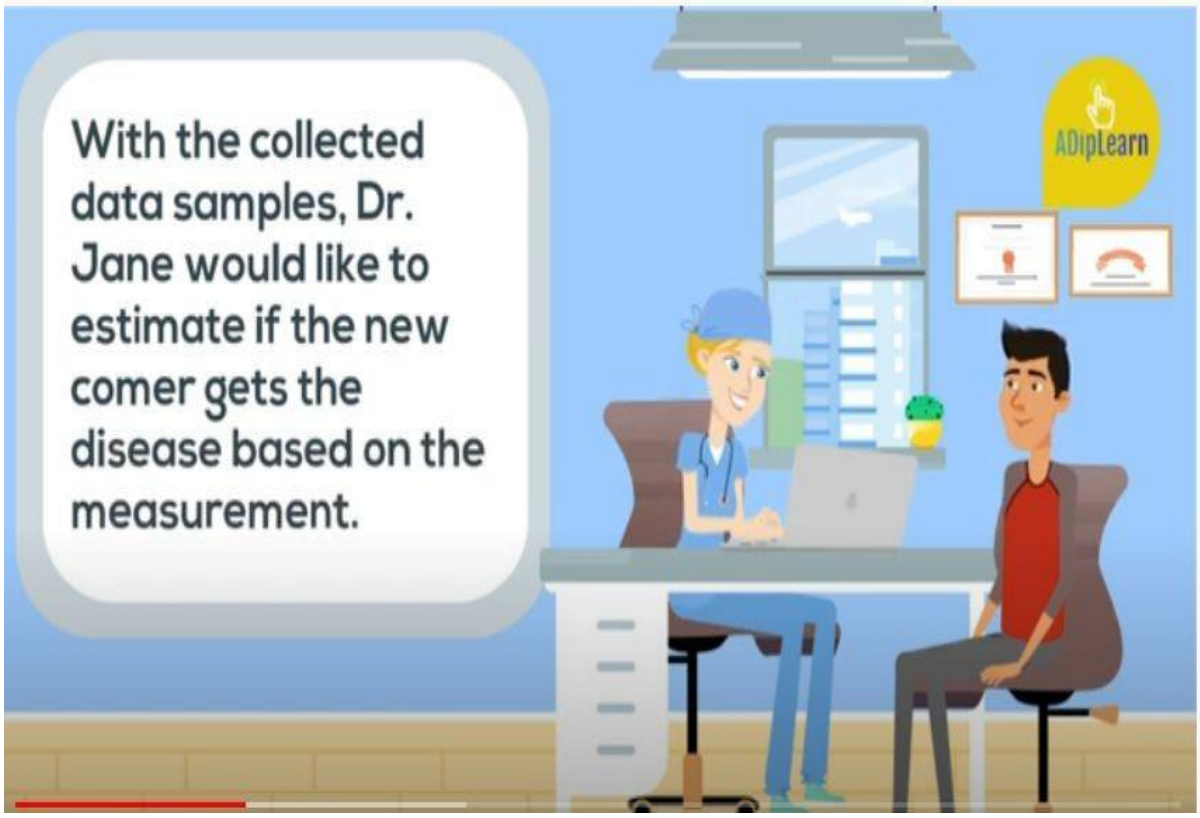
- What is the difference between LDA & PCA?

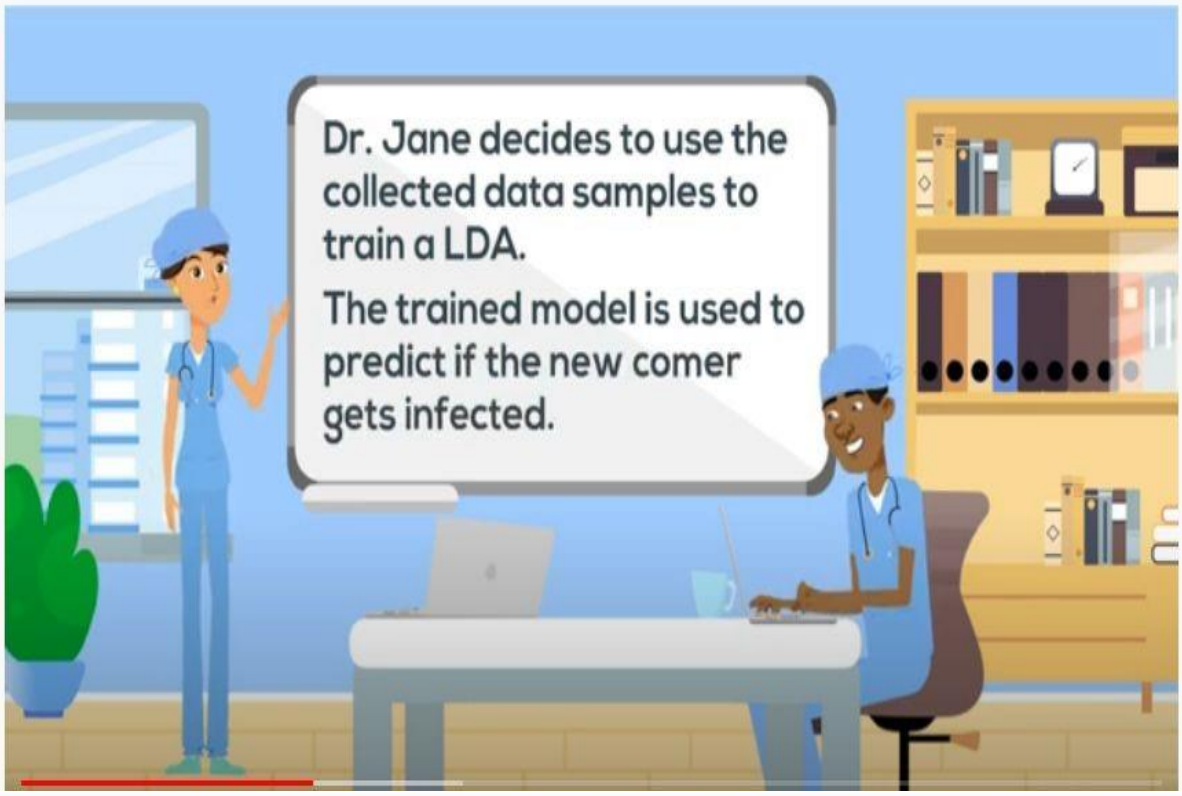
LDA	PCA
Discovers relationship between Dependent & independent variables	Discovers relationship between independent variables
Used for variable reduction based on strength of relationship between independent n dependent variable	Used for reducing variables based on collinearity of independent variables
Used for prediction of classes	
Finds the direction that maximizes difference between two classes	Finds direction that maximizes the variance in the data

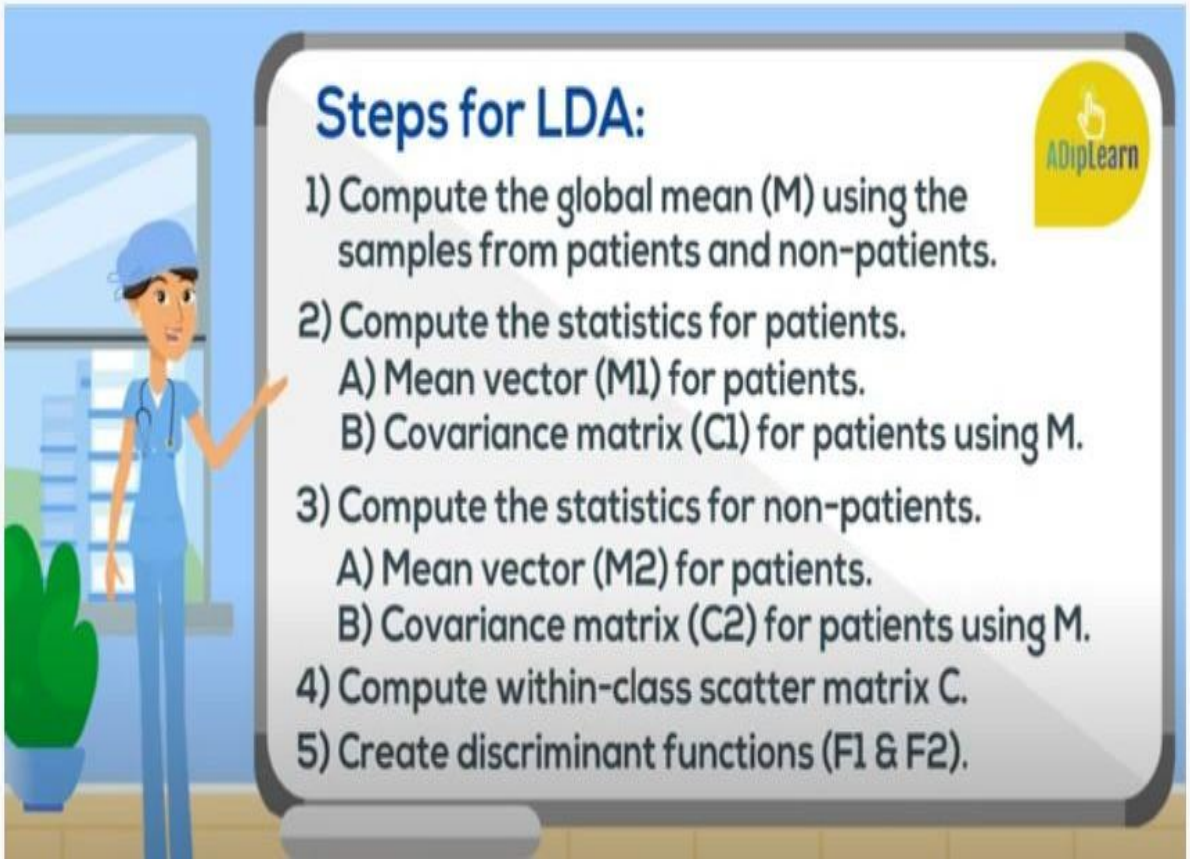

HOSPITAL
The symptoms are high blood pressure, pulse rate and temperature.

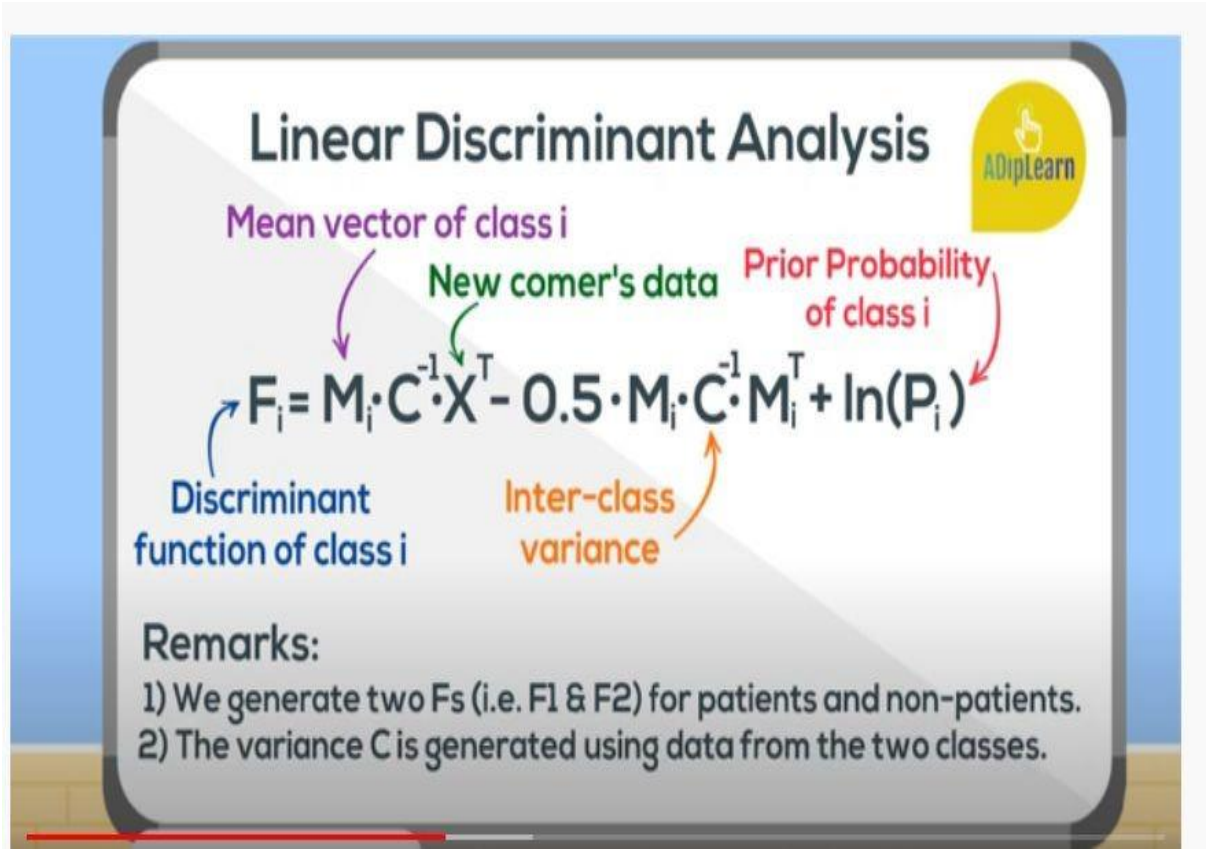

EMERGENCY
More than 100 patients were sent to the emergency room in the last 2 weeks.

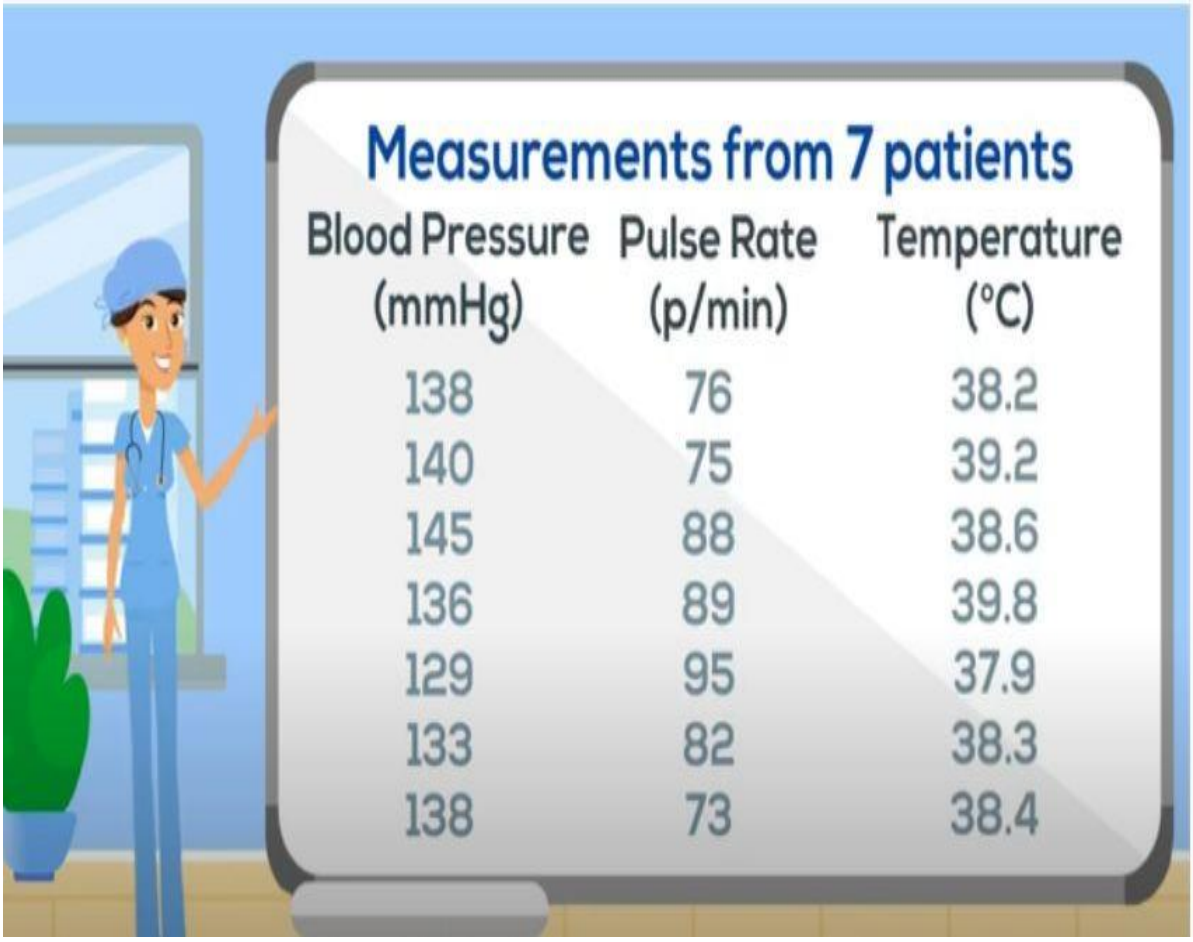

Dr. Jane collects blood pressure, pulse rate and temperature from patients and non-patients for further medical analysis.
ADipLearn

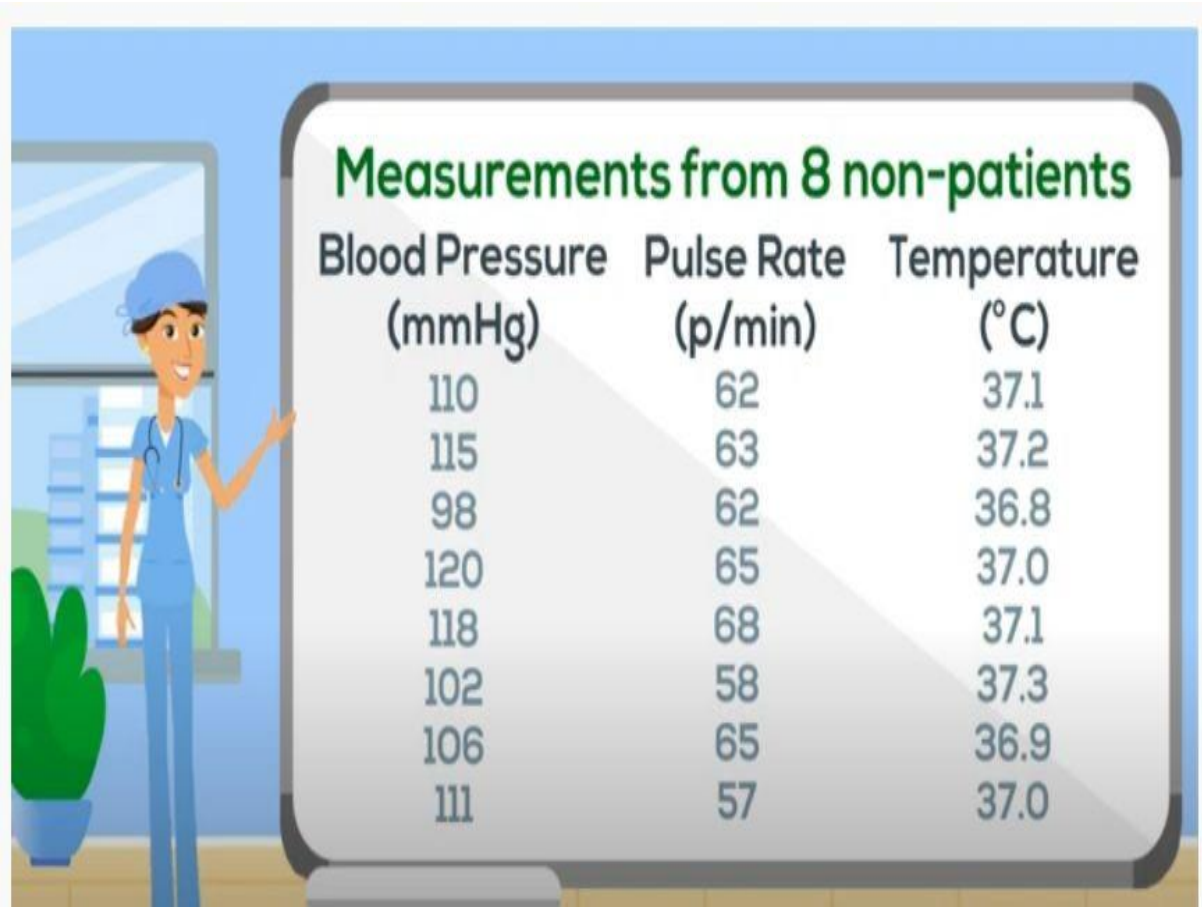

With the collected data samples, Dr. Jane would like to estimate if the new comer gets the disease based on the measurement.
ADipLearn

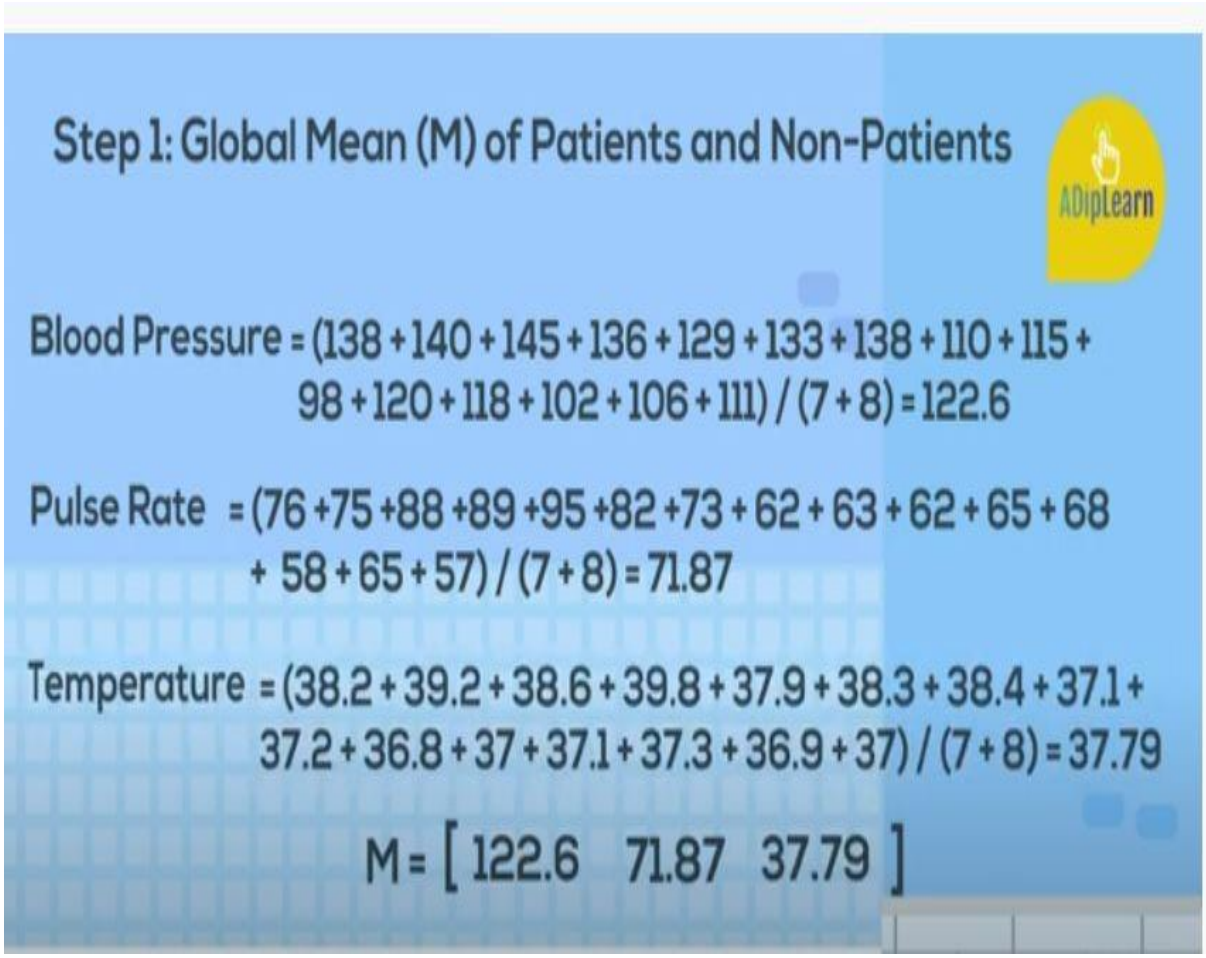

Dr. Jane decides to use the collected data samples to train a LDA.
The trained model is used to predict if the new comer gets infected.

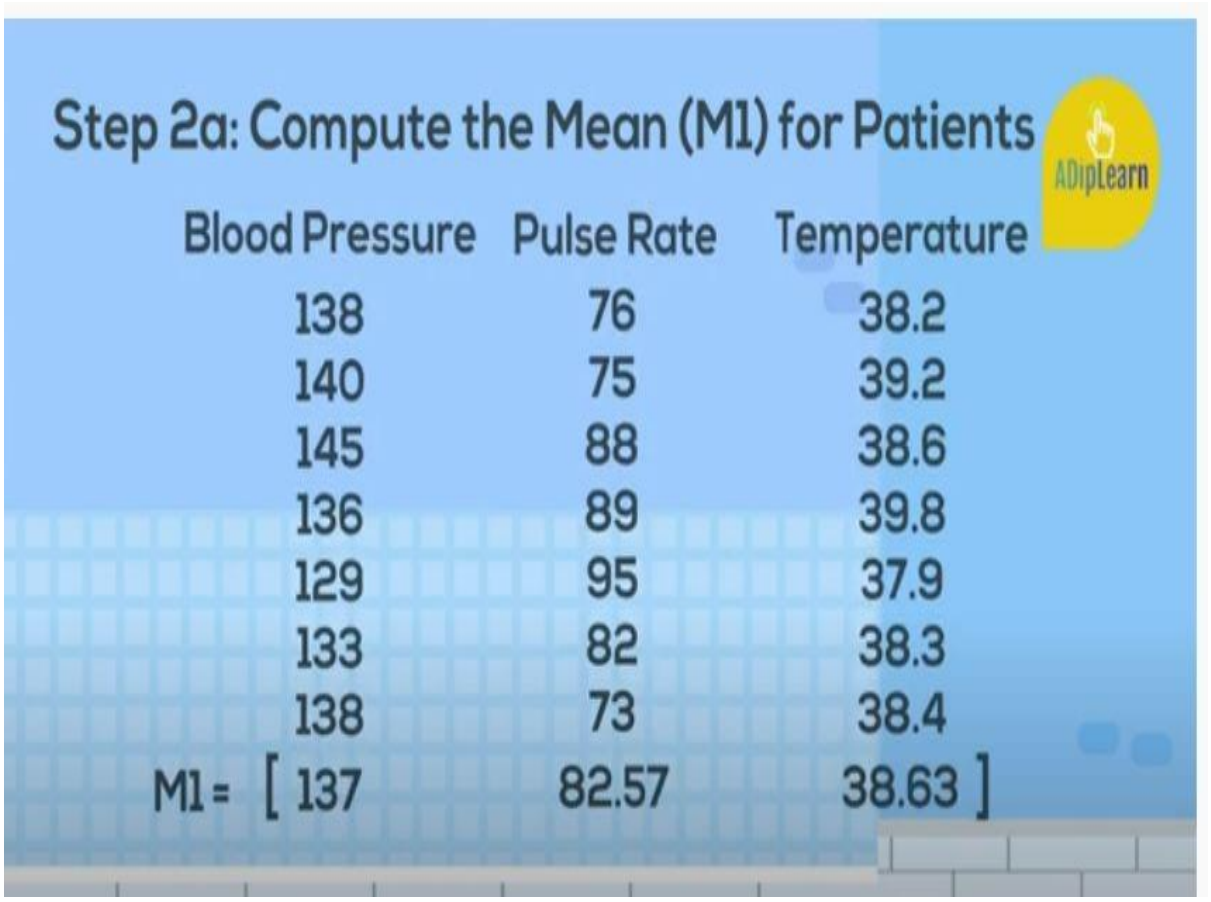

Steps for LDA:
ADipLearn
1) Compute the global mean (M) using the samples from patients and non-patients.
2) Compute the statistics for patients.
A) Mean vector (M1) for patients.
B) Covariance matrix (C1) for patients using M.
3) Compute the statistics for non-patients.
A) Mean vector (M2) for patients.
B) Covariance matrix (C2) for patients using M.
4) Compute within-class scatter matrix C.
5) Create discriminant functions (F1 & F2).

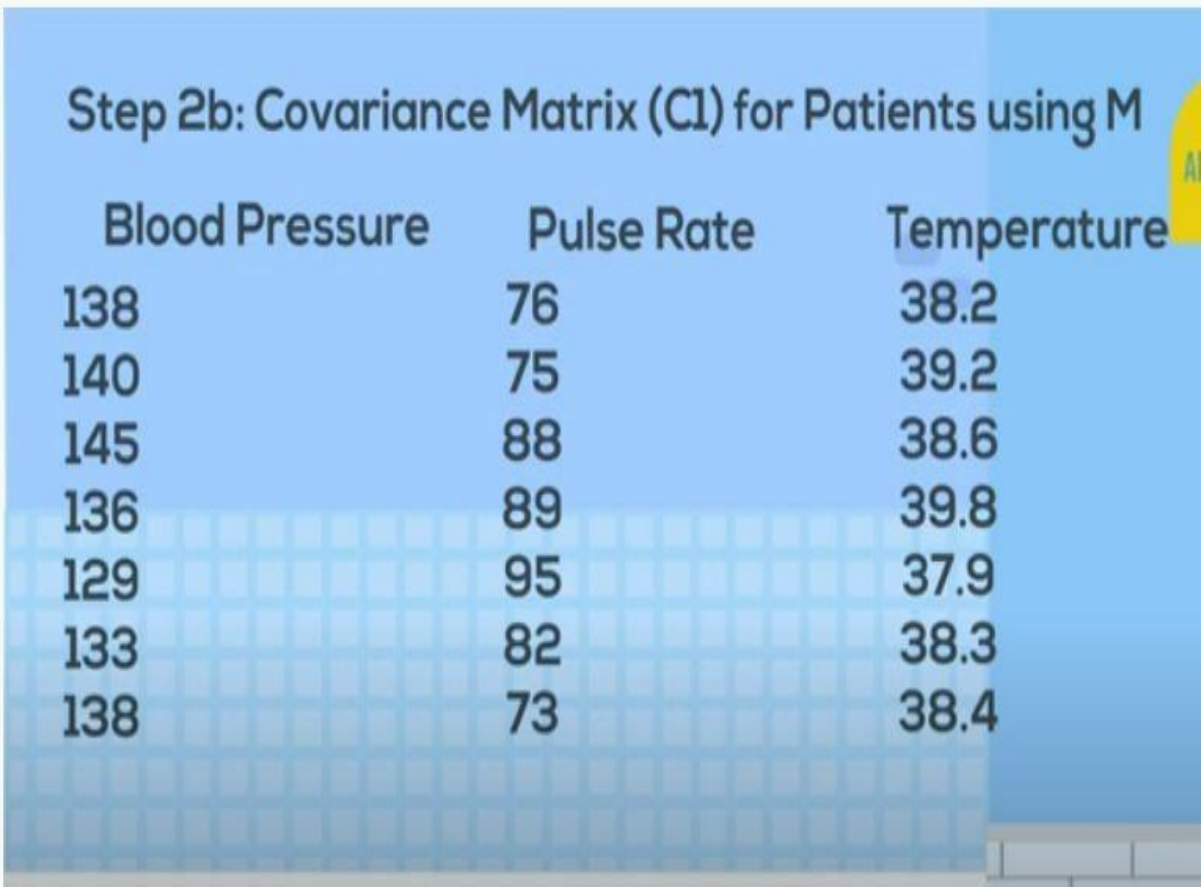

Linear Discriminant Analysis
ADipLearn
Mean vector of class i
New comer's data
Prior Probability of class i
$F_i = M_i \cdot C^{-1} \cdot X^T - 0.5 \cdot M_i \cdot C^{-1} \cdot M_i^T + \ln(P_i)$
Discriminant function of class i
Inter-class variance
Remarks:
1) We generate two Fs (i.e. F1 & F2) for patients and non-patients.
2) The variance C is generated using data from the two classes.

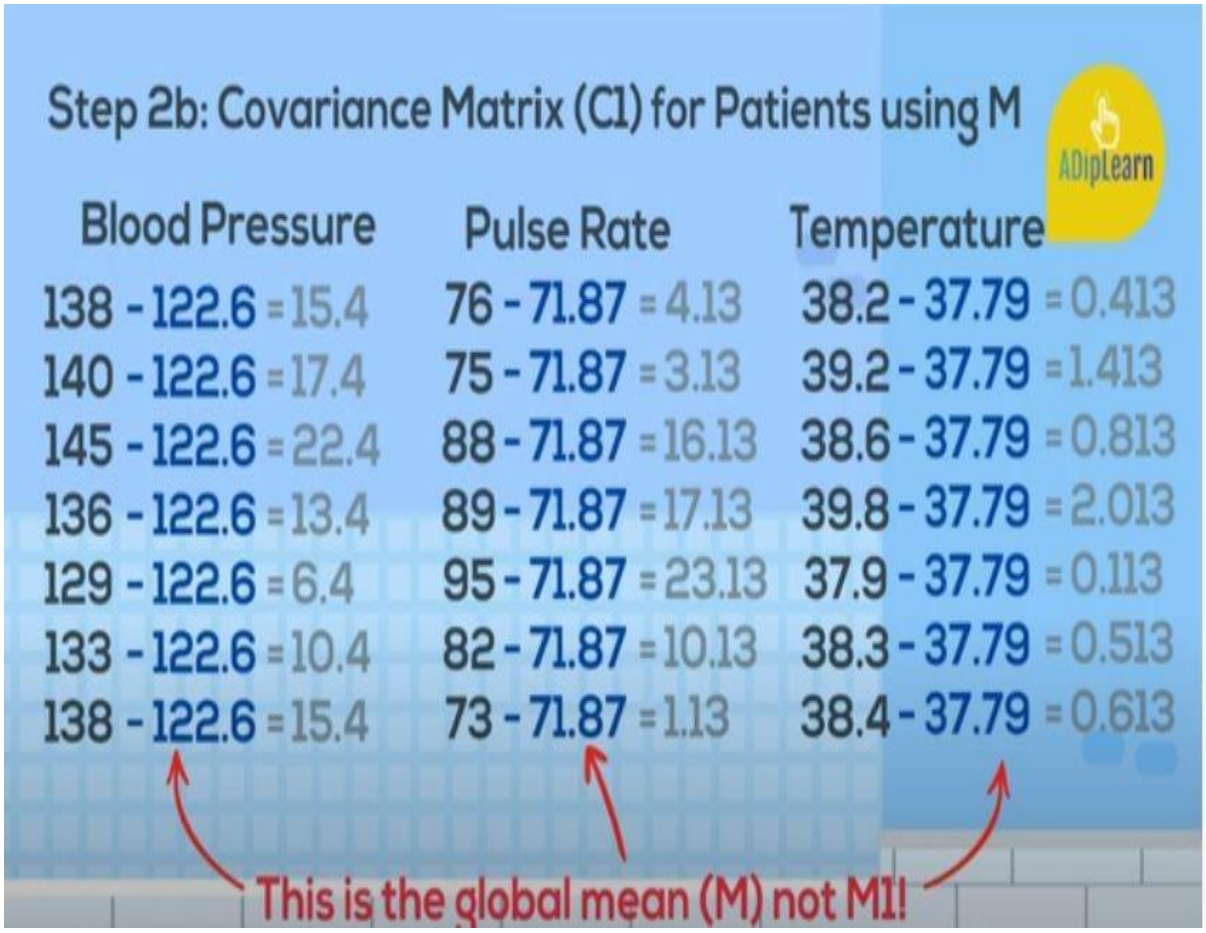

Measurements from 7 patients
Blood Pressure (mmHg) | Pulse Rate (p/min) | Temperature (°C)
138 | 76 | 38.2
140 | 75 | 39.2
145 | 88 | 38.6
136 | 89 | 39.8
129 | 95 | 37.9
133 | 82 | 38.3
138 | 73 | 38.4


Measurements from 8 non-patients
Blood Pressure (mmHg)
Pulse Rate (p/min)
Temperature (°C)
110 62 37.1
115 63 37.2
98 62 36.8
120 65 37.0
118 68 37.1
102 58 37.3
106 65 36.9
111 57 37.0

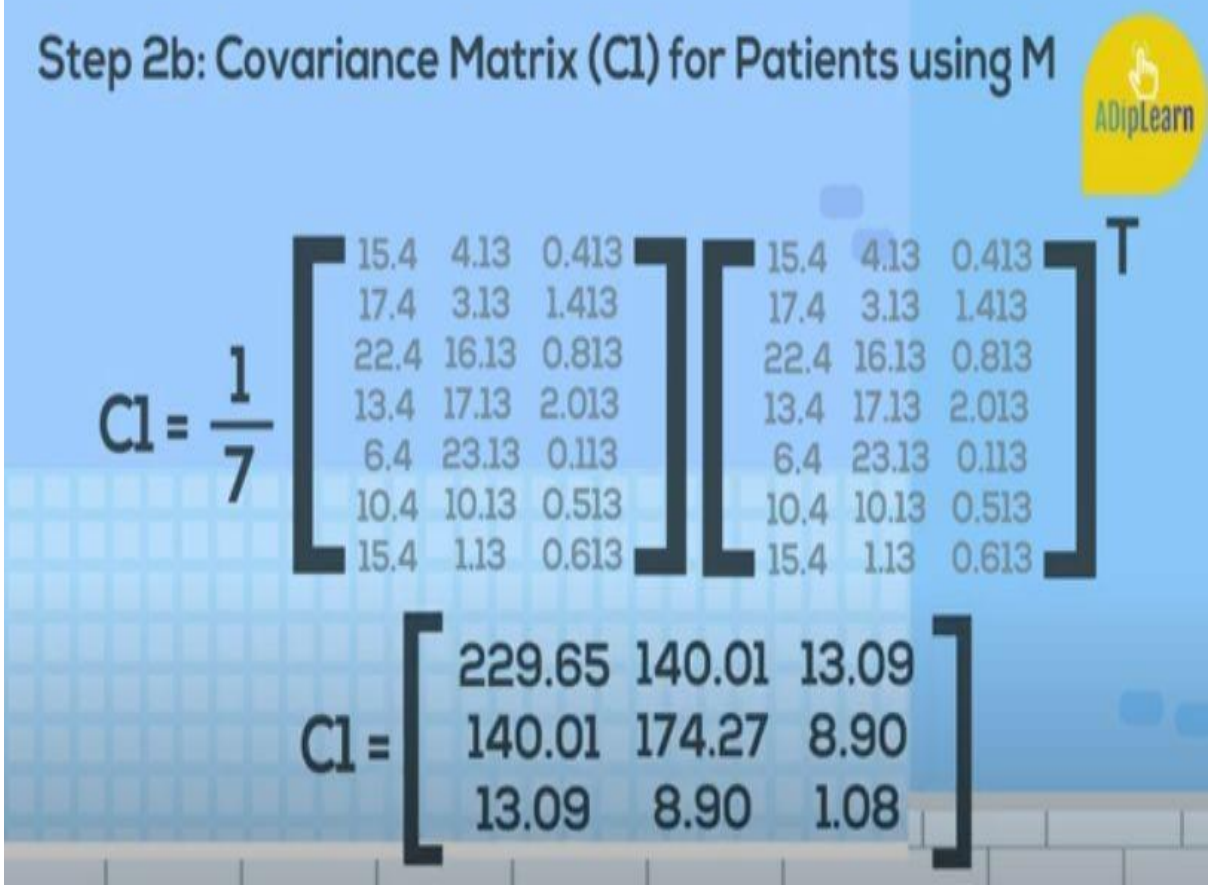

Step 1: Global Mean (M) of Patients and Non-Patients
ADipLearn
Blood Pressure = (138 + 140 + 145 + 136 + 129 + 133 + 138 + 110 + 115 + 98 + 120 + 118 + 102 + 106 + 111) / (7 + 8) = 122.6
Pulse Rate = (76 + 75 + 88 + 89 + 95 + 82 + 73 + 62 + 63 + 62 + 65 + 68 + 58 + 65 + 57) / (7 + 8) = 71.87
Temperature = (38.2 + 39.2 + 38.6 + 39.8 + 37.9 + 38.3 + 38.4 + 37.1 + 37.2 + 36.8 + 37 + 37.1 + 37.3 + 36.9 + 37) / (7 + 8) = 37.79
M = [122.6 71.87 37.79]


Step 2a: Compute the Mean (M1) for Patients
ADipLearn
Blood Pressure Pulse Rate Temperature
138 76 38.2
140 75 39.2
145 88 38.6
136 89 39.8
129 95 37.9
133 82 38.3
138 73 38.4
M1 = [137 82.57 38.63]


Step 2b: Covariance Matrix (C1) for Patients using M
Blood Pressure Pulse Rate Temperature
138 76 38.2
140 75 39.2
145 88 38.6
136 89 39.8
129 95 37.9
133 82 38.3
138 73 38.4

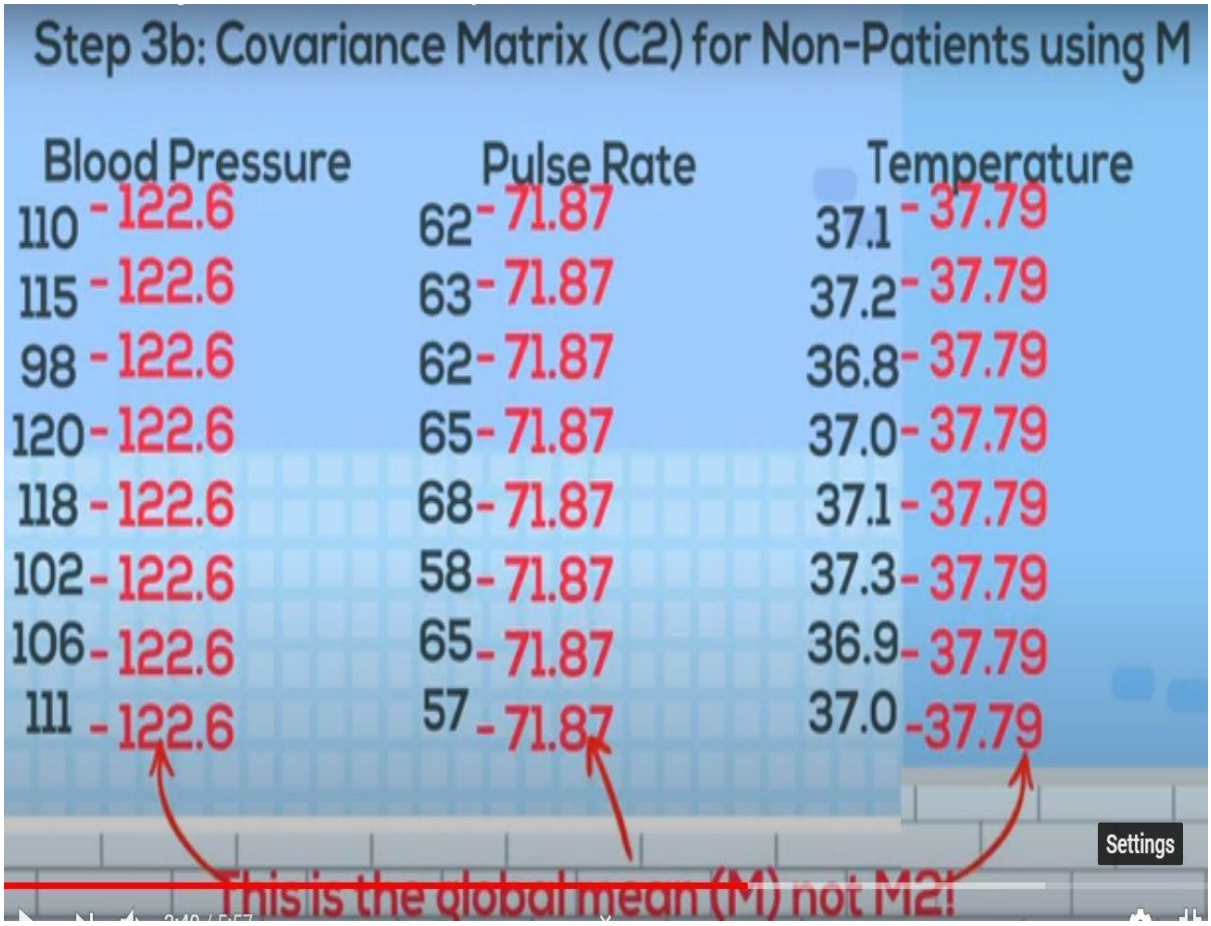

Step 2b: Covariance Matrix (C1) for Patients using M
ADipLearn
Blood Pressure
Pulse Rate
Temperature
138 - 122.6 = 15.4
76 - 71.87 = 4.13
38.2 - 37.79 = 0.413
140 - 122.6 = 17.4
75 - 71.87 = 3.13
39.2 - 37.79 = 1.413
145 - 122.6 = 22.4
88 - 71.87 = 16.13
38.6 - 37.79 = 0.813
136 - 122.6 = 13.4
89 - 71.87 = 17.13
39.8 - 37.79 = 2.013
129 - 122.6 = 6.4
95 - 71.87 = 23.13
37.9 - 37.79 = 0.113
133 - 122.6 = 10.4
82 - 71.87 = 10.13
38.3 - 37.79 = 0.513
138 - 122.6 = 15.4
73 - 71.87 = 1.13
38.4 - 37.79 = 0.613
This is the global mean (M) not M1!

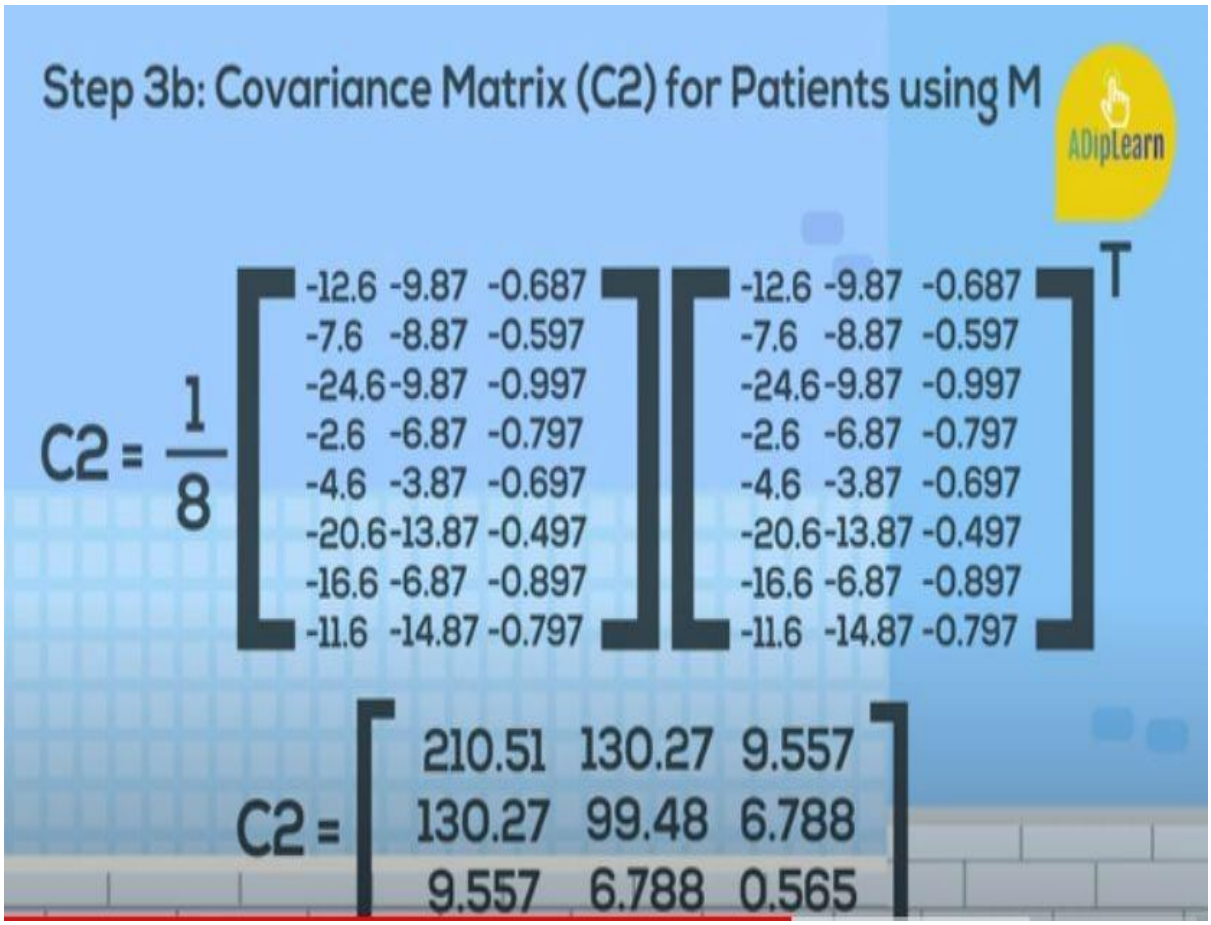
Step 2b: Covariance Matrix (C1) for Patients using M



Step 3a: Compute the Mean (M2) for Non-Patients

Blood Pressure	Pulse Rate	Temperature
110	62	37.1
115	63	37.2
98	62	36.8
120	65	37.0
118	68	37.1
102	58	37.3
106	65	36.9
111	57	37.0


Step 3b: Covariance Matrix (C2) for Non-Patients using M
Blood Pressure
Pulse Rate
Temperature
110 - 122.6
62 - 71.87
37.1 - 37.79
115 - 122.6
63 - 71.87
37.2 - 37.79
98 - 122.6
62 - 71.87
36.8 - 37.79
120 - 122.6
65 - 71.87
37.0 - 37.79
118 - 122.6
68 - 71.87
37.1 - 37.79
102 - 122.6
58 - 71.87
37.3 - 37.79
106 - 122.6
65 - 71.87
36.9 - 37.79
111 - 122.6
57 - 71.87
37.0 - 37.79
Settings
This is the global mean (M) not M2!


Step 3b: Covariance Matrix (C2) for Patients using M
ADiplearn
C2 = 1/8
-12.6 -9.87 -0.687
-7.6 -8.87 -0.597
-24.6 -9.87 -0.997
-2.6 -6.87 -0.797
-4.6 -3.87 -0.697
-20.6 -13.87 -0.497
-16.6 -6.87 -0.897
-11.6 -14.87 -0.797
T
C2 =
210.51 130.27 9.557
130.27 99.48 6.788
9.557 6.788 0.565

Step 4: Within-Class Scatter Matrix (C)

ADipLearn

$$C1 = \begin{bmatrix} 229.65 & 140.01 & 13.09 \\ 140.01 & 174.27 & 8.90 \\ 13.09 & 8.90 & 1.08 \end{bmatrix} \quad C2 = \begin{bmatrix} 210.51 & 130.27 & 9.557 \\ 130.27 & 99.48 & 6.788 \\ 9.557 & 6.788 & 0.565 \end{bmatrix}$$

$$C = \frac{7}{7+8} \times C1 + \frac{8}{7+8} \times C2 = \begin{bmatrix} 219.44 & 134.82 & 11.208 \\ 134.82 & 134.38 & 7.772 \\ 11.208 & 7.772 & 0.804 \end{bmatrix}$$

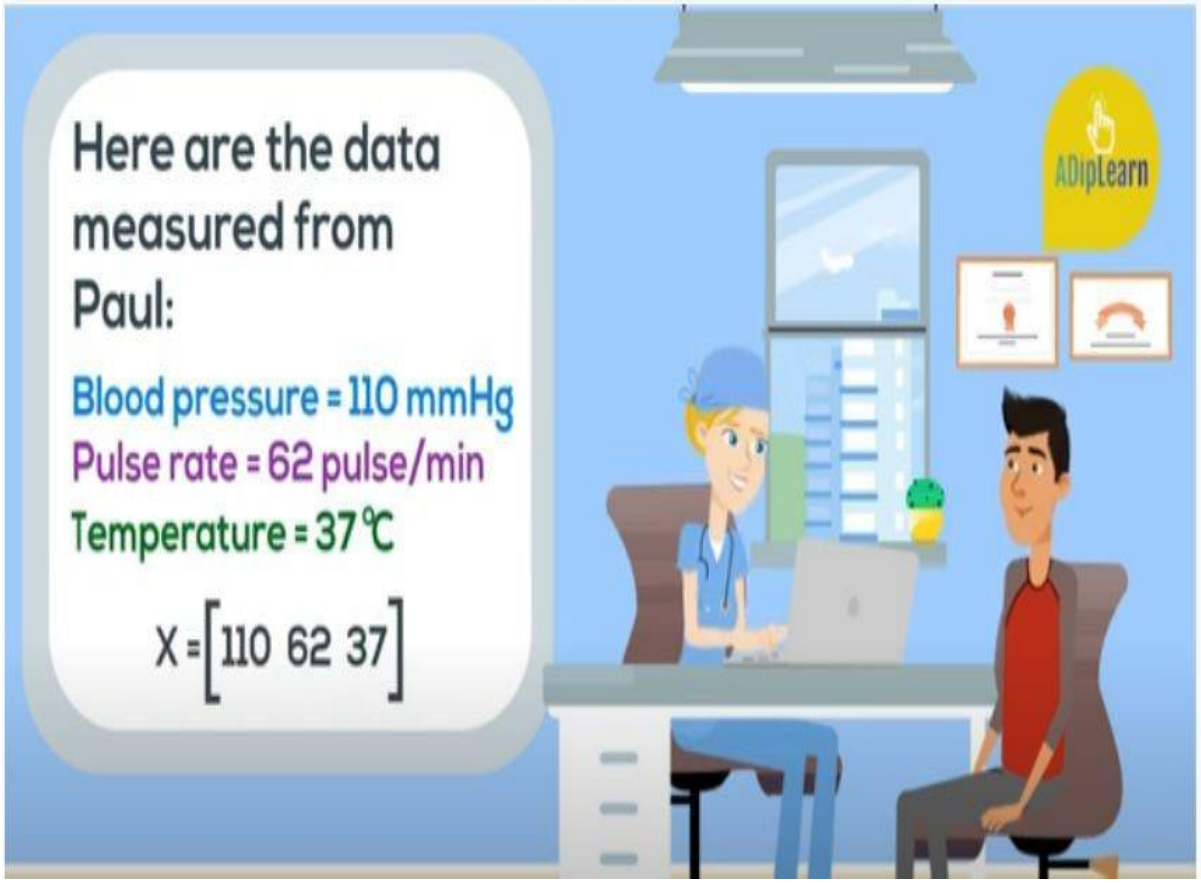
Here are the data measured from Paul:

Blood pressure = 110 mmHg

Pulse rate = 62 pulse/min

Temperature = 37 ℃

$$X = \begin{bmatrix} 110 & 62 & 37 \end{bmatrix}$$



Step 5a: Compute F1 for Patients

$$F1 = M1 \cdot C^{-1} \cdot X^{T} - 0.5 \cdot M1 \cdot C^{-1} \cdot M1^{T} + \ln(P1)$$

$$= 4679.31 - 0.5 \times 4722.57 + \ln\left[\frac{7}{7+8}\right]$$

$$= 2317.26$$

Step 5b: Compute F2 for Non-Patients

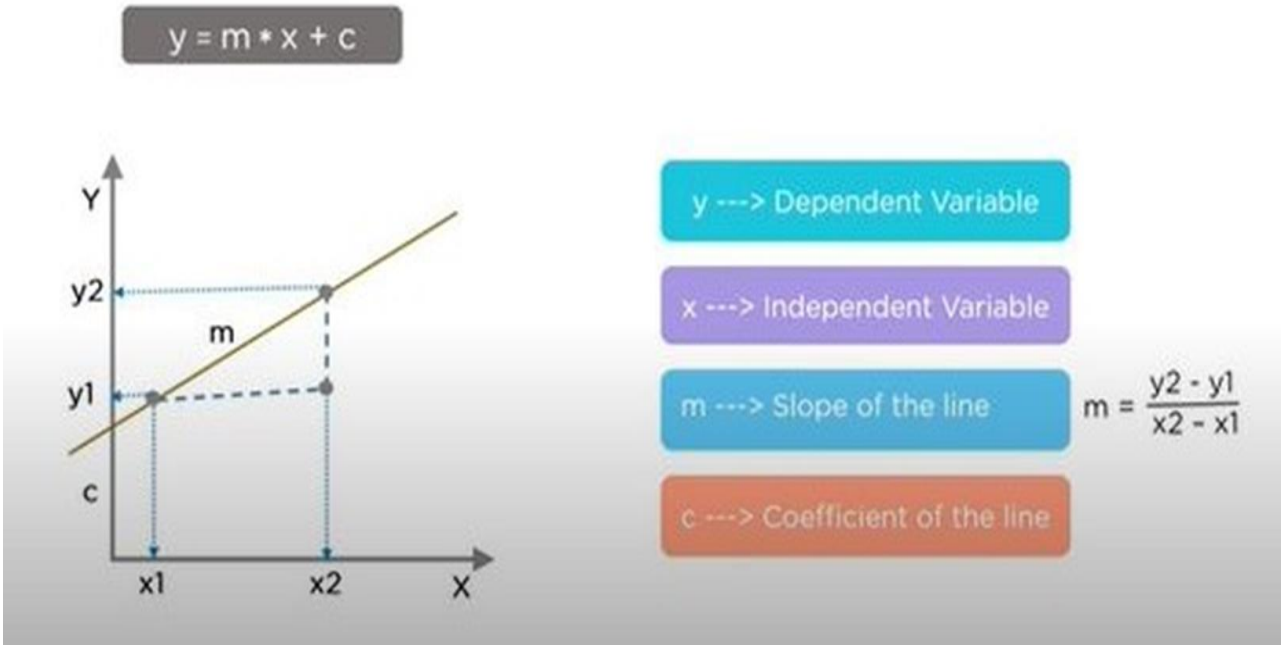
$$M2 \cdot C^{-1} \cdot X^{T} = \begin{bmatrix} 110 & 62.5 & 37.05 \end{bmatrix} \begin{bmatrix} 219.44 & 134.82 & 11.208 \\ 134.82 & 134.38 & 7.772 \\ 11.208 & 7.772 & 0.804 \end{bmatrix}^{-1} \begin{bmatrix} 110 & 62 & 37 \end{bmatrix}^{T}$$

$$= 4645.81$$

$$M2 \cdot C^{-1} \cdot M2^{T} = \begin{bmatrix} 110 & 62.5 & 37.05 \end{bmatrix} \begin{bmatrix} 219.44 & 134.82 & 11.208 \\ 134.82 & 134.38 & 7.772 \\ 11.208 & 7.772 & 0.804 \end{bmatrix}^{-1} \begin{bmatrix} 110 \\ 62.5 \\ 37.05 \end{bmatrix}$$

Linear Regression Analysis

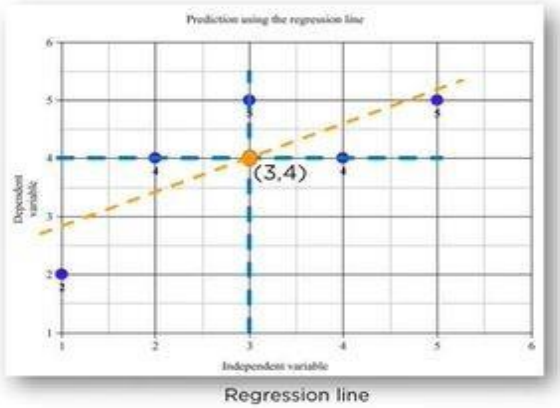
- Linear regression that attempts to show the relationship between two variables with the linear equation.
- The simples form of a simple linear regression equation with one dependent and one independent variable is represented by



Intuition behind the Regression line

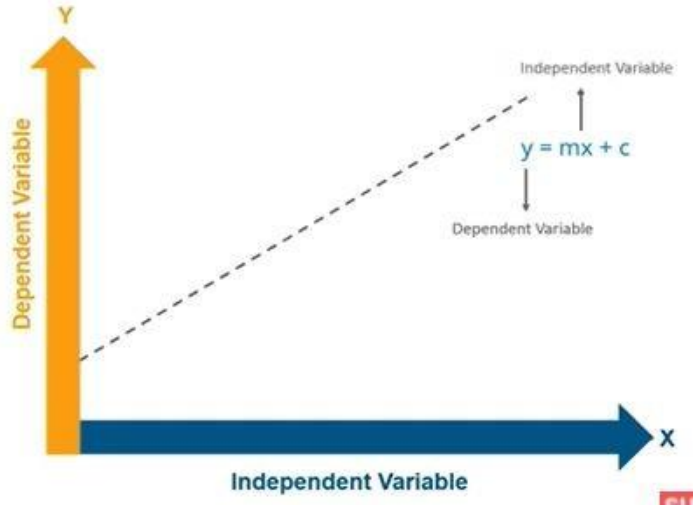
Regression line should ideally pass through the mean of X and Y

Independent variable	Dependent variable
X	Y
1	2
2	4
3	5
4	4
5	5
Mean	3	4



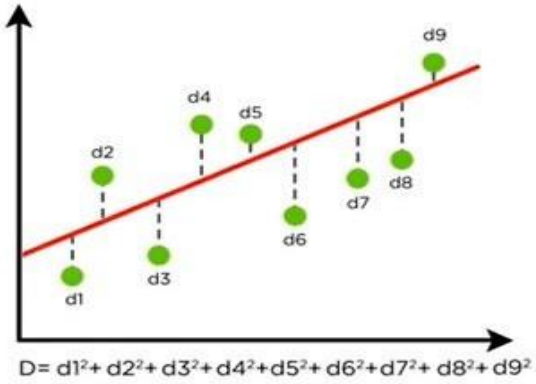
Regression line

Understanding Linear Regression Algorithm



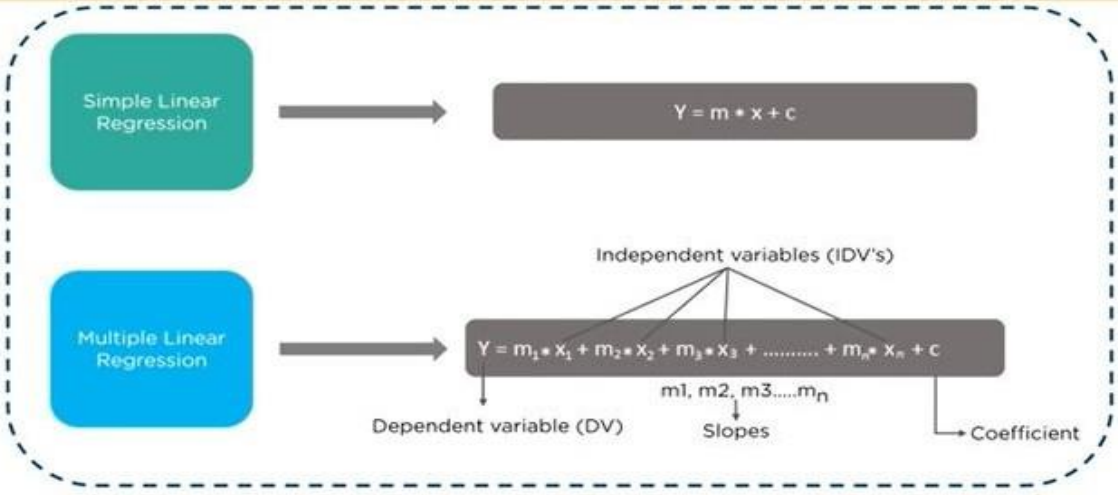
Finding the Best fit line

Minimizing the Distance: There are lots of ways to minimize the distance between the line and the data points like Sum of Squared errors, Sum of Absolute errors, Root Mean Square error etc.



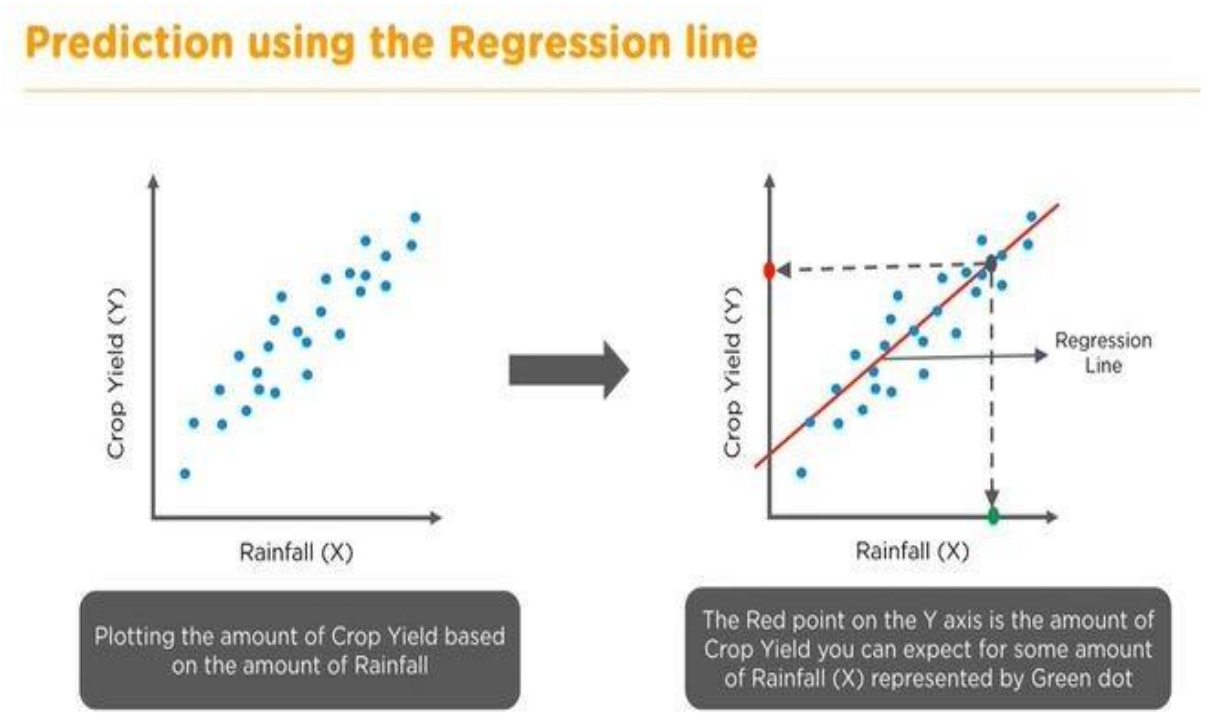
We keep moving this line through the data points to make sure the Best fit line has the least square distance between the data points and the regression line

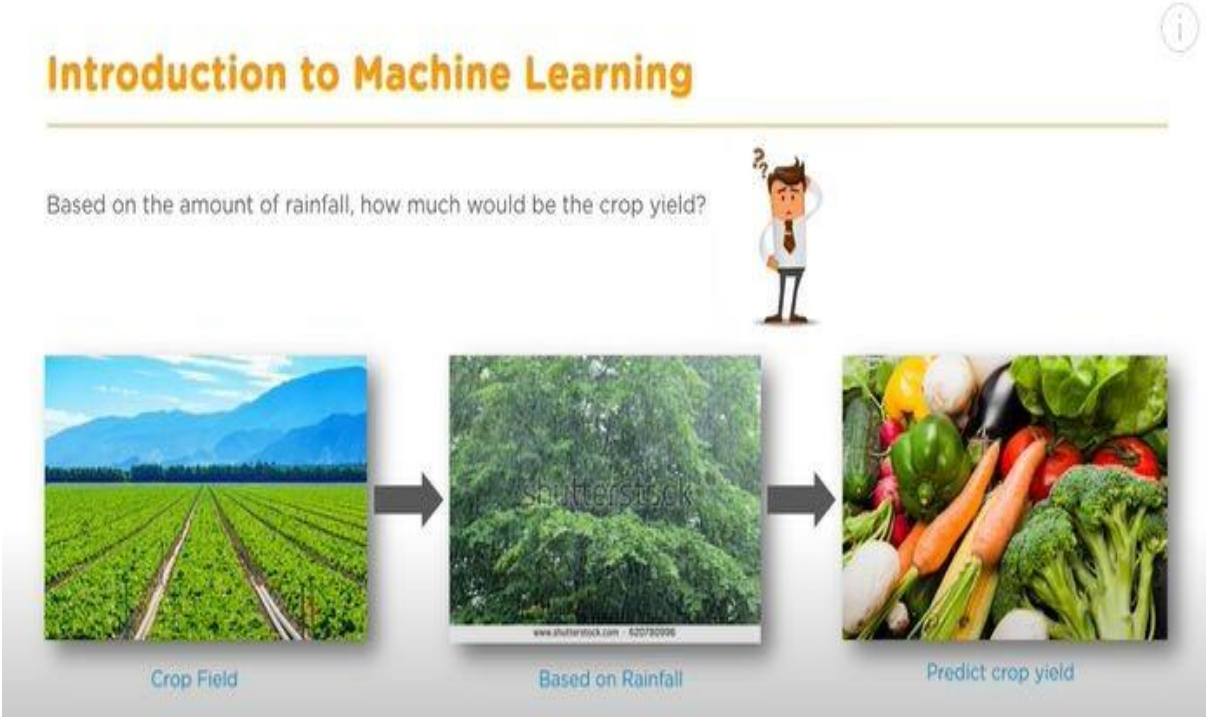
Multiple Linear Regression



- Product price- Sale
- Height-Weight
- Temperature-Ice Cream sales
- Process- RAM

Linear Regression formula: y=mx+b





Independent and Dependent Variables

Independent variable

A variable whose value does not change by the effect of other variables and is used to manipulate the dependent variable. It is often denoted as **X**.

Dependent variable

A variable whose value change when there is any manipulation in the values of independent variables. It is often denoted as **Y**.

In our example:

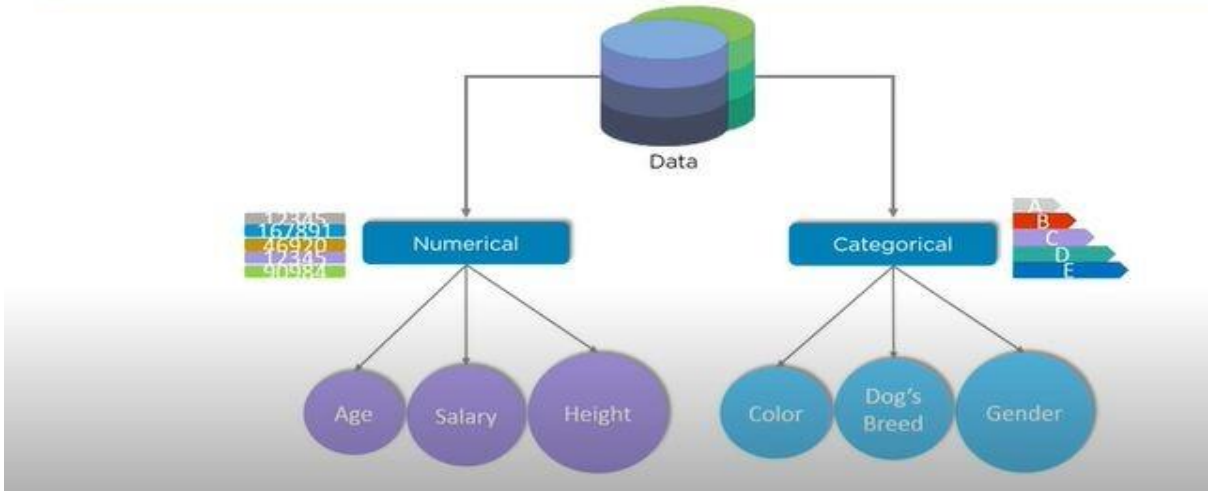
Rainfall – Independent variable

Crop yield depends on the amount of rainfall received

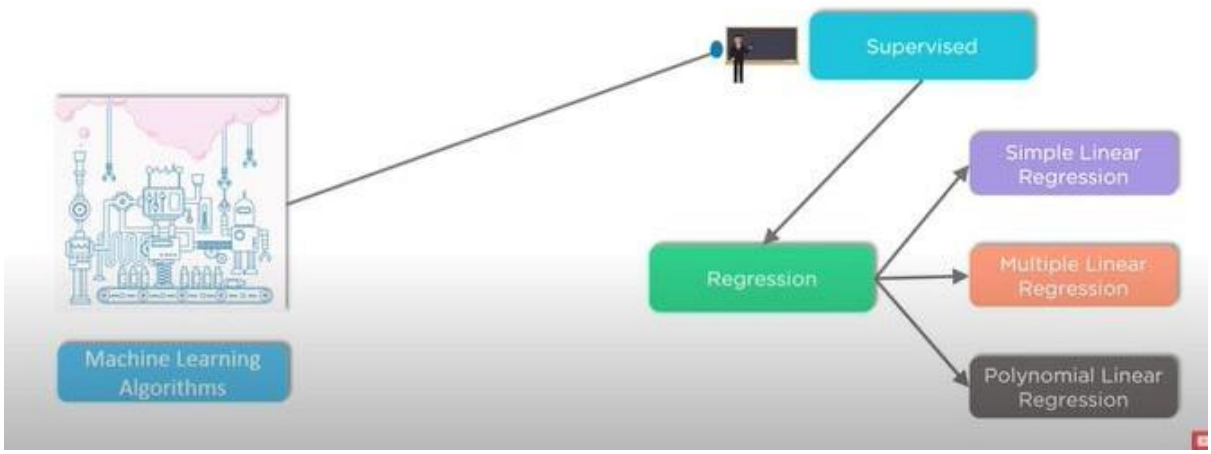
Crop yield – Dependent variable

Suggested: What is Machine Learning? | Machine Learning Tutorial

Numerical and Categorical values



Machine Learning Algorithms



Applications of Linear Regression

Economic Growth

Used to determine the Economic Growth of a country or a state in the coming quarter, can also be used to predict the GDP of a country

Applications of Linear Regression

Product price

Can be used to predict what would be the price of a product in the future

Applications of Linear Regression

Housing sales

To estimate the number of houses a builder would sell and at what price in the coming months

Applications of Linear Regression

Score Prediction

To predict the number of runs a player would score in the coming matches based on previous performance

Understanding Linear Regression

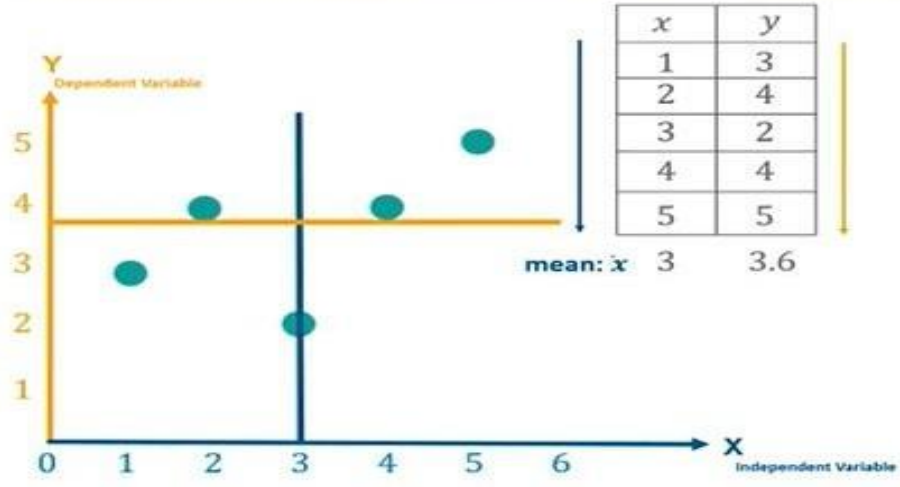
Linear Regression is a statistical model used to predict the relationship between independent and dependent variables.

Examine 2 factors

1. Which variables in particular are significant predictors of the outcome variables?

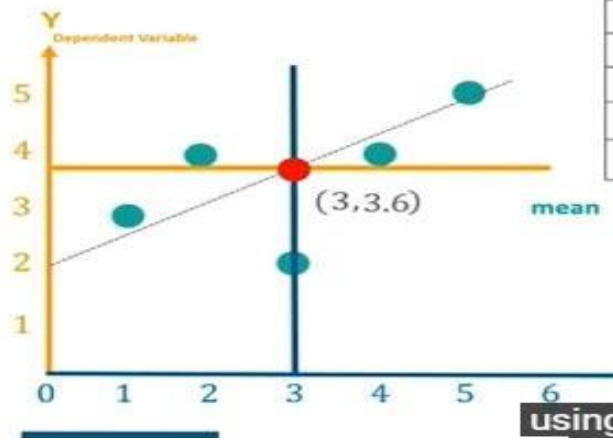
2. How significant is the Regression line to make predictions with highest possible accuracy

Understanding Linear Regression Algorithm



x	y
1	3
2	4
3	2
4	4
5	5
mean: $\bar{x}$ 3	3.6

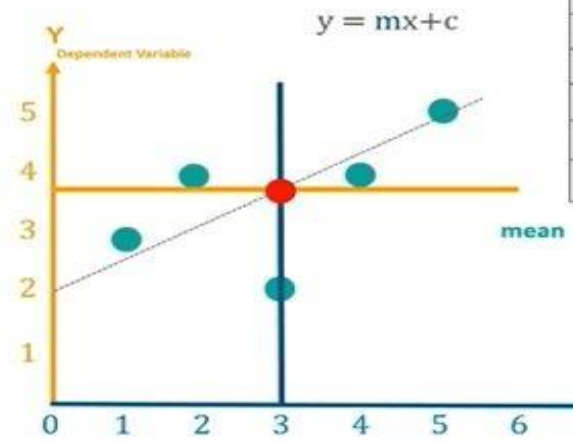
Understanding Linear Regression Algorithm



x	y
1	3
2	4
3	2
4	4
5	5

using the least Square

Understanding Linear Regression Algorithm

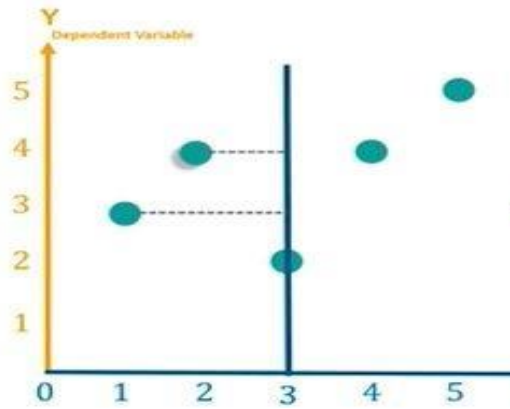


x	y
1	3
2	4
3	2
4	4
5	5

$$m = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

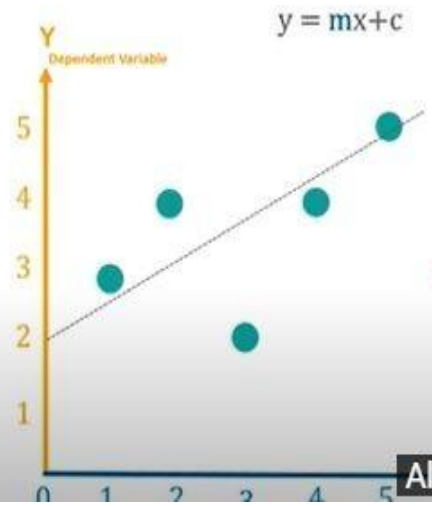
where m equals summation of x

Understanding Linear Regression Algorithm



x	y	$x - \bar{x}$
1	3	-2
2	4	$2 - 3$
3	2	
4	4	
5	5	
3	3.6	

Understanding Linear Regression Algorithm

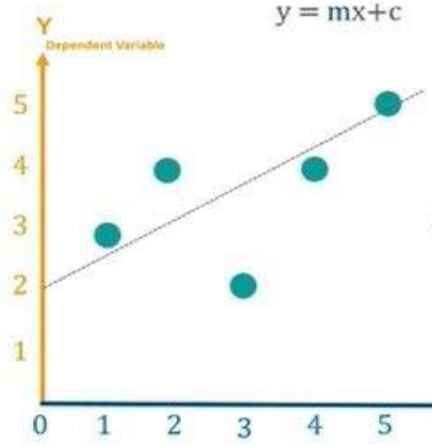


x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
mean	3	3.6		$\Sigma = 10$	$\Sigma = 4$

$$m = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2}$$

All right, so the summation

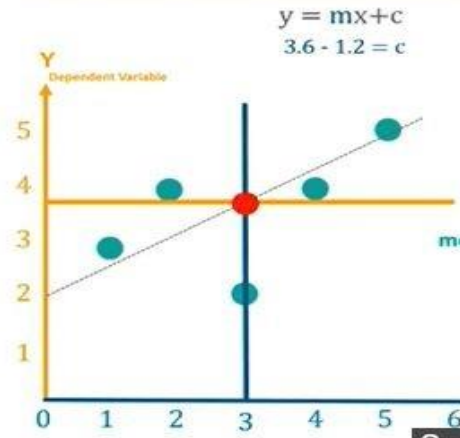
Understanding Linear Regression Algorithm



x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
mean	3	3.6		$\Sigma = 10$	$\Sigma = 4$

$$m = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2} = \frac{4}{10}$$

Understanding Linear Regression Algorithm



x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
mean	3	3.6		$\Sigma = 10$	$\Sigma = 4$

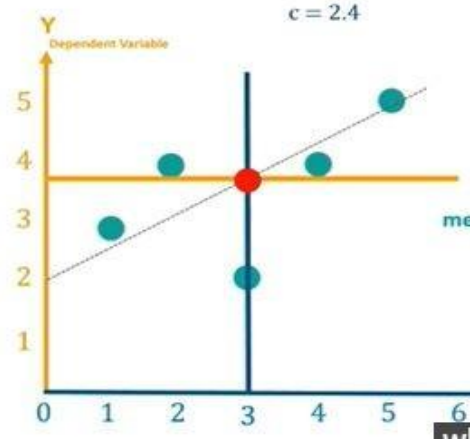
$$m = \sum \frac{(x-\bar{x})(y-\bar{y})}{(x-\bar{x})^2} = \frac{4}{10}$$

So what is the value of C

Understanding Linear Regression Algorithm

$y = mx+c$

$c = 2.4$



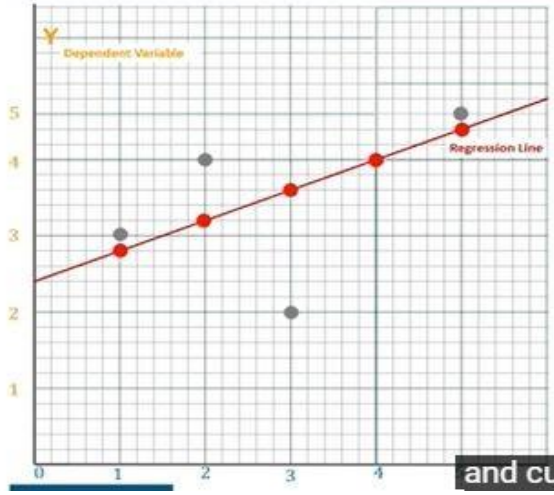
x	y	$x-\dot{x}$	$y-\dot{y}$	$(x-\dot{x})^2$	$(x-\dot{x})(y-\dot{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
mean	3	3.6		$\Sigma = 10$	$\Sigma = 4$

$$m = \sum \frac{(x-\dot{x})(y-\dot{y})}{(x-\dot{x})^2} = \frac{4}{10}$$

$m = 0.4$
$c = 2.4$
$y = 0.4x + 2.4$

what we get is y equal

Mean Square Error



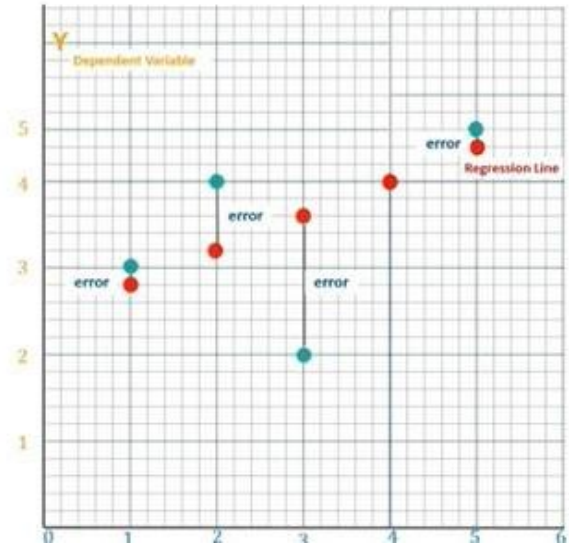
and cutting y-axis

$m = 0.4$
$c = 2.4$
$y = 0.4x + 2.4$

For given $m = 0.4$ & $c = 2.4$, lets predict values for y for $x = \{1,2,3,4,5\}$

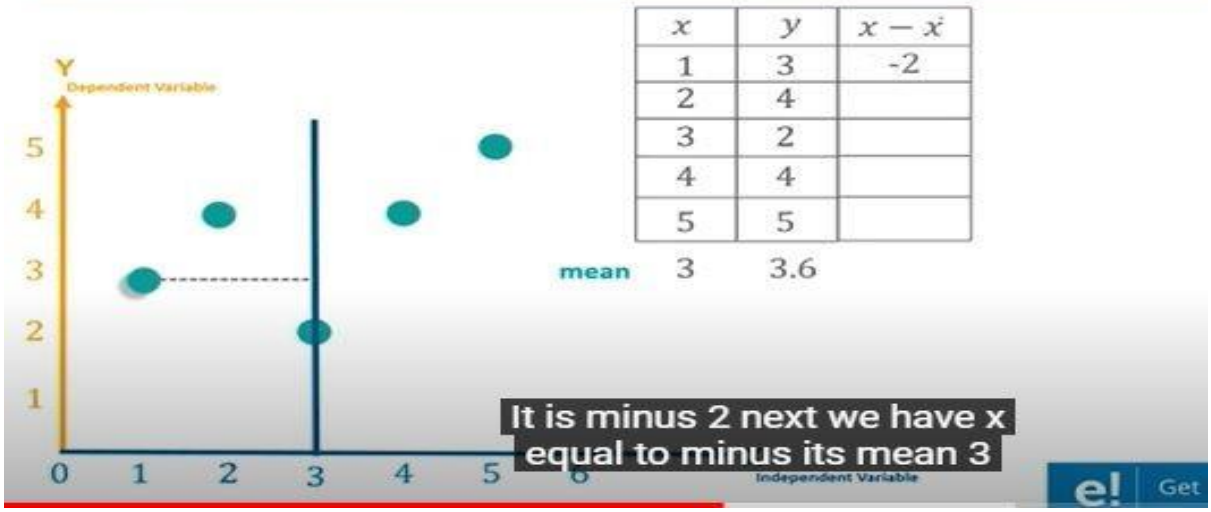
$y = 0.4 \times 1 + 2.4 = 2.8$
$y = 0.4 \times 2 + 2.4 = 3.2$
$y = 0.4 \times 3 + 2.4 = 3.6$
$y = 0.4 \times 4 + 2.4 = 4.0$
$y = 0.4 \times 5 + 2.4 = 4.4$

Mean Square Error



Distance between actual & predicted value

Understanding Linear Regression Algorithm



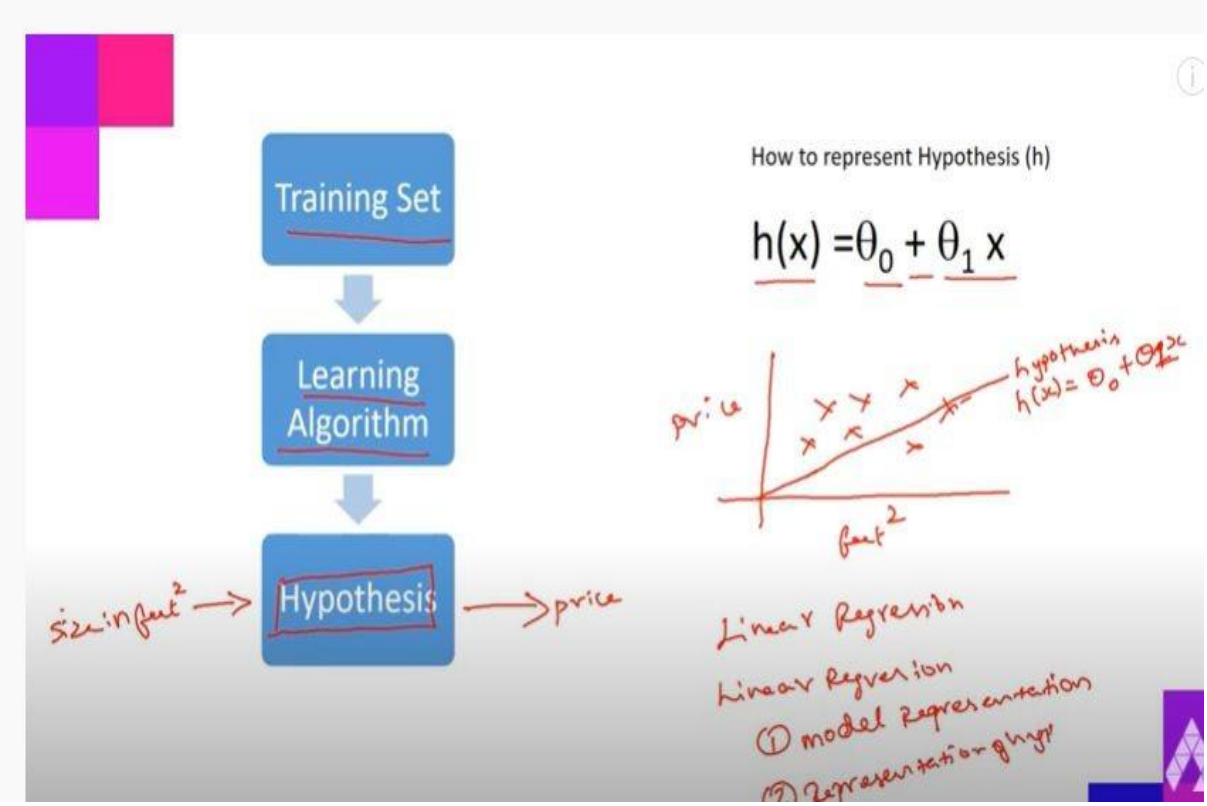
x	y	$x - \bar{x}$
1	3	-2
2	4	
3	2	
4	4	
5	5	
mean	3	3.6

Model Representation

Model Representation

Model Representation

Model Representation



1. Why was Machine Learning Introduced? ...
2. What are Different Types of Machine Learning algorithms? ...
3. What is Supervised Learning? ...
4. What is Unsupervised Learning? ...
5. How is neuroscience related to machine learning?
6. How does machine learning work similar to a brain?
7. What is the main difference between human brain and a computer?
8. What is concept learning task in machine learning?
9. What is hypothesis concept learning?
10. What are the objectives of machine learning?
11. What is the goal of concept learning Search task?
12. Solved Numerical Example (Candidate Elimination Algorithm):

Example	Citations	Size	InLibrary	Price	Editions	Buy
1	Some	Small	No	Affordable	One	No
2	Many	Big	No	Expensive	Many	Yes
3	Many	Medium	No	Expensive	Few	Yes
4	Many	Small	No	Affordable	Many	Yes

1. What is linear discriminant analysis in machine learning?
2. What is the purpose of linear discriminant analysis?
3. How do you do linear discriminant analysis? What is a Linear Regression?
4. Can you list out the critical assumptions of linear regression?
5. What is Heteroscedasticity?
6. What is the primary difference between R square and adjusted R square?
7. Can you list out the formulas to find RMSE and MSE?

UNIT – II LINEAR MODELS

Multi – Layer Perceptron in Practice

To solve real problems, MLP find solutions to four different types of problem: regression, classification, time-series prediction, and data compression.

2.1 Amount of Training Data

- For the MLP - (L+1)×M +(M +1)×N weights, where L,M,N are the number of nodes in the input, hidden, and output layers, respectively.
- +1s -> bias nod es, which is **adjustable weights**. Huge number of adjustable parameters that we need to set during the training phase.
- Setting the values of adjustable weights is the job of the back-propagation algorithm, which is driven by the errors coming from the training data.
- More training data is the better for learning, algorithm takes to learn increases.
- Minimum amount of data required is, it depends on the problem.
- Use a number of training examples is 10 times the number of weights.
- If MLP has large number of examples, so neural network training has expensive operation
- Two hidden layers is the need for normal MLP learning. This result can be strengthened by showing mathematically that one hidden layer with lots of hidden nodes is sufficient. This is known as the Universal Approximation Theorem
- Training networks with different numbers of hidden nodes and then choosing the one that gives the best results.
- Using back-propagation algorithm for a network with as many layers, it harder to keep track of which weights are being update.
- The basic idea is that by combining sigmoid functions we can generate ridge-like functions, and by combining ridge-like functions, generate functions with a unique maximum.
- By combining these and transforming them using another layer of neurons, we obtain localized

response (a ‘bump’ function), and any functional mapping can be approximated to arbitrary accuracy using a linear combination of such bumps.

- Two hidden layers are sufficient to compute these bump functions for different inputs, and so if the function learns (approximate) is continuous, the network can compute it.

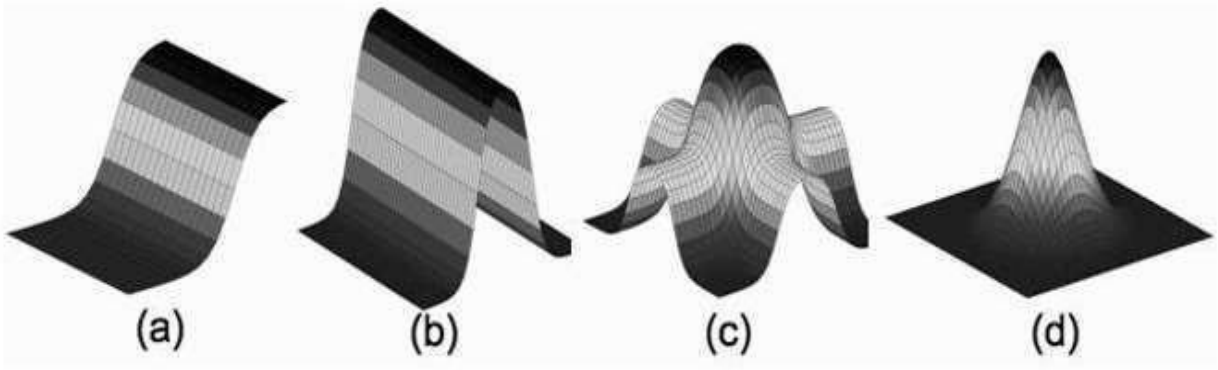


FIGURE 4.9 The learning of the MLP can be shown as the output of a single sigmoidal neuron (a), which can be added to others, including reversed ones, to get a hill shape (b). Adding another hill at 90° produces a bump (c) , which can be sharpened to any extent we want (d), with the bumps added together in the output layer. Thus the MLP learns a local representation of individual inputs.

2.3 When to Stop Learning

- The training of the MLP requires that the algorithm runs over the entire data set many times, with the weights changing as the network makes errors in each iteration.
- Set predefined N number of iteration, the network has overfitted (Overfitting (overtraining): when the NN learns too many I/O examples it may end up memorizingthe training data) or learn sufficiently and when it stops, some predefined minimum error is reached that means algorithm never terminates. Sum of squares errors during training
- At some stage the error on the validation set will start increasing again, because the network has stopped learning about the function that generated the data, andstarted to learn about the noise. At this stage we stop the training. This technique is called early stopping

When a model fits more data than it needs, it starts catching the noisy data and inaccurate values in the data. As a result, the efficiency and accuracy of the model decrease.

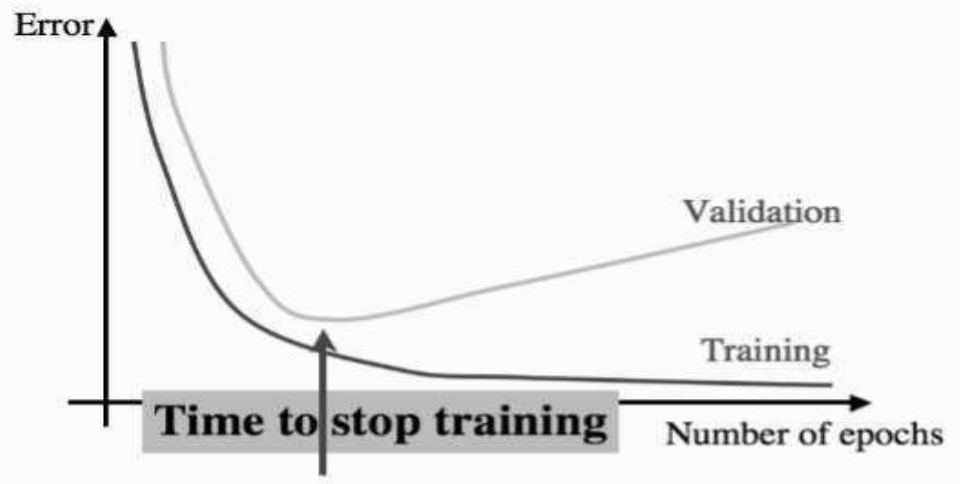


FIGURE 4.11 The effect of overfitting on the training and validation error curves, with the point at which early stopping will stop the learning marked.

Cross-Validation Method

- **Early-stopping Method**
- **(Holdout method)**
  - The training is stopped periodically, i.e., after so many epochs, and the network is assessed using the validation subset.
  - When the validation phase is complete, the estimation (training) is resumed for another period, and the process is repeated.
  - The best model (parameters) is that at the minimum validation error.

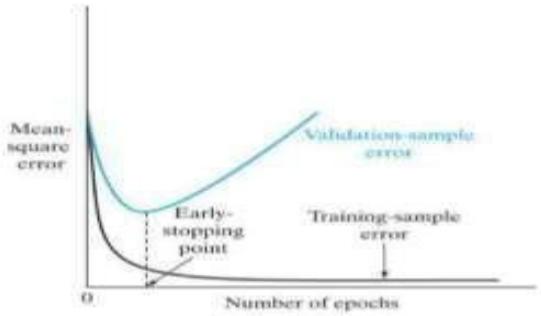


Figure 4.17 Illustration of the early-stopping rule based on cross-validation.

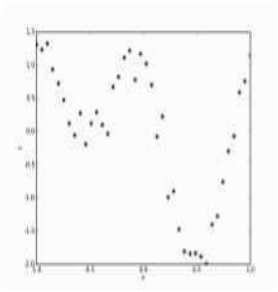


FIGURE 4.12 The data that we will learn using an MLP, consisting of some samples from a sine wave with Gaussian noise added.

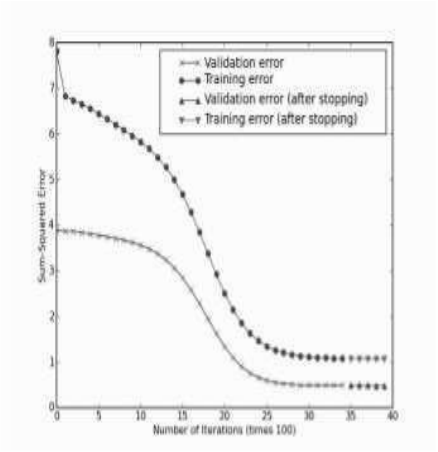


FIGURE 4.13 Plot of the error as the MLP learns (top line is total error on the training set; bottom line is on the validation set; it is larger on the training set because there are more datapoints in this set). Early-stopping halts the learning at the point where there is no line, where the crosses become triangles. The learning was continued to show that the error got slightly worse afterwards.

A Regression Problem: Find the values of any inputs and train the data Function –sin wave

Train an MLP on the data. There is one input value, x and one output value t, so the neural network will have one input and one output. Before getting started, we need to normalise the data, and then separate the data into training, testing, and validation sets. In given example there are only 40 datapoints and use half of them as the training set Split the data in the ratio 50:25:25 by using the odd-numbered elements as training data, the even-numbered ones that do not divide by 4 for testing, and the rest for validation Construct a network with three nodes in the hidden layer, and run it for 101 iterations with a learning rate of 0.25. The output: Iteration: 0 Error: 12.3704163654

Iteration: 100 Error: 8.2075961385 so that the network is learning, since the error is decreasing.

To do two things: how many hidden nodes we need, and decide how long to train the network for.

In order to solve the first problem, need to test out different networks and see which get lower errors, but to do that properly need to know when to stop training. Solve the second problem first, which is to implement early stopping. keep track of the validation error and stop when it starts to increase.

2.3.2 Classification with the MLP

$$\text{Class is: } \begin{cases} C_1 & \text{if } y \leq -0.5 \\ C_2 & \text{if } -0.5 < y \leq 0 \\ C_3 & \text{if } 0 < y \leq 0.5 \\ C_4 & \text{if } y > 0.5 \end{cases}$$

Figure gives an example of the output of running the function.

It plots the training and validation errors.

- The point at which early stopping makes the learning finish is the point where there is a missing validation datapoint.

- The validation error did not improve after that, and so early stopping found the correct point .
  - Problem of finding the right size of network.
  - Each network size is run 10 times, and the average is monitored.
  - The following table shows the results of doing this, reporting the sum-of-squares validation error, for a few different sizes of network:

No. of hidden nodes	1	2	3	5	10	25	50
Mean error	2.21	0.52	0.52	0.52	0.55	1.35	2.56
Standard deviation	0.17	0.00	0.00	0.02	0.00	1.20	1.27
Max error	2.31	0.53	0.54	0.54	0.60	3.230	3.66
Min error	2.10	0.51	0.50	0.50	0.47	0.42	0.52

Couple of choices for the outputs.

- First-use a single linear node for the output, y, and put some thresholds on theactivation value of that node. For example, for a four-class problem,
- Close to a boundary, say y = 0.5? It belongs to class C3, close to the boundary in theoutput.
- A more suitable output encoding is called1-of-N encoding. e.g., (0,0,1,0) means that the correct resultis the 3rd class out of 4 Element yk of the output vector that is the largest element of y(in mathematical notation, pick the yk for which yk > yj Aj ≠ k;A means for all.

  Two output neurons will have identical largest output values.

  - This is known as the hard-max activation function (since the neuron with the highest activation is chosen to fire and the rest are ignored).
  - Two-class classification, 90% of our data belongs to class 1. (This scan happens: for example, in medical data, most tests are negativein general.)
  - There is an alternative solution, known as novelty detection, which is to train the data on the data in the negative class only, and to assume that anything that looks different to that is a positive example.

2.3.3 A Classification Example: The Iris Dataset

- Three types of iris (flower) by the length and width of the sepals and petals and is called iris.
- **stext1 = 'Iris-setosa' ,stext2 = 'Iris-versicolor' ,stext3 = 'Iris-virginica' .**
- Need to separate the data into training, testing, and validationsets.
- There are 150 examples in the dataset, and they are split evenly amongst the three classes, so the three classes are the same size.
- Split them into ½ training, and ¼ each testing and validation.50 are class1, 50class 2, etc.,

2.3.4 Time – Series Prediction

- Stock market, disease pattern, seasonal variation

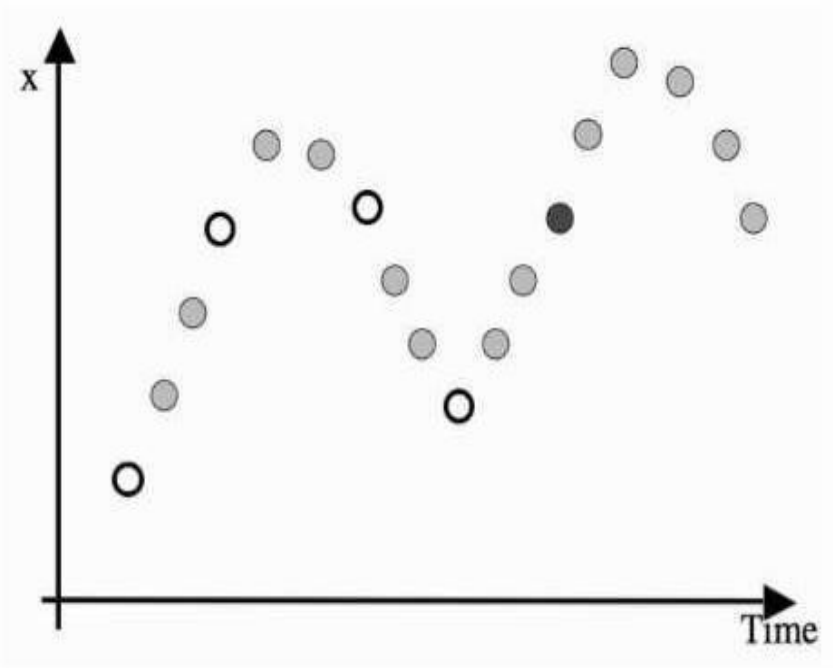


FIGURE 4.14 Part of a time-series plot, showing the datapoints and the meanings of τ and k.

Decision boundary

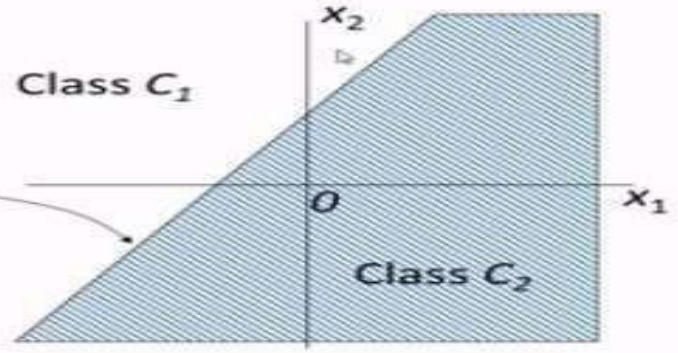
- The hyper-plane

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

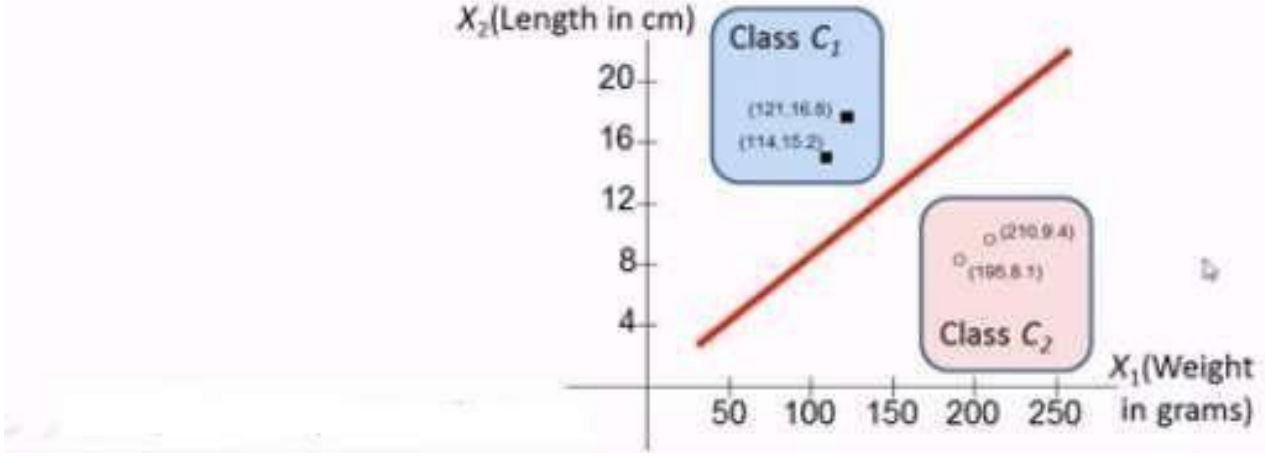
or

$$w_1 x_1 + w_2 x_2 + b = 0$$

is the decision boundary for a two class classification problem.



	Weight (grams)	Length (cm)
Fruit 1 (Class C1)	121	16.8
	114	15.2
Fruit 2 (Class C2)	210	9.4
	195	8.1



simplest form of a neural network used for the classification of patterns said to be linearly separable. Basically, it consists of a single neuron with adjustable synaptic weights and bias.

$$\left.\begin{array}{l} w_1(0) = -30,\ w_2(0) = 300, \\ b(0) = 50,\ \eta = 0.01 \end{array}\right\} \text{given}$$

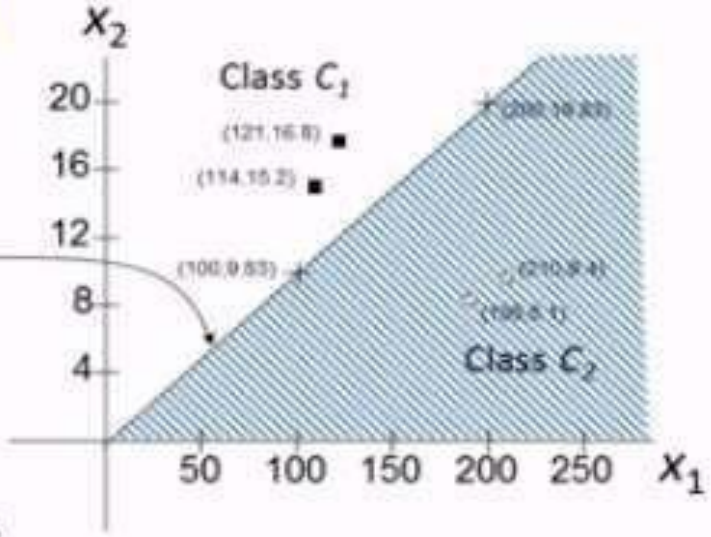
Therefore the Initial Decision Boundary for this example is:

$$w_1 x_1 + w_2 x_2 + b = 0$$

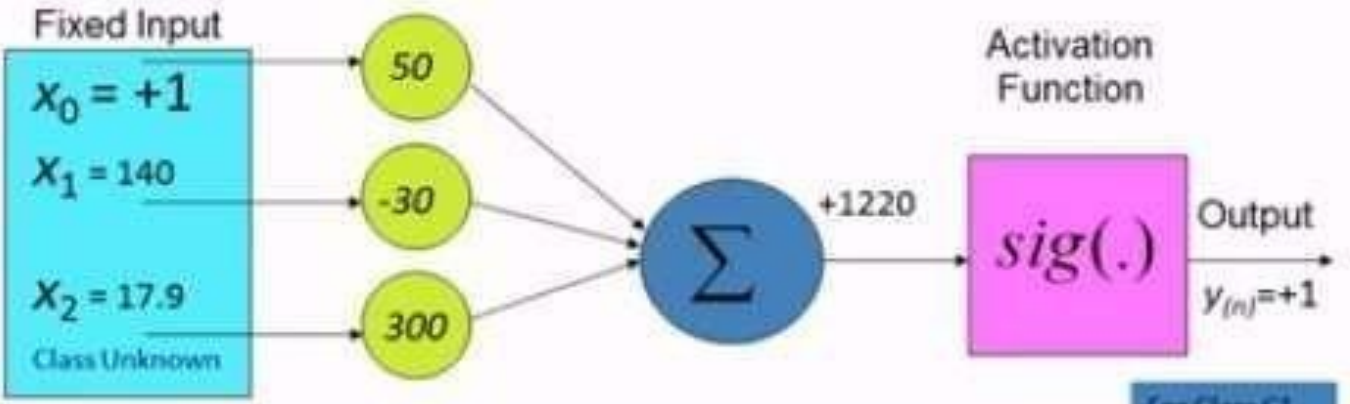
$$-30x_1 + 300x_2 + 50 = 0$$

$$x_1 = 100,\ x_2 = \frac{30 \times 100 - 50}{300} = 9.83$$

$$x_1 = 200,\ x_2 = \frac{30 \times 200 - 50}{300} = 19.83$$



Initial hyper-plane does separate the two classes.



Now use the above model to classify the unknown fruit.

For Class C1, Output = +1

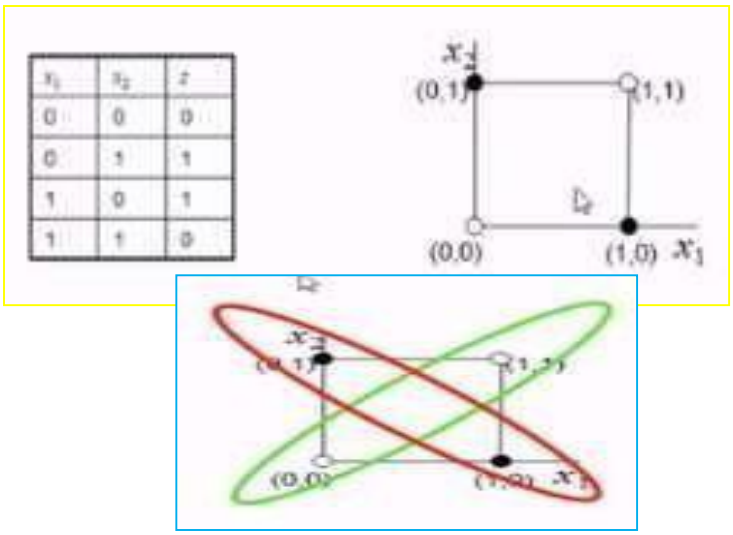
$$\mathbf{x}(\text{unknown}) = [+1, 140, 17.9]^T$$

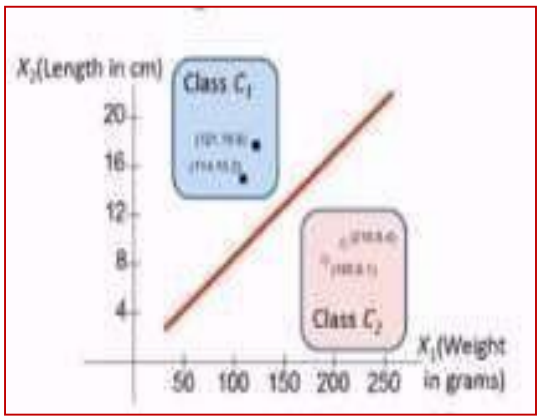
$$\mathbf{w}(3) = [50, -30, 300]^T$$

$$y(\text{unknown}) = \text{sgn}\left(\mathbf{w}^T(3)\mathbf{x}(\text{unknown})\right) = \text{sgn}(50 \times 1 - 30 \times 140 + 300 \times 17.9)$$

$$= \text{sgn}(1220) = +1$$

$\therefore$ this unknown fruit belongs to the class C_1.





If there is a solution to be found then the single layer perceptron learning algorithm will find it.

- It can separate classes that lie either side of a straight line easily.
- But in reality, division between classes are much more complex.
- Take for example the classical exclusive-or (XOR) problem.
- XOR logic function has two inputs and one output.
- It has limited set of functions Decision boundaries must be hyperplanes
- It can only perfectly separate linearly separable data

- We consider this as a problem in which we want the perceptron to learn to solve:
- Output 1 if x1 is on and x2 is off, or is x2 is on and x1 is off, otherwise output a 0.
- This appears a simple problem but there is no linear solution and this problem is linearly inseparable.

The XOR Problem

- **A two-layer Network to solve the XOR Problem**

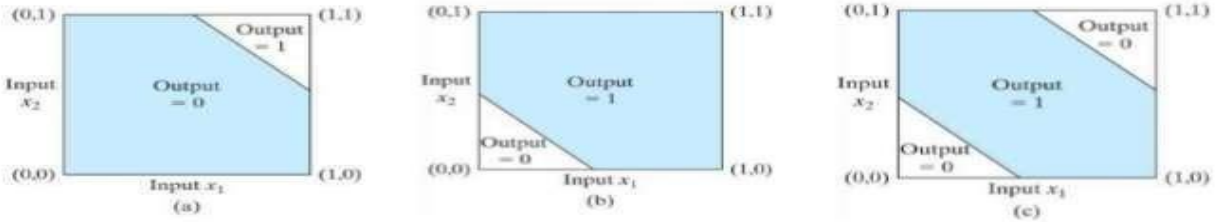
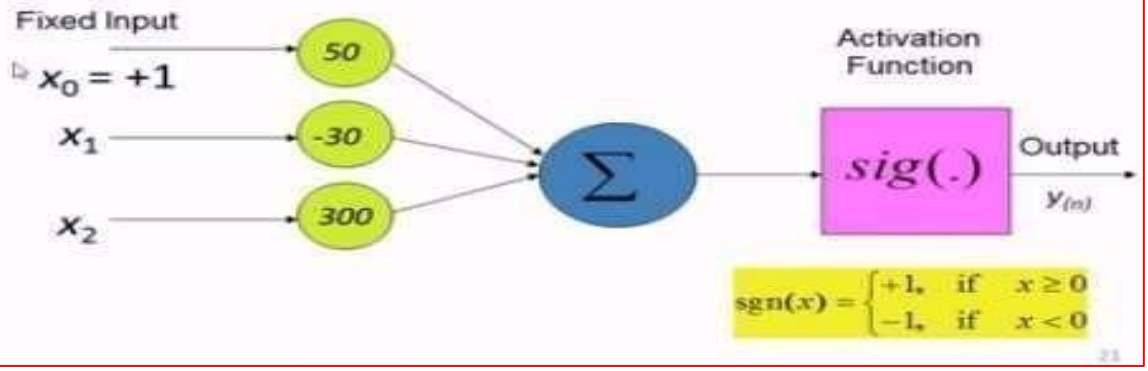


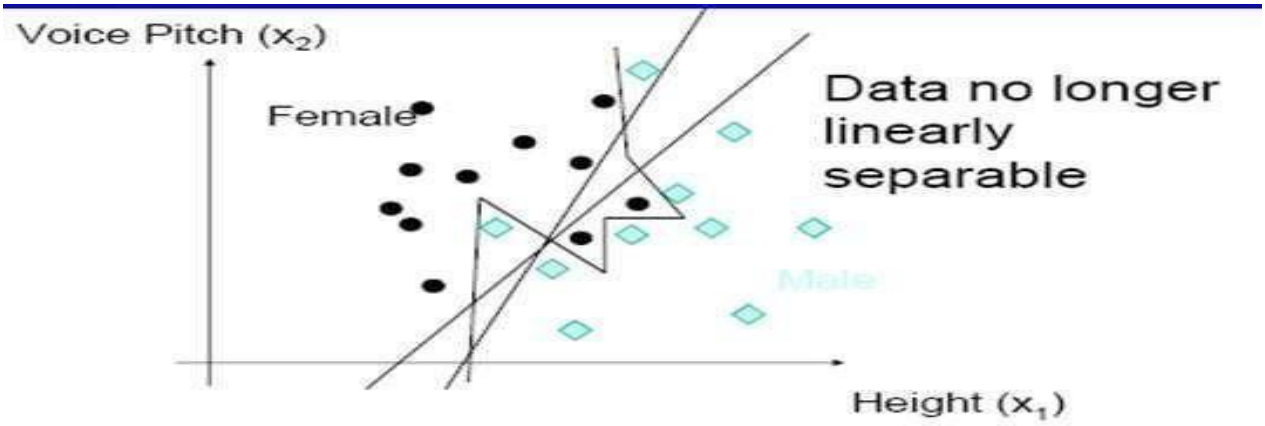
Figure 4.9 (a) Decision boundary constructed by hidden neuron 1 of the network in Fig. 4.8. (b) Decision boundary constructed by hidden neuron 2 of the network. (c) Decision boundaries constructed by the complete network.

Example single layer perceptron

With correct initial weights and bias

$$\left.\begin{array}{l} w_1(0) = -30,\ w_2(0) = 300, \\ b(0) = 50,\ \eta = 0.01 \end{array}\right\} \text{given}$$

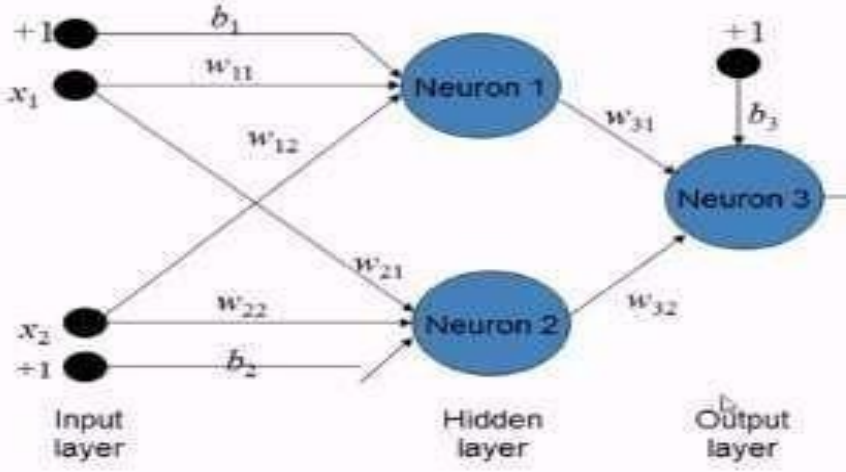




What is a good decision boundary ?

XOR Problem – with two layers

Now consider the following network with two inputs, 1 hidden layer and 1 output layer.



x_1	x_2	z
0	0	0
0	1	1
1	0	1
1	1	0

(1) $w_{11}= w_{12} = w_{21} = w_{22} = w_{32} = +1$, $w_{31} =-2$, $b_1=-1.5$ and $b_2=b_3=-0.5$

For [0,1] input:

$$w_H(n)=\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\0\\1\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix}^T\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-1.5 & 1 & 1\\ -0.5 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi\begin{bmatrix}-0.5\\0.5\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & -2 & 1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi[0.5]=[1]$$

For [0,0] input:

$$w_H(n)=\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix}^T\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-1.5 & 1 & 1\\ -0.5 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi\begin{bmatrix}-1.5\\-0.5\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & -2 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi[-0.5]=[0]$$

For [1,0] input:

$$w_H(n)=\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\1\\0\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix}^T\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-1.5 & 1 & 1\\ -0.5 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi\begin{bmatrix}-0.5\\0.5\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & -2 & 1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi[0.5]=[1]$$

(2) $w_{11}= w_{12} = w_{21} = w_{22} = w_{32} = -1$, $w_{31} =1$, $b_1=1.5$, $b_2=0.5$ and $b_3=-0.5$

For [0,1] input:

$$w_H(n)=\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\0\\1\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix}^T\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}1.5 & -1 & -1\\ 0.5 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\right)=\varphi\begin{bmatrix}0.5\\-0.5\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & 1 & -1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi[0.5]=[1]$$

For [1,1] input:

$$w_H(n)=\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}-1.5 & -0.5\\ 1 & 1\\ 1 & 1\end{bmatrix}^T\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-1.5 & 1 & 1\\ -0.5 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi\begin{bmatrix}0.5\\1.5\end{bmatrix}=\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & -2 & 1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi[-1.5]=[0]$$

For [0,0] input:

$$w_H(n)=\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix}^T\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}1.5 & -1 & -1\\ 0.5 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi\begin{bmatrix}1.5\\0.5\end{bmatrix}=\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & 1 & -1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi[-0.5]=[0]$$

For [1,0] input:

$$w_H(n)=\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\1\\0\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix}^T\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}1.5 & -1 & -1\\ 0.5 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi\begin{bmatrix}0.5\\-0.5\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & 1 & -1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\right)=\varphi[0.5]=[1]$$

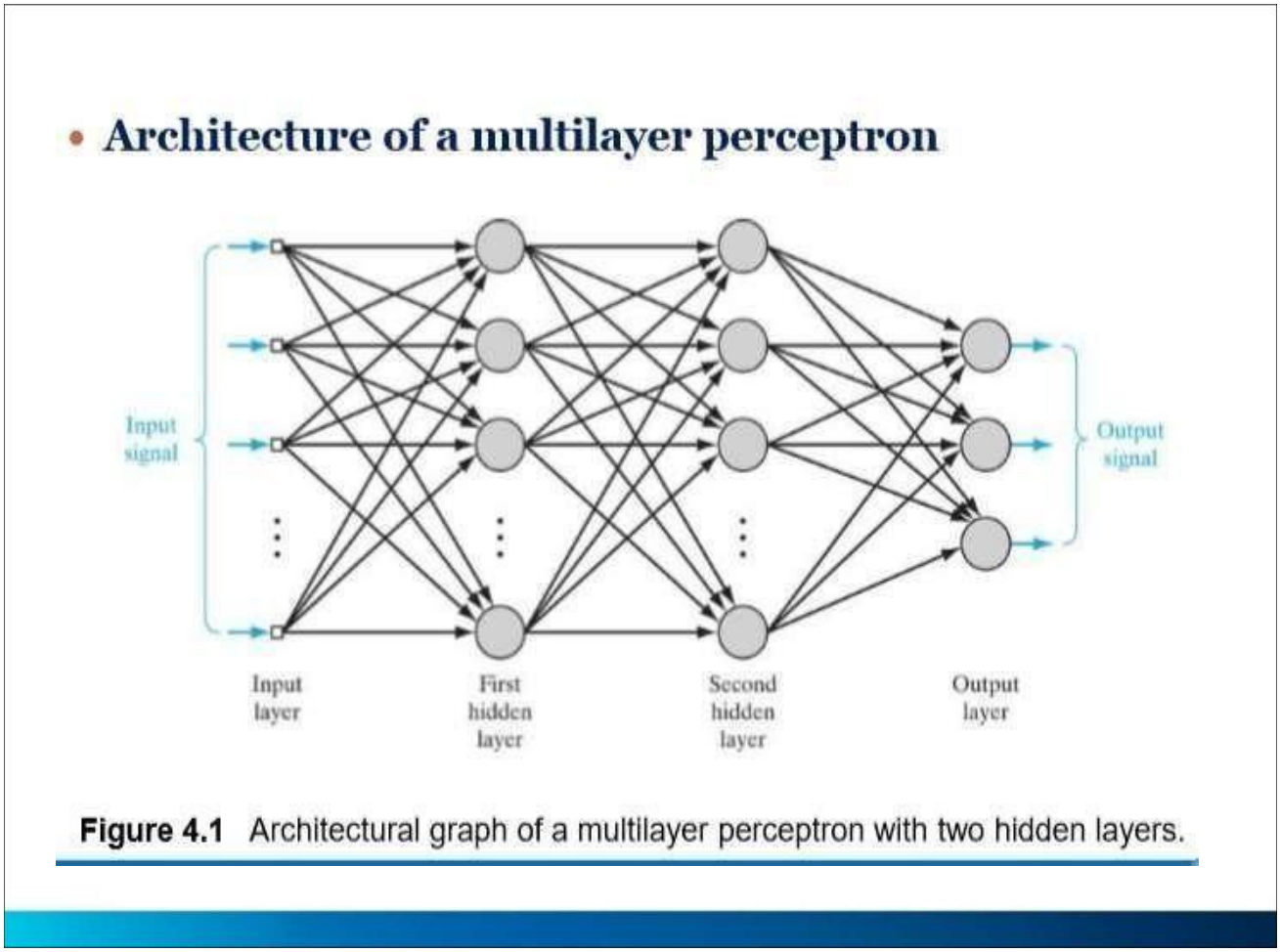
For [1,1] input:

$$w_H(n)=\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix} \qquad x(n)=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

$$y^H(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}1.5 & 0.5\\ -1 & -1\\ -1 & -1\end{bmatrix}^T\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}1.5 & -1 & -1\\ 0.5 & -1 & -1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\right)=\varphi\begin{bmatrix}-0.5\\-1.5\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}=\begin{bmatrix}y_1^H\\y_2^H\end{bmatrix}$$

$$y^o(n)=\varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]=\varphi\left(\begin{bmatrix}b_3\\w_{31}\\w_{32}\end{bmatrix}^T\begin{bmatrix}1\\y_1^H\\y_2^H\end{bmatrix}\right)=\varphi\left(\begin{bmatrix}-0.5 & 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)=\varphi[-0.5]=[0]$$

Multi-Layer Perceptron(MLP)



- ✓ MLP have been applied to solve some difficult problems.
- ✓ This consist of input layer, one or more hidden layer and an output layer.
- ✓ The training of the network is done by the highly popular algorithm known as error back propagation algorithm
- ✓ This algorithm is based on the error correcting learning rule. Basically, there are two passes through the different layers of the network: forward pass and backward pass.
- ✓ The Perceptron, training the MLP consists of two parts:
- ✓ working out what the outputs are for the given inputs and current weights
- ✓ Update the weight according to the error, which is a function of the difference between the outputs and the targets.

- ✓ These are generally known as going forwards and backwards through the network

- **Function Signal:**
  - is the input signal that comes in at the input end of the network, ***propagates forward*** (neuron by neuron) through the network, and emerges at the output of the network as an ***output signal***.
- **Error Signal:**
  - originate at the output neuron of the network and ***propagates backward*** (layer by layer) through the network.
- **Each *hidden* or *output neuron* computes these two signals.**

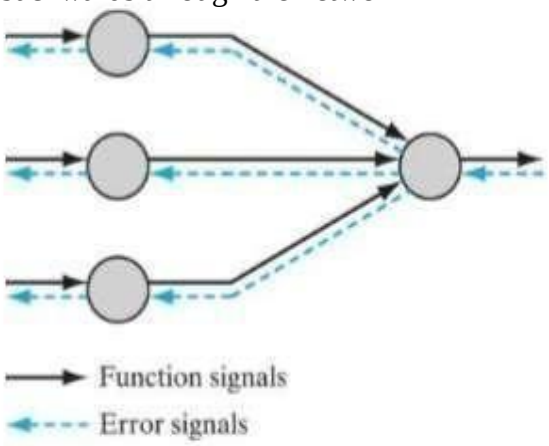
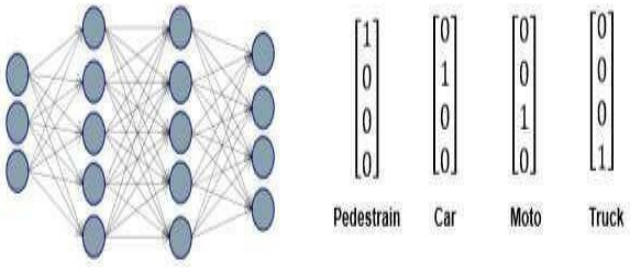


Figure 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.



two phases:

- In the ***forward*** *phase*, the weights of the network are *fixed* and the input signal is *propagated* through the network, layer by layer, until it reaches the *output*.
- In the ***backward*** *phase*, the error signal, which is produced by comparing the output of the network and the desired response, is *propagated* through the network, again layer by layer, but in the *backward direction*.

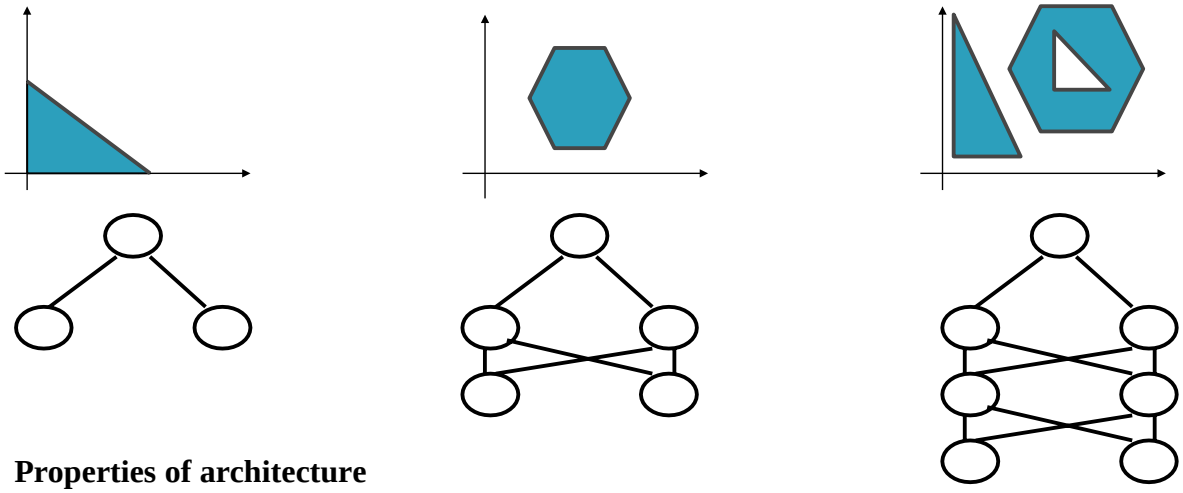
Training of Multilayer Perceptron

The training process of the MLP occurs by continuous adjustment of the weights of the connections after each processing.

This adjustment is based on the error in output(which is the different between the expected result and the output).

This continuous adjustment of the weights is a supervised learning process called 'backpropagation'.

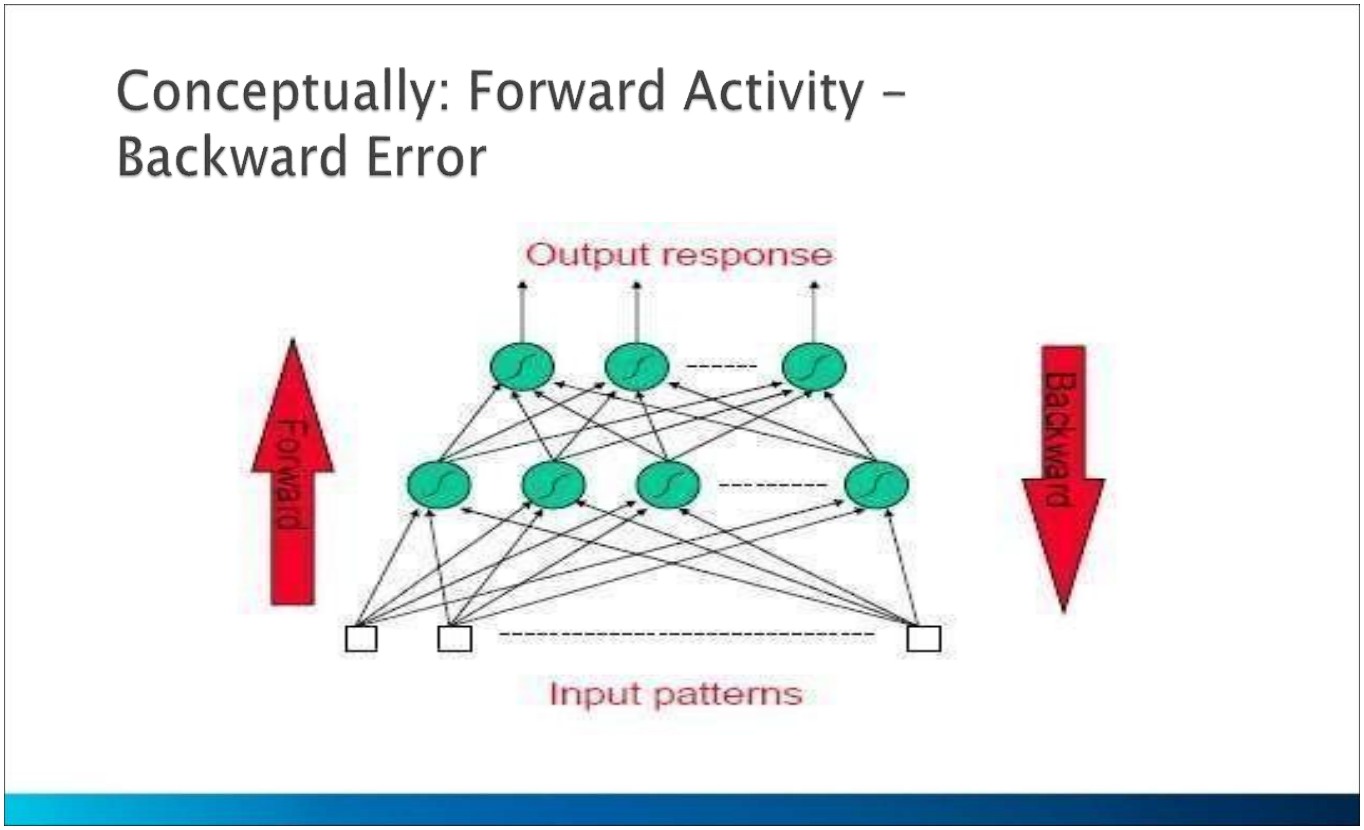
- **The basic features of the multilayer perceptrons:**
  - Each neuron in the network includes a nonlinear activation function that is *differentiable*.
  - The network contains one or more layers that are *hidden* from both the input and output nodes.
  - The network exhibits a high degree of *connectivity*.



Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units

1st layer draws linearboundaries 2nd layer combines the boundaries 3rd layer can generate arbitrarily complex boundaries



2.1 Going Forward

- Start at the left by filling in the values for the inputs.
- Use these inputs and the first level of weights to calculate the activations of the hidden layer
- Use those activations and the next set of weights to calculate the activations of the output layer.
- Then got the outputs of the network, we can compare them to the targets and compute the error. Biases
- Include a bias input to each neuron as such in Perceptron. by having an extra input that is
- permanently set to -1, and adjusting the weights to each neuron as part of the training.
- Thus, each neuron in the network (whether it is a hidden layer or the output) has 1 extra input, with
- fixed value.

2 .2 GOINGBACKWARDS: BACK-PROPAGATION OF ERROR

- Computing the errors at the output is no more difficult in Perceptron, but working out what to do with those errors is more difficult.
- The method is called back-propagation of error, that the errors are sent backwards through the network.
- The best way to describe back-propagation properly is mathematically, by choose an error function
- k: Ek = yk −tk for each neuron and tried to make it as small as possible.
- If it has only one set of weights in the network, it was sufficient to train the network.
- But, with the addition of extra layers of weights, this is harder to arrange.
- The problem is that try to adapt the weights of the Multi-layer Perceptron, it has to work out which weights caused the error. This could be the weights connecting the inputs to the hidden layer, or the weights connecting the hidden layer to the output layer.
- The error function that used for the Perceptron was where N is the number of output nodes.

If MLP has two errors,

1.The target is bigger than the output

2.The output is bigger than the target.

If these two errors are the same size, then add them up to get 0, which means there was no error.

- To get no errors make all errors have the same sign.
- It will be done in a few different ways, but the one that will turn out to be best is the sum-of-squares error function, which calculates the difference between y and t for each node, squares them, and adds them all together: ½ makes it easier when differentiate the function.

If differentiate a function, then it is called gradient of function, which is the direction along which it increases and decreases the most.

If differentiate an error function, it gets the gradient of the error. Since the purpose of learning is to minimize the error, following the error function downhill (in other words, in the direction of the negative gradient) .

Imagine a ball rolling around on a surface that looks like the line in Figure 4.3. Gravity will make the ball roll downhill (follow the downhill gradient) until it ends up in the bottom of one of the hollows.

The places where the error is small, that algorithm is called gradient descent.

Differentiate with respect to three things in the network that change:

The inputs

The activation function that decides whether or not the node fires

The weights.

- It saturates(reaches its constant values) at ± 1 instead of 0 and1.
- It also has a relatively simple derivative: d/dx tanh x = (1−tanh2(x)).
- It can convert between the two easily, because if the saturation points are (±1), then it can convert to (0,1) by using 0.5×(x+1).

New form of error computation and new activation function decided whether or not a neuron should fire.

If change the weights means improving the error function of the network.

Fed inputs forward through the network and worked out which nodes are firing.

At the output, computed the errors as the sum squared difference between the outputs and the target.

When the output is computing the gradient of these errors and to decide how much update each weight in the network. Inputs connected to the output layer and after updated means, it will work backwards through the network until get back to the inputs again.

It raises two problems

For the output neurons, don’t know which input.

For the hidden neurons, don’t know the target.

The Multi-layer Perceptron Algorithm

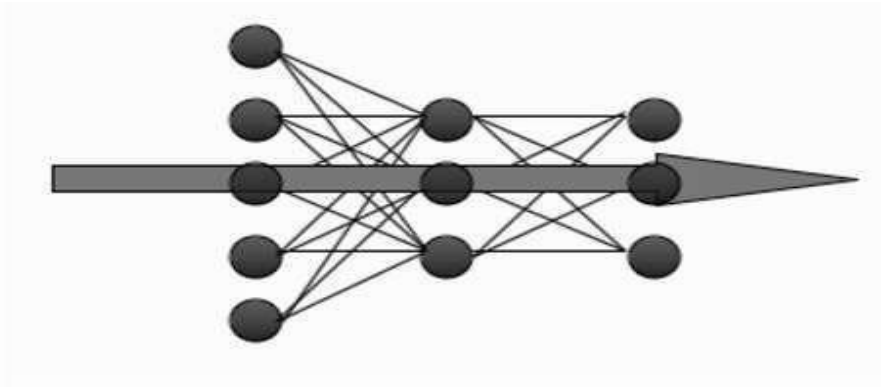


FIGURE 4.6 The forward direction in a Multi-layer Perceptron.

Assume

L input nodes, plus the bias

M hidden nodes, plus a bias

N output nodes

$(L+1)\times M$ weights between the input and the hidden layer

$(M+1)\times N$ between the hidden layer and the output.

$x0 = -1$ is the bias input

$a0 = -1$ is the bias hidden node.

i,j,k to index the nodes in each layer in the sums, and the corresponding Greek letters (ι,ζ,κ) for fixed indices

.

The Multi-layer Perceptron Algorithm

MLP training algorithm using back-propagation of error is described.

1.An input vector is put into the input nodes 2. the inputs are fed forward through the network (Figure 4.6)

2.The inputs and the first-layer weights (here labelled as v) are used to decide whether the hidden nodes fire or not. The activation function g(·) is the sigmoid function given in Equation (4.2)above. The outputs of these neurons and the second-layer weights (labelled as w)are used to decide if the output neurons fire or not

3.Error is computed as the sum-of-squares difference between the network outputs and the targets

4.This error is fed backwards through the network in order to

- first update the second-layer weights
- and then afterwards, the first-layer weights

The Multi-layer Perceptron Algorithm

- **Initialisation**
  - initialise all weights to small (positive and negative) random values
- **Training**
  - repeat:
    - for each input vector:

      Forwards phase:

      · compute the activation of each neuron j in the hidden layer(s) using:

$$h_\zeta = \sum_{i=0}^{L} x_i v_{i\zeta} \tag{4.4}$$

$$a_\zeta = g(h_\zeta) = \frac{1}{1+\exp(-\beta h_\zeta)} \tag{4.5}$$

      · work through the network until you get to the output layer neurons, which have activations (although see also Section 4.2.3):

$$h_\kappa = \sum_{j} a_j w_{j\kappa} \tag{4.6}$$

$$y_\kappa = g(h_\kappa) = \frac{1}{1+\exp(-\beta h_\kappa)} \tag{4.7}$$

Backwards phase:

· compute the error at the output using:

$$\delta_o(\kappa) = (y_\kappa - t_\kappa)\, y_\kappa(1 - y_\kappa) \tag{4.8}$$

· compute the error in the hidden layer(s) using:

$$\delta_h(\zeta) = a_\zeta(1 - a_\zeta) \sum_{k=1}^{N} w_\zeta \delta_o(k) \tag{4.9}$$

· update the output layer weights using:

$$w_{\zeta\kappa} \leftarrow w_{\zeta\kappa} - \eta \delta_o(\kappa) a_\zeta^{\text{hidden}} \tag{4.10}$$

· update the hidden layer weights using:

$$v_\iota \leftarrow v_\iota - \eta \delta_h(\kappa) x_\iota \tag{4.11}$$

∗ (if using sequential updating) randomise the order of the input vectors so that you don't train in exactly the same order each iteration

– until learning stops (see Section 4.3.3)

- **Recall**
  - use the Forwards phase in the training section above

Initialising method:

weights are initialized to small random numbers, both positive and negative.

If the initial weights values are close to 1 or -1 then the inputs to the sigmoid are also likely to be close to ±1 and so the output of the neuron is either 0 or 1 (the sigmoid has saturated, reached its maximum or minimum value).

If the weights are very small (close to zero) then the input is still close to 0 and so the output of the neuron is just linear, so gets a linear model.

Input to the neuron will be w√n, where w is the initialization value of the weights.

Set the weights in the range $-1/\sqrt{n} < w < 1/\sqrt{n}$, where n is the number of nodes in the input layer.

β in the logistic function (say β =3.0 or less) are more effective.

3 Different Output Activation Functions

Sigmoid neurons in hidden and output layer- 0 and 1 ,Regression problem- continuous range

Soft-max activation – 1 of N output encoding. The soft-max function rescales the outputs by calculating the exponential of the inputs to that neuron and dividing by the total sum of the inputs to all of the neurons, so that the activations sum to 1 and lie between 0 and 1.

As an activation function it can be written as:

$$y_\kappa = g(h_\kappa) = \frac{\exp(h_\kappa)}{\sum_{k=1}^{N} \exp(h_k)}. \tag{4.12}$$

Of course, if we change the activation function, then the derivative of the activation function will also change, and so the learning rule will be different. The changes that need to be made to the algorithm are in Equations (4.7) and (4.8), and are derived in Section 4.6.5. For the linear activation function the first is replaced by:

$$y_\kappa = g(h_\kappa) = h_\kappa, \tag{4.13}$$

while the second is replaced by:

$$\delta_o(\kappa) = (y_\kappa - t_\kappa)\,. \tag{4.14}$$

For the soft-max activation, the update equation that replaces (4.8) is

$$\delta_o(\kappa) = (y_\kappa - t_\kappa) y_\kappa (\delta_{\kappa K} - y_K), \tag{4.15}$$

where $\delta_{\kappa K} = 1$ if $\kappa = K$ and 0 otherwise; see Section 4.6.5 for further details. However, if we modify the error function as well, to have the cross-entropy form (where ln is the natural logarithm):

$$E_{\text{ce}} = -\sum_{k=1}^{N} t_k \ln(y_k), \tag{4.16}$$

2.2.4 Sequential and Batch Training

The MLP is designed to be a batch algorithm.

All of the training examples are presented to the neural network, the average sum-of-squares error is then computed, and this is used to update the weights.

Thus there is only one set of weight updates for each epoch (pass through all the training examples).

This means that only update the weights once for each iteration of the algorithm, which means that the weights are moved in the direction that most of the inputs want them to move, rather than being pulled around by each input individually.

The batch method performs a more accurate estimate of the error gradient, and will thus converge to the local minimum more quickly

2.2.5 Local minima

The learning rule is the minimisation of the network error by gradient descent (using the derivative of the error function to make the error smaller).

Perform an optimisation-adapting the values of the weights in order to minimise the error function.

2.2.6 Picking Up Momentum

A local minimum of a function is a point where the function value is smaller than at nearby points, but possibly

• Neural network learning by adding in some contribution from the previous weight change that made to the current one

•Benefit to momentum: Use a smaller learning rate, which means that the learning is more stable.

• Weight decay- reduces the size of the weights as the number of iterations increases. This gives better result to lead a network

ie., small weights are closer to linear greater than at a distant point.

A global minimum is a point where the function value is smaller than at all other feasible points.

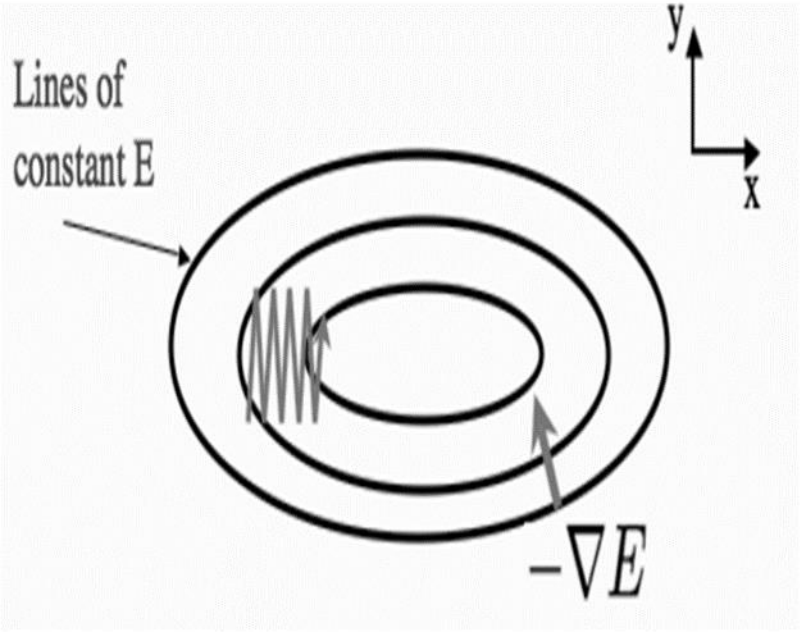


FIGURE 4.7 In 2D, downhill means at right angles to the lines of constant contour. Imagine walking down a hill with your eyes closed. If you find a direction that stays flat, then it is quite likely that perpendicular to that the ground goes uphill or downhill. However, this is not the direction that takes you directly towards the local minimum.

· MLP algorithm is in Equations (4.10) and (4.11), where we need to add a second term to the weight updates so that they have the form:

$$w_{\zeta\kappa}^{t} \leftarrow w_{\zeta\kappa}^{t-1} + \eta\delta_o(\kappa)a_{\zeta}^{\text{hidden}} + \alpha\Delta w_{\zeta\kappa}^{t-1}, \tag{4.17}$$

where t is used to indicate the current update and $t-1$ is the previous one. $\Delta w_{\zeta\kappa}^{t-1}$ is the previous update that we made to the weights (so $\Delta w_{\zeta\kappa}^{t} = \eta\delta_o(\kappa)a_{\zeta}^{\text{hidden}} + \alpha\Delta w_{\zeta\kappa}^{t-1}$) and $0 < \alpha < 1$ is the momentum constant. Typically a value of $\alpha = 0.9$ is used. This is a very easy addition to the code, and can improve the speed of learning a lot.

Minibatches and Stochastic Gradient Descent

Batch algorithm converges to a local minimum faster than the sequential algorithm, which computes the error for each input individually and then does a weight update, but latter stuck in local minima. The idea of a minibatch method is by splitting the training set into random batches, estimating the gradient based on one of the subsets of the training set, performing a weight update, and then using the next subset to estimate a new gradient and using that for the weight update, until all of the training set has been used. The training set are then randomly shuffled into new batches and the next iteration takes place. A more extreme version of the minibatch idea is to use just one piece of data to estimate the gradient at each iteration of the algorithm, and to pick that piece of data uniformly at random from the training set. So a single input vector is chosen from the training set, and the output and hence the error for that one vector computed, and this is used to estimate the gradient and so update the weights. A new random input vector (which could be the same as the previous one) is then chosen and the process repeated. This is known as stochastic gradient descent. t It is often used if the training set is very large, since it would be very expensive to use the whole dataset to estimate the gradient in that case.

Other Improvements

One is to reduce the learning rate as the algorithm progresses. The network making large-scale changes to the weights at the beginning, when the weights are random. Results gives larger performance gains the second derivatives of the error with respect to the weights. In the back-propagation algorithm - use the first derivatives to drive the learning. Knowledge of the second derivatives , it helps to improve the network

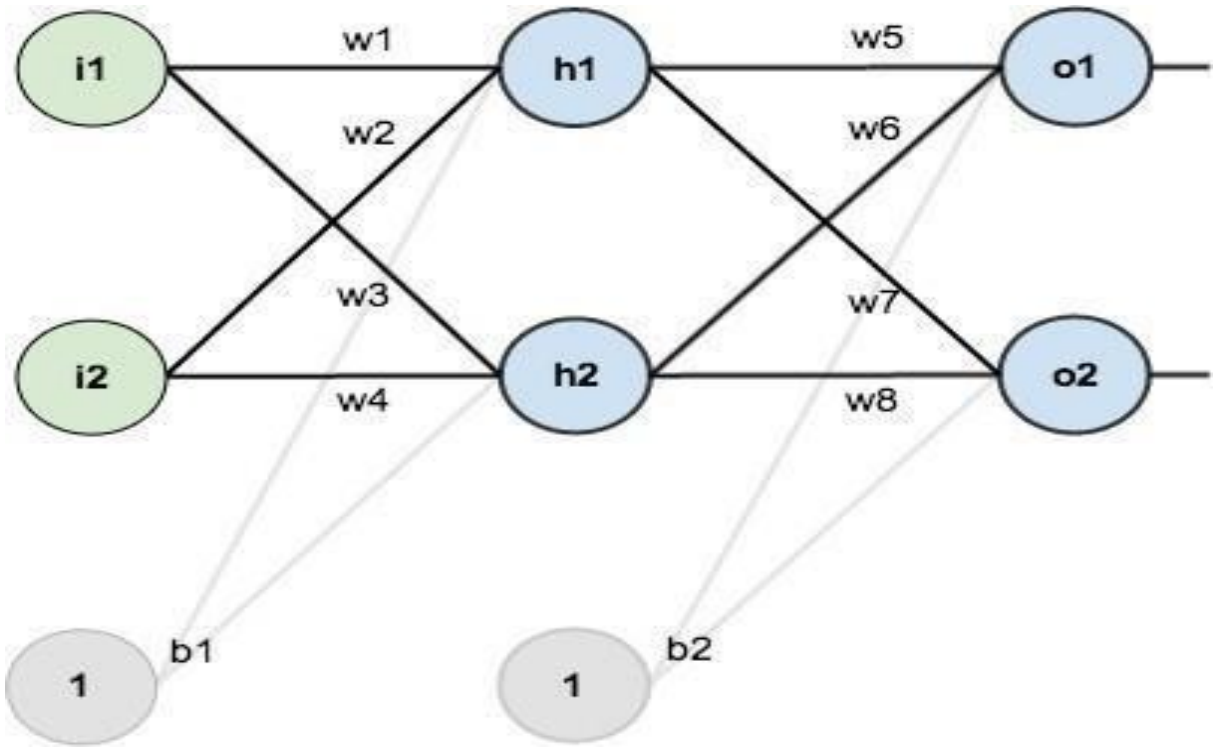
A Step by Step Backpropagation Example

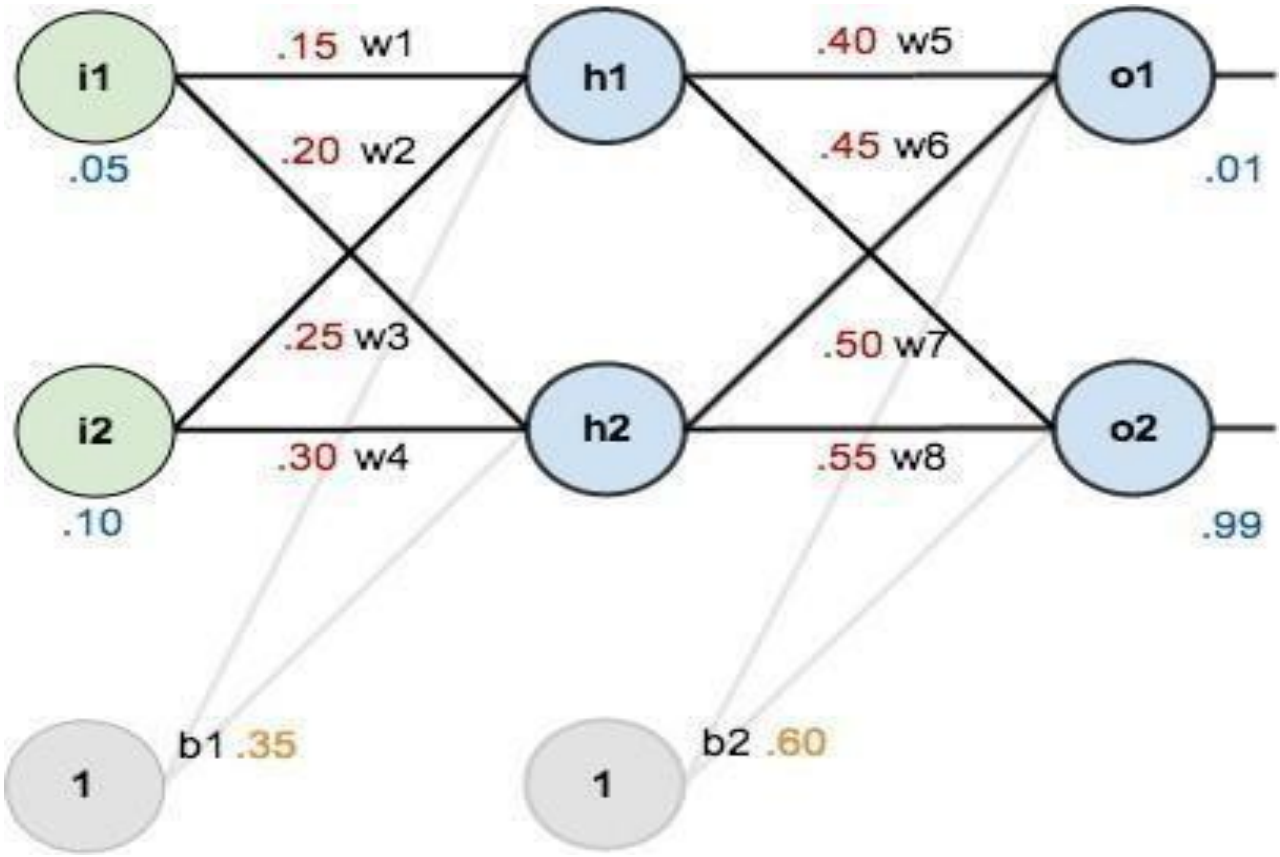
Background

Backpropagation is a common method for training a neural network.

Overview

For this tutorial, we're going to use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.
Here's the basic structure:





The goal of backpropagation is to optimize the weights so that the neural network can learn howto correctly map arbitrary inputs to outputs.

Single training set:

- Inputs 0.05 and 0.10
- Expected output 0.01 and 0.99.

The Forward Pass

- To begin, let's see what the neural network currently predicts given the weights andbiases above and inputs of 0.05 and 0.10.
- To do this we'll feed those inputs forward though the network.

We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.

Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here's the output for o_1:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

And carrying out the same process for we get:

$$out_{o2} = 0.772928465$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.1059059967}} = 0.75136507$$

Calculating the Total Error

Calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Some sources refer to the target as the *ideal* and the output as the *actual*.

The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here [1].

For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they causethe actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

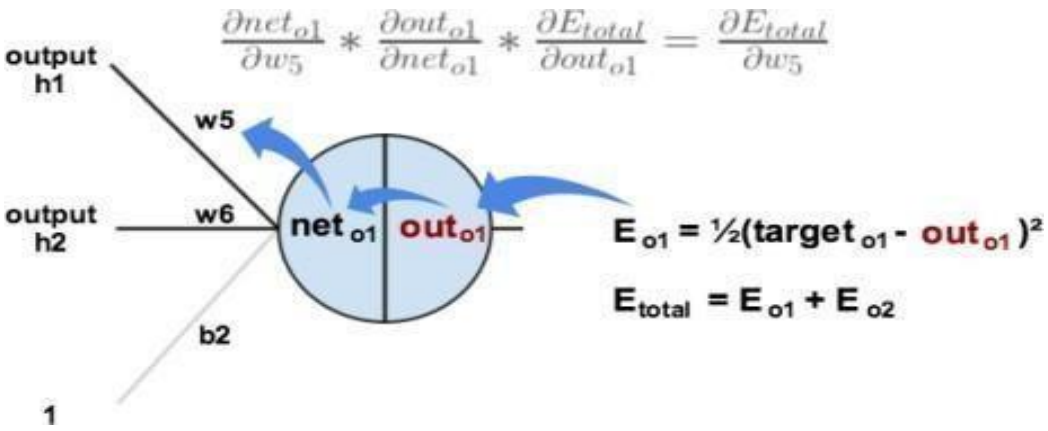
Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, $\frac{\partial E_{total}}{\partial w_5}$.

$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the chain rule we know that:

Visually, here's what we're doing:



$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they causethe actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error,

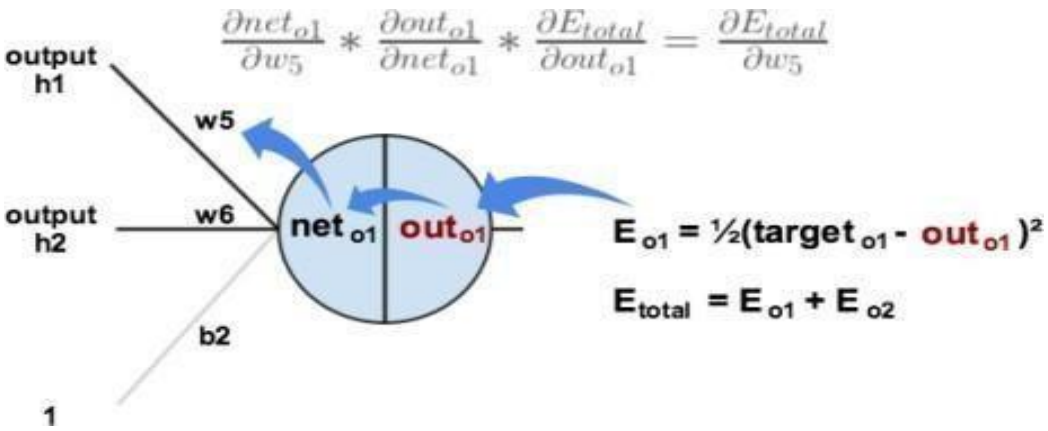
$$\frac{\partial E_{total}}{\partial w_5}$$

.

$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the chain rule we know that:

Visually, here's what we're doing:



Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

To decrease the error, we then subtract this value from the current weight(optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Some sources use α (alpha) to represent the learning rate, others use η (eta), and others even use ϵ (epsilon).

We can repeat this process to get the new weights w_6, w_7, and w_8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for , w_{2}, w_{1}, and w_{4}, w_{3}

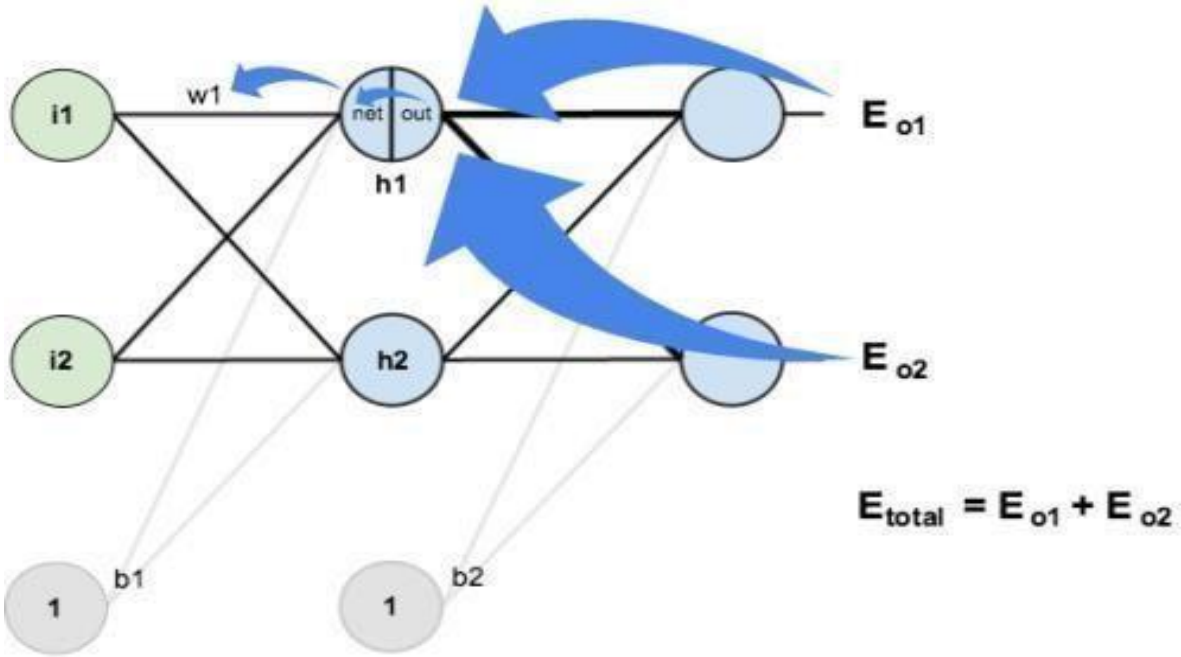
Big picture, here's what we need to figure out:

Visually:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that affects both and out_{o2} therefore out_{h1} out_{o1} the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

Therefore:

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

We calculate the partial derivative of the total net input to h_1 with respect tow_1 the same as we did for the output neuron:

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

You might also see this written as:

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \delta_o * w_{ho}) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

We can now update w_1:

$$w_2^+ = 0.19956143$$

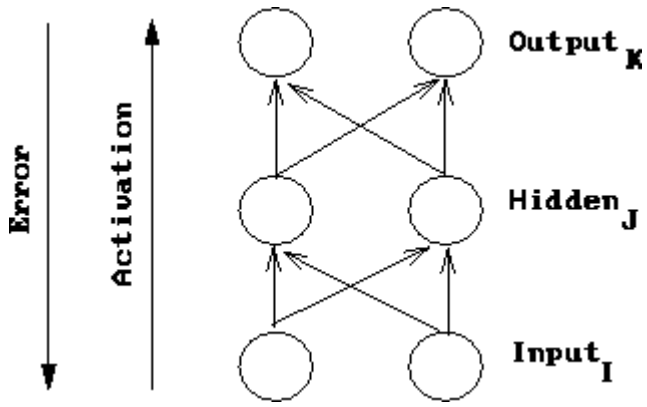
$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round

of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating thisprocess 10,000 times, for example, the error plummets to 0.0000351085. At this point, when wefeed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

1 Introduction



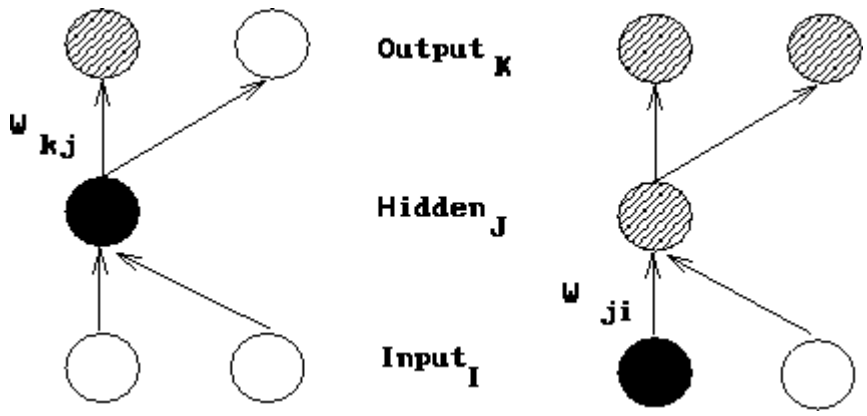
Conceptually, a network forward propagates activation to produce an output and it backward propagates error to determine weight changes (as shown in Figure 1). The weightson the connections between neurons mediate the passed values in both directions.

The Backpropagation algorithm is used to learn the weights of a multilayer neural network with a fixed architecture. It performs gradient descent to try to minimize the sum squared error between the network's output values and the given target values. Figure 2 depicts the network components which affect a particular weight change. Noticethat all the necessary components are locally related to the weight being updated. This is one feature of backpropagation that seems biologically plausible. However, brain connections appear to be unidirectional and not bidirectional as would be required to implement backpropagation.

1 Notation

For the purpose of this derivation, we will use the following notation:

- The subscript *k* denotes the output layer.
- The subscript *j* denotes the hidden layer.
- The subscript *i* denotes the input layer.



Linear models

- Radial Basis Functions and Splines – Concepts
- RBF Network
- Curse of Dimensionality
- Interpolations and Basis

Radial Basis Functions and Splines – Concepts

An RBFN performs classification by measuring the input's similarity to examples from the training set. Each RBFN neuron stores a prototype, which is just one of the examples from the training set.

When we want to classify a new input, each neuron computes the Euclidean distance between the input and its prototype. Ie., If the input more closely resembles the Class A prototypes than the Class B prototypes, it is classified as Class A.

Classification:

Purpose: assign previously unseen patterns to their respective classes.

Training: Previous examples of each class. Output: A class out of a discrete set of classes. Classification problems can be made to look like nonparametric regression.

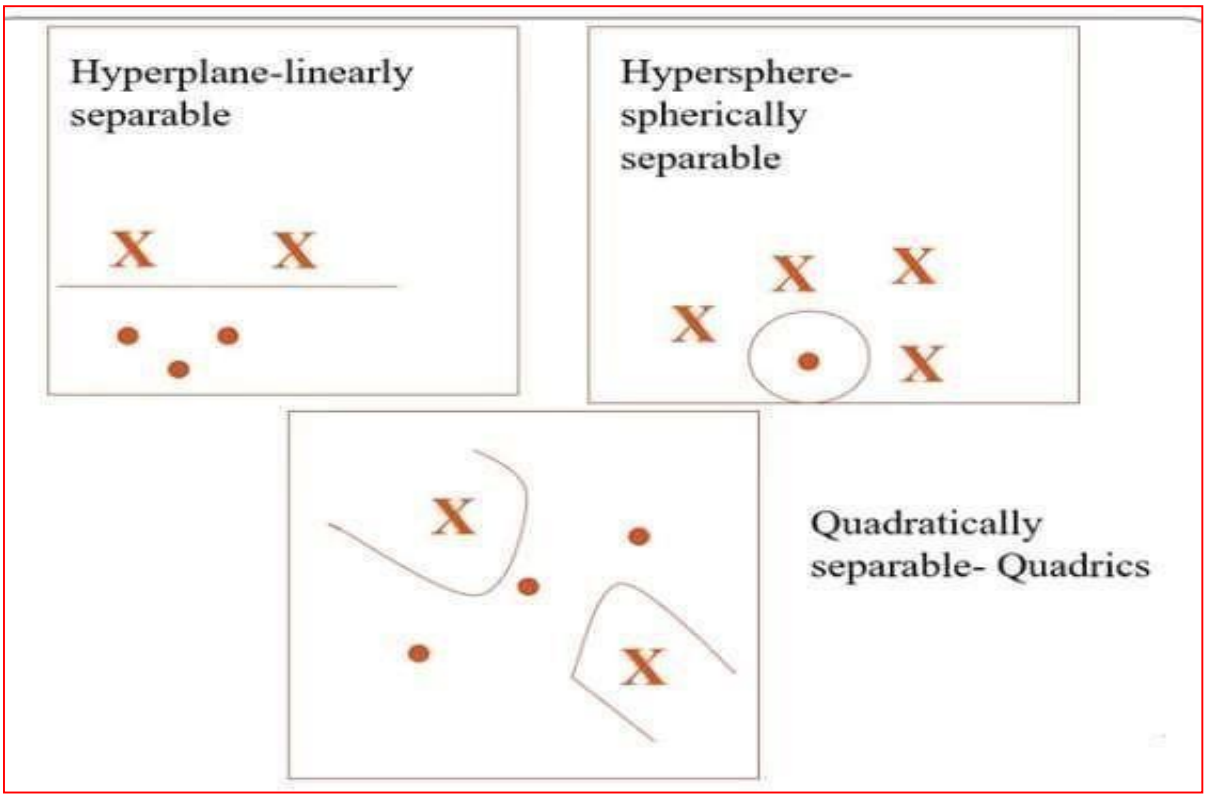
RBF would be separate class distributions by localizing radial basis functions.

Types of separating surfaces are

Hyperplane- linearly separable

Spherically separable-Hypersphere

Quadratically separable- Quadrics



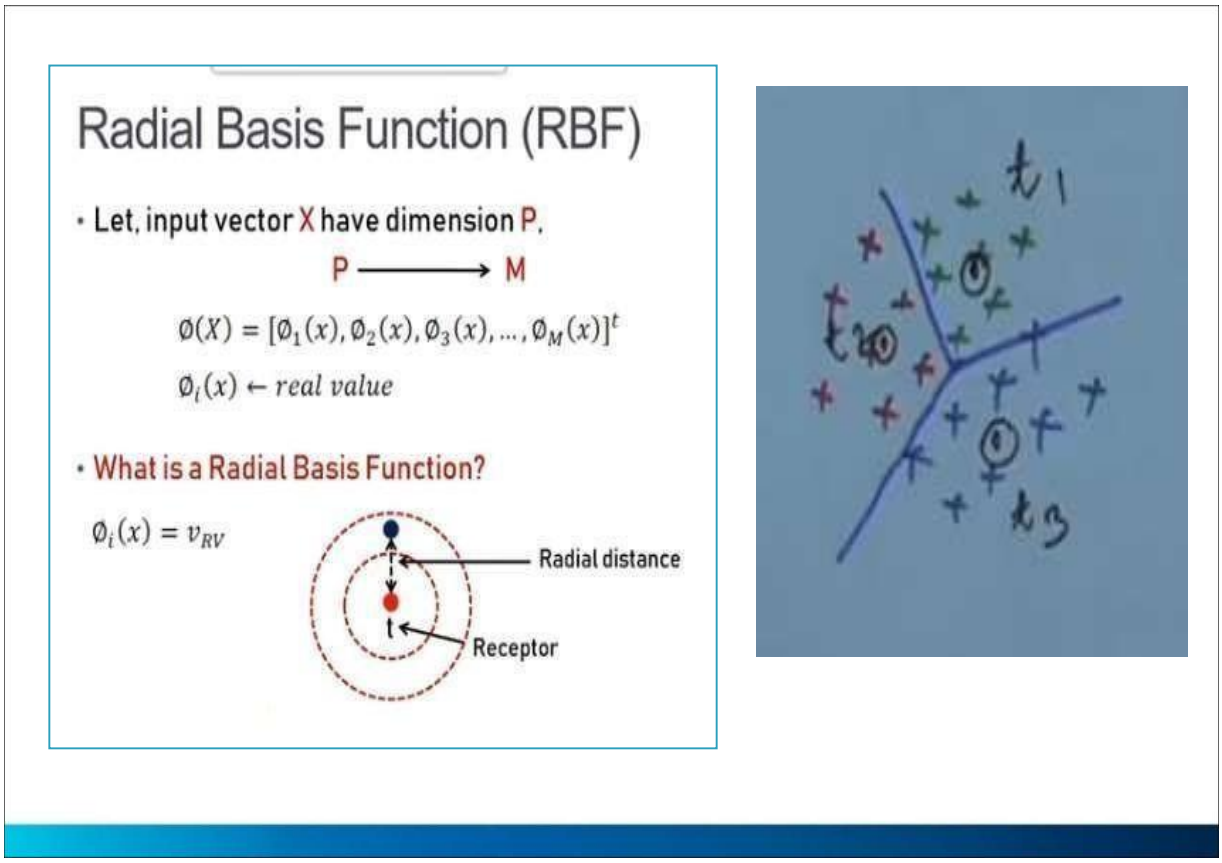
■ Input layer should mapping with hidden layer , there they provide a set of functions which forms a based for mapping into the hidden layer space.

■ To do mapping from input space to the hidden layer , it is need some basis functions providing the neurons in the hidden layer. Hidden neurons in hidden layer providing the basis functions and this kind of architecture is called Radial Basis function networks.

- Approximate function with linear combination of Radial basis functions
  - $F(x) = \sum w_i h(x)$
  - h(x) is mostly Gaussian function
- Three layers

- Input layer – Source nodes that connect to the network to its environment / hidden layer. (Non linear mapping)
- Hidden layer – Hidden units provide a set of basis function– Nonlinear transformation
- Ie Input space -> Hidden space(High dimensionality)
- Output layer – Linear combination of hidden functions

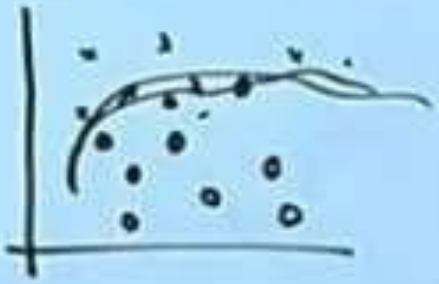
■ Cover's Theorem : A pattern classification problem cast in high dimensional space is more likely to be linearly separable than in a low dimension space.



Set
H of N patterns

$\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$

Each can be assigned to H_1 or H_2.



For each

$\vec{x} \in H$, define a vector made of a set of real-valued functions

$$\{\phi_i(\vec{x}) \mid i = 1, 2, \ldots, m_1\}.$$

ϕ_i's
Hidden functions

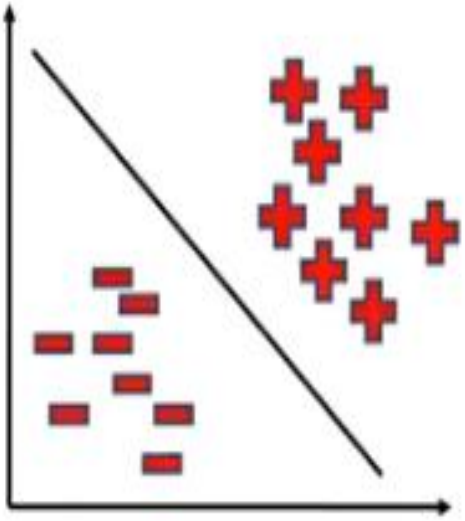
$\vec{x}$ → $\phi_1(\vec{x})$, $\phi_2(\vec{x})$, ⋮, $\phi_{m_1}(\vec{x})$

Linear separability

- Data distribution as per its feature space in 2D space.

- Data

- Separate data linearly



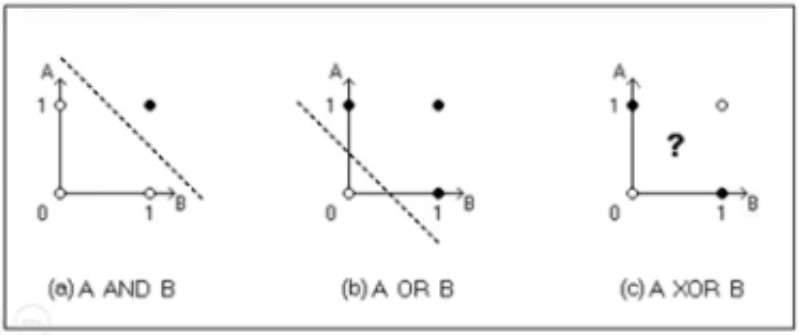
Need of RBFN

- In Single Perceptron, we only have linear separability because they are composed of input and output layers.

A	B	AND
1	1	1
1	0	0
0	1	0
0	0	0

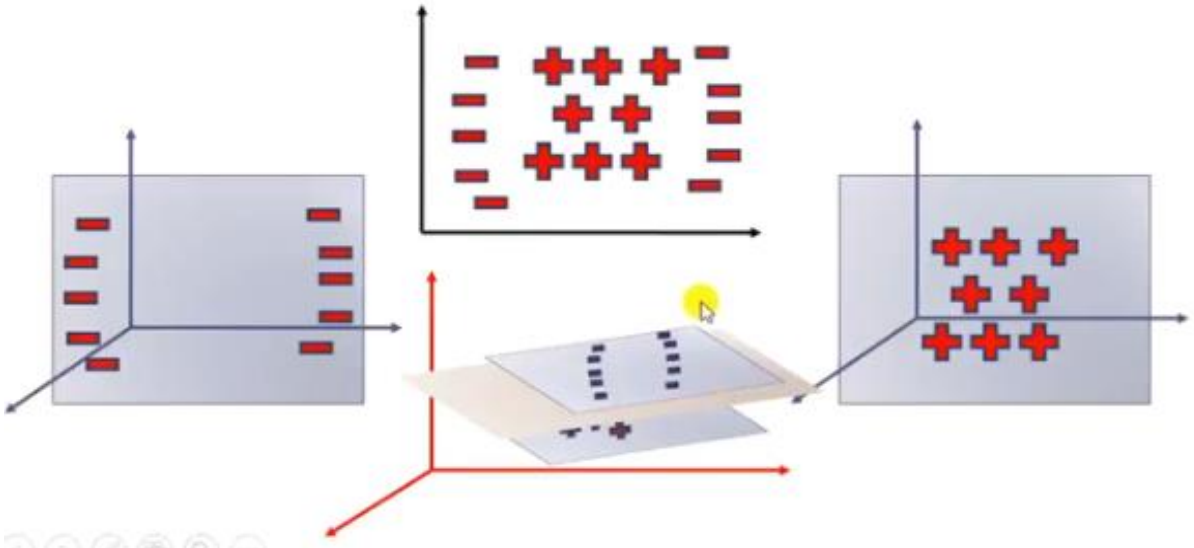
A	B	OR
1	1	1
1	0	1
0	1	1
0	0	0

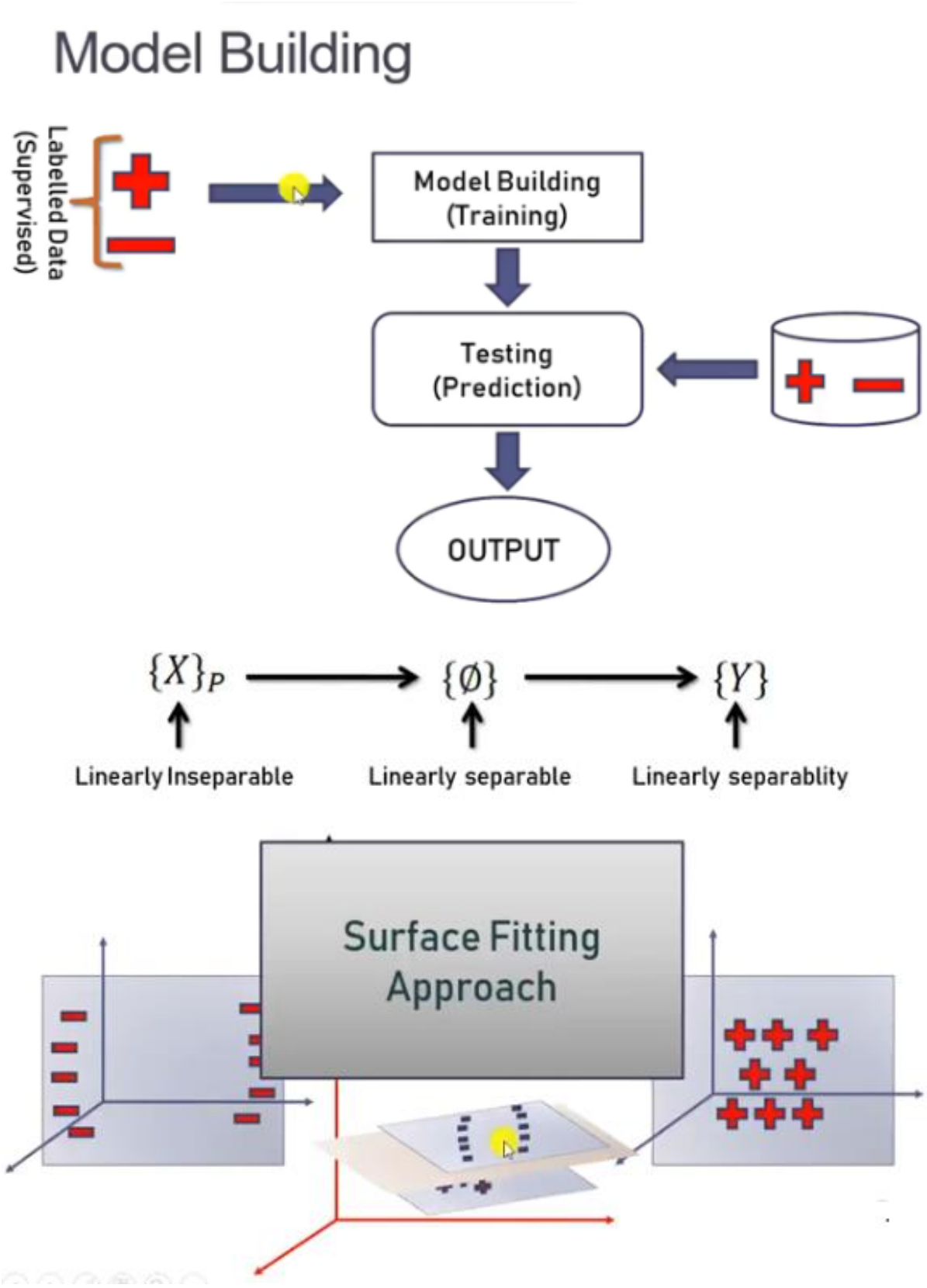
A	B	XOR
1	1	0
1	0	1
0	1	1
0	0	0



$\{X\}_P$ → $\{\emptyset\}$ → $\{Y\}$

Linearly Inseparable — Linearly separable — Linearly separablity




Model Building
Labelled Data
(Supervised)
Model Building
(Training)
Testing
(Prediction)
OUTPUT
$\{X\}_P$
$\{\emptyset\}$
$\{Y\}$
Linearly Inseparable
Linearly separable
Linearly separablity
Surface Fitting
Approach

What happens in Hidden layer?

The patterns in the input space form clusters. If the centres of these clusters are known then the distance from the cluster centre can be measured. The most commonly used radial basis function is a **Gaussian function.** The other function are **multi quadrics function and inverse multi quadric function.** In an RBF network **r** is the distance from the cluster centre.

Radial functions

- Gaussian RBF:
  c: center, *r*: radius

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right)$$

- monotonically decreases with distance from center

- Multiquadric RBF

$$h(x) = \frac{\sqrt{r^2 + (x-c)^2}}{r}$$

- monotonically increases with distance from center

Distance measure

- The distance measured from the cluster centre is usually the Euclidean distance
- For each neuron in the hidden layer, the weights represent the co-ordinates from the centre of the cluster
- When the neuron receives an input pattern X, the distance is found using the equation

$$r_j = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$$

RBF for a gaussian function

- **Centers: are selected at random**
  - **centers** are chosen randomly from the training set
- **Spreads:** are chosen by **normalization**:

$$\sigma = \frac{\text{Maximum distance between any 2 centers}}{\sqrt{\text{number of centers}}} = \frac{d_{max}}{\sqrt{m_1}}$$

- Then the activation function of hidden neuron becomes: i

$$\varphi_i(\|x\|) = \exp\left(-\frac{m_1}{d_{max}^2}\|x - \mu_i\|^2\right)$$

Width of hidden unit

$$\phi_j = \exp\left(-\frac{\sum_{i=1}^{n}(xi - \mu_j)^2}{2\sigma^2}\right) \quad \text{---- 1}$$

where $\sigma = \frac{d_{max}}{\sqrt{2M}}$ ---- 2

σ Is the width or radius of the bell shape and has to be determined empirically

M=no. of basis function μ_j =basis function centre

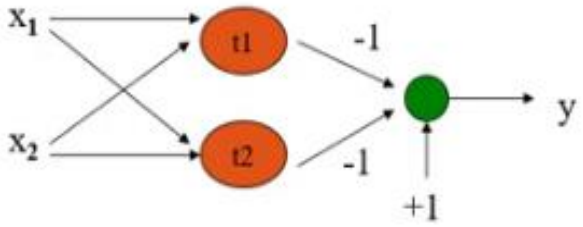
D_{max}=distance between them

$$\phi_j = \exp\left(-\frac{M}{d^2_{max}}\sum_{i=1}^{n}(xi - \mu_j)^2\right) \quad \text{---- 3}$$

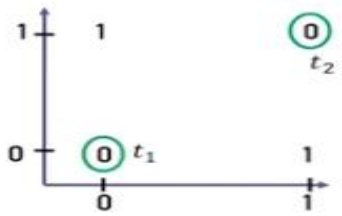
RBF NN for the XOR problem

$$\varphi_1(x) = e^{-\|x-\mu_1\|^2} \qquad \text{with } \mu_1 = (0,0) \text{ and } \mu_2 = (1,1)$$

$$\varphi_2(x) = e^{-\|x-\mu_2\|^2}$$



Pattern	X1	X2	φ1	φ2
1	0	0	1	0.135
2	0	1	.36	.36
3	1	0	.36	.36
4	1	1	.135	1



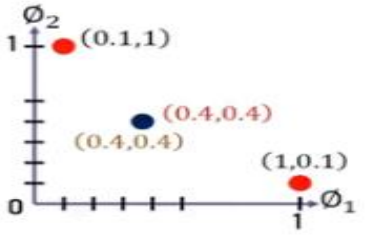
$$\emptyset_i(r) = \frac{exp[-r^2]}{2\sigma^2}, \quad \sigma > 0$$

$t_1 = (0,0)$ $\quad \emptyset(X) = e^{-\|x-t\|^2}$ $\quad$ Let, $2\sigma^2 = 1$

$t_2 = (1,1)$ $\quad \emptyset_1 = e^{-\|x-t_1\|^2}$

$\emptyset_2 = e^{-\|x-t_2\|^2}$

		$\emptyset_1$	$\emptyset_2$
0	0	1	0.1
0	1	0.4	0.4
1	0	0.4	0.4
1	1	0.1	1

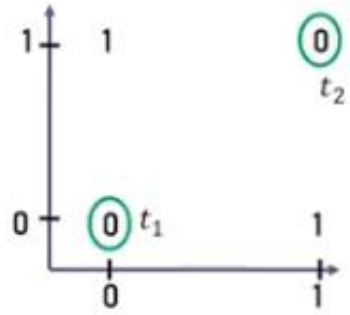


$\emptyset_1(X) = e^{-((0-0)^2+(0-0)^2)} = 1$

where$(x_1, x_2) = (0,0)$and $t_1 = (0,0)$

$\emptyset_2(X) = e^{-((0-1)^2+(0-1)^2)} = 0.1$

where$(x_1, x_2) = (0,0)$and $t_2 = (1,1)$



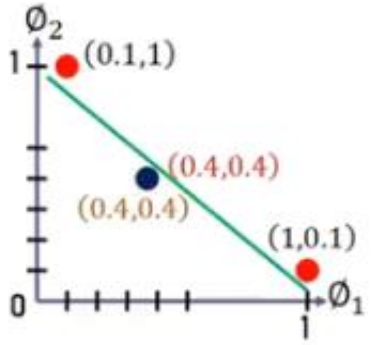
$$\emptyset_i(r) = \frac{exp[-r^2]}{2\sigma^2}, \quad \sigma > 0$$

$t_1 = (0,0)$ $\quad \emptyset(X) = e^{-\|x-t\|^2}$ $\quad$ Let, $2\sigma^2 = 1$

$t_2 = (1,1)$ $\quad \emptyset_1 = e^{-\|x-t_1\|^2}$

$\emptyset_2 = e^{-\|x-t_2\|^2}$

		$\emptyset_1$	$\emptyset_2$
0	0	1	0.1
0	1	0.4	0.4
1	0	0.4	0.4
1	1	0.1	1



$\emptyset_1(X) = e^{-((0-0)^2+(0-0)^2)} = 1$

where$(x_1, x_2) = (0,0)$and $t_1 = (0,0)$

$\emptyset_2(X) = e^{-((0-1)^2+(0-1)^2)} = 0.1$

where$(x_1, x_2) = (0,0)$and $t_2 = (1,1)$

Training

- Self organized selection of centres
- K-means algorithm
  - Initialization : Create random cluster centre points
  - Sampling: Draw a sample vector x from a input space
  - Similarity Matching: Index of the winning cluster centre

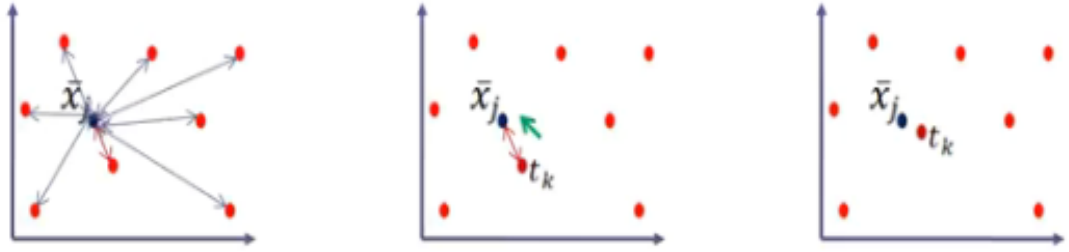
$$K\left(\underset{x}{\rightarrow}\right) = arg\ \min_{k} \|X - t_k\|$$

  - Updation:

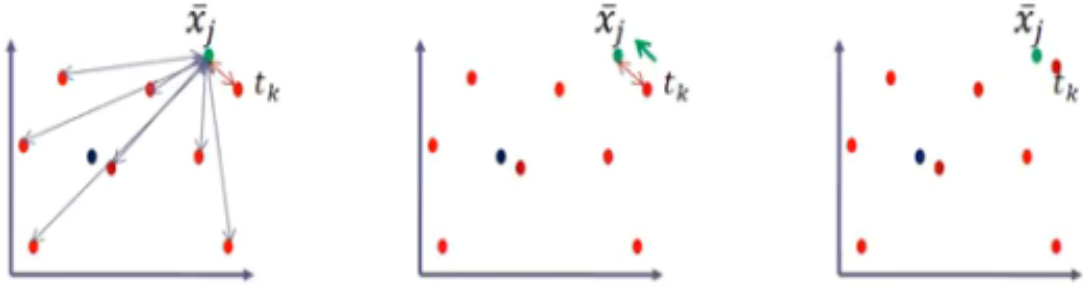
$$t_k(n+1) = \begin{cases} t_k(n) + \eta\left[\bar{x}(n) - t_{k(n)}\right], & if\ K = K(x) \\ t_{k(n)}, & if\ K \neq K(x) \end{cases}$$

$$S = \{\bar{x}_1, \bar{x}_2, \ldots \bar{x}_n\}$$

I/P 01



I/P 02



Learning Algorithm

Step 1: Set the weights to small random values.
Step 2: Perform steps 3-9 when the stooping conditions is false.
Step 3: Perform steps 4-8 for each input.
Step 4: Each input unit (Xi for all i=1 to n) receives input signals and transmits to the next hidden layer unit.
Step 5: Calculate the radial basis function.
Step 6: Select the centers for the radial basis function. The centers are selected from the set of input vector.
Step 7: Calculate the output from the hidden layer unit:

$$\emptyset_i(x_i) = \frac{exp\left[-\sum_{j=1}^{r}(x_{ij} - \hat{x}_{ij})^2\right]}{2\sigma^2}, \quad \sigma > 0$$

Step 8: Calculate the output of the neural network:

$$y_{net} = \sum_{i=1}^{m} w_{ih}\, \emptyset_i(x_i) + w_0$$

Step 9: Calculate error and test for the stopping criteria.

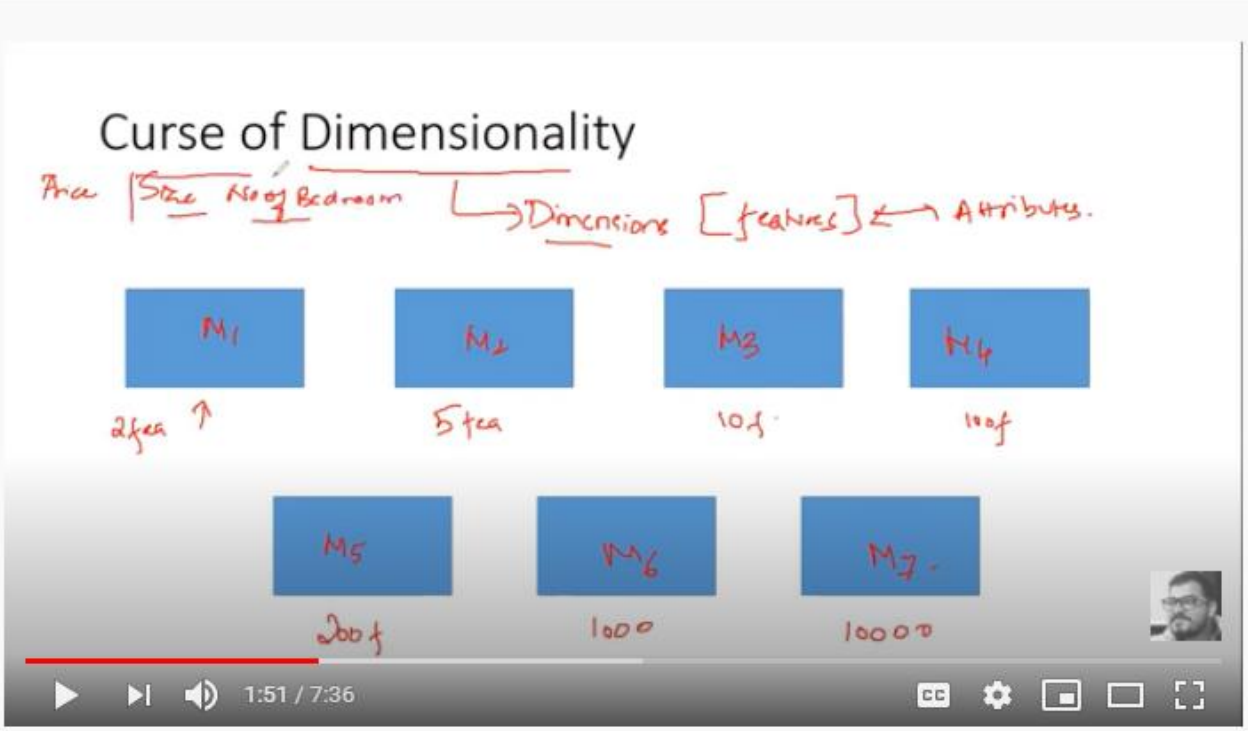
Differences between MLP and RBF

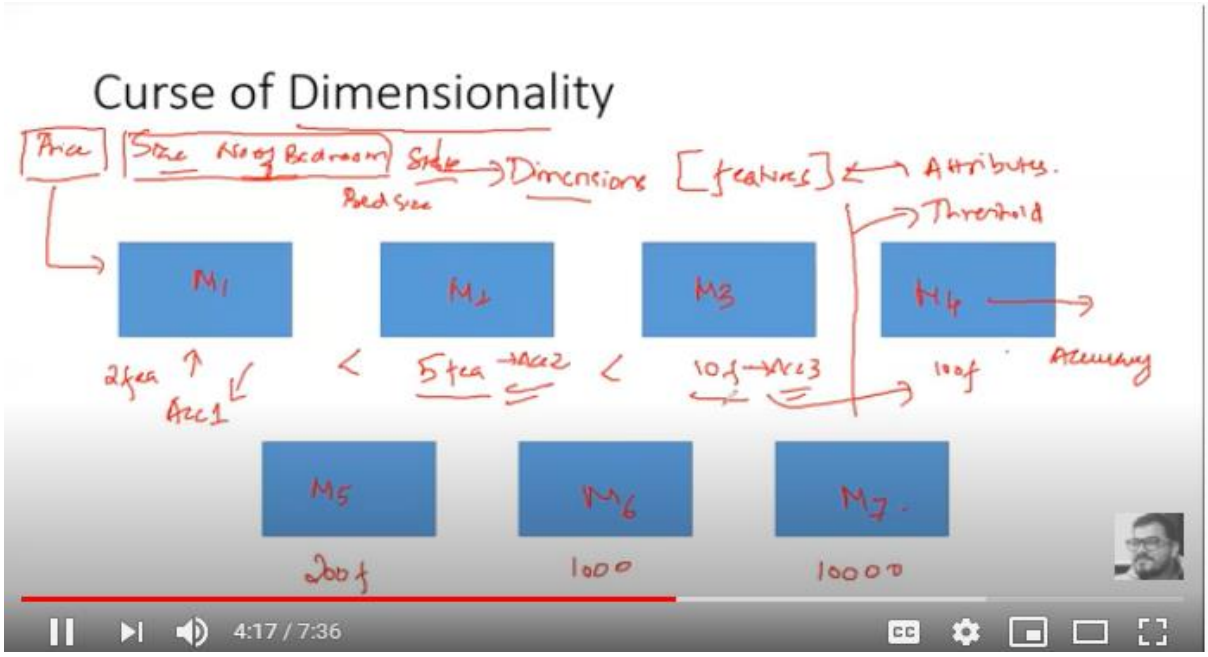
MLP	RBF
Can have any number of hidden layer	Can have only one hidden layer
Can be fully or partially connected	Has to be mandatorily completely connected
Processing nodes in different layers shares a common neural model	Hidden nodes operate very differently and have a different purpose
Argument of hidden function activation function is the inner product of the inputs and the weights	The argument of each hidden unit activation function is the distance between the input and the weights
Trained with a single global supervised algorithm	RBF networks are usually trained one later at a time
Training is slower compared to RBF	Training is comparitely faster than MLP
After training MLP is much faster than RBF	After training RBF is much slower than MLP

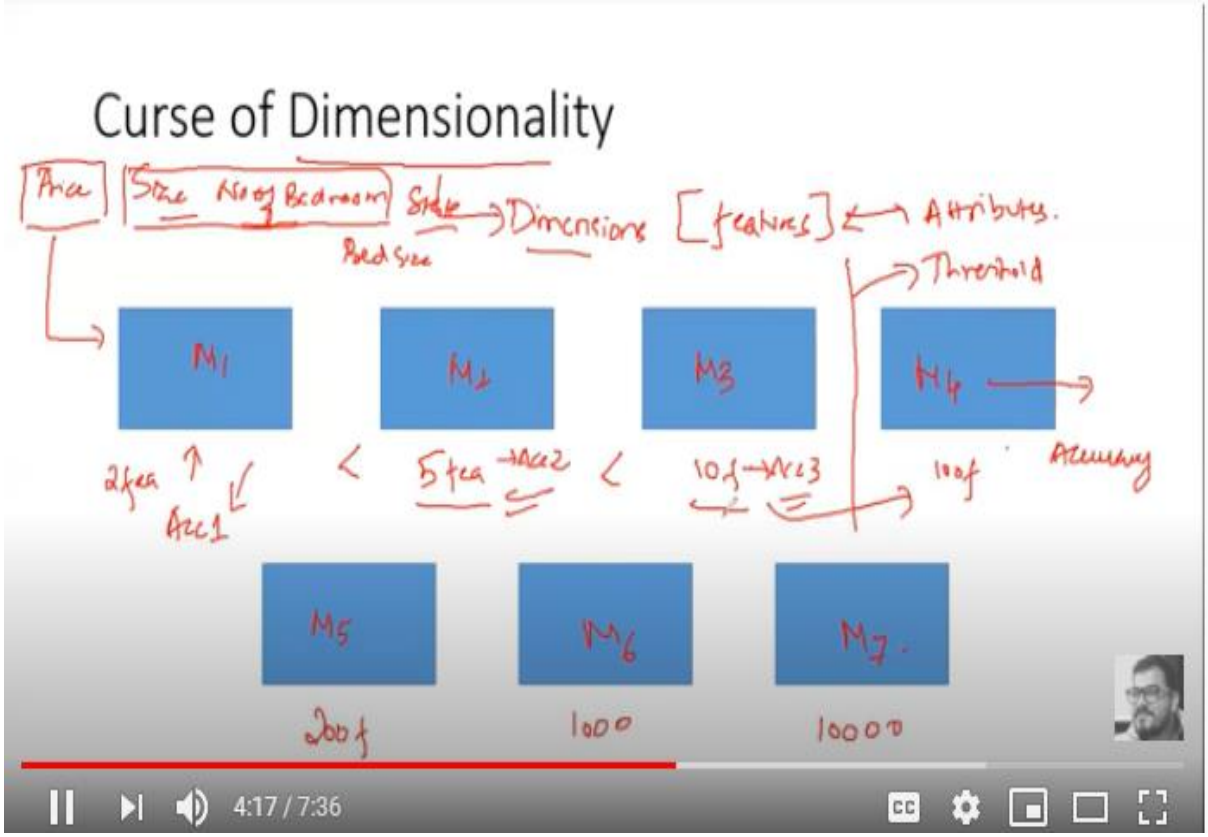
MLP	RBFN
One or More hidden layer	Only one hidden layer
Back propagation training algorithm	K-Mean and RLS training algorithm
Non-linear Output	Linear output
Activation function of hidden unit perform inner product of input vector and weight	Activation function of hidden unit perform Euclidean norm of input vector and the centre of the unit
Sigmoid and tanh activation function	Gaussian activation function
Slower training	Faster training

Curse of Dimensionality

As the Number of features or dimension grows. The amount of data we need to generate accurately grows exponentially. Feature selection and feature engineering. The dimension also called features. The features independent features or target output features. Features basically attributes

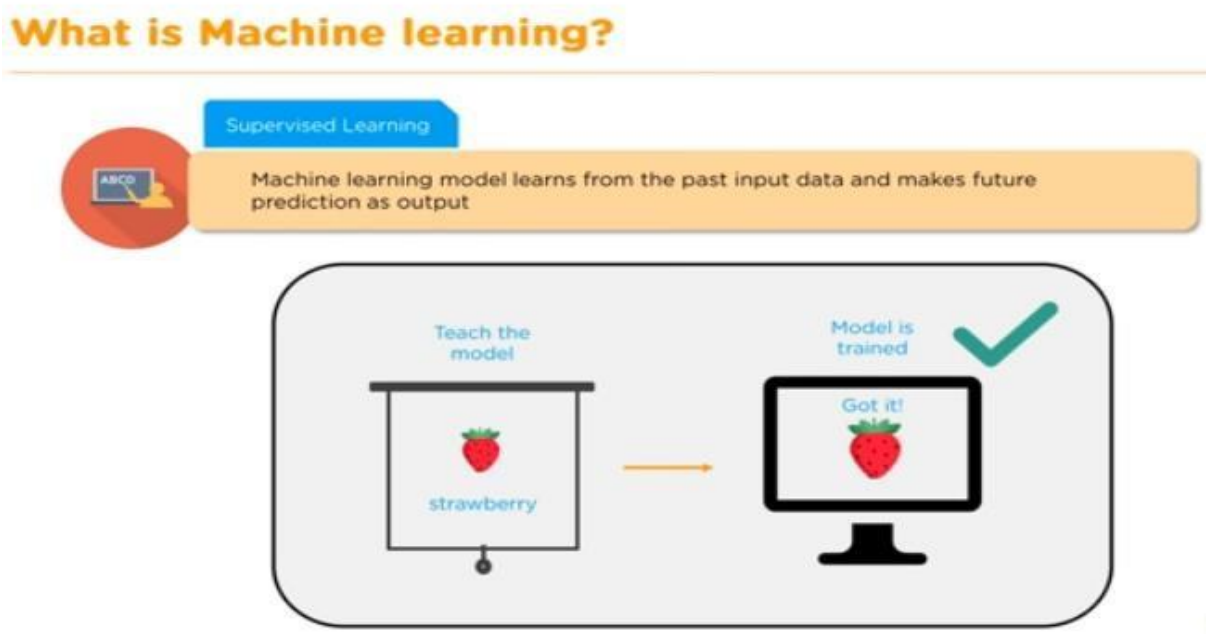


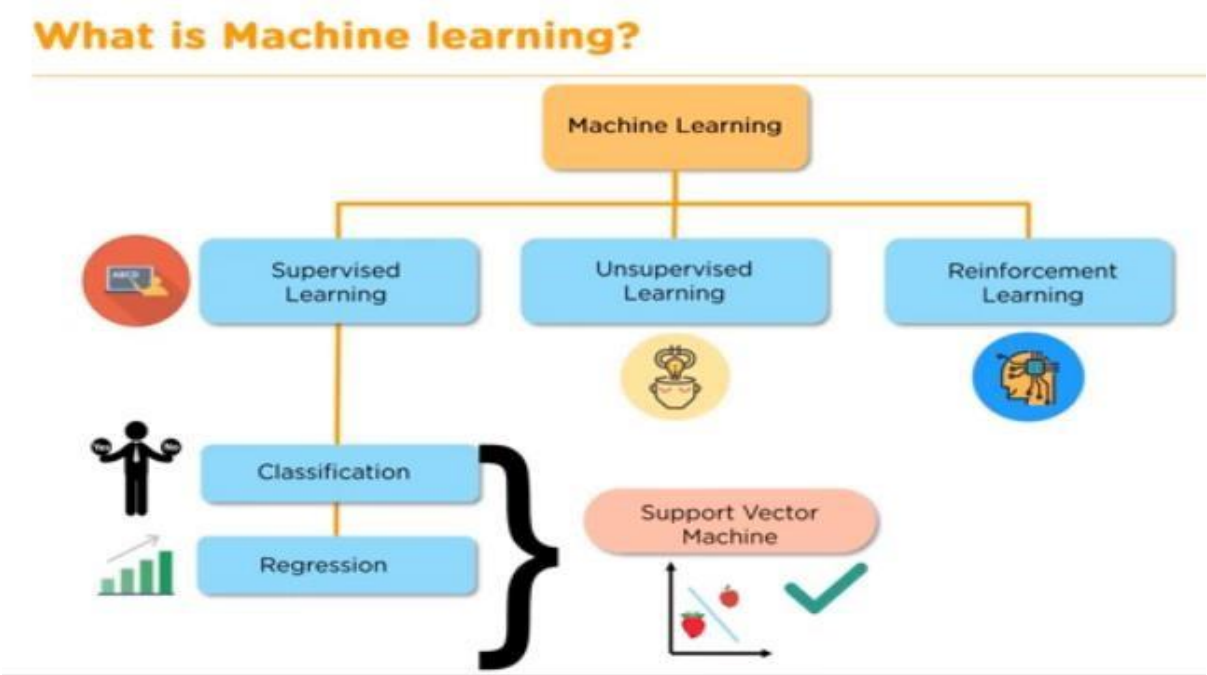

Curse of Dimensionality
Area
Size
No of Bedroom
Bed Size
Style
Dimensions
[features]
Attributes.
Threshold
M1
M2
M3
M4
M5
M6
M7
2 feet
Acc1
5 feet
Acc2
10 f
Acc3
100 f
Accuracy
200 f
1000
10000
4:17 / 7:36


Curse of Dimensionality
Area
Size
No of Bedroom
Bed Size
Style
Dimensions
[features]
Attributes.
Threshold
M1
M2
M3
M4
M5
M6
M7
2 feet
Acc1
5 feet
Acc2
10 f
Acc3
100 f
Accuracy
200 f
1000
10000
4:17 / 7:36

Model learn more features exponentially get confusion. Once it reach threshold value, The accuracy not changed. If feature is increasing exponentially from 100 to 200 or 1000. The accuracy is decreased is called curse of dimensionality,

Support Vector Machine





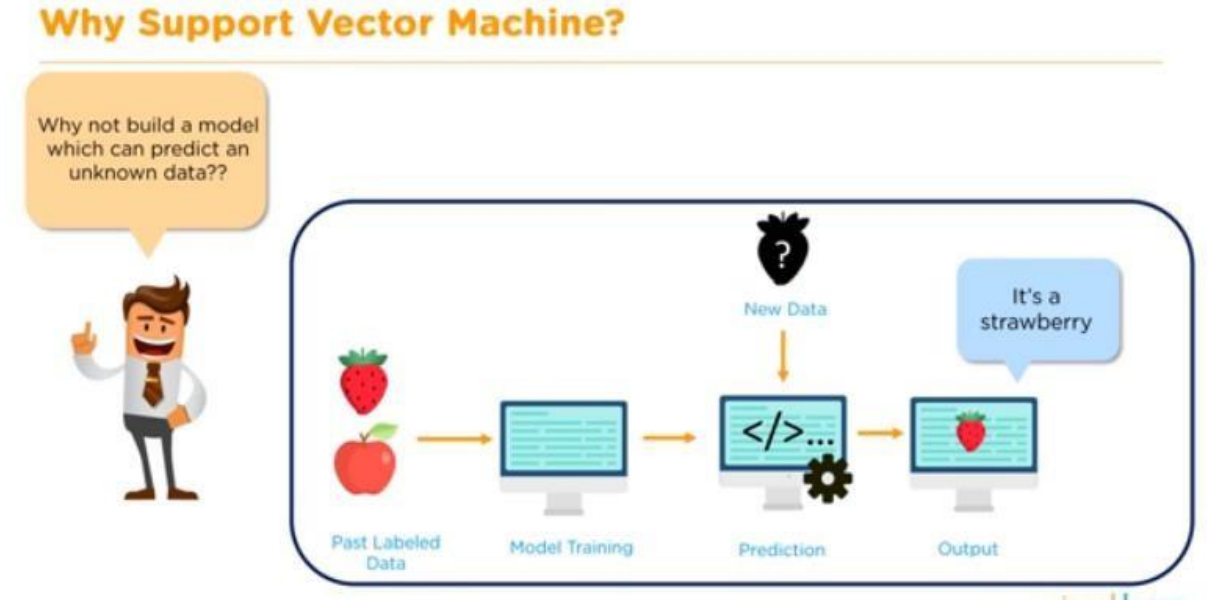
Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression.

But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized.

The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

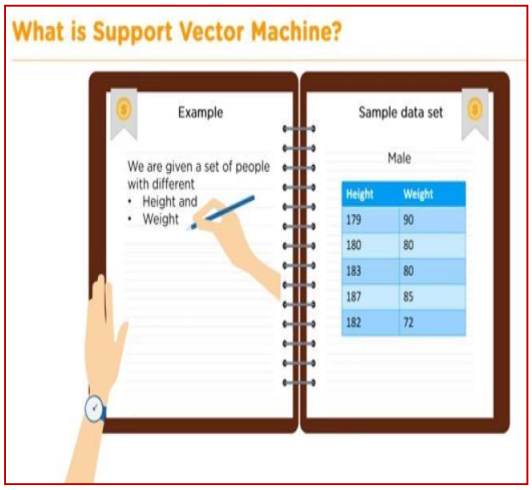


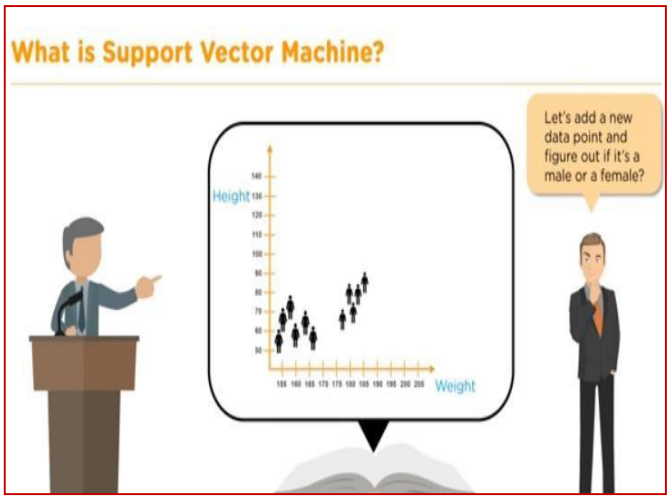
•Support Vectors − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.

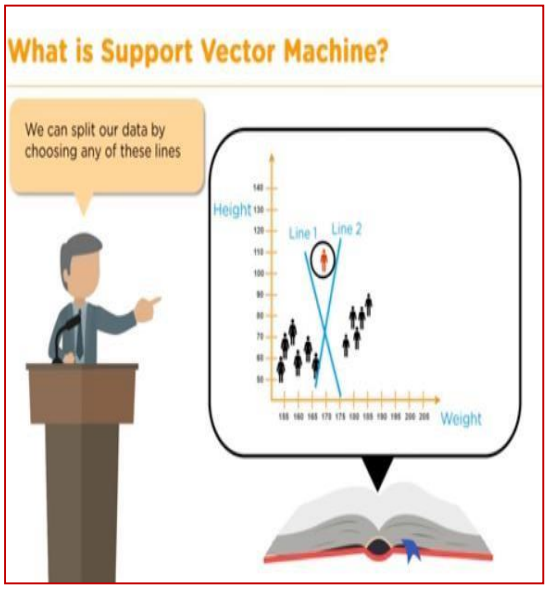
•Hyperplane − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

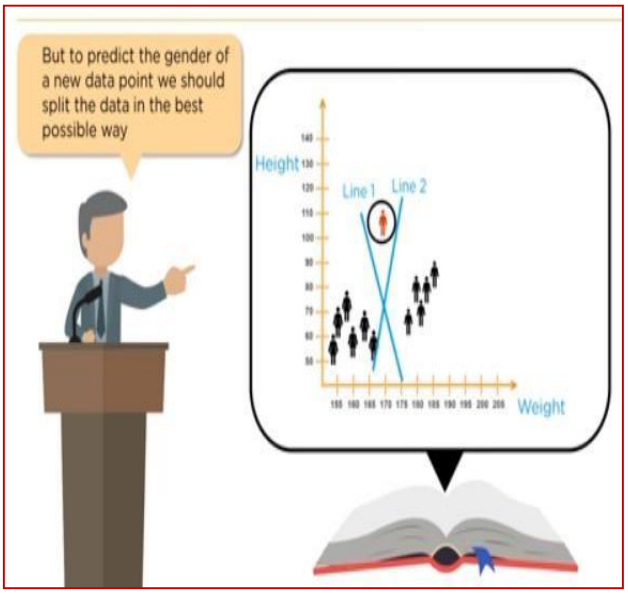
•Margin − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin. Support Vector Machine is a supervised learning method , it is a discriminative classifier that is formally designed by a separative hyperplane.

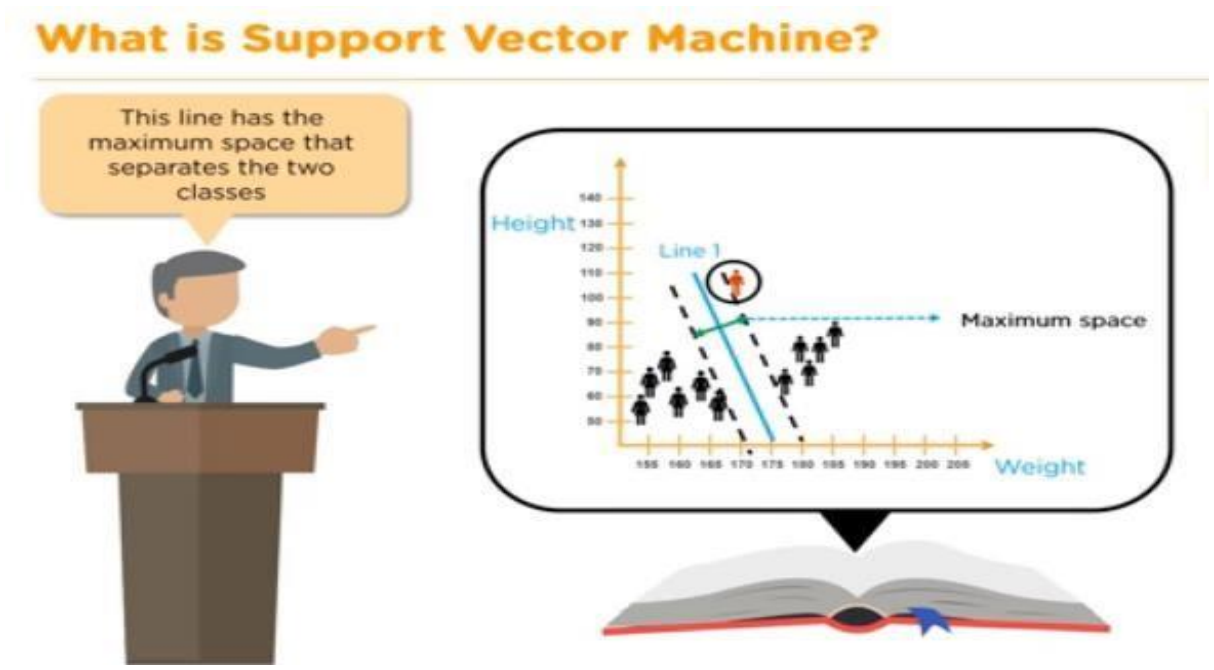
It is a representation of examples as points in space that are mapped so that the points of different categories are separated by a gap as SVM is the extreme points in the dataset Hyperplane is the maximum distance to the support vectors of any class.

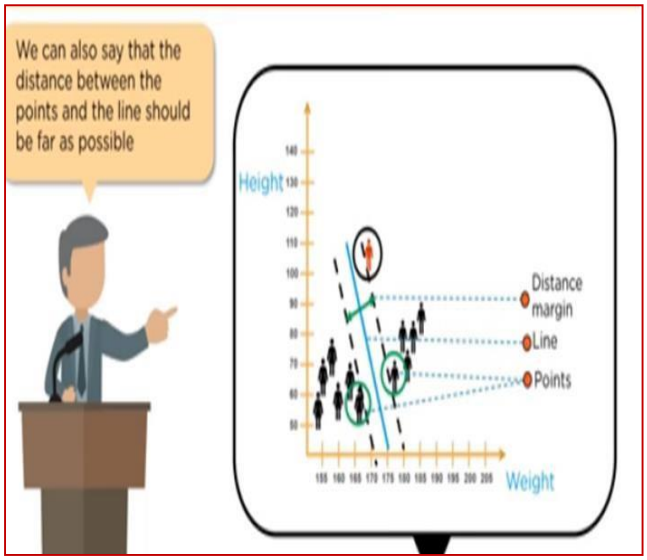

What is Support Vector Machine?
Example
We are given a set of people with different
• Height and
• Weight
Sample data set
Male
Height Weight
179 90
180 80
183 80
187 85
182 72

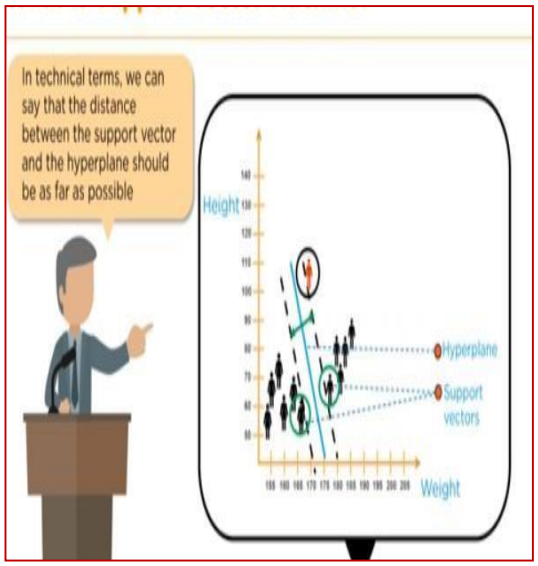

What is Support Vector Machine?
Let's add a new data point and figure out if it's a male or a female?
Height
Weight

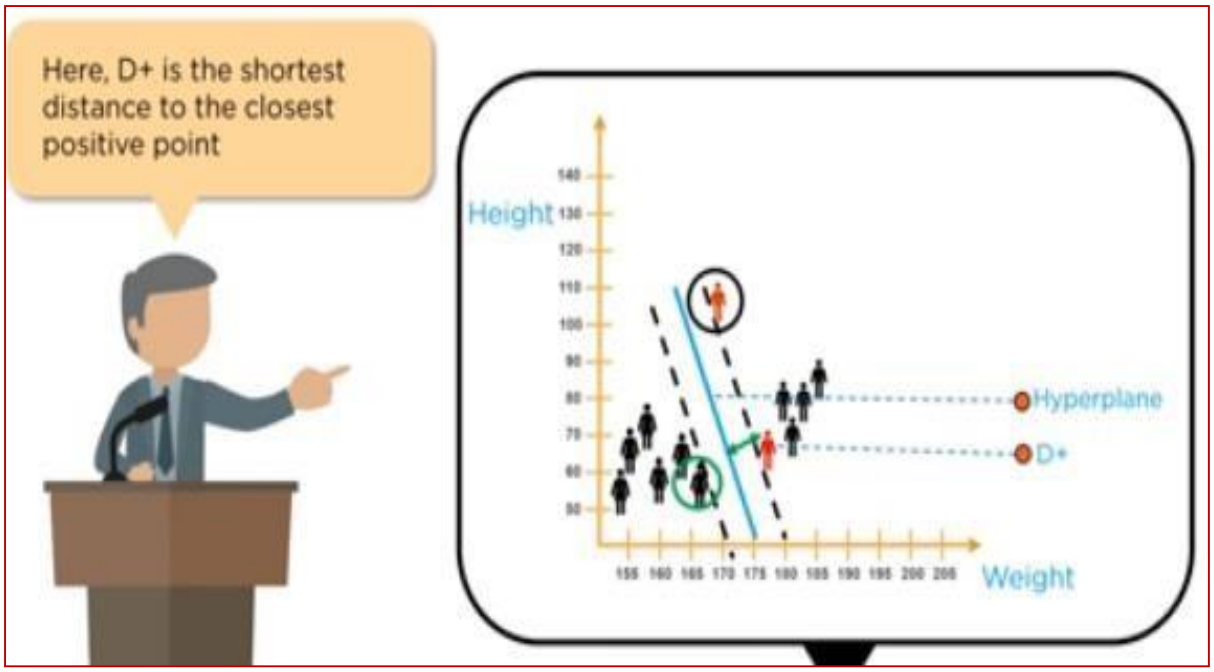

What is Support Vector Machine?
We can split our data by choosing any of these lines
Height
Line 1
Line 2
Weight

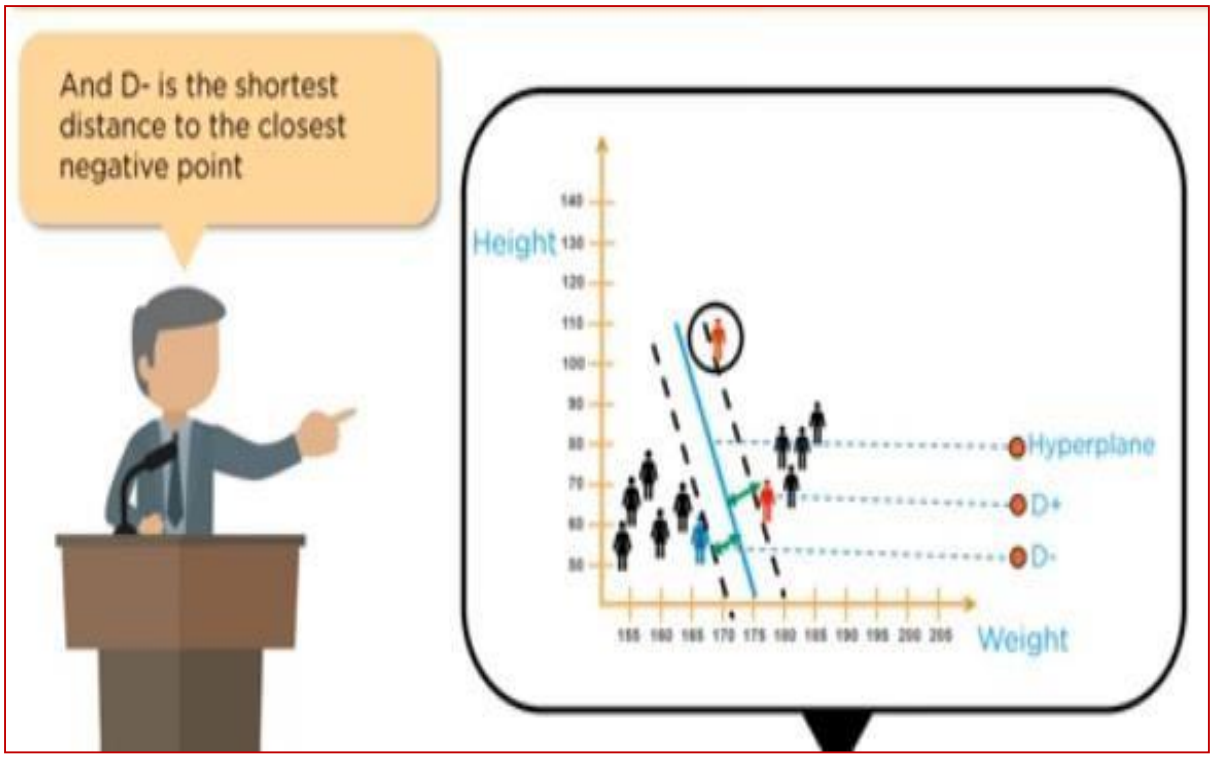

But to predict the gender of a new data point we should split the data in the best possible way
Height
Line 1
Line 2
Weight

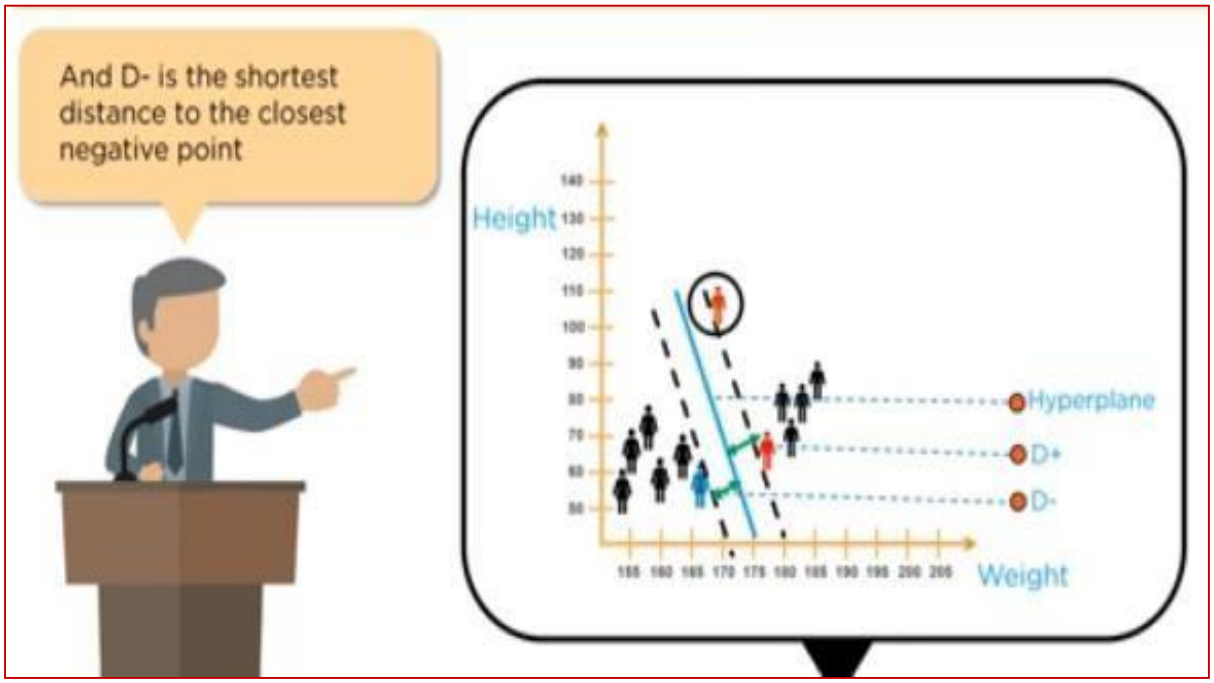

What is Support Vector Machine?
This line has the maximum space that separates the two classes
Height
Line 1
Maximum space
Weight


We can also say that the distance between the points and the line should be far as possible
Height
Distance margin
Line
Points
Weight


In technical terms, we can say that the distance between the support vector and the hyperplane should be as far as possible
Height
Hyperplane
Support vectors
Weight

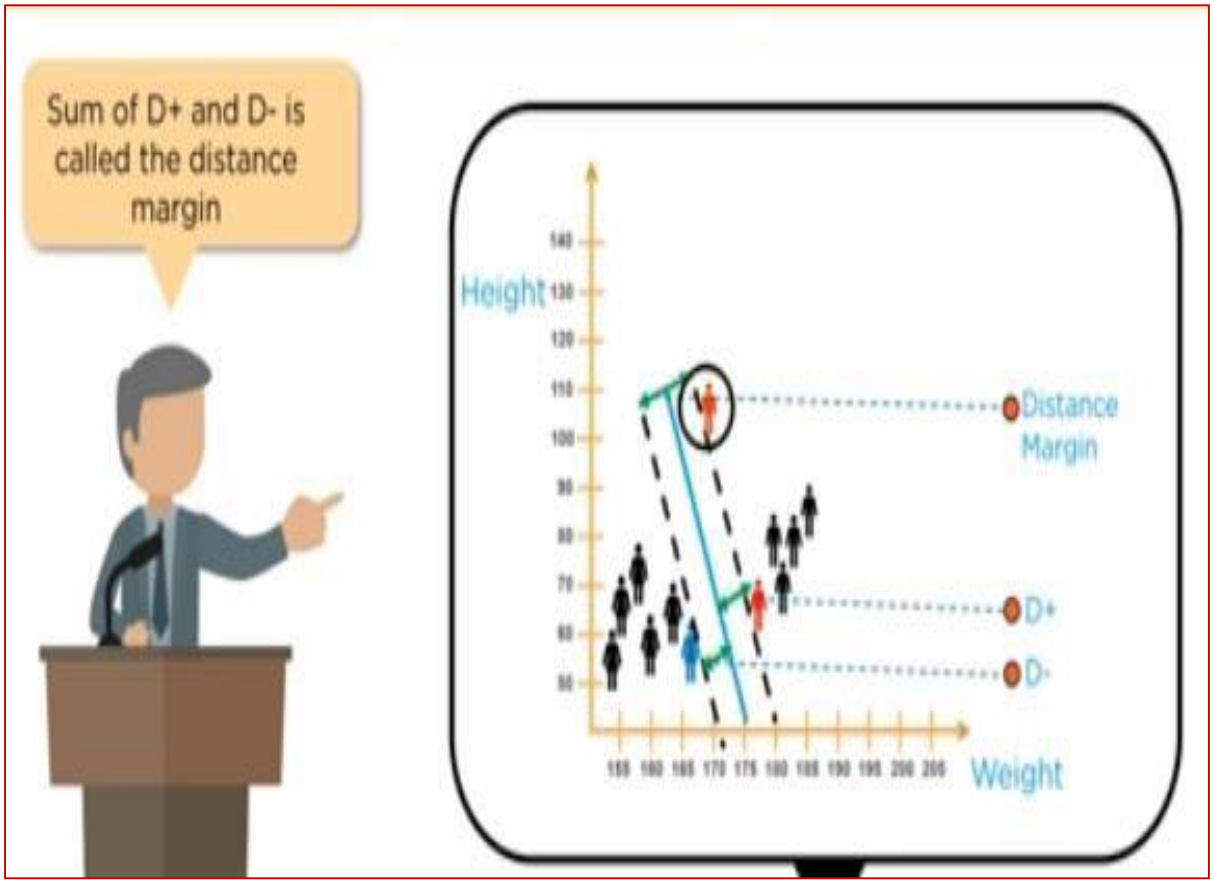

Here, D+ is the shortest distance to the closest positive point
Height
Hyperplane
D+
Weight

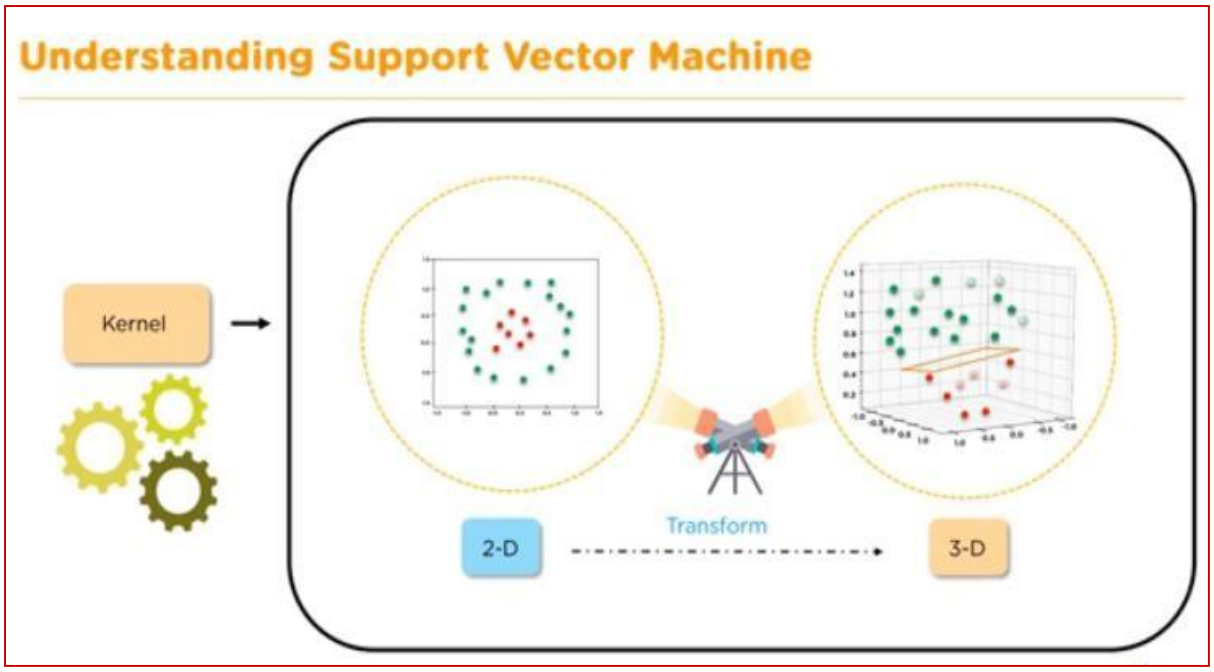


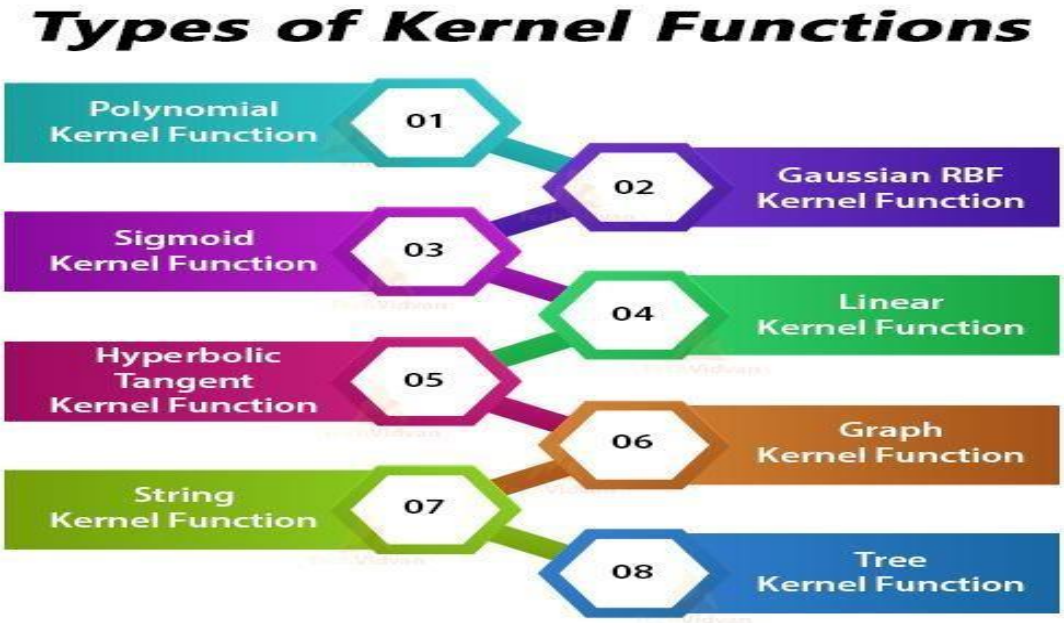


- From the distance margin – it get the optimal hyperplane.
- Based on the hyperplane , it can say the new data point belongs to male gender.

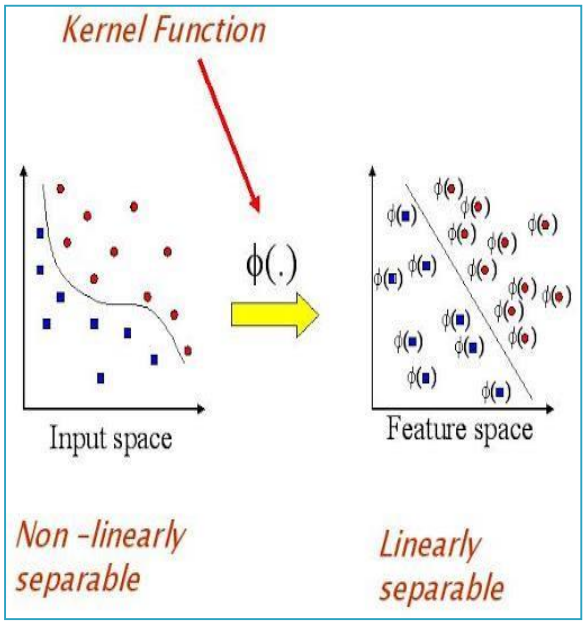
- If select a hyperplane having low margin then there is high chance of misclassification

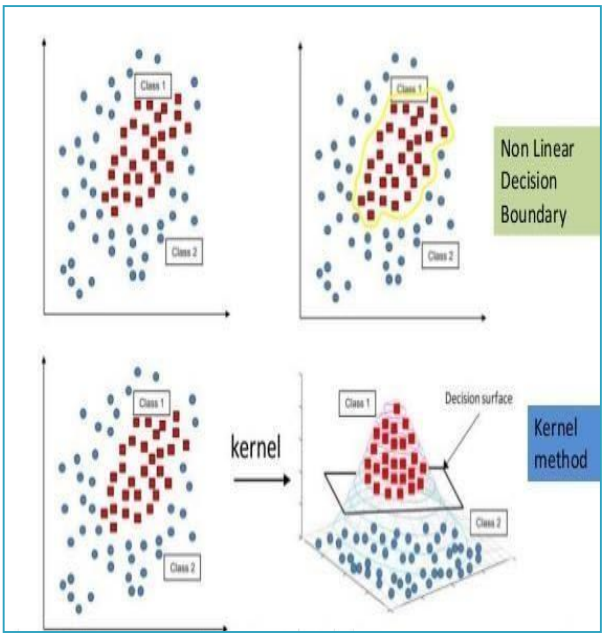


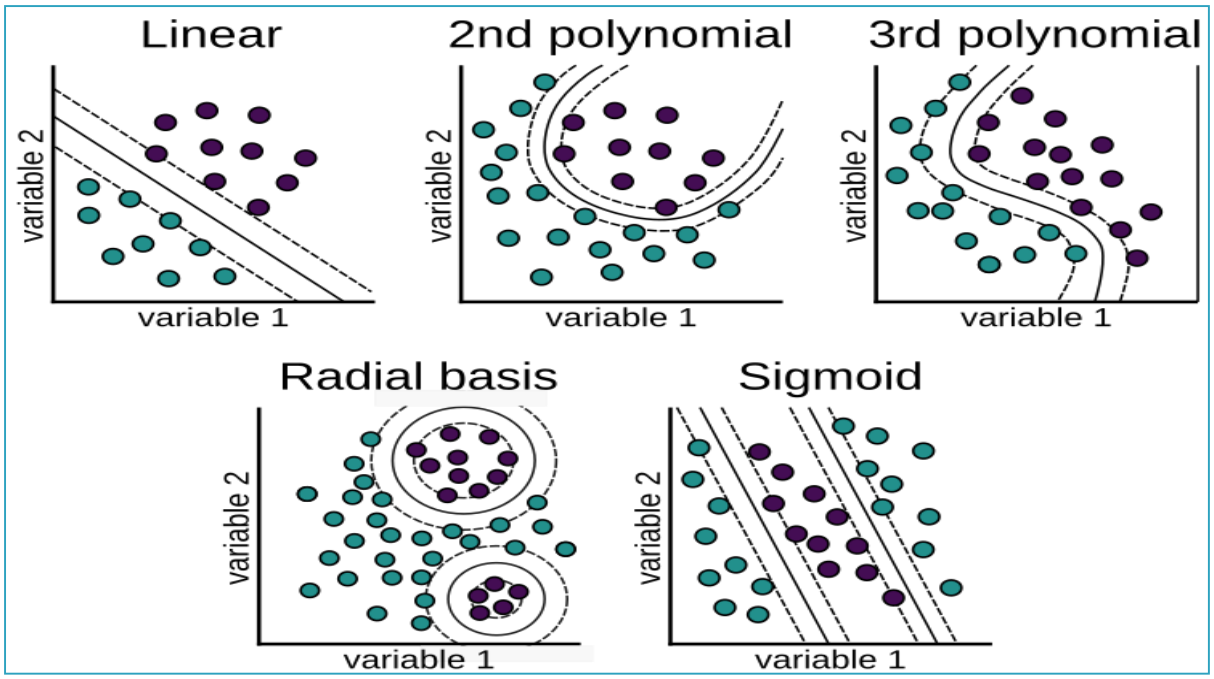


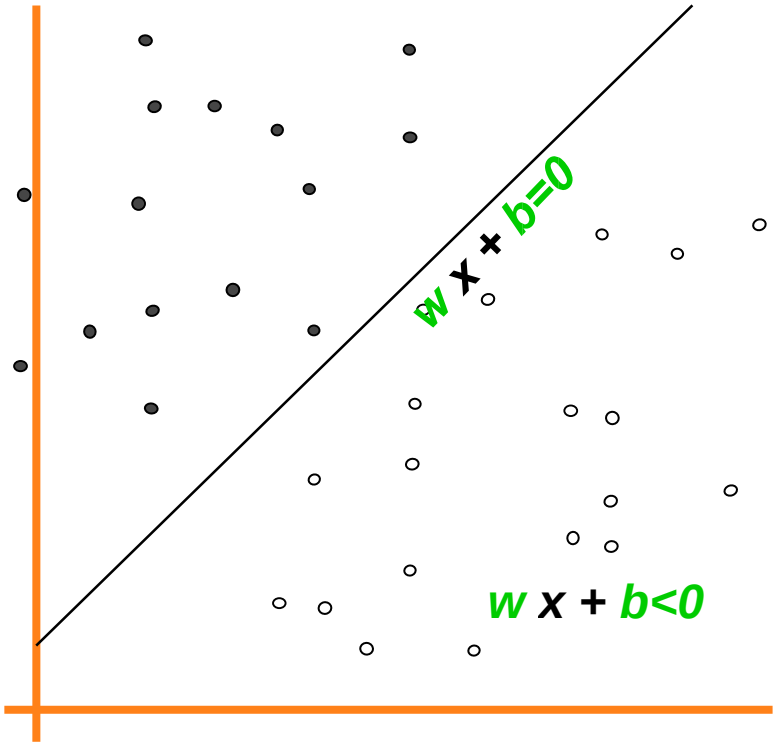


- Common kernel functions for SVM
  - linear $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
  - polynomial $k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
  - Gaussian or radial basis $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_2\right\|^2\right)$
  - sigmoid $k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$


Kernel Function
ϕ(.)
Input space
Feature space
Non -linearly separable
Linearly separable


Class 1
Class 2
Non Linear Decision Boundary
kernel
Decision surface
Kernel method


Linear
2nd polynomial
3rd polynomial
variable 2
variable 1
Radial basis
Sigmoid


w x + b=0
w x + b<0

Support Vectors with the maximum are those margin. datapoints that the margin This is the pushes up simplest kind of against SVM (Called an LSVM)Linear SVM

Linear SVM Mathematically

■ **Goal: 1) Correctly classify all training data**

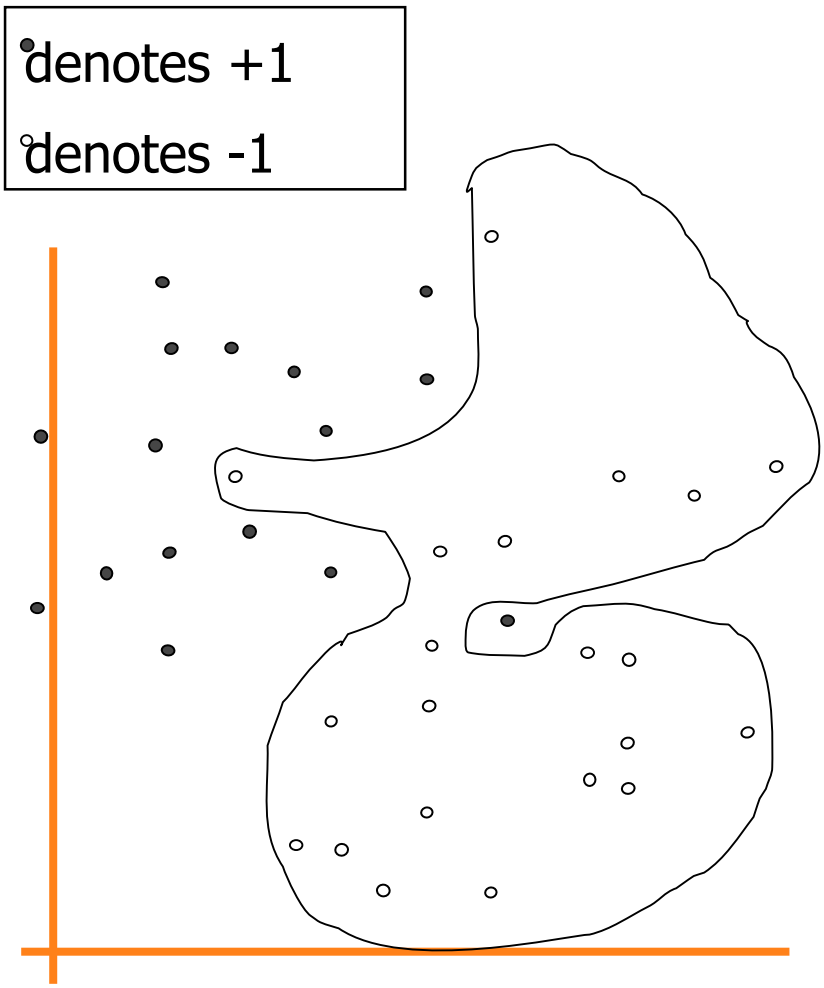
2) Maximize the Margin same as minimize

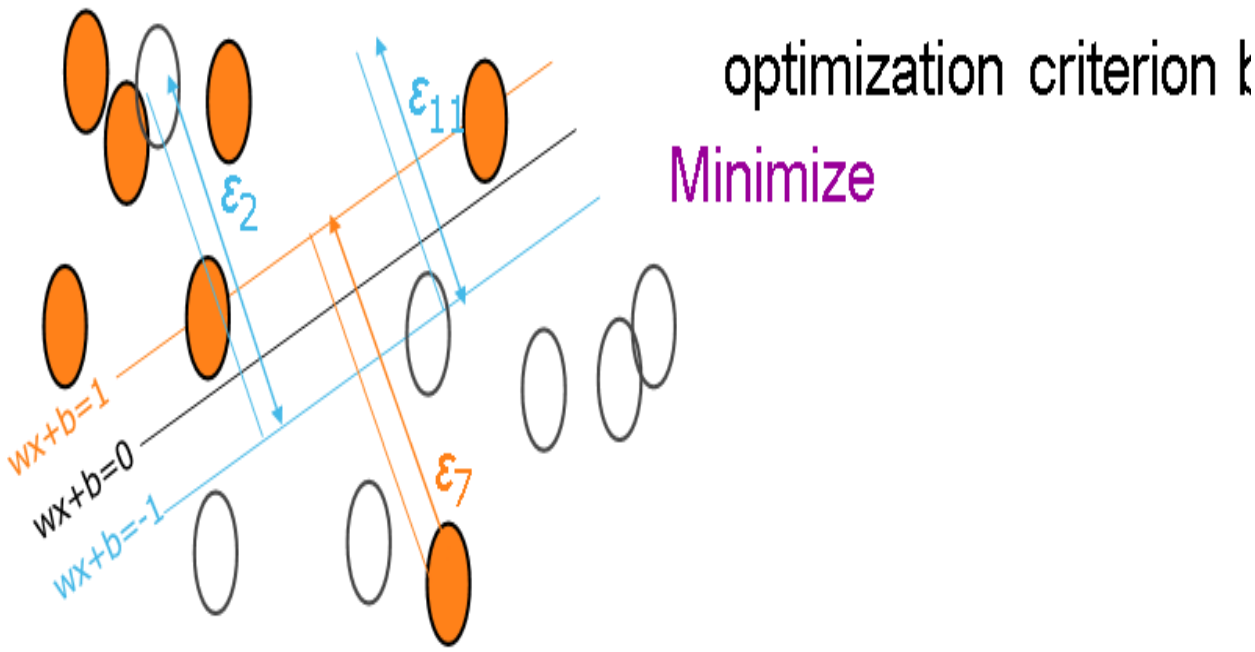
■ **We can formulate a Quadratic Optimization Problem and solve for w and b** What if the training set is noisy? - Solution 1: use very powerful kernels

OVERFITTI NG!

Soft Margin Classification

Slack variables ξi can be added to allow misclassification of difficult or noisy examples





Hard Margin v.s. Soft Margin

■ The old formulation: Find w and b such that

$\Phi(w) = \frac{1}{2}\, w^T w$ is minimized and for all $\{(x_i, y_i)\}$ $y_i (w^T x_i + b) \geq 1$

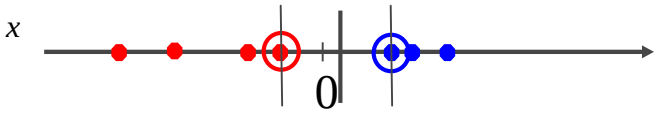
■ The new formulation incorporating slack variables: Find w and b such that

$\Phi(w) = \frac{1}{2}\, w^T w + C\Sigma\xi_i$ is minimized and for all $\{(x_i, y_i)\}$ $y_i (w^T x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

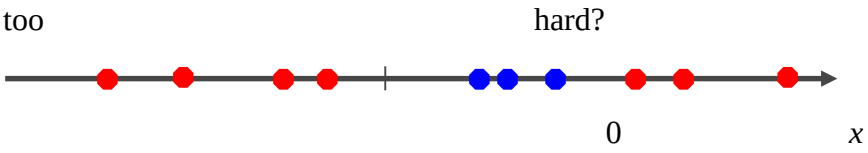
■ **Parameter *C* can be viewed as a way to control overfitting.**

Non-linear SVMs

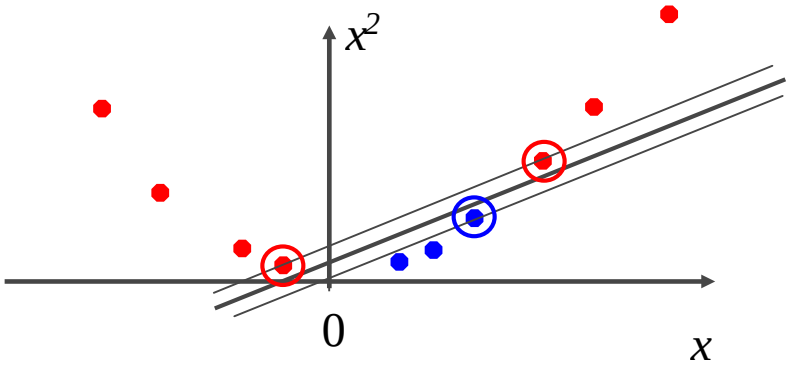
■ Datasets that are linearly separable with some noise work out great:



■ But what are we going to do if the dataset is just too hard?



■ How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

■ General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable: Nonlinear SVM - Overview

■ SVM locates a separating hyperplane in the feature space and classify points in that space

■ It does not need to represent the space explicitly, simply by defining a kernel function

■ The kernel function plays the role of the dot product in the feature space.

SVM Applications

SVM has been used successfully in many real-world problems

Weakness of SVM

It is sensitive to noise

A relatively small number of mis labelled examples can dramatically decrease the performance. It only considers two classes, how to do multi-class classification with SVM?

Answer:

1) with output parity m, learn m SVM's

◦ SVM 1 learns "Output==1" vs "Output != 1"

◦ SVM 2 learns "Output==2" vs "Output != 2" ◦ :

◦ SVM m learns "Output==m" vs "Output != m"

2)To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Application 2: Text Categorization

Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. email filtering, web searching, sorting documents by topic, etc. A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Representation of Text

IR's vector space model (aka bag-of-words representation) ■ A doc is represented by a vector indexed by a pre-fixed set or dictionary of terms

- Values of an entry can be binary or weights
- Normalization, stop words, word stems
- Doc x => φ(x)

Text Categorization using SVM

The distance between two documents is φ(x)·φ(z) K(x,z) = ⟨φ(x)·φ(z) is a valid kernel, SVM can be used with K(x,z) for discrimination.

Why SVM?

High dimensional input space

Few irrelevant features (dense concept)

Sparse document vectors (sparse instances)

Text categorization problems are linearly separable, Some Issues

Choice of kernel

Gaussian or polynomial kernel is default if ineffective, more elaborate kernels are needed domain experts can give assistance in formulating appropriate similarity measures Choice of kernel parameters. e.g. σ in Gaussian kernel, σ is the distance between closest points with different classifications . In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

Pros and Cons of SVM Classifiers

Pros of SVM classifiers

SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

Cons of SVM classifiers

They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.

Key terms include:

- **Epoch** — an arbitrary cut off, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation. In layman's term, a number of epochs means how many times you go through your training set.
- **Learning Rate** — "a scalar used to train a model via gradient descent. During each iteration, the **gradientdescent** algorithm multiplies the learning rate by the gradient. The resulting product is called the **gradient step**. Learning rate is a key **hyperparameter**." Specifying the learning rate is equivalent to determining **how fast** weights change for each iteration. In Tensorflow playground, the learning rate ranges from 0.00001 to 10.
- **Activation Function** — the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found. Available activation functions in Tensorflow playground are ReLU, Tanh, Sigmoid, and Linear.
- **Regularization** — a hyperparameter to prevent overfitting. Available values are L1 and L2. **L1 computes the sum of the weights, whereas L2 takes the sum of the square of the weights.**

- **Regularization Rate** — a scalar used to specify the rate at which the model applies the regularization, ranging from 0 to 10.
- **Problem Type** — classification (categorical output) vs. regression (numerical output)
- **Ratio of the Training and Testing Sets —** the proportion of a subset to train a model and a subset to test a model. *I usually set it to 80/20*
- **Noise** — a distortion in data that is construed to be extraneous to the original data.
- **Batch Size** — "a small, randomly selected subset of the entire batch of **examples** run together in a single iteration of training or inference. The **batch size** of a mini-batch is usually between 10 and 1,000."
- **Features** — represents *an input layer* to feed in.
- **Hidden Layer** — a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. In this context, you can specify as many as you want, but bear in mind that the more hidden layer you add, the more complex the model becomes.
- **Output** — an output layer in the neural network, often involving the loss evaluation. *Loss function (or a cost function) is a method of evaluating how well the neural network performs in the given data*. If predictions deviates too much from actual results, loss function will be high. We often evaluate the losses both on training and testing sets.

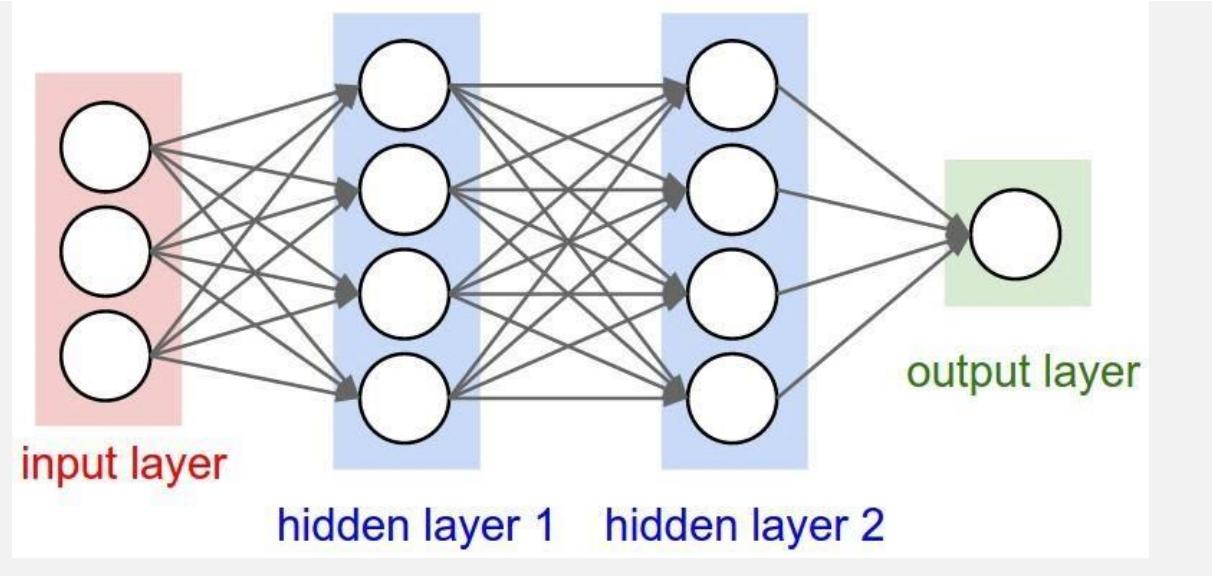


Figure 2: represents an artificial **neural network** (ANN) with multiple layers between the input and output layers. For example, given input data of image pixels from MNIST dataset, we can specify 2 hidden layers, each of which has 4 hidden neurons. Ultimately, we have the predicted probabilities of the possible number for the given image. Image Source: Deep learning — Convolutional neural networks and feature extraction with Python, Perone (2015)

UNIT – III TREE AND PROBABILISTIC MODELS

Learning with Trees – Decision Trees – Constructing Decision Trees – Classification and Regression Trees – Ensemble Learning – Boosting – Bagging – Different ways to Combine Classifiers – Probability and Learning – Data into Probabilities – Basic Statistics – Gaussian Mixture Models – Nearest Neighbor Methods – Unsupervised Learning – K means Algorithms – Vector Quantization – Self Organizing Feature Map.

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning.

Problems in Machine learning

Classification:

Problems with categorical solutions like 'yes' or 'No' ,'True' or 'False','1' or '0'.

Regression:

Problems wherein continuous value needs to be predicted like 'Product Prices', 'Profit'.

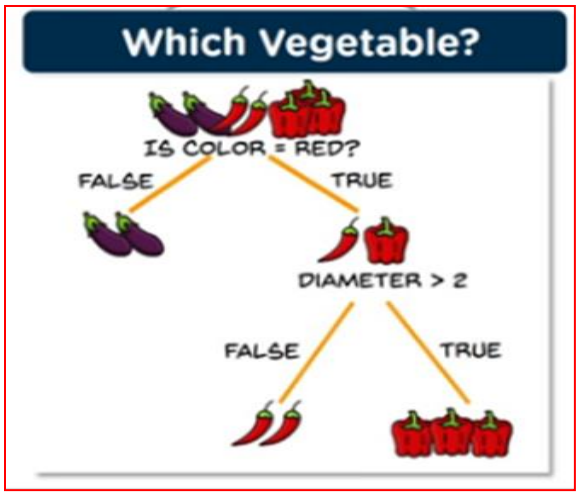
Clustering:

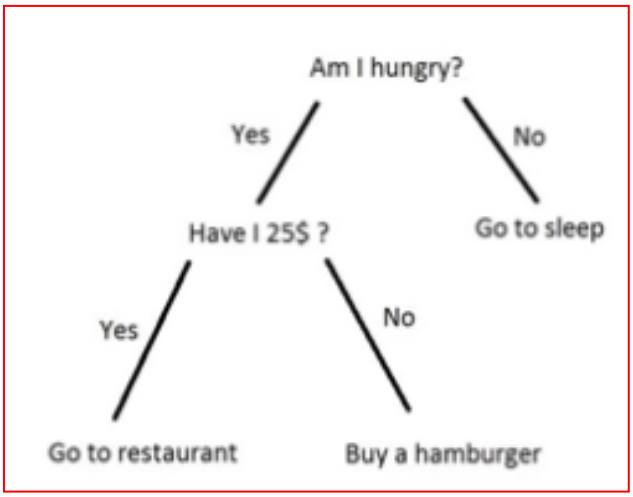
Problems wherein the data needs to be organized to find specific patterns like in the case of 'Product' Recommendation

- Classification is the process of dividing the datasets into different categories or groups by adding label. Ie., It adds the data point to a particular labelled group on the basis of some condition.
- Types of Classification
    - **Decision Tree**
    - Random Forest
    - Naïve Bayes
    - K Nearest Neighbour
    - Logistic Regression

A classification tree will determine a set of logical if then conditions to classify problems. For example, discriminating between three types of flowers based on certain features.

- A decision tree is a graphical representation of all possible solutions to a decision based on certain conditions.
- It is a tree shaped diagram used to determine a course of action. Each branch of the tree represents a possible decision, occurrence or relation.





Advantages of Decision Tree

- Simple to understand, interpret and visualize
- Little effort for data preparation and less requirement of data cleaning
- Can handle both numerical and categorical data
- Useful for solving decision related problems.
- Non linear parameters don't effect its performance

Disadvantages

- Over fitting occurs when the algorithm captures noise in the data
- High variance- The model can get unstable due to small variation in data
- Low biased tree- A highly complication DT tends to have a low bias which makes it difficult for the model to work with new data

Motivation for tree based models

- Handling of categorical variables

- Handling of missing values and unknown levels
- Detection of nonlinear relationships
- Visualization and interpretation in decision trees.

3.1 Decision Tree- Terminology

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

Building a Decision tree

There are several algorithms to build a decision tree.

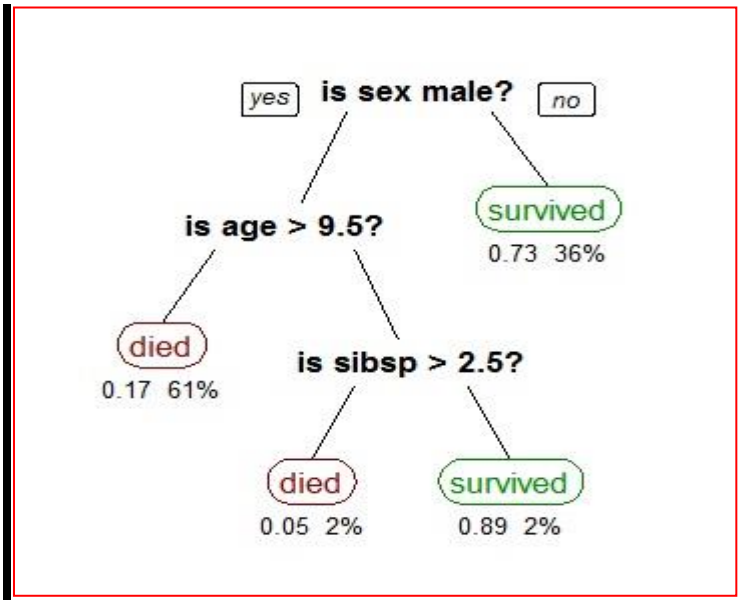
- CART-Classification And Regression Trees –Gini Index
- ID3-Iterative Dichotomiser 3
    - Entropy function
    - Information Gain
- C4.5
- CHAID-Chi-squared Automatic Interaction Detection

Only CART and ID3 algorithms as they are the ones majorly used.

A Decision tree is tree which each node represents a Feature(Attribute) , each link (branch) represents a Decision (Rule) and each leaf represents an outcome.

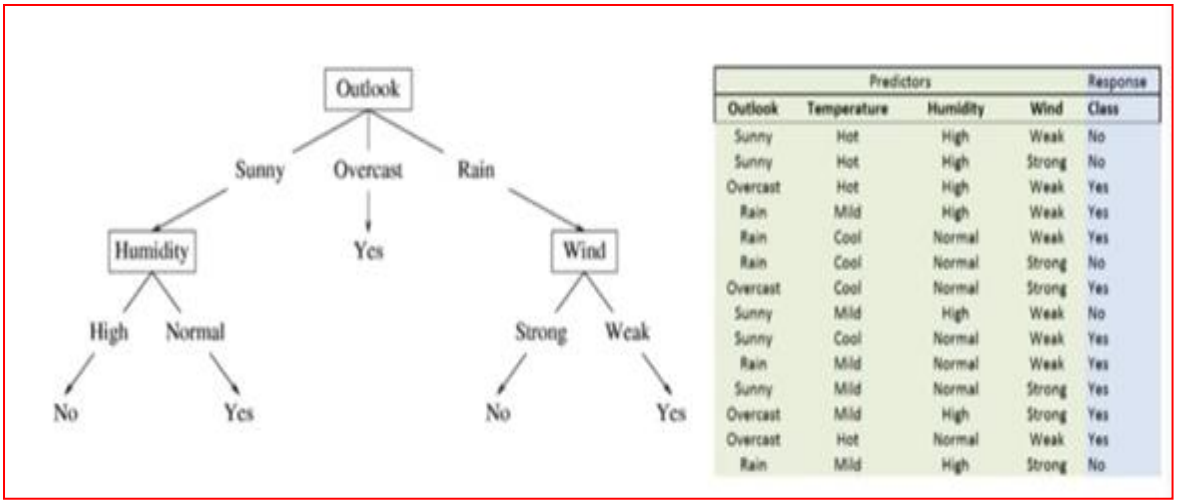
How can an algorithm be represented as a tree?

For this let's consider a very basic example that uses titanic data set for predicting whether a passenger will survive or not. Below model uses 3 features/attributes/columns from the data set, namely sex, age and sibsp (number of spouses or children along).



A possible decision tree for the data:

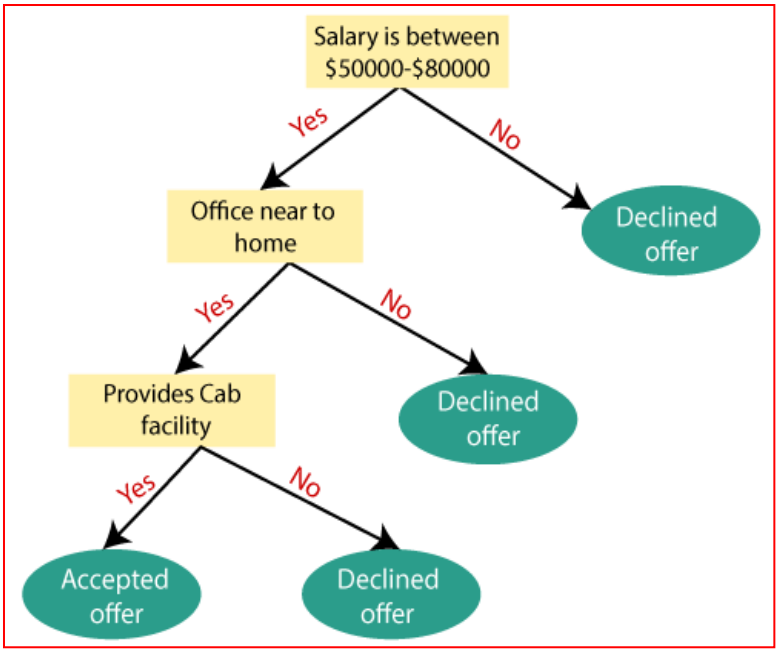
- Each internal node: test on attribute X_i
- Each branch from a node: selects one value for X_i
- Each leaf node: Predict Y



Predictors				Response
Outlook	Temperature	Humidity	Wind	Class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the **root node of the tree.** This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the **attribute value** with the other **sub-nodes** and move further. It continues the process until it reaches the **leaf node** of the tree.



The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.

- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Attribute Selection Measures

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

These are popular techniques for ASM:

- **Information Gain-ID3**
- **Gini Index-CART**
- **Entropy-ID3**
- Gain Ratio-C4.5
- Reduction in Variance-C4.5
- Chi-Square-CHAID

Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, split the node and build the decision tree.

ie., decide which attribute should be selected as the decision node

- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)-[{Weighted avg)* Entropy(Each feature)]

For example

IG(T,X)=Entropy(T)-Entropy(T,X)

IG(PlayGolf,Outlook)=E(PlayGolf)-E(PlayGolf,Outlook)

= 0.940-0.693=0.247

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

Entropy can be calculated as:

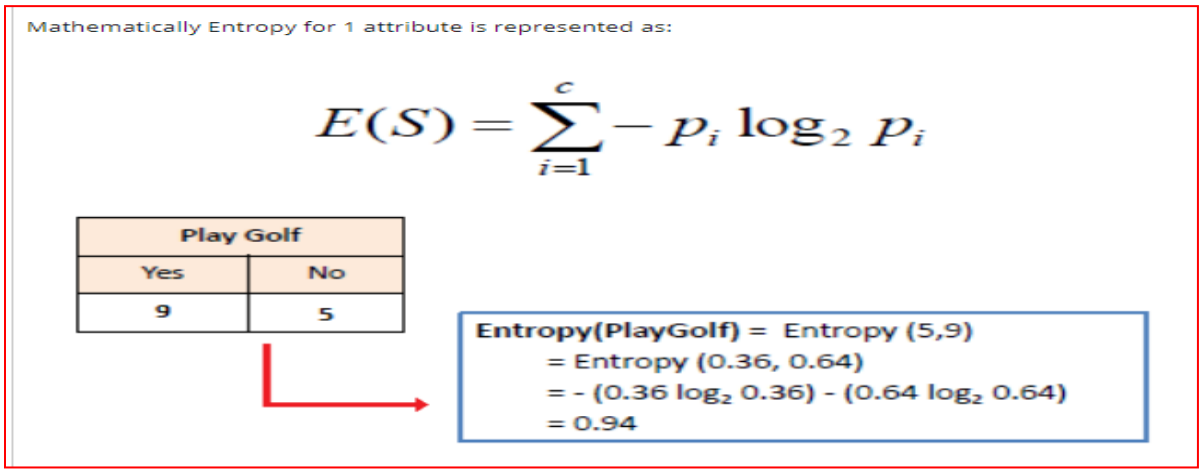
$$\text{Entropy}(s) = P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

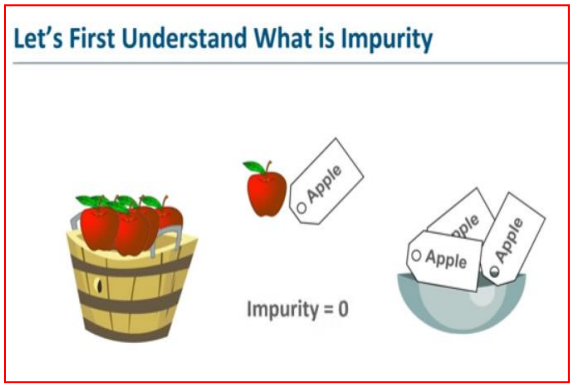
Where,

S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no





Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Steps to Calculate Gini index for a split

- Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p^2+q^2).
- Calculate the Gini index for split using the weighted Gini score of each node of that split.

CART (Classification and Regression Tree) uses the Gini index method to create split points.

Pruning: Getting an Optimal Decision tree

- *Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning.

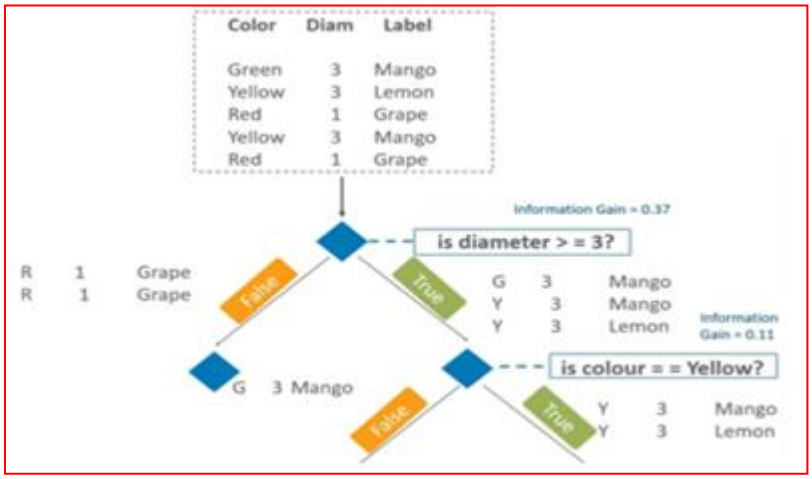
There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

What is a CART in Machine Learning?

- A Classification and Regression Tree(CART) is a predictive algorithm used in machine learning. It explains how a target variable's values can be predicted based on other values.
- It is a decision tree where each fork is a split in a predictor variable and each node at the end has a prediction for the target variable.
- The CART algorithm is an important decision tree algorithm that lies at the foundation of machine learning.
- Moreover, it is also the basis for other powerful machine learning algorithms like bagged decision trees, random forest and boosted decision trees.

The Classification and regression tree(CART) methodology is one of the oldest and most fundamental algorithms. It is used to predict outcomes based on certain predictor variables.



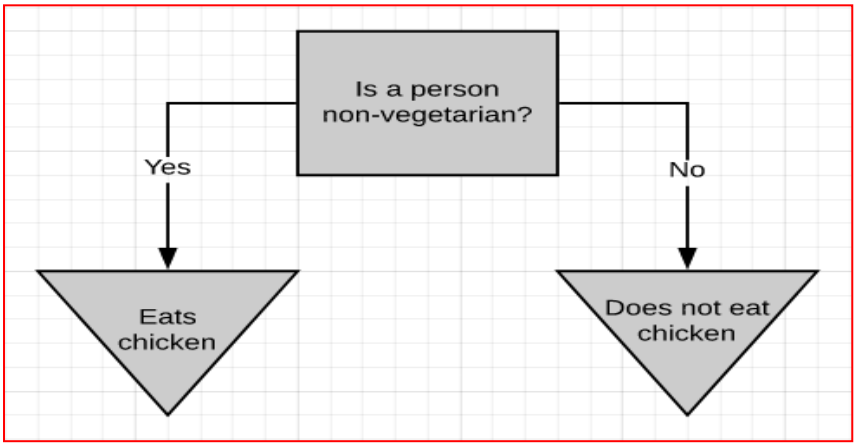
Sample Dataset(was Tennis Played?)

- Columns denote features Xi
- Rows denote labeled instances xi,yi
- Class label denotes whether a tennis game was played

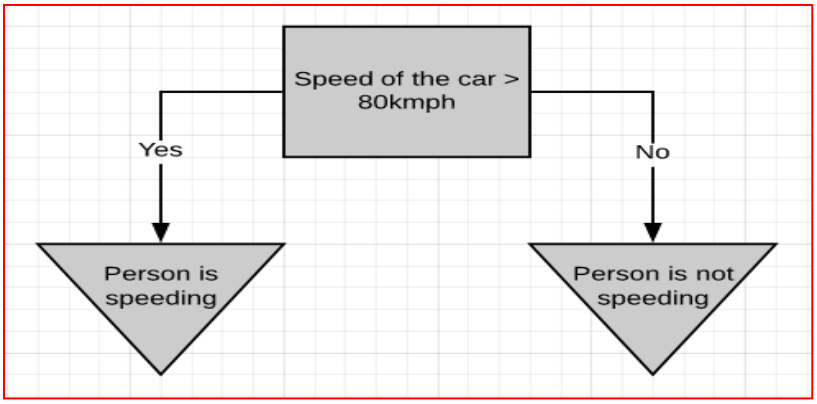
x_i, y_i

Predictors				Response
Outlook	Temperature	Humidity	Wind	Class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

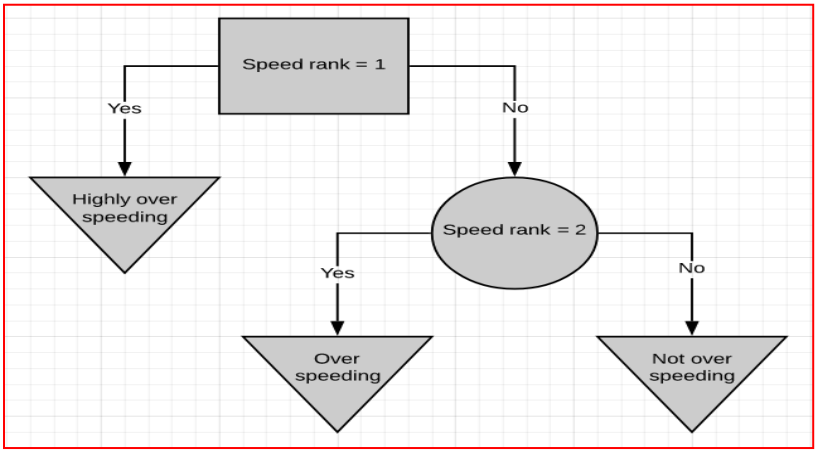
- **Decision Trees (DTs)** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.



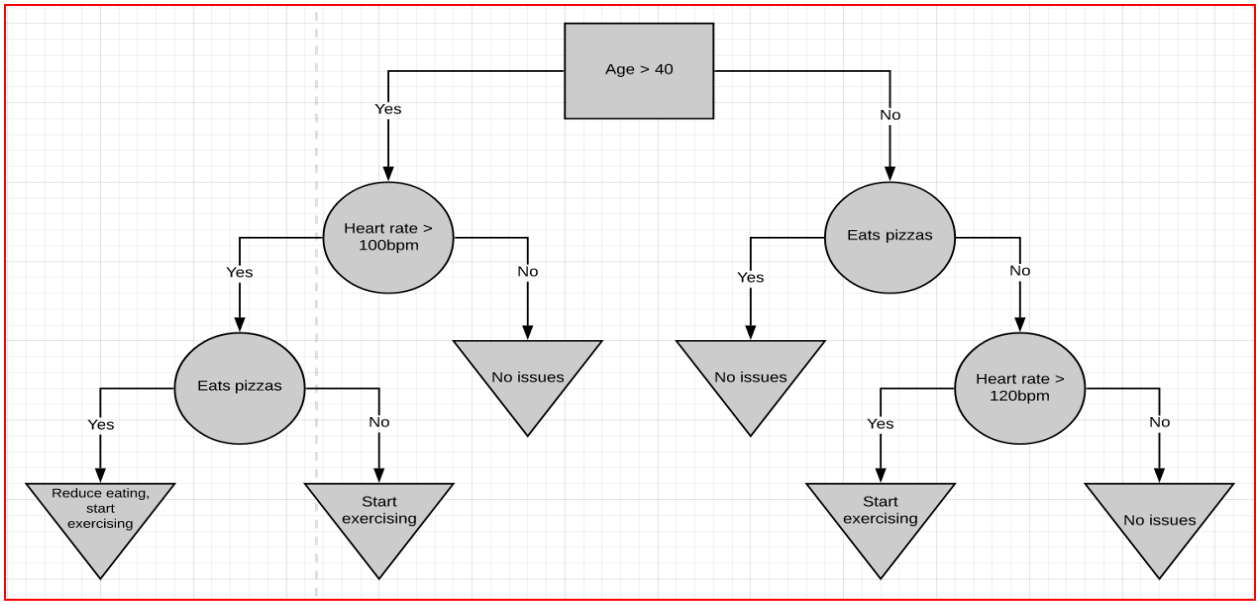
Example 2: Decision tree is based on numeric data. If a person is driving above 80kmph, we can consider it as over-speeding, else not.



Example 3: If a person is driving above 80kmph, we can consider it as over-speeding, else not.Here is one more simple decision tree. This decision tree is based on ranked data, where 1 means the speed is too high, 2 corresponds to a much lesser speed. If a person is speeding above rank 1 then he/she is highly over-speeding. If the person is above speed rank 2 but below speed rank 1 then he/she is over-speeding but not that much. If the person is below speed rank 2 then he/she is driving well within speed limits.



The classification in a decision tree can be both categorical or numeric which is used in Bioinformatics



- The tree is called the ***root node*** . The nodes in between are called ***internal nodes***. Internal nodes have arrows pointing to them and arrows pointing away from them. The end nodes are called the ***leaf nodes*** or just ***leaves***. Leaf nodes have arrows pointing to them but no arrows pointing away from them.

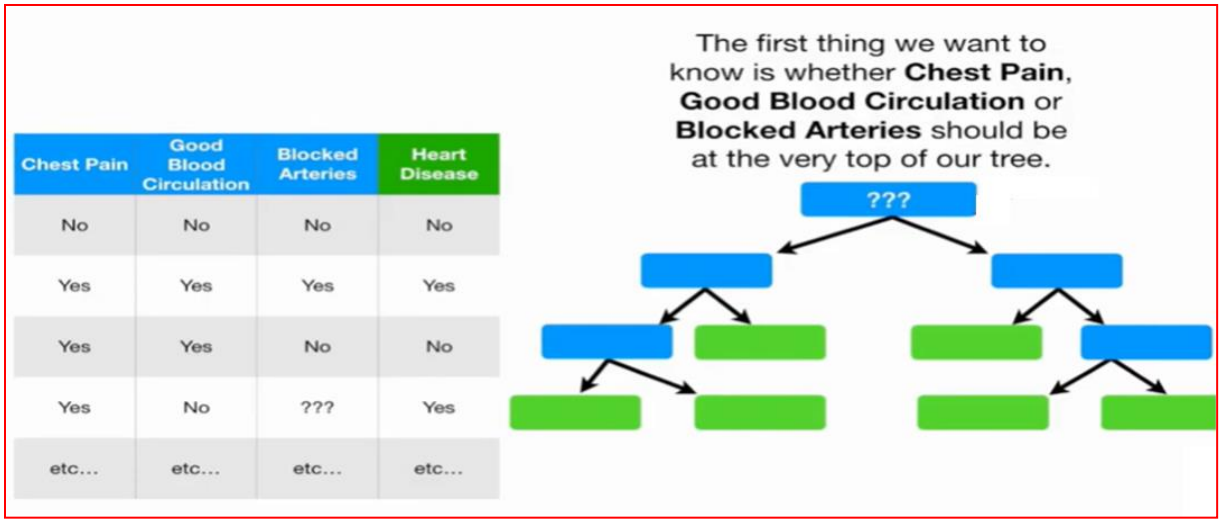
- In the above diagrams, root nodes are represented by rectangles, internal nodes by circles, and leaf nodes by inverted-triangles.

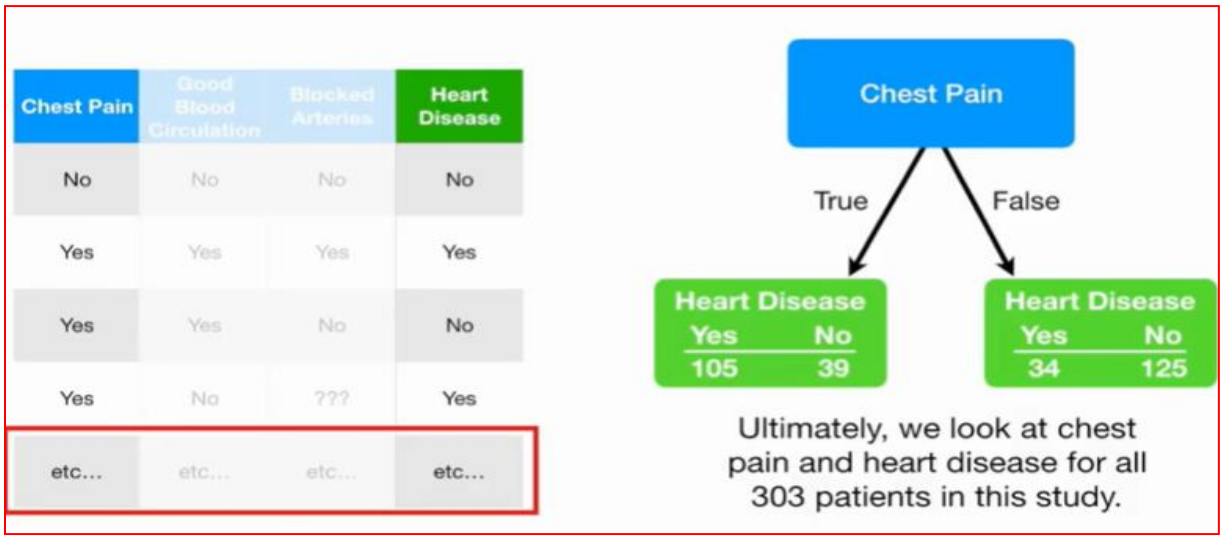
Classification and Regression Trees

CART is a DT algorithm that produces binary Classification or Regression Trees, depending on whether the dependent (or target) variable is categorical or numeric, respectively. It handles data in its raw form (no preprocessing needed) and can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables.

- In the example given, build a decision tree that uses chest pain, good blood circulation, and the status of blocked arteries to predict if a person has heart disease or not.

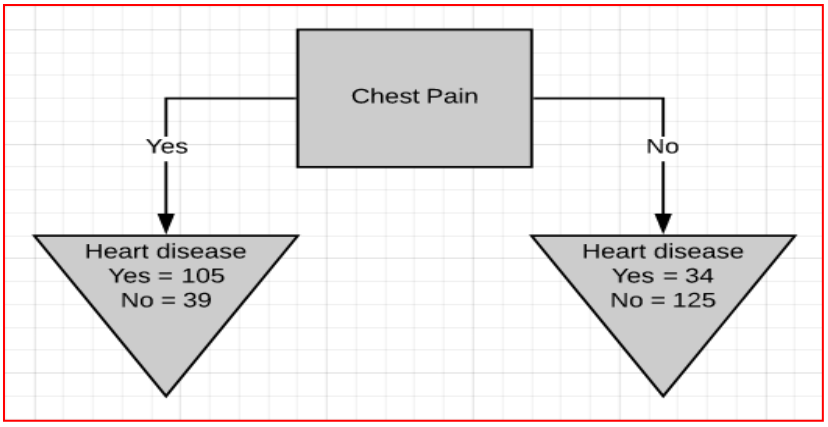
Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
NO	NO	NO	NO
YES	YES	YES	YES
YES	YES	NO	NO
YES	NO	YES	YES
etc.	etc.	etc.	etc.





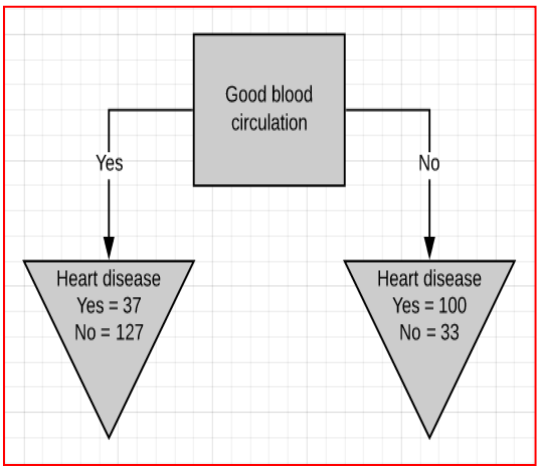
There are two leaf nodes, one each for the two outcomes of chest pain. Each of the leaves contains the no. of patients having heart disease and not having heart disease for the corresponding entry of chest pain

Chest pain as the root node

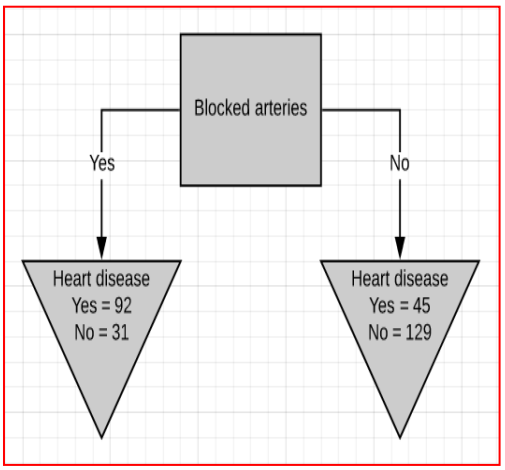


There are two leaf nodes, one each for the two outcomes of chest pain. Each of the leaves contains the no. of patients having heart disease and not having heart disease for the corresponding entry of chest pain.

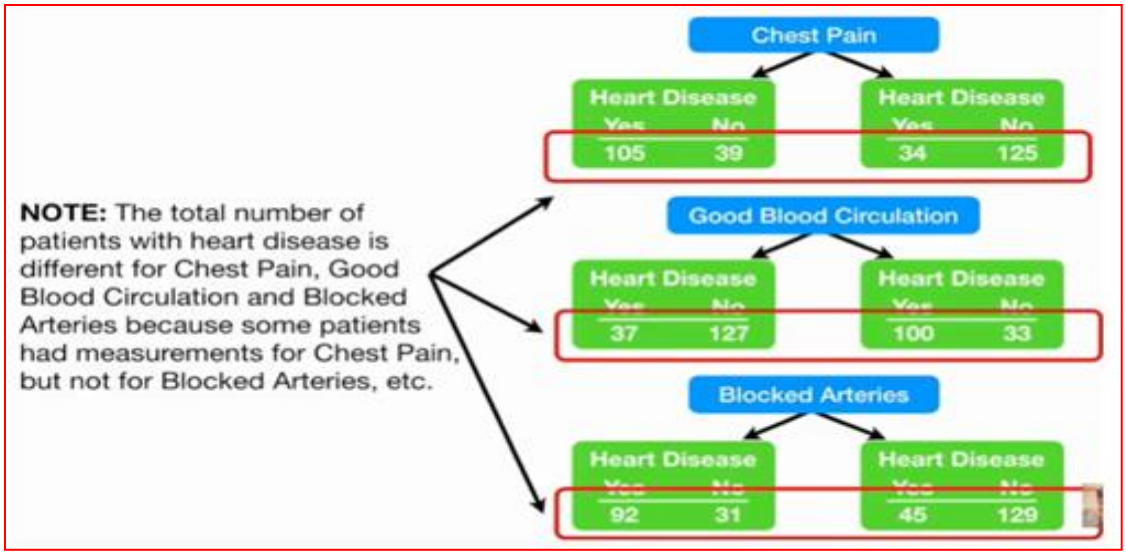
- **The same thing for good blood circulation and blocked arteries.**

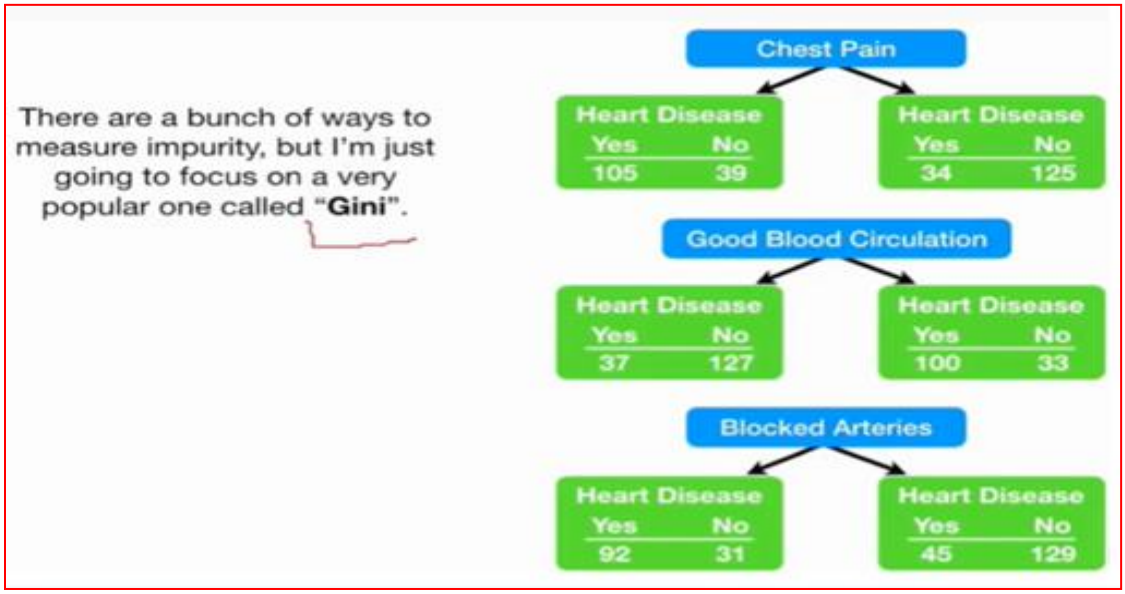


Good blood circulation as the root node



Blocked arteries as the root node





- The 3 features separates the patients having heart disease from the patients not having heart disease perfectly. It is to be noted that the total no. of patients having heart disease is different in all three cases. This is done to simulate the missing values present in real-world datasets.
- Because none of the leaf nodes is either 100% 'yes heart disease' or 100% 'no heart disease', they are all considered *impure.* To decide on which separation is the best, we need a method to measure and compare *impurity.*
- The metric used in the CART algorithm to measure impurity is the *Gini impurity score*.

  Gini impurity = 1 - (probability of 'yes')² - (probability of 'no')²

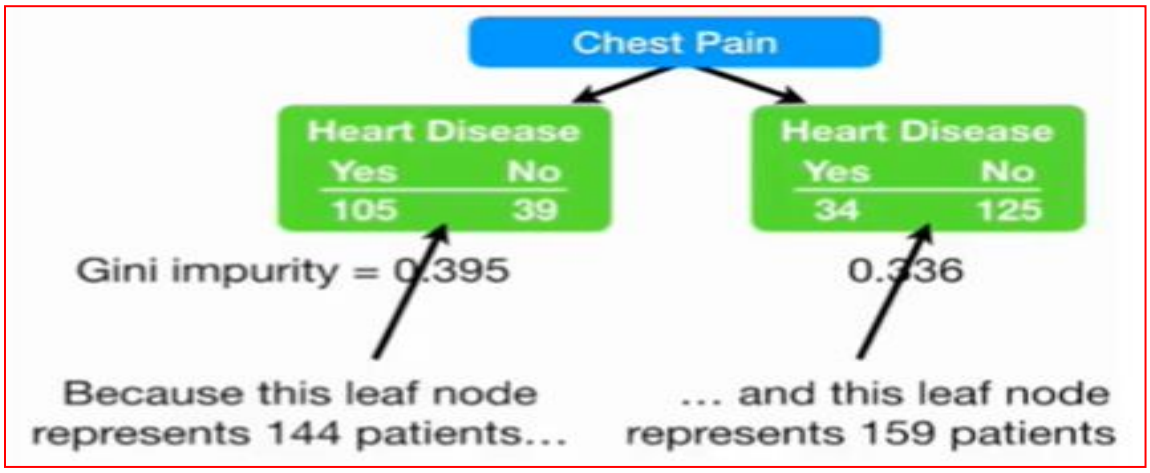
Chest Pain

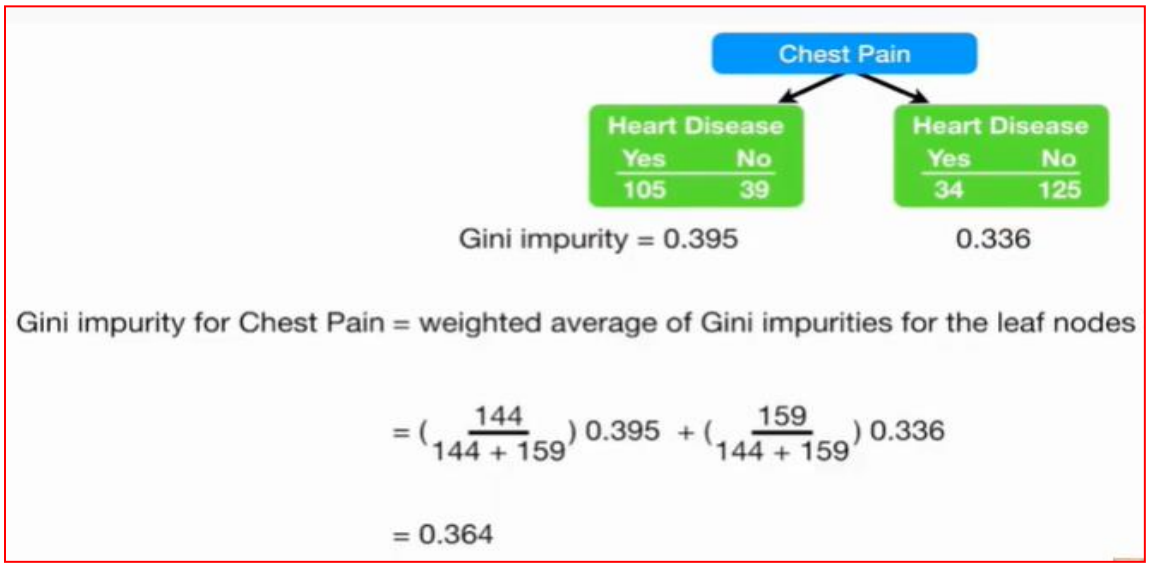
Calculating the Gini impurity for chest pain for the left leaf,

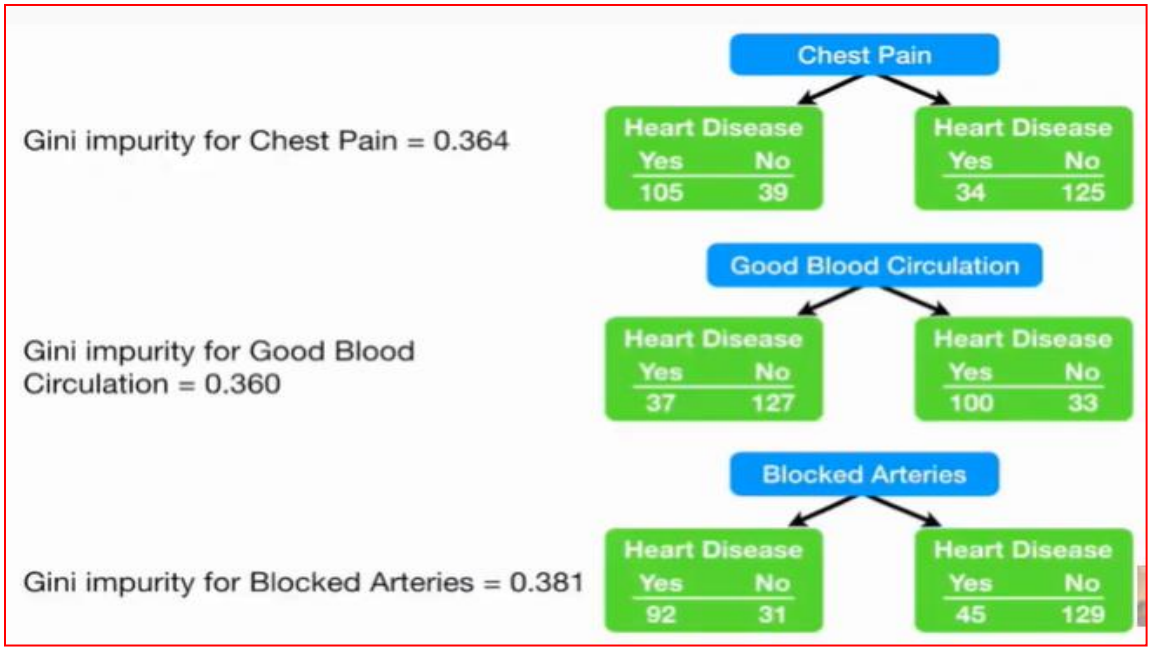
- Gini impurity = 1 - (probability of 'yes')² - (probability of 'no')²
  = 1 - (105/105+39)² - (39/105+39)²
  Gini impurity = 0.395

Similarly, calculate the Gini impurity for the right leaf node.

- Gini impurity = 1 - (probability of 'yes')² - (probability of 'no')²
  = 1 - (34/34+125)² - (125/34+125)²
  Gini impurity = 0.336

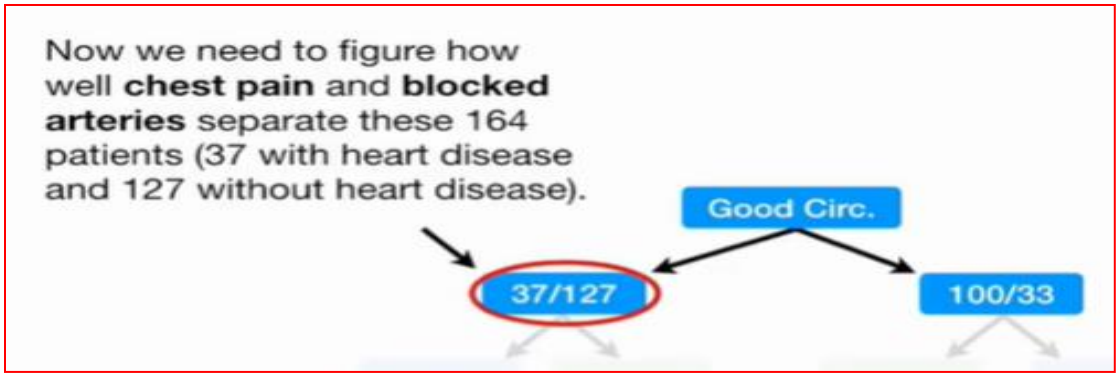


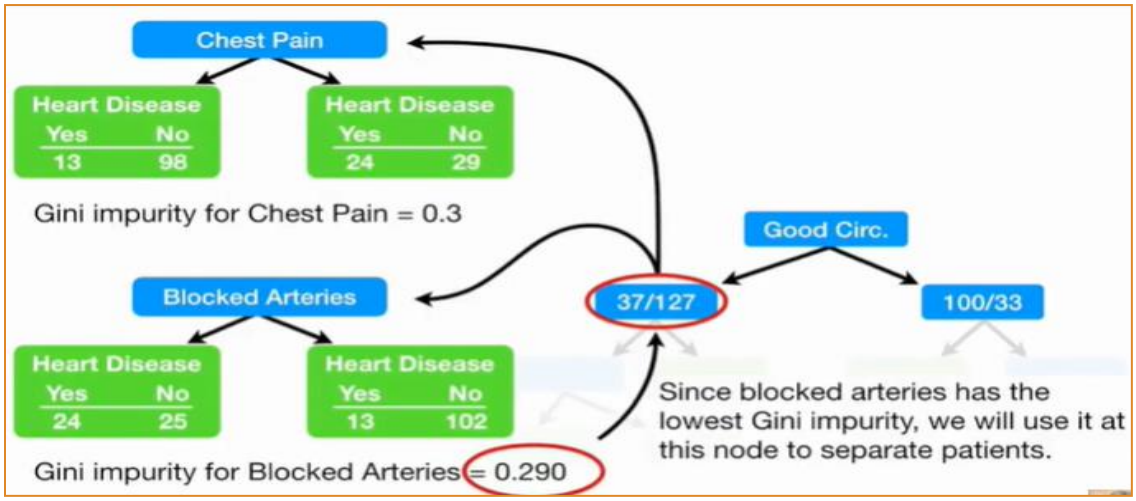


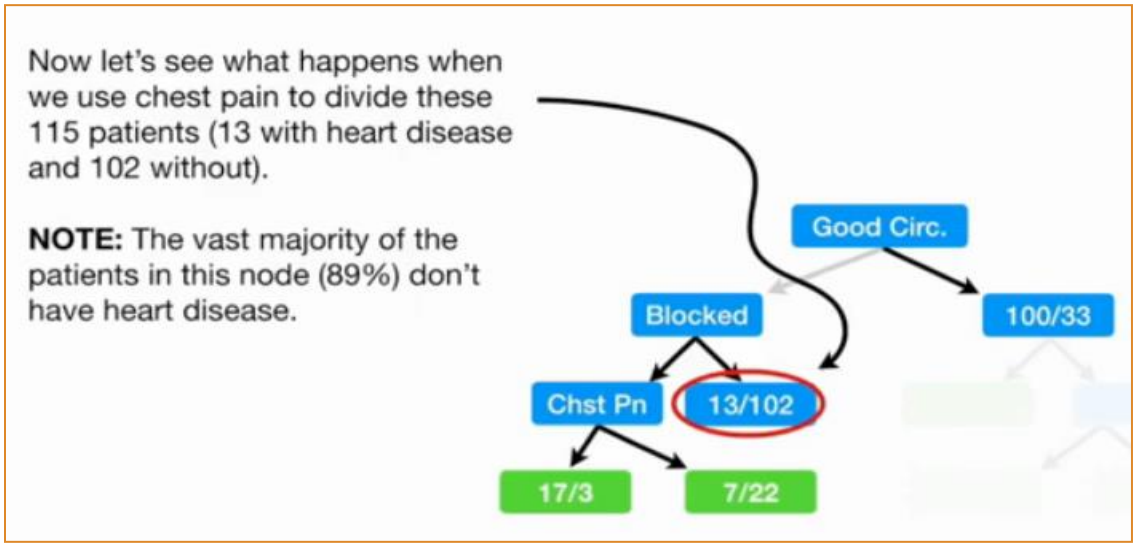


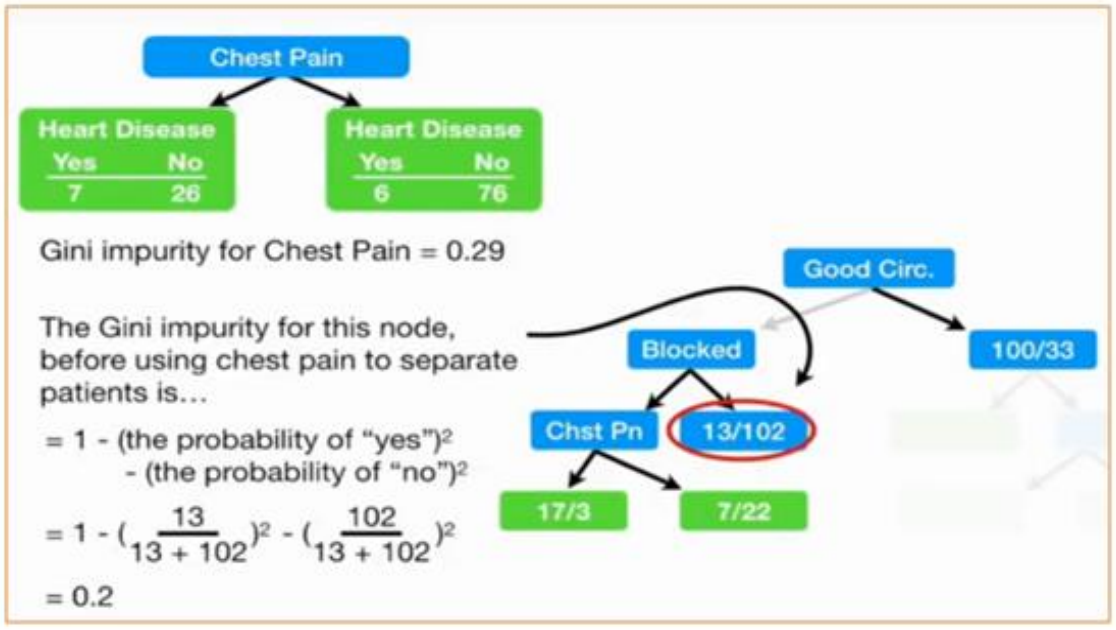
- Calculate the total Gini impurity for using chest pain to separate patients with and without heart disease.
- The leaf nodes do not represent the same no. of patients as the left leaf represents 144 patients and the right leaf represents 159 patients. Thus the total Gini impurity will be the weighted average of the leaf node Gini impurities.
- Gini impurity = (144/144+159)*0.395 + (159/144+159)*0.336 = 0.364
- Similarly the total Gini impurity for 'good blood circulation' and 'blocked arteries' is calculated as
- Gini impurity for 'good blood circulation' = 0.360
  Gini impurity for 'blocked arteries' = 0.381

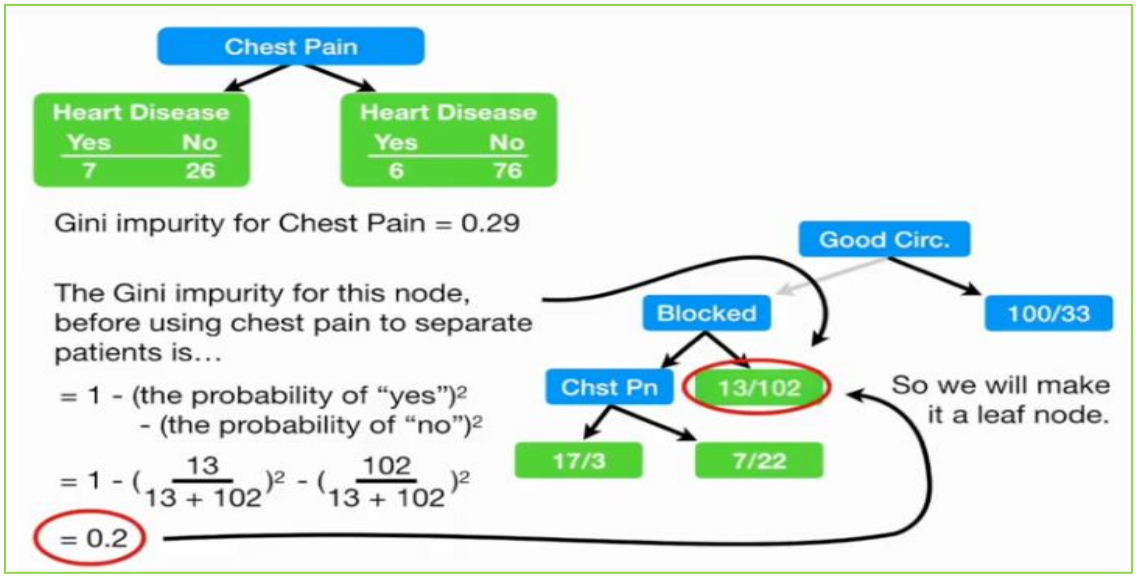
'Good blood circulation' has the lowest impurity score among the tree which symbolizes that it best separates the patients having and not having heart disease, so we will use it at the root node.


Now we need to figure how well **chest pain** and **blocked arteries** separate these 164 patients (37 with heart disease and 127 without heart disease).
Good Circ.
37/127
100/33

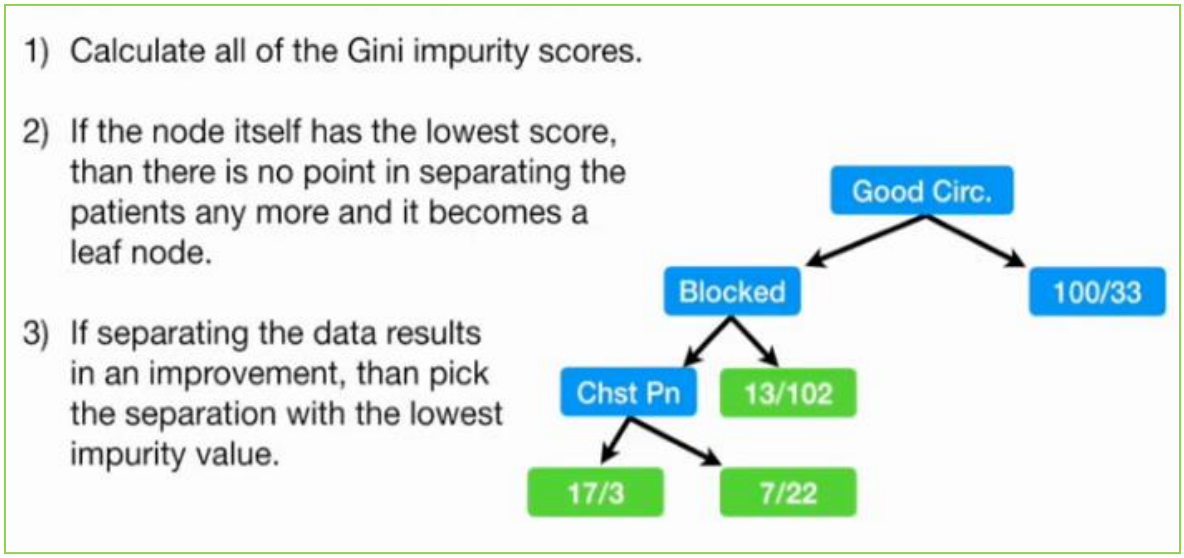

Chest Pain
Heart Disease
Yes No
13 98
Heart Disease
Yes No
24 29
Gini impurity for Chest Pain = 0.3
Good Circ.
Blocked Arteries
37/127
100/33
Heart Disease
Yes No
24 25
Heart Disease
Yes No
13 102
Since blocked arteries has the lowest Gini impurity, we will use it at this node to separate patients.
Gini impurity for Blocked Arteries = 0.290

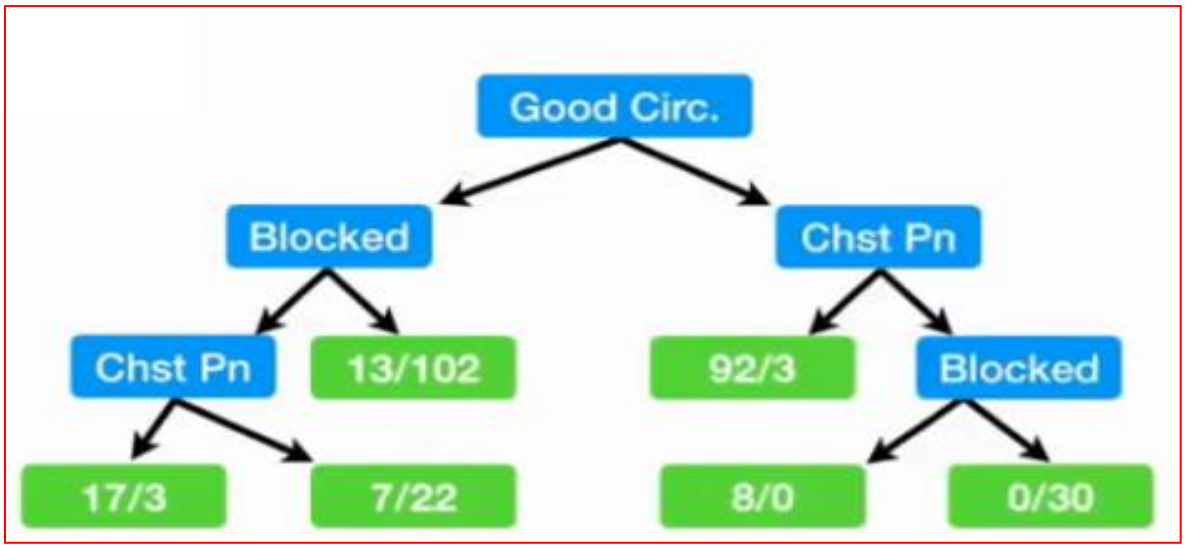

Now let's see what happens when we use chest pain to divide these 115 patients (13 with heart disease and 102 without).
NOTE: The vast majority of the patients in this node (89%) don't have heart disease.
Good Circ.
Blocked
100/33
Chst Pn
13/102
17/3
7/22


Chest Pain
Heart Disease
Yes No
7 26
Heart Disease
Yes No
6 76
Gini impurity for Chest Pain = 0.29
The Gini impurity for this node, before using chest pain to separate patients is...
= 1 - (the probability of "yes")²
- (the probability of "no")²
= 1 - (13/(13 + 102))² - (102/(13 + 102))²
= 0.2
Good Circ.
Blocked
100/33
Chst Pn
13/102
17/3
7/22

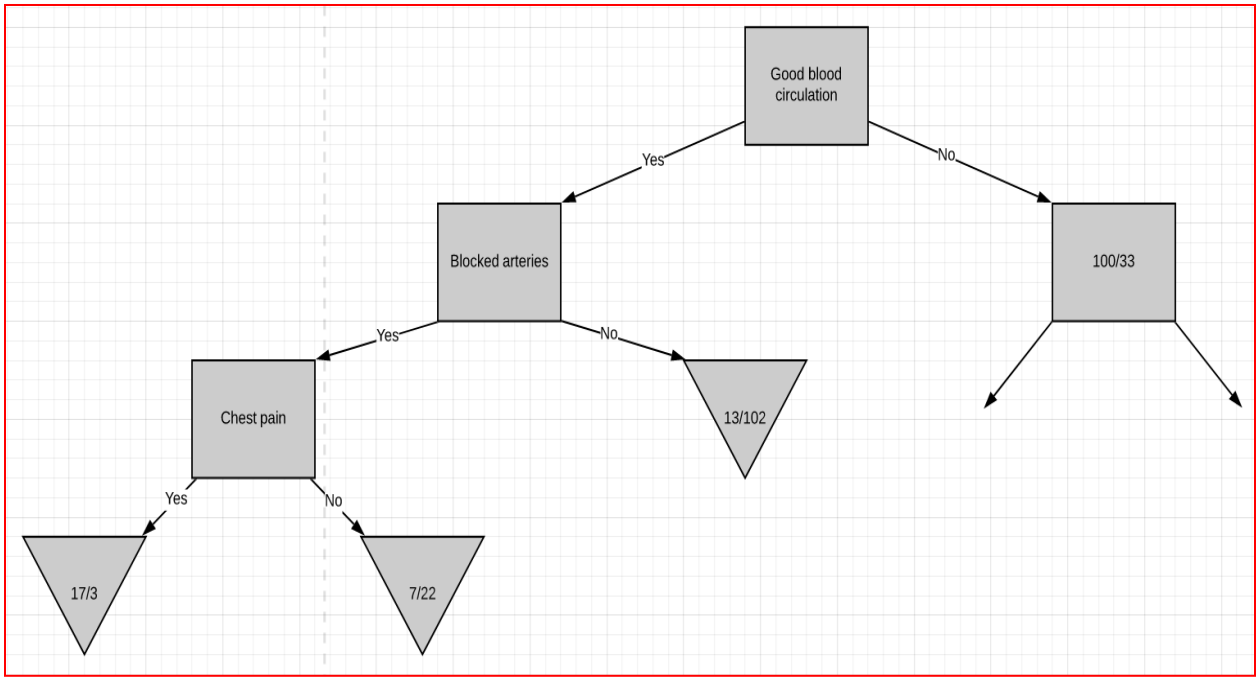

Chest Pain
Heart Disease
Yes No
7 26
Heart Disease
Yes No
6 76
Gini impurity for Chest Pain = 0.29
The Gini impurity for this node, before using chest pain to separate patients is...
= 1 - (the probability of "yes")²
- (the probability of "no")²
= 1 - (13/(13 + 102))² - (102/(13 + 102))²
= 0.2
Good Circ.
Blocked
100/33
Chst Pn
13/102
17/3
7/22
So we will make it a leaf node.

Steps to be repeated on the left side

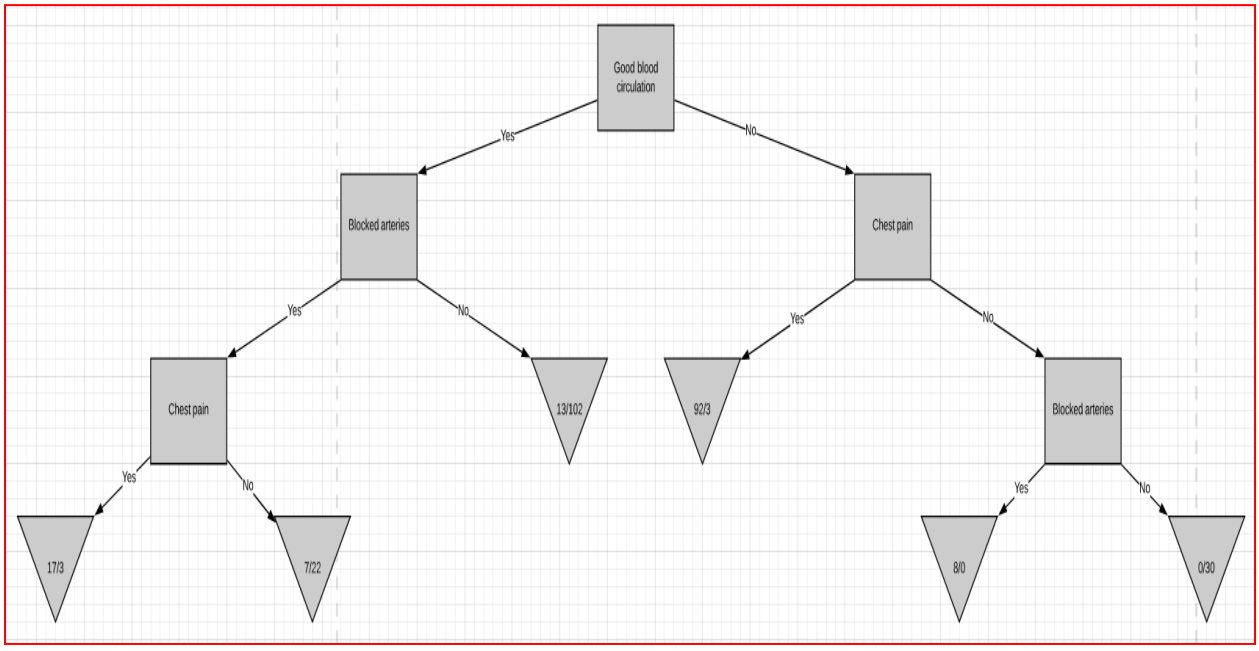




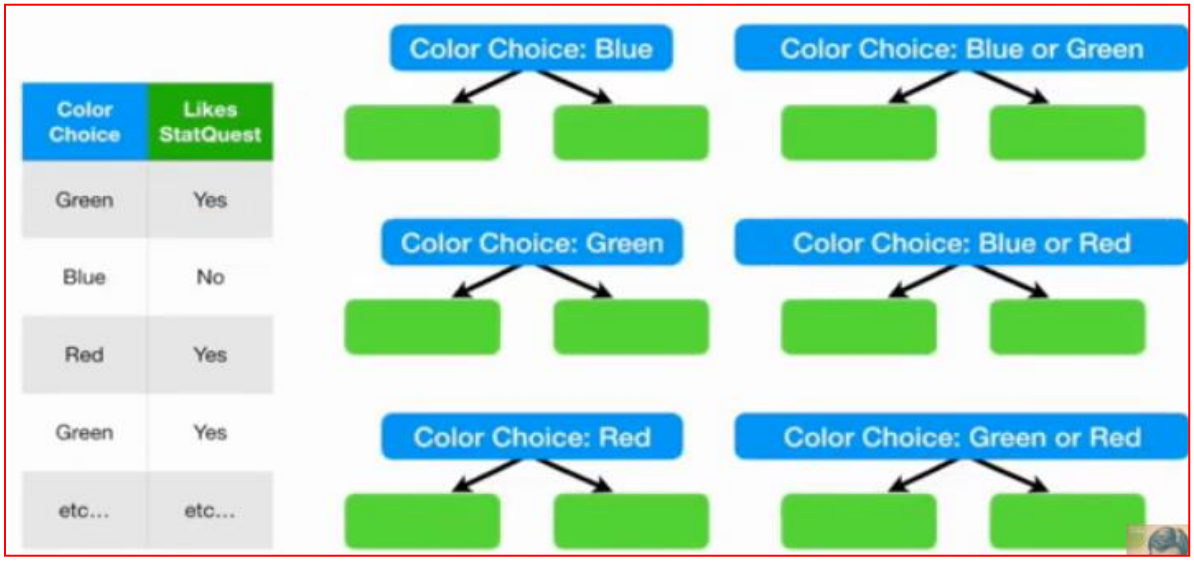
- Calculate the Gini impurity scores.
- If the node itself has the lowest score, then there is no point in separating the patients anymore and it becomes a leaf node.
- If separating the data results in improvement then pick the separation with the lowest impurity value.



Complete Decision tree



Some examples



Classification And Regression Trees (CART) for Machine Learning

Classification Trees	**Regression Tress**
It is used when dependent variables is categorical	It is used when dependent variable is continuous

Use Mode/Class(Mode- happens most often)	Use Mean/ Average
A classification tree is an algorithm where the target variable is fixed or categorical.	A regression tree refers to an algorithm where the target variable is and the algorithm is used to predict it's value.
Classification trees are used when the dataset needs to be split into classes which belong to the response variable. In many cases, the classes Yes or No.	Regression trees, on the other hand, are used when the response variable is continuous. For instance, if the response variable is something like the price of a property or the temperature of the day, a regression tree is used.
classification trees are used for classification-type problems.	Regression trees are used for prediction-type problems
A classification tree splits the dataset based on the homogeneity of data.	In a regression tree, a regression model is fit to the target variable using each of the independent variables.
Measures of impurity like entropy or Gini index are used to quantify the homogeneity of the data when it comes to classification trees.	The error between the predicted values and actual values is squared to get "A Sum of Squared Errors"(SSE). The SSE is compared across the variables and the variable or point which has the lowest SSE is chosen as the split point. This process is continued recursively.

3.2 Ensemble Learning –Boosting-Bagging -Different ways to Combine Classifiers

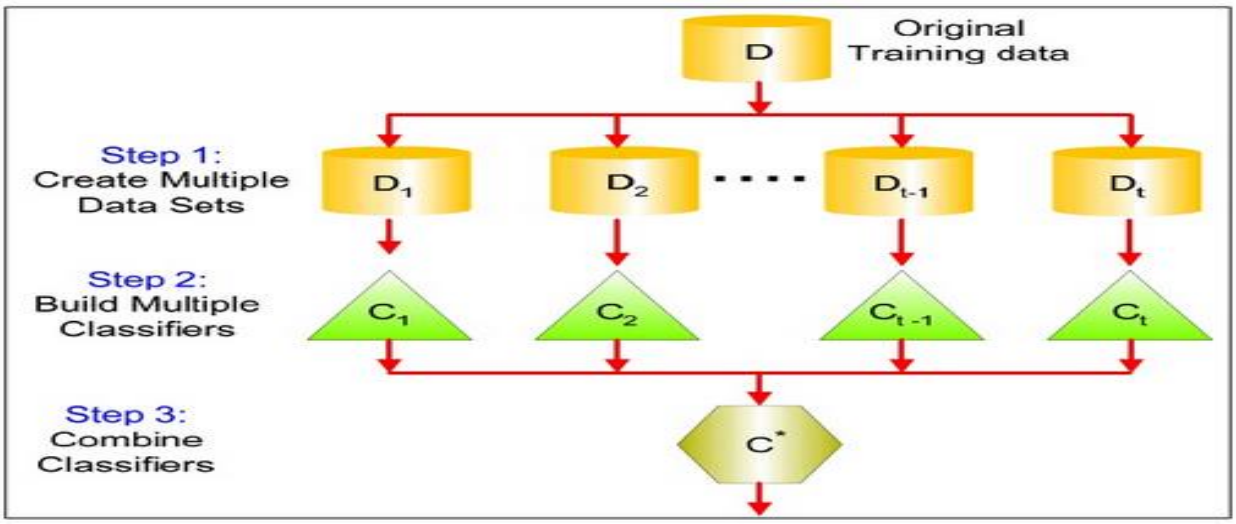
- **First level of Machine Learning**
  - Linear Regression
  - Logistics Regression
  - Support Vector Machine
  - Naive Bayes
  - Decision Tree
- **Ensemble Learning(2nd level)**
  - Weak Learner to Strong Learners
  - Group/ collection of Models/Predictors

- Wisdom of Crowd
- Voting
- Combining different models

What is Classification

“Classification is the process of grouping things according to similar features they share”

- Ensemble Learning is a technique combines individual models together to improve the stability and predictive power of the model.
- This technique permits higher predictive performance
- It combines multiple machine learning models into one predictive model.



- Certain models do well in modelling one aspect of the data, while others do well in modelling another.
- Learn several simple models and combine their output to produce the final decision.
- The combined strength of the models offsets individual model variances and biases.
- This provides a composite prediction where the final accuracy is better than the accuracy of individual models.

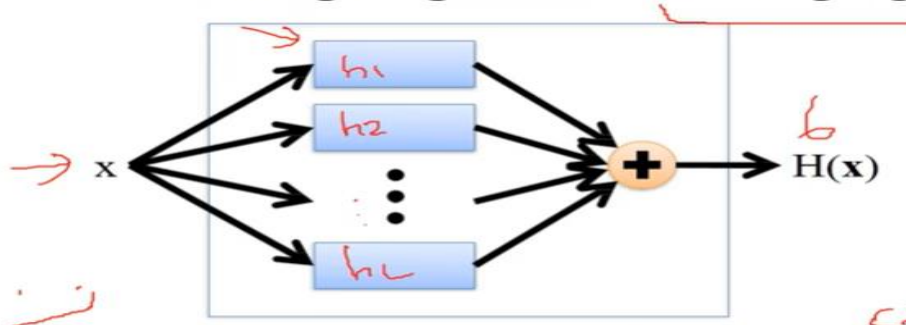
Significance

- This model is the application of multiple models to obtain better performance than from a single model
- Robustness- Ensemble models incorporate the predictions from all the base learners
- Accuracy-Ensemble models deliver accurate predictions and have improved performances
- Consider a set of classifiers h_1 , …., h_L

Construct a classifier H(x) that combines the individual decisions of h_1 , …., h_L

- Successful ensembles requires diversity
    - Classifiers should make different mistakes
    - Can have different types of base learners

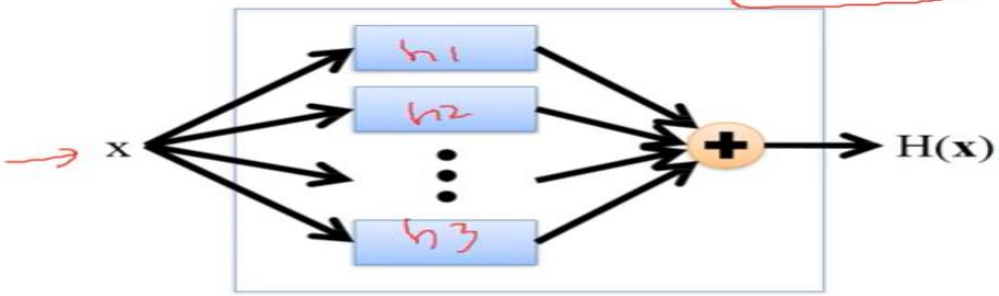
Combining Regressors: Averaging



h1 LR 5 | h2 PR 8 | h3 SVR 6 | h4 DT 7

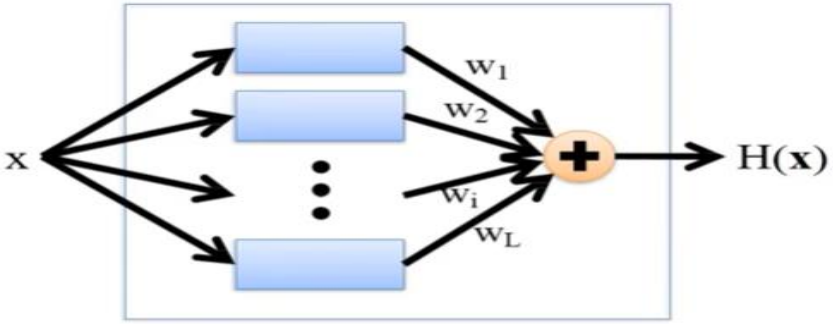
$\frac{5+6+6+7}{4} = 6$

Combining Classifiers: Voting

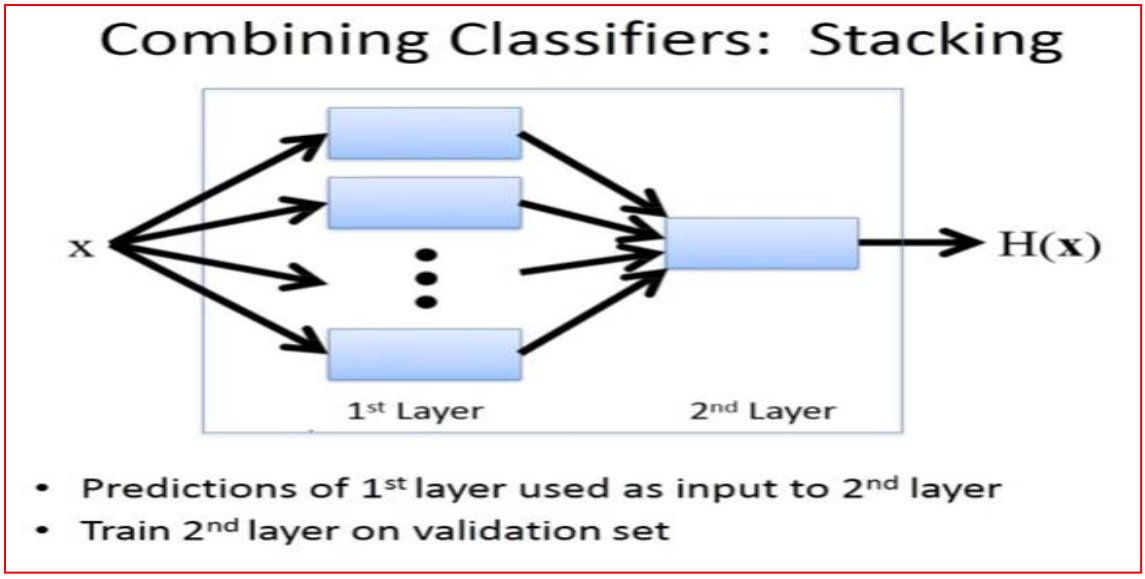


- Final hypothesis is a simple vote of the members

Combining Classifiers: Weighted Average

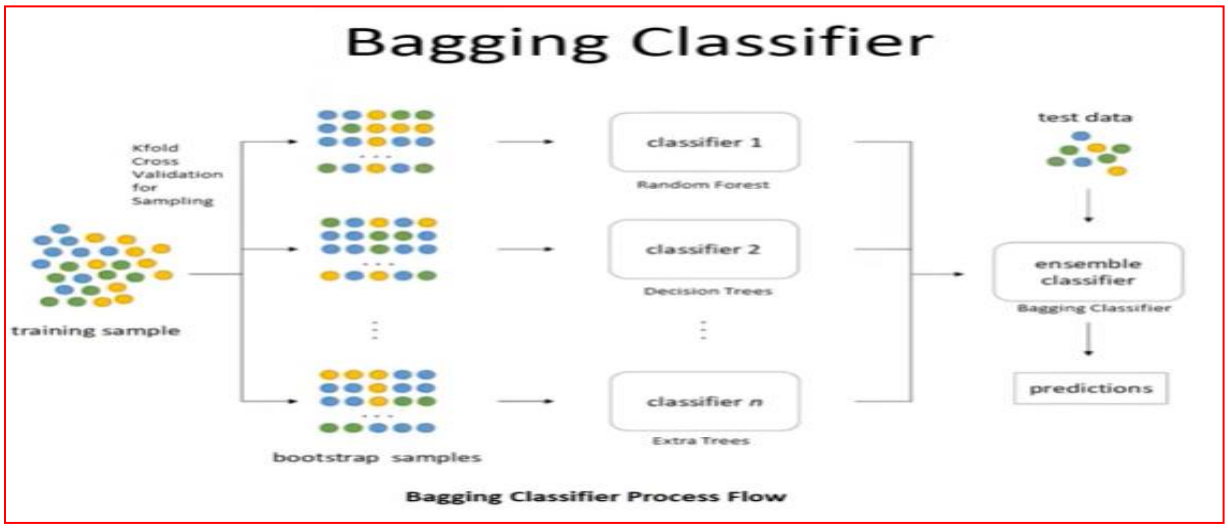


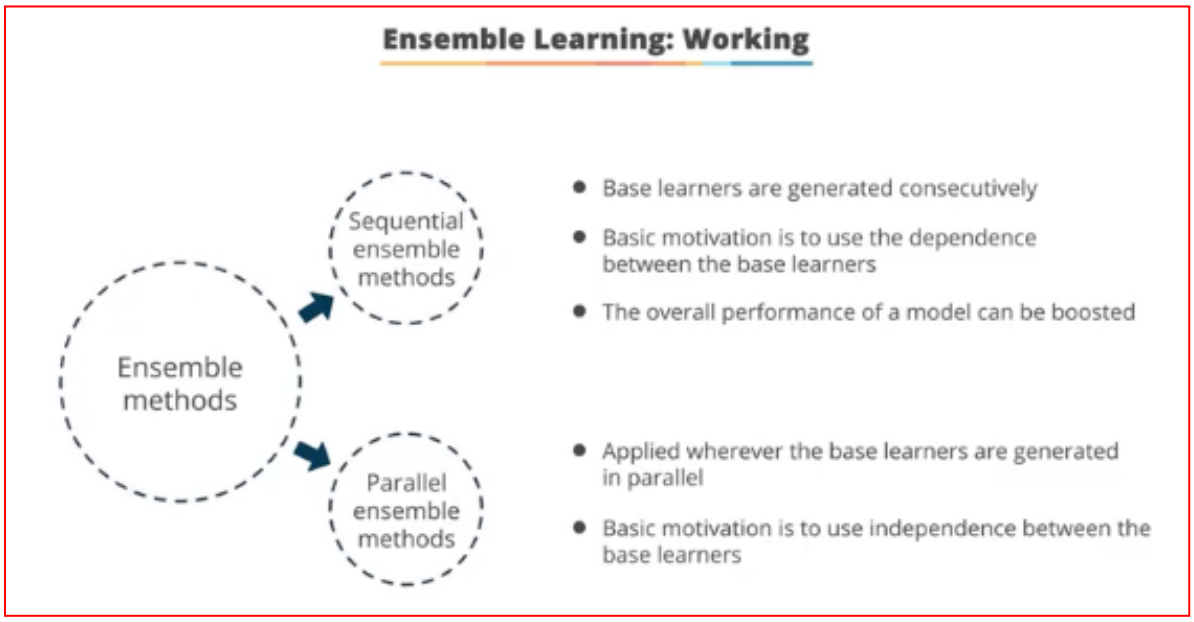
- Coefficients of individual members are trained using a validation set



Manipulating the training data

- **Bootstrap replication:**
  - Given n training examples, construct a new training set by sampling n instances with replacement
  - Excludes 30% of the training instances
- **Bagging:**
  - Create bootstrap replicates of training set
  - Train a classifier(e.g., a decision tree) for each replicate
  - Estimate classifier performance using out-of-bootstrap data
  - Average output of all classifiers

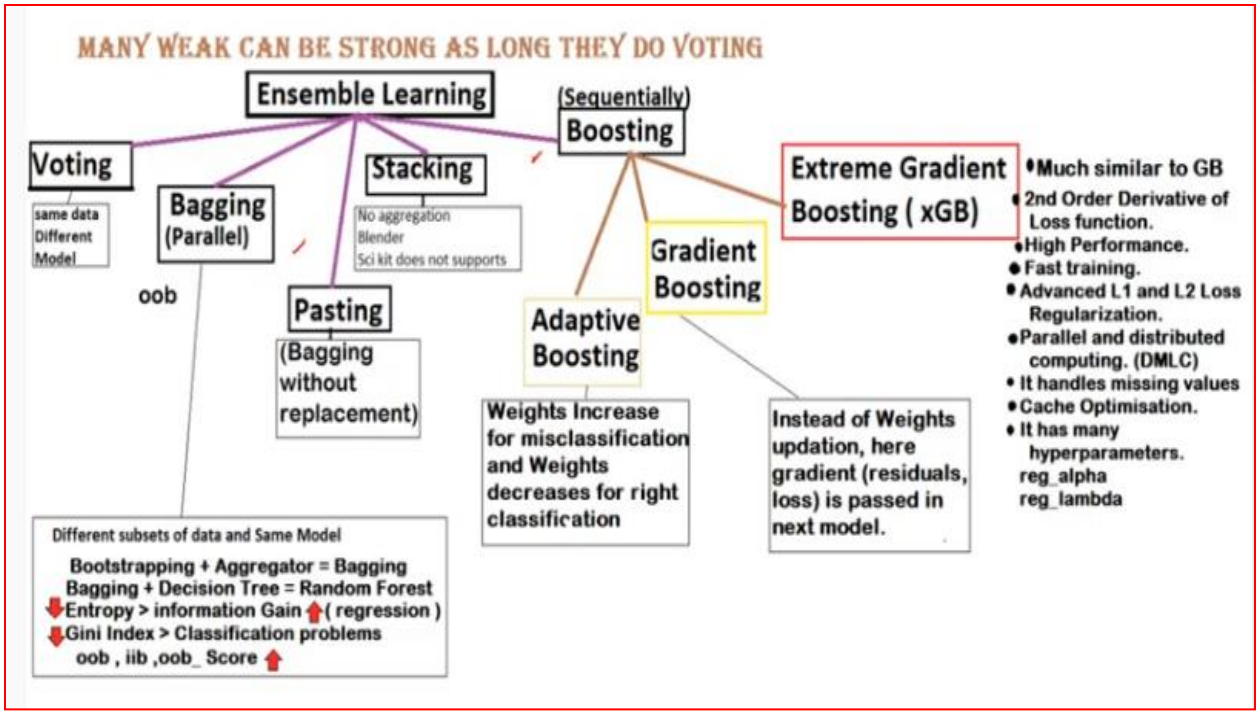




Labelled images of Hot DOGS and other objects

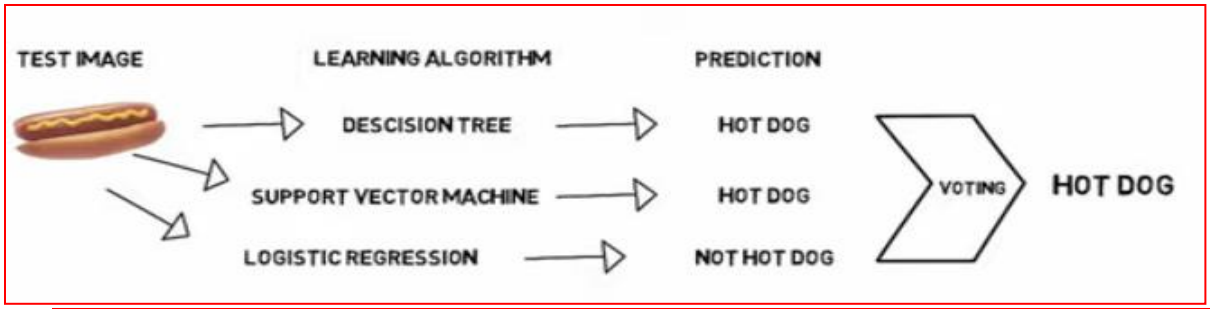
- Images of hot dogs labelled 1
- Images of other objects labelled 0

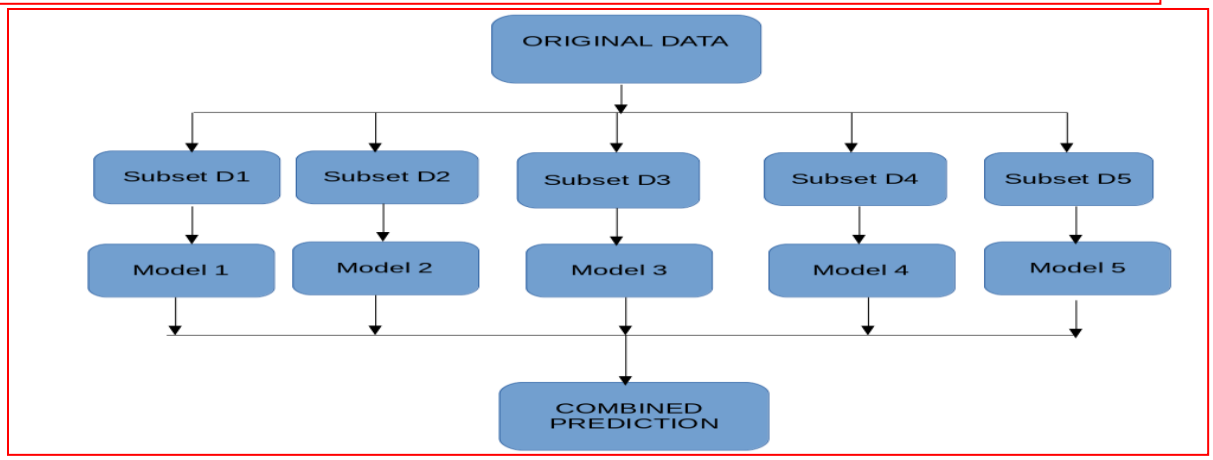
- **Why use Ensemble models?**
  - Better Accuracy(Low Error)
  - Higher Consistency(Avoids Over fitting)
  - Reduces bias and Variance errors
- **When and Where to use Ensemble models**
  - Single model overfits
  - Results worth the extra trining.
  - Can be used for Classification as well as Regression
- **Ensemble Learning Methods**
  - Combine all weak learners to form an ensemble (or) Create an ensemble of well chosen strong and diverse models
  - This models gain more accuracy and robustness by combining data from numerous modelling approaches



Bagging

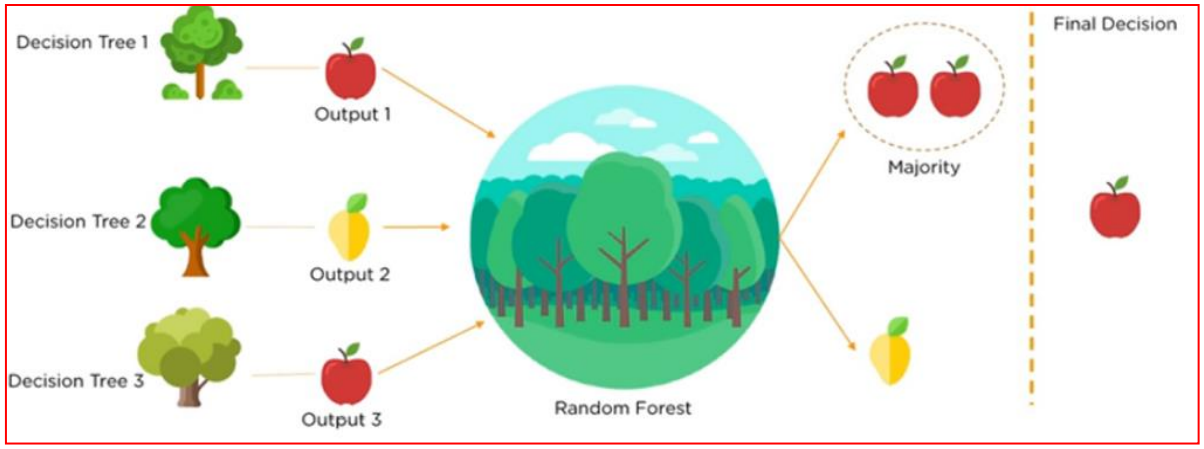
- The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result.
- If create all the models on the same set of data and combine it, Whether these models will give the same result ,since they are getting the same input. One of the techniques is **bootstrapping.**
- Bootstrapping is a sampling technique in which create subsets of observations from the original dataset, **with replacement**. The size of the subsets is the same as the size of the original set.
- Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.
- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.





What is Random forest?

Random forest or Random decision forest is a method that operates by constructing multiple decision trees during training phase. The decision of the majority of the tree is chosen by the random forest as the final decision.



Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

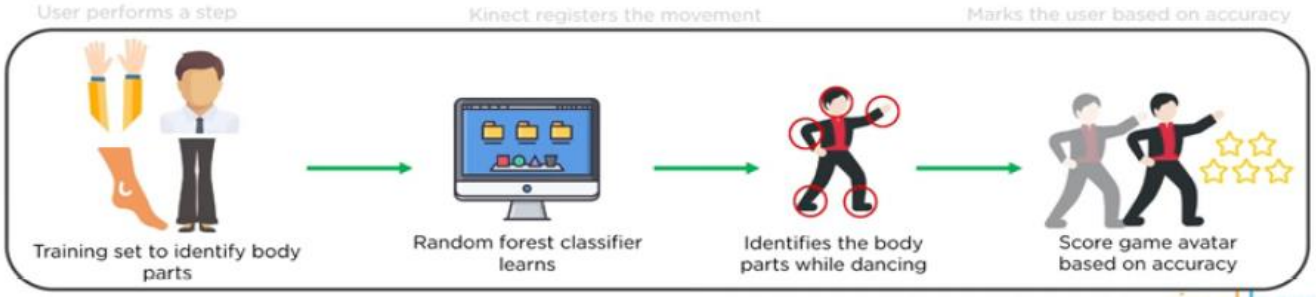
Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations**.**

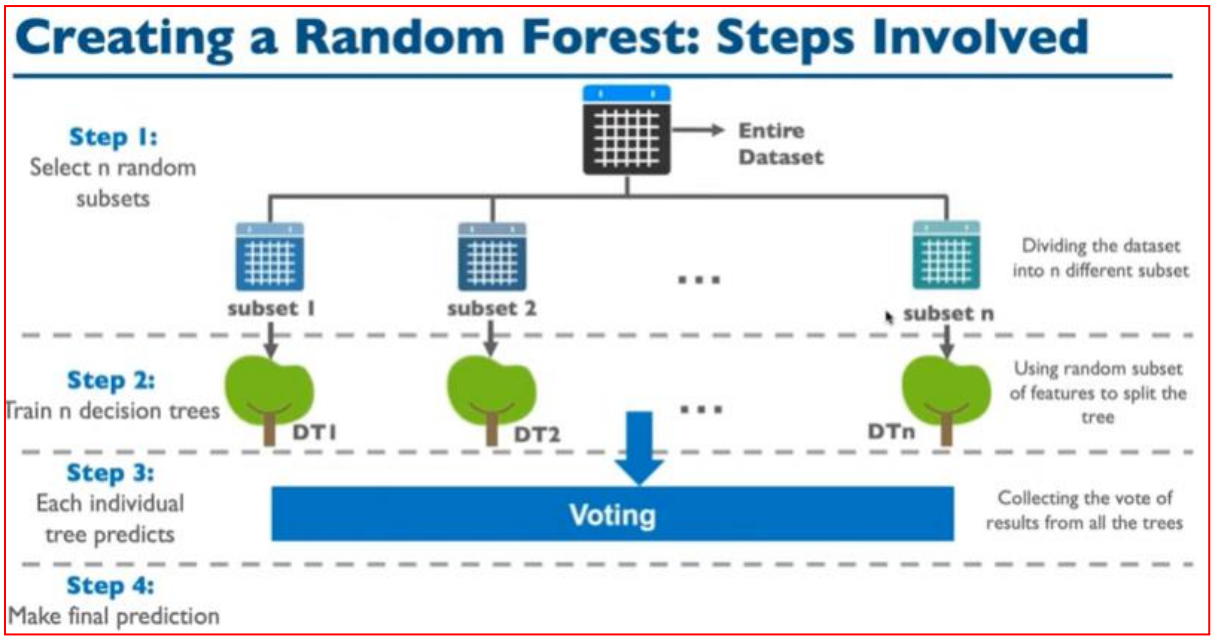
Why Random Forest



- Random forest is the most used supervised machine learning algorithm for classification and regression
- RF uses ensemble learning method in which the predictions are based on the combined results of various individual models
- No Overfitting
    - Use of multiple trees reduce the risk of overfitting

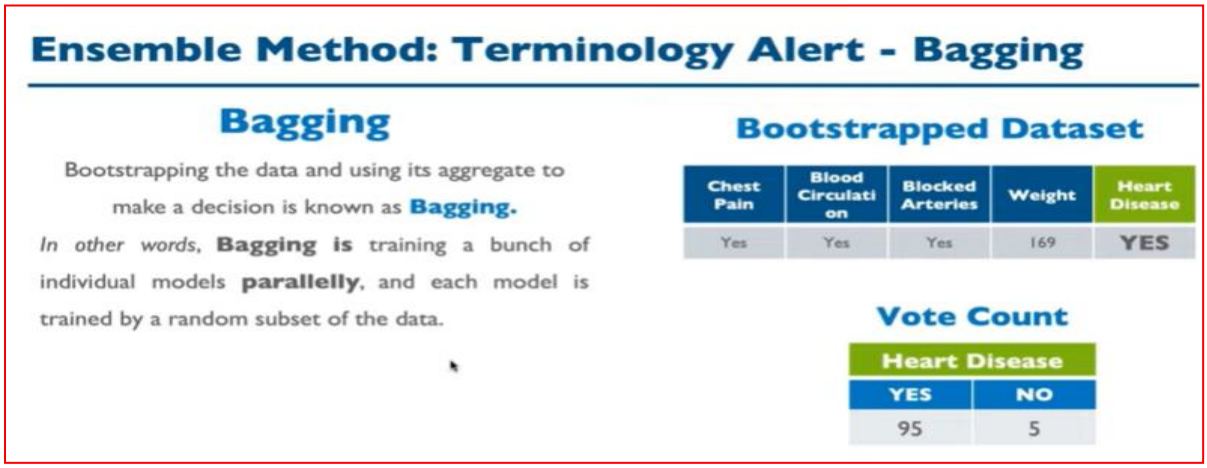
- Training time is less

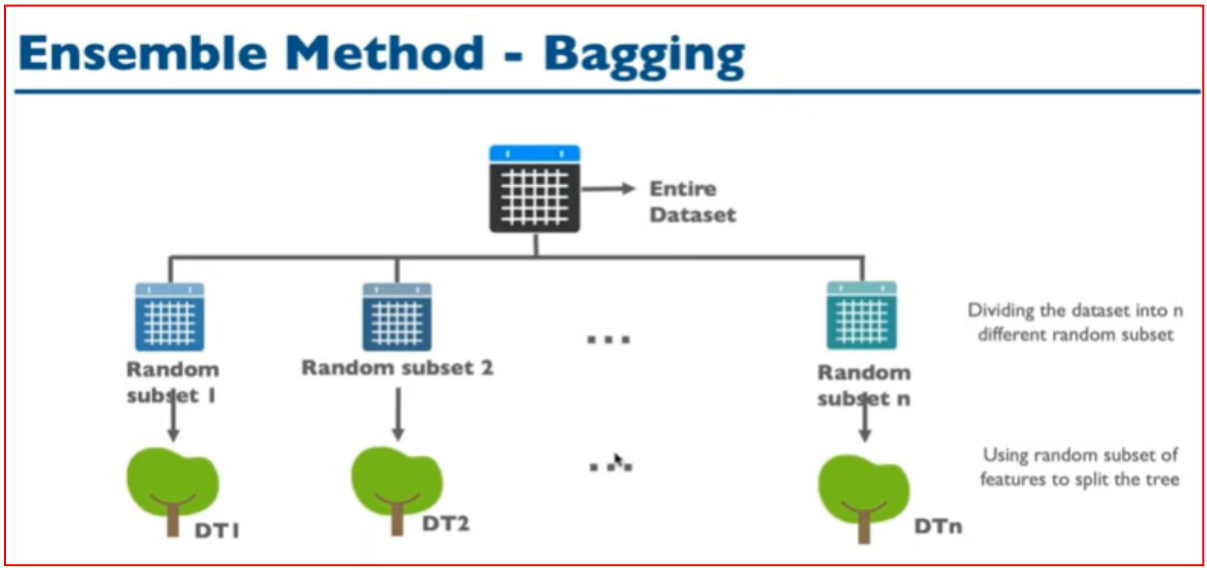
- High Accuracy
  - Runs efficiently on large database
  - For large data. It produces highly accurate predictions
- Estimates missing data
  - Random forest can maintain accuracy when a large proportion of data is missing

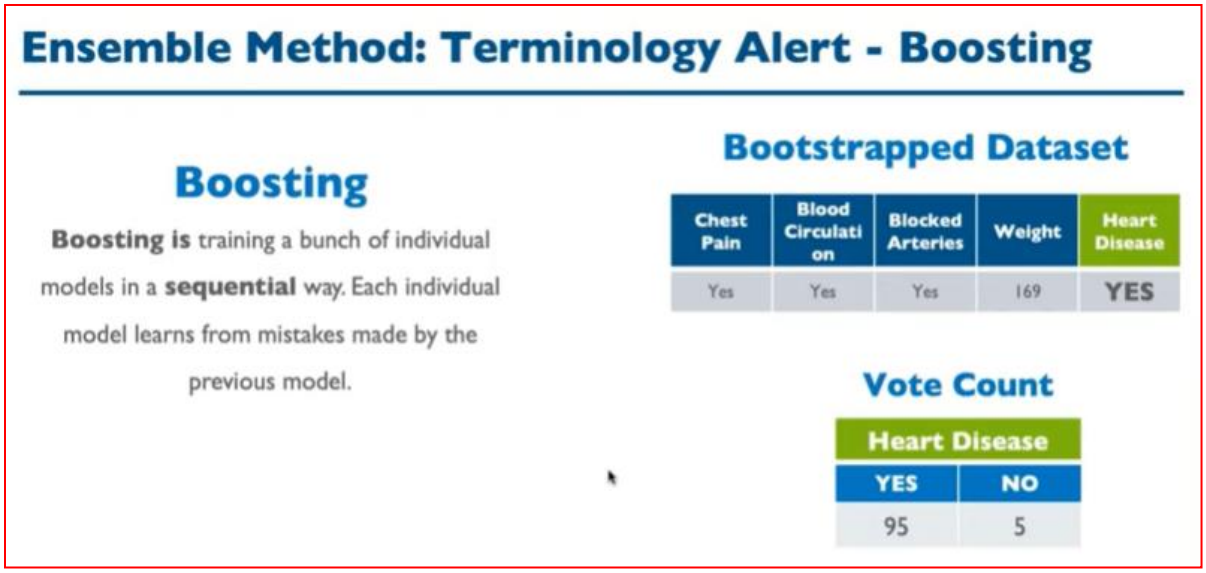


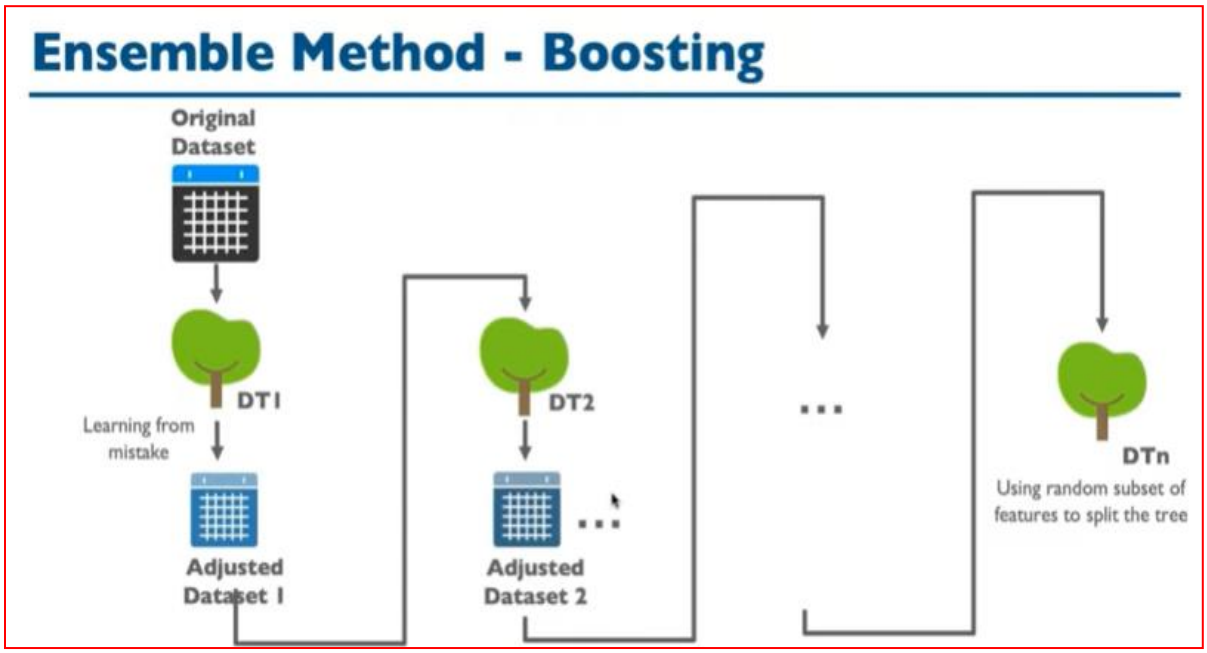
How does Random Forest algorithm work?

- Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.
- The Working process can be explained in the below steps and diagram:
- **Step-1:** Select random K data points from the training set.
- **Step-2:** Build the decision trees associated with the selected data points (Subsets).
- **Step-3:** Choose the number N for decision trees that you want to build.
- **Step-4:** Repeat Step 1 & 2.
- **Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.









How Boosting Algorithm Works?

- The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strong rule.
- These weak rules are generated by applying base Machine Learning algorithms on different distributions of the data set. These algorithms generate weak rules for each iteration.
- After multiple iterations, the weak learners are combined to form a strong learner that will predict a more accurate outcome.

The algorithm :

- **Step 1:** The base algorithm reads the data and assigns equal weight to each sample observation.
- **Step 2:** False predictions made by the base learner are identified. In the next iteration, these false predictions are assigned to the next base learner with a higher weightage on these incorrect predictions.
- **Step 3:** Repeat step 2 until the algorithm can correctly classify the output.

Therefore, the main aim of Boosting is to focus more on miss-classified predictions

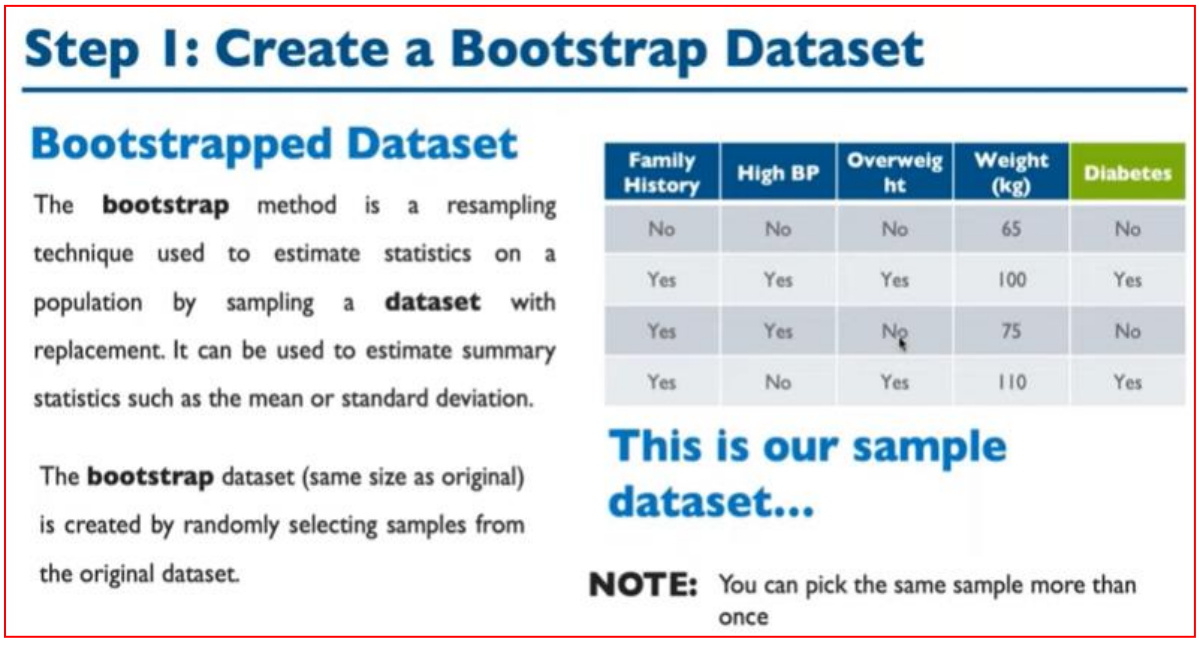
Algorithms based on Bagging and Boosting

- Bagging and Boosting are two of the most commonly used techniques in machine learning. Following are the algorithms will be focusing on:
- Bagging algorithms:
    - Bagging meta-estimator
    - **Random forest**
- Boosting algorithms:
    - **AdaBoost(Adaptive Boosting)**
    - GBM(Gradient Boosting)
    - XGBM
    - Light GBM
    - CatBoost

Similarities Between Bagging and Boosting –

- Both are ensemble methods to get N learners from 1 learner.
- Both generate several training data sets by random sampling.
- Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
- Both are good at reducing variance and provide higher stability.

S.No	Boosting	Bagging
1.	Simplest way of combining predictions that belong to the same type	A way of combining predictions that belong to the different types
2.	Aim to decrease variance, not bias	Aim to decrease bias, not variance
3.	Each model receives equal weight	Models are weighted according to their performance
4.	Each model is built independently	New model are influenced by performance of previously built models
5.	Different training data subsets are randomly drawn with replacement from the entire training dataset.	Every new subsets contains the elements that were misclassified by previous models.
6.	Bagging tries to solve overfitting problem	Boosting tries to reduce bias
7.	If the classifier is unstable (high variance) then apply bagging	If the classifier is stable and simple(high bias) the apply boosting
8.	Random forest	Gradient boosting



Step 1: Create a Bootstrap Dataset

Original Dataset

Family History	High BP	Overweight	Weight (kg)	Diabetes
No	No	No	65	No
Yes	Yes	Yes	100	Yes
Yes	Yes	No	75	No
Yes	No	Yes	110	Yes

Bootstrapped Dataset

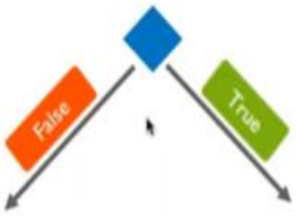
Family History	High BP	Overweight	Weight (kg)	Diabetes
Yes	Yes	Yes	100	Yes
No	No	No	65	No
Yes	Yes	No	75	No
Yes	Yes	No	75	No

Creating a bootstrapped dataset with randomly selected sample from the original dataset

Step 2: Creating Decision Tree: Bootstrapped Dataset

Bootstrapped Dataset

Family History	High BP	Overweight	Weight (kg)	Diabetes
Yes	Yes	Yes	100	Yes
No	No	No	65	No
Yes	Yes	No	75	No
Yes	Yes	No	75	No



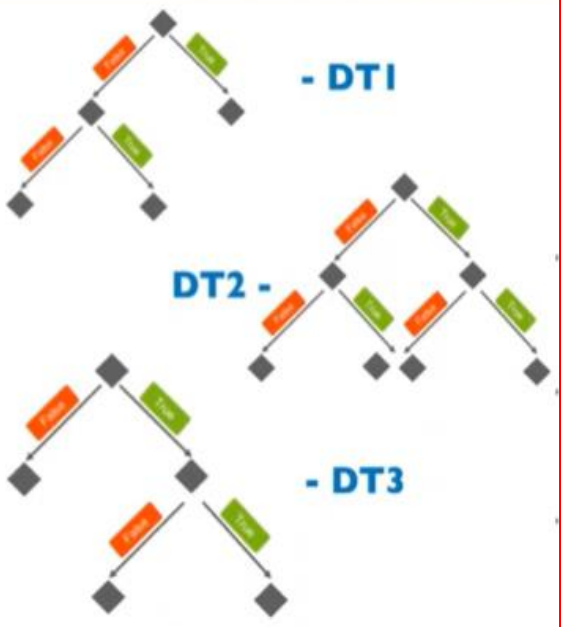
Create a decision tree (eg: Root node - *Overweight*) using the bootstrapped dataset. Use only a random subset of variables/column at each step

Instead of considering all the 4 variables to figure out how to split the root node. **In this example**, consider only 2 *variables* at each step

Step 2: Creating Decision Tree: Bootstrapped Dataset

Original Dataset

Family History	High BP	Overweight	Weight (kg)	Diabetes
No	No	No	65	No
Yes	Yes	Yes	100	Yes
Yes	Yes	No	75	No
Yes	No	Yes	110	Yes



Recursively splitting sample at other nodes

Get back to Step 1 and repeat: Build new bootstrapped dataset and rebuild decision trees considering subset of variables at each step (ideally you have to repeat this step 100's of time)

and

Step 3 & 4: Counting the Votes for Predicting

Bootstrapped Dataset

Family History	High BP	Overweight	Weight (kg)	Diabetes
Yes	Yes	No	75	?

Predict if the patient has diabetes or not

Vote Count

Diabetes	
YES	NO
95	5

In this case, '**YES**' received most of the votes,
→ **patient has heart disease**

Let's say we created 100 decision tree and this is the overall vote count of the predictions from all the decision tree. Next thing is to find out the option which received most votes

How Good is Your Model: Out-of-bag Dataset

Predicted Diabetes: **YES**

Predicted Diabetes: **NO**

Predicted Diabetes: **NO**

Out-of-Bag Dataset

Family History	High BP	Overweight	Weight (kg)	Diabetes
Yes	No	Yes	110	Yes

Run this Out-of-Bag sample data through all the trees that were built without it

How Good is Your Model: Out-of-bag Dataset

Original Dataset

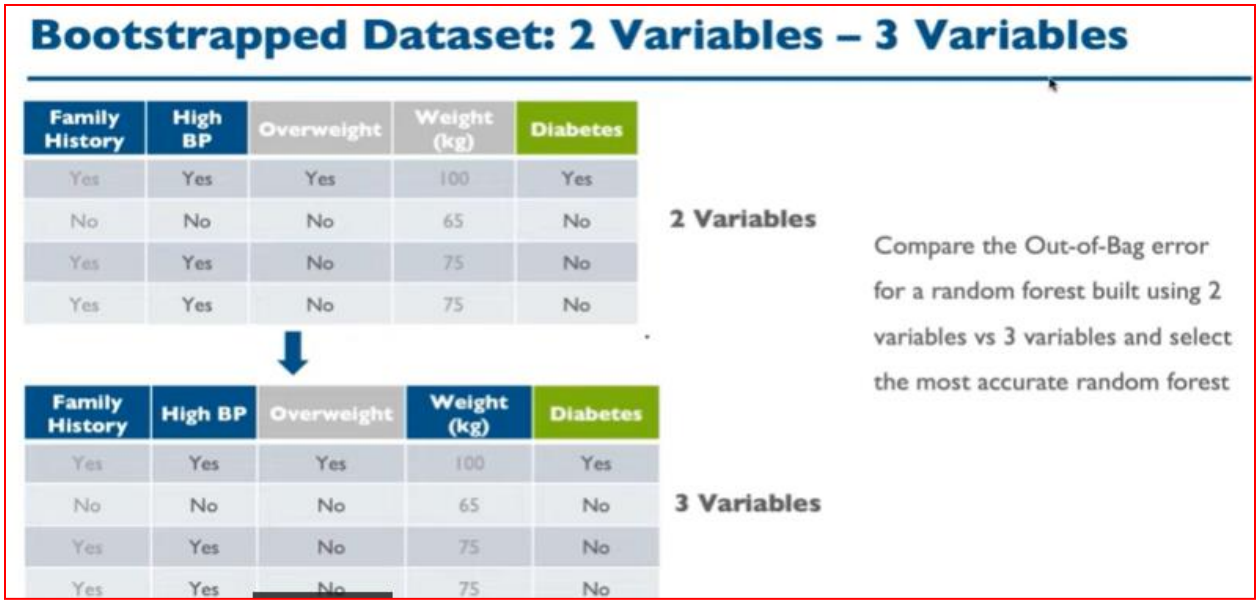
Family History	High BP	Overweight	Weight (kg)	Diabetes
No	No	No	65	No
Yes	Yes	Yes	100	Yes
Yes	Yes	No	75	No
Yes	No	Yes	110	Yes

Out-of-Bag Dataset

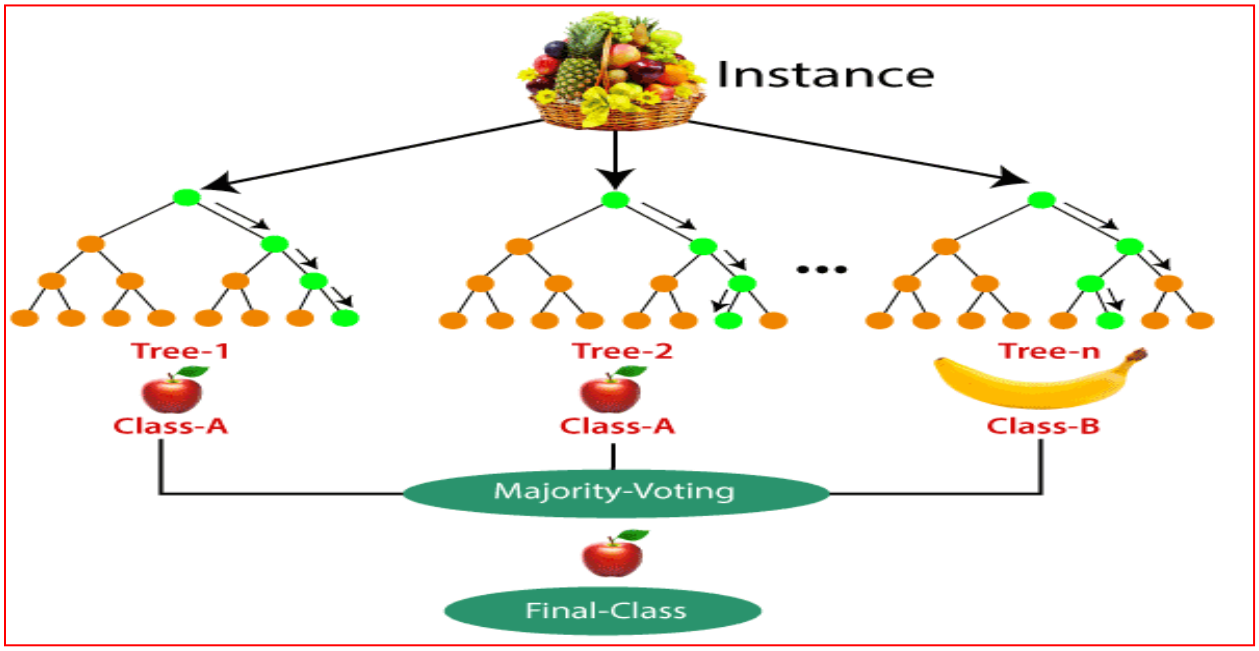
Family History	High BP	Overweight	Weight (kg)	Diabetes
Yes	No	Yes	110	Yes

This is the entry that did not end up in the bootstrapped dataset and the collection of such entries as a dataset is known as **" Out of Bag Dataset"**

NOTE: Just in case, the original dataset were larger, we would have more than 1 entry over here



Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



How does a Decision Tree work?

PROBLEM STATEMENT

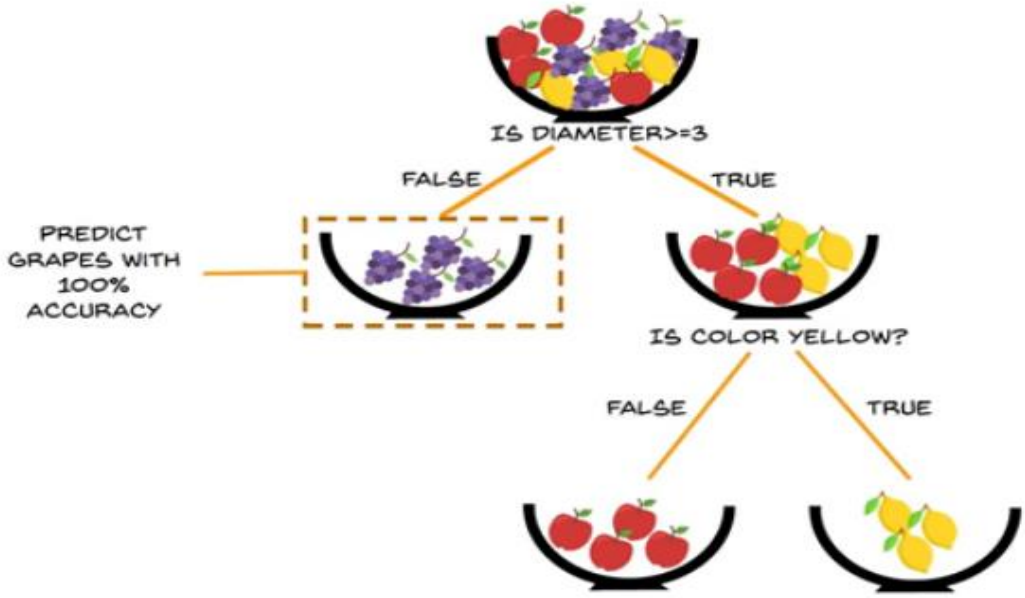
TO CLASSIFY THE DIFFERENT TYPES OF FRUITS IN THE BOWL BASED ON DIFFERENT FEATURES

THE DATASET(BOWL) IS LOOKING QUITE MESSY AND THE ENTROPY IS HIGH IN THIS CASE

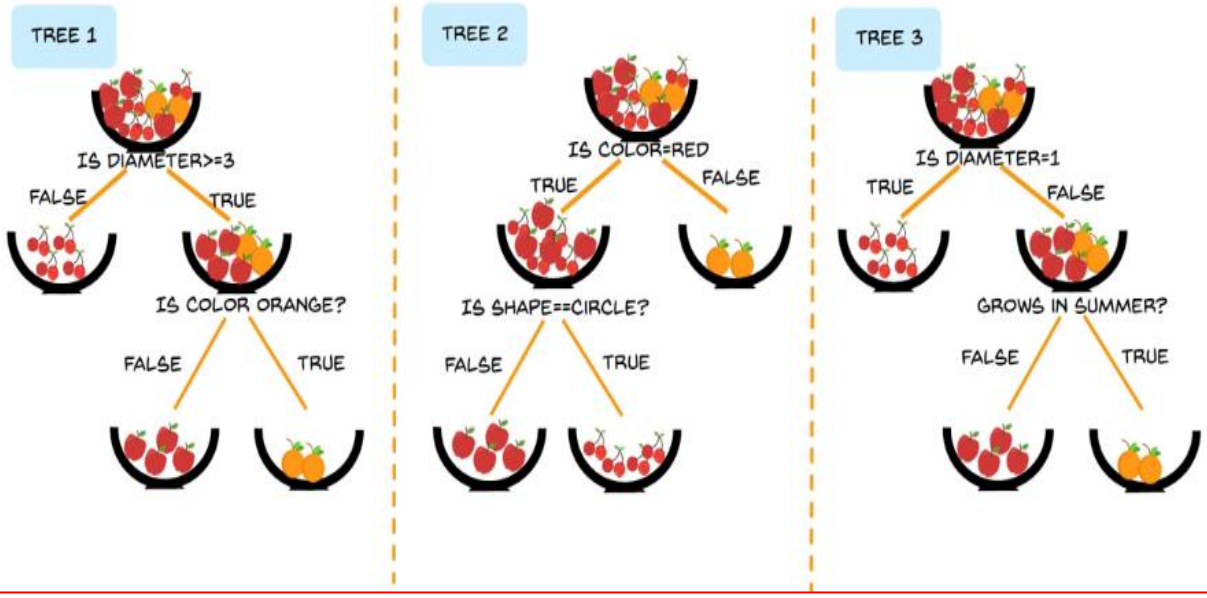
TRAINING DATASET

COLOR	DIAMETER	LABEL
RED	3	APPLE
YELLOW	3	LEMON
PURPLE	1	GRAPES
RED	3	APPLE
YELLOW	3	LEMON
PURPLE	1	GRAPES

How does a Decision Tree work?



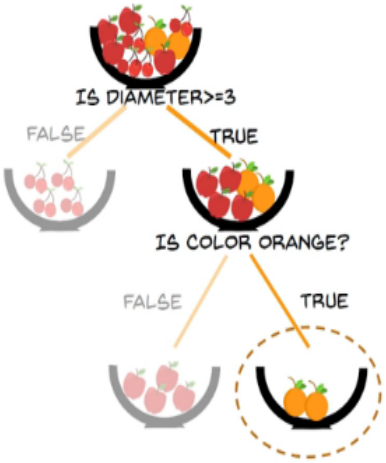
How does a Random Forest work?

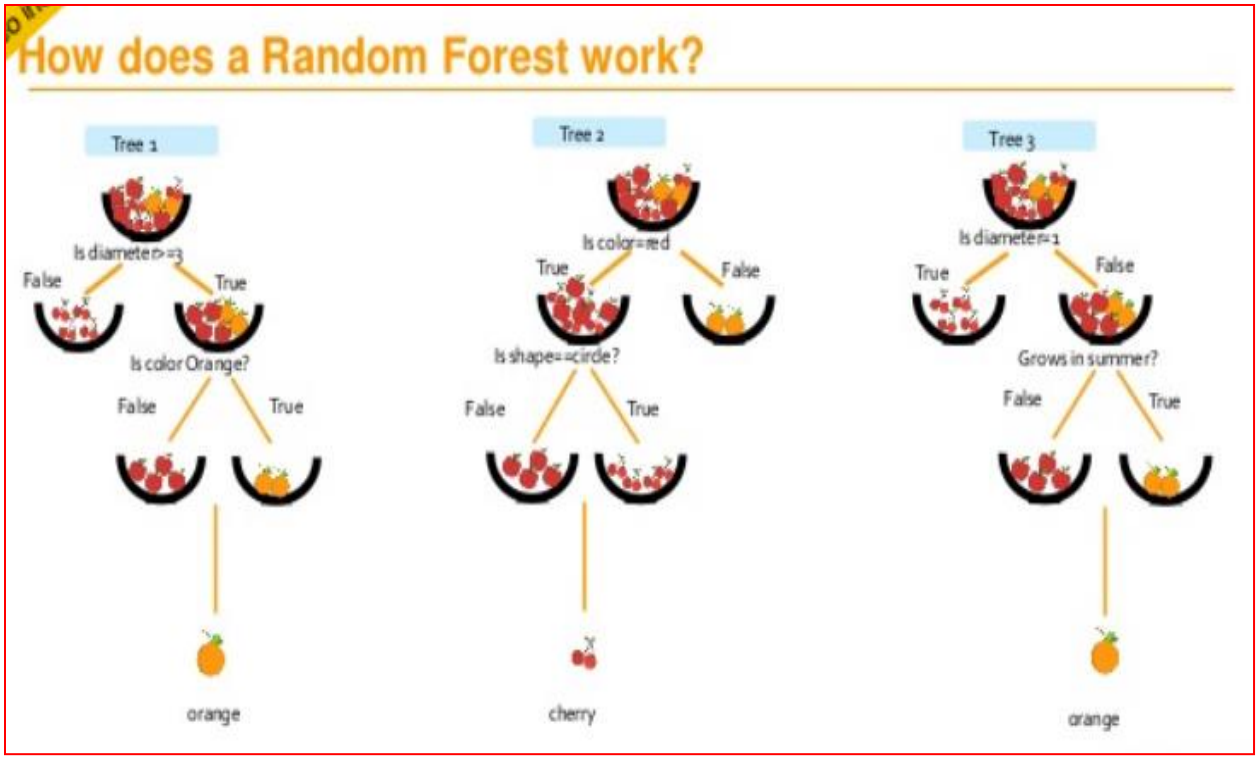


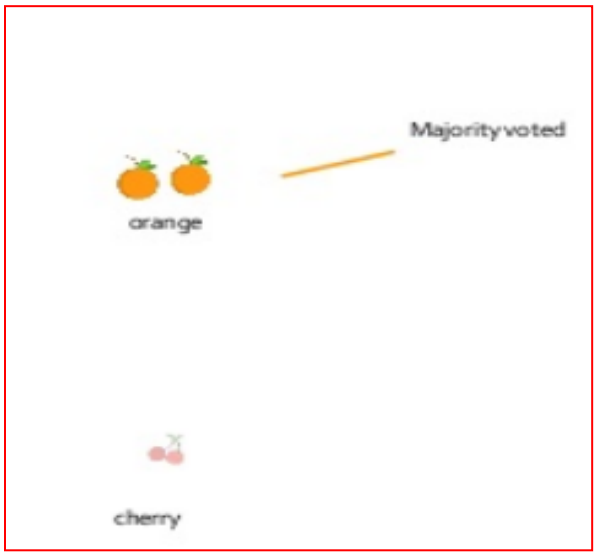
How does a Random Forest work?

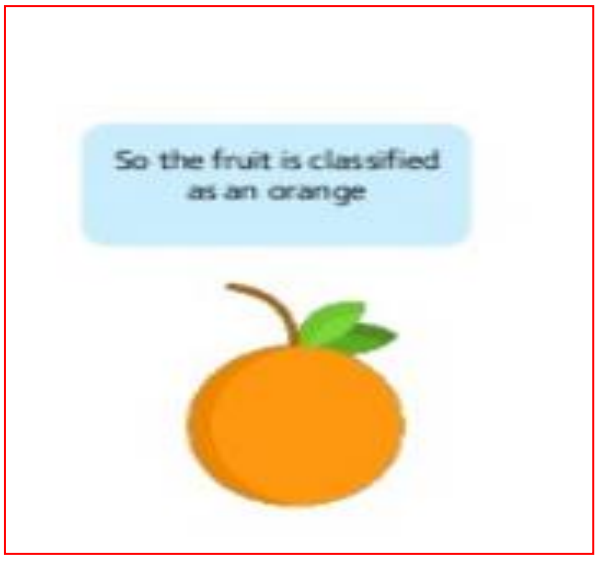
TREE 1 CLASSIFIES IT AS AN ORANGE

DIAMETER = 3
COLOUR = ORANGE
GROWS IN SUMMER = YES
SHAPE = CIRCLE









Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

- **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
- **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
- **Land Use:** We can identify the areas of similar land use by this algorithm.

- **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

3.3 Probability & Statistics

- Machine Learning is an interdisciplinary field that uses statistics, probability, algorithms to learn from data and provide insights which can be used to build intelligent applications.

Key concepts widely used in machine learning.

- Probability and statistics are related areas of mathematics which concern themselves with analyzing the relative frequency of events.
- *Probability* deals with predicting the likelihood of future events, while *statistics* involves the analysis of the frequency of past events.

Probability

- Most people have an intuitive understanding of degrees of probability, which is why we use words like “probably” and “unlikely” in our daily conversation, but we will talk about how to make quantitative claims about those degrees .
- In probability theory, an **event** is a set of outcomes of an experiment to which a probability is assigned. If **E** represents an event, then **P(E)** represents the probability that **E** will occur. A situation where **E** might happen (*success*) or might not happen (*failure*) is called a ***trial***.
- This event can be anything like *tossing a coin, rolling a die* or *pulling a colored ball out of a bag.* In these examples the outcome of the event is random, so the variable that represents the outcome of these events is called a **random variable.**
- Let us consider a basic example of tossing a coin. If the coin is fair, then it is just as likely to come up heads as it is to come up tails. In other words, if we were to repeatedly toss the coin many times, we would expect about about half of the tosses to be heads and and half to be tails. In this case, we say that the probability of getting a head is 1/2 or 0.5 .