Machine Learning

Introduction to Machine learning (for understanding purpose)

In simple words, we can say that machine learning is the competency of the software to perform a single or series of tasks intelligently without being programmed for those activities.
This is part of **Artificial Intelligence**. Normally, the software behaves the way the programmer programmed it; while machine learning is going one step further by making the software capable of accomplishing intended tasks by using statistical analysis and predictive analytics techniques.

What is the Need of Machine Learning?

Generally, the software works the way it is programmed, there are hundreds and thousands of complex codes written to execute the desired works. The programmer writes the codes and the software performs the planned task wonderfully provided there is no defect inside. So, when everything is going well by programming the software, then what is the need of machine learning? Why is it necessary to make the software recognize the vast data and determine the next course of actionable task?

Let's discuss one simple example which we normally perform every day as part our online activities, that is searching something on the Google or any other Search Engines. So, whenever we search something, the search engine displays the links of related web pages within seconds. So, can that be possible just by programming the search results to display the related web links based on the words or sentences we typed in on the search bar?

There are millions of people searching the web at a time, and there are many new websites launching every minute. So, can Google or any other search engine providers able to decide which related search contents will be shown on the search results page just by doing the programming at runtime? That is completely impossible. So, how it works? Let's find out how it is possible using the machine learning techniques.

How Machine Learning Works?

Machine Learning is a technique which works intelligently by using some complex algorithms and set of predefined rules. It uses the past data to read the patterns and then based on the analysis it generates the relevant data or performs the intended task abiding the defined rules and algorithms.

As we discussed earlier in the search example, whenever we typed in something on the search bar, the search engines uses this machine learning technique to display the related contents. It intelligently reads the vast data on the web, indexes, ranking, and based on the defined rules and algorithm it displays the search results.

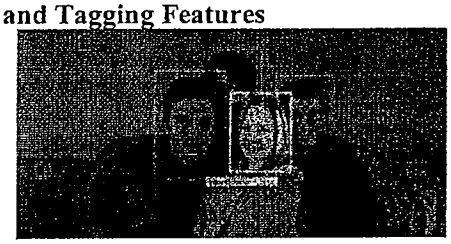
Some of the machine learning products
Moley
Deep Blue
IBM Watson - youngest programmer working with Watson – Tanmay
Nadine social robot - Singapore

Examples of Machine Learning

- **Google and other search engines** providers use machine learning techniques to populate the relevant data. Google recommends so many keywords when we try to search in search box. Ranking web pages.
- **Diagnosis of Deadly Diseases**
  The machine learning technique is being used in the field of healthcare domain. The software helps to detect the deadly diseases such as cancers by reading the past data of the patients and matching that with the symptoms defined in the algorithms.

  To predict recovery rates of pneumonia patients.

  In medicine, learning programs are used for medical diagnosis.
  Computers learning from medical records which treatments are most effective for new diseases
- **Face Reorganization and Tagging Features**



You may have noticed that, while uploading the images into your social media accounts, the software suggests the name of your friends present in the images for tagging. It asks you to whether you want to tag the person whose face is in the uploaded images.

This is possible by using the machine learning techniques. It recognizes the faces of the people present in the images by using the past data of those users and suggests you the accurate names.

- **Detection of Spam Emails**

This is another example which we experienced in our email account. There are emails which directly go to the spam folders instead of inbox, why? That is again the machine learning techniques being used by the software.

The software model identifies the email messages based on the nature of the content it carries and decides whether it is a spam or a genuine email to be allowed to the inbox. The software reads the past data feeds and the set of rules and algorithms to identify the nature of the emails.

- **Displaying the Related Advertisements**

While reading an online article or seeing images or videos, you may have noticed that the related advertisements from various advertisers to buy or see a product keep on coming on the side bars and elsewhere on the web pages. That is again an example of machine learning. The

software recognizes the data used in the articles and shows the related products which the users may be interested in buying or seeing.

- **Speech Recognition**
  To recognize spoken words (Subtitle in youtube videos, individual speakers, vocabularies, microphone characteristics, background noise etc).
  Translation of one language to other language – google translator

- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper.
- Shopping - amazon , Netflix recommends many products
- To find the shortest path between 2 places
- Many industrial purposes- Finding and targeting potential customers
- Drive autonomous vehicles on public highways (ML methods have been used to train computer –controlled vehicles to steer correctly when driving on a variety of road types)
- Postal department uses for Hand writing recognition
- Database mining – to extract information (large databases from growth of automation/web)
- Play games such as backgammon at levels approaching the performance of human world champions (the world's top computer program for backgammon, learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion.)

- Learning to classify new astronomical structures (decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey (Fayyad et al. 1995). This system is now used to automatically classify **all** objects in the Sky Survey, which consists of three terrabytes of image data.)
  Decision tree learning algorithms have been used by NASA to learn how to classify celestial objects.

- Robots to the astronomy and the medical science fields where machine learning techniques are being used.
- Banking- fraud detection (credit cards usage)
- Stock market
- In manufacturing, learning models are used for optimization, control, and troubleshooting.
- In telecommunications, call patterns are analysed for network optimization and maximizing the quality of service.
- In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers.

Areas of Influence for Machine Learning

- **Statistics:** How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn?
- **Brain Models:** Non-linear elements with weighted inputs (Artificial Neural Networks) have been suggested as simple models of biological neurons.
- **Adaptive Control Theory:** How to deal with controlling a process having unknown parameters that must be estimated during operation?
- **Psychology:** How to model human performance on various learning tasks?
- **Artificial Intelligence:** How to write algorithms to acquire the knowledge humans are able to acquire, at least, as well as humans?
- **Evolutionary Models:** How to model certain aspects of biological evolution to improve the performance of computer programs?

Definition: A field of study that gives the computers the ability to learn without being explicitly programmed. (Arthus Samuel)

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

So learning is about learning from past experience

Examples

- **A checkers learning problem**:
  - Task T: playing checkers
  - Performance measure P: percent of games won against opponents

    (probability that wins the next game of checkers against some new opponent)
  - Training experience E: playing practice games against itself

    (experience of having the program play tens of some games against itself)

- **Handwriting recognition learning problem:**
  - Task T: recognizing and classifying handwritten words within images
  - Performance measure P: percent of words correctly classified
  - Training experience E: a database of handwritten words with given classifications

- **A robot driving learning problem:**
  - Task T: driving on public four-lane highways using vision sensors
  - Performance measure P: average distance traveled before an error (as judged by human overseer)
  - Training experience E: a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

- Consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament
- So we want to solve the following learning problem

Problem Description: A Checker Learning Problem

- § ***Task T: Playing Checkers***
- § ***Performance Measure P: Percent of games won against opponents***
- § ***Training Experience E: To be selected ==> Games Played against itself***

- Requires the following steps

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
5. Final Design

Step 1: Choosing the Training Experience

As a first step of the design, we have to choose the type of training experience from which our system will learn.
Type of training experience will have great impact on the outcome, success or failure.
There are three attributes which will affect the outcome.

§ **Direct versus Indirect Experience**

Whether the training experience provide *direct* or *indirect* feedback regarding the choices made by the performance system

- **Direct Feedback:** Individual checkers board states and the correct move for each will be given as input. So system learns from direct training examples.
- **Indirect** Feedback: A bunch of recorded games, where the correctness of the moves is inferred by the result of the game.
  Sequences of moves and final outcomes of various games played will be given as input.

  ***Credit assignment problem*:** Value of early states must be inferred from the outcome (for each move, how good or bad)

Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable direct training examples would be difficult. Instead we will need to work with indirect knowledge of how the game turned out.
Indirect Experience gives rise to the **credit assignment** problem and is thus more difficult.

§ **Teacher versus Learner Controlled Experience**

What amount of interaction should there be between the system and the supervisor/ teacher?
Degree to which the learner controls the sequence of training examples
Choice #1: No freedom. Supervisor/Teacher provides all training examples.
Choice #2: Semi-free. The learner might suggest *interesting* examples and ask the teacher for their outcome; [Supervisor provides training examples, system constructs its own examples too, and asks questions to the supervisor in cases of doubt.]
Choice #3: Total-freedom. The learner can be completely on its own with no access to correct outcomes. System learns to play completely unsupervised.

§ **How Representative is the Experience?**

How well the training experience represents the distribution of examples over which the final system performance *P* will be measured

- If training the checkers program consists only of experiences played against itself, it may never encounter crucial board states that are likely to be played by the human checkers champion
- Most theory of machine learning rests on the assumption that the distribution of training examples is identical to the distribution of test examples

Challenge: Design a Learning System for Checkers

- There is a huge number of possible games.
- No time to try all possible games.
- System should learn with examples that it will encounter in the future. For example, if the goal is to beat humans, it should be able to do well in situations that humans encounter when they play (this is hard to achieve in practice).

Step 2: Choosing the Target Function

§ **Given a set of legal moves, we want to learn how to choose the best move [since the best move is not necessarily known, this is an *optimization* problem]**

§ ***ChooseMove*: B --> M is called a Target Function**
ChooseMove, however, is difficult to learn.
An easier and related target function to learn is *V*: B --> R, which assigns a numerical score to each board. The better the board, the higher the score.

- An evaluation function that assigns a numerical score to any given board state
  $V : B \rightarrow \mathcal{R}$ is a target function(where $\mathcal{R}$ is the set of real numbers)

Knowing the 'value' of the board can be used to pick the best move. Choose the move that leads to the board with the highest value.

V is the name we are giving to this function
B is the set of all possible checker board states
R is the real numbers

§ **Operational versus Non-Operational Description of a Target Function**

$V(b)$ for an arbitrary board state b in B

- if b is a final board state that is won, then $V(b) = 100$
  (if black can guarantee a win starting from board b)
- if b is a final board state that is lost, then $V(b) = -100$
  (if red can guarantee a win starting from board b)
- if b is a final board state that is drawn, then $V(b) = 0$ (if the game is drawn)
- if b is not a final state, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

With this definition, every board b has a value of 100, -100, or 0. To compute it we need to run all possibilities until the final state. So this is a non-operational definition. because we can't work with it in practice.

Finding an operational definition may be even impossible. In practice, we will try to find an operational definition of an approximation to V.

- Goal of learning is to discover an operational description of V
- Learning the target function is often called **function approximation**
  - Referred to as $\hat{V}$

The actual function can often not be learned and must be approximated.

Step 3: Choosing a Representation for the Target Function

Choice of representations involve trade offs

1. A large table with boards b and values $\hat{V}(b)$ or....

2.An artificial neural network that implements it. Imagine a neuron for every cell of the board, firing when a piece is present, and many connections between neurons, or.....

3.A polynomial function of predefined board features ... etc

There is a trade-off between very expressive representations that can get very close to V but are nearly non-operational or simpler representations that are efficiently computable

§ ***Example of a representation:***

- Use linear combination of the following board features:
  - *x1*: 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

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

xi is for some attribute of the board state (i.e. how many red pieces)

wi's are adjustable or "learnable" coefficients

A particular setting of the wi 's is called a hypothesis
The set of all possible settings for all weight assignments wi is called the hypothesis space
We want to find a hypothesis so that $\hat{V}$ best approximates V

Step 4: Choosing a Function Approximation Algorithm

To learn we require a set of training examples describing the board b and the training value $V_{train}(b)$

§ Generating Training Examples of the form (ordered pair)

$$\langle b, V_{train}(b) \rangle$$

$$\langle \langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0 \rangle, +100 \rangle$$ (=blacks won)

Estimating Training Values

- Need to assign specific scores to *intermediate* board states
- Approximate intermediate board state b using the learner's current approximation of the next board state following b

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

  - Simple and successful approach
  - More accurate for states closer to end states
  -

Adjusting the Weights

- Choose the weights w_i to best fit the set of training examples
- Minimize the squared error E between the train values and the values predicted by the hypothesis

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

- Require an algorithm that
  - will incrementally refine weights as new training examples become available
  - will be robust to errors in these estimated training values
- Least Mean Squares (LMS) is one such algorithm

LMS Weight Update Rule

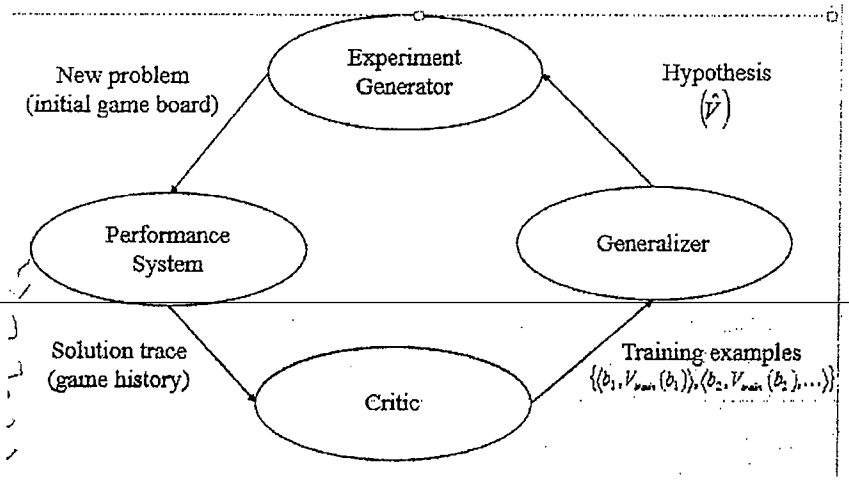
- For each train example $\langle b, V_{train}(b) \rangle$
  - Use the current weights to calculate
- To update the weights, we look at the discrepancy between the training example and the function

  We adjust the weights wi in the \right direction" to make up for the discrepancy...
  - ...but only a little bit

  This is essentially a gradient descent search
  - For each weight w_i, update it as $w_i \leftarrow w_i + \eta \left(V_{train}(b) - \hat{V}(b) \right) x_i$
    - Where η is a small constant (e.g. 0.1)

$\left(V_{train}(b) - \hat{V}(b) \right)_i$ if this is equal to zero , no need to change the value of weight w_i

Step 5 : Final Design for checkers learning



Four main components of the agent

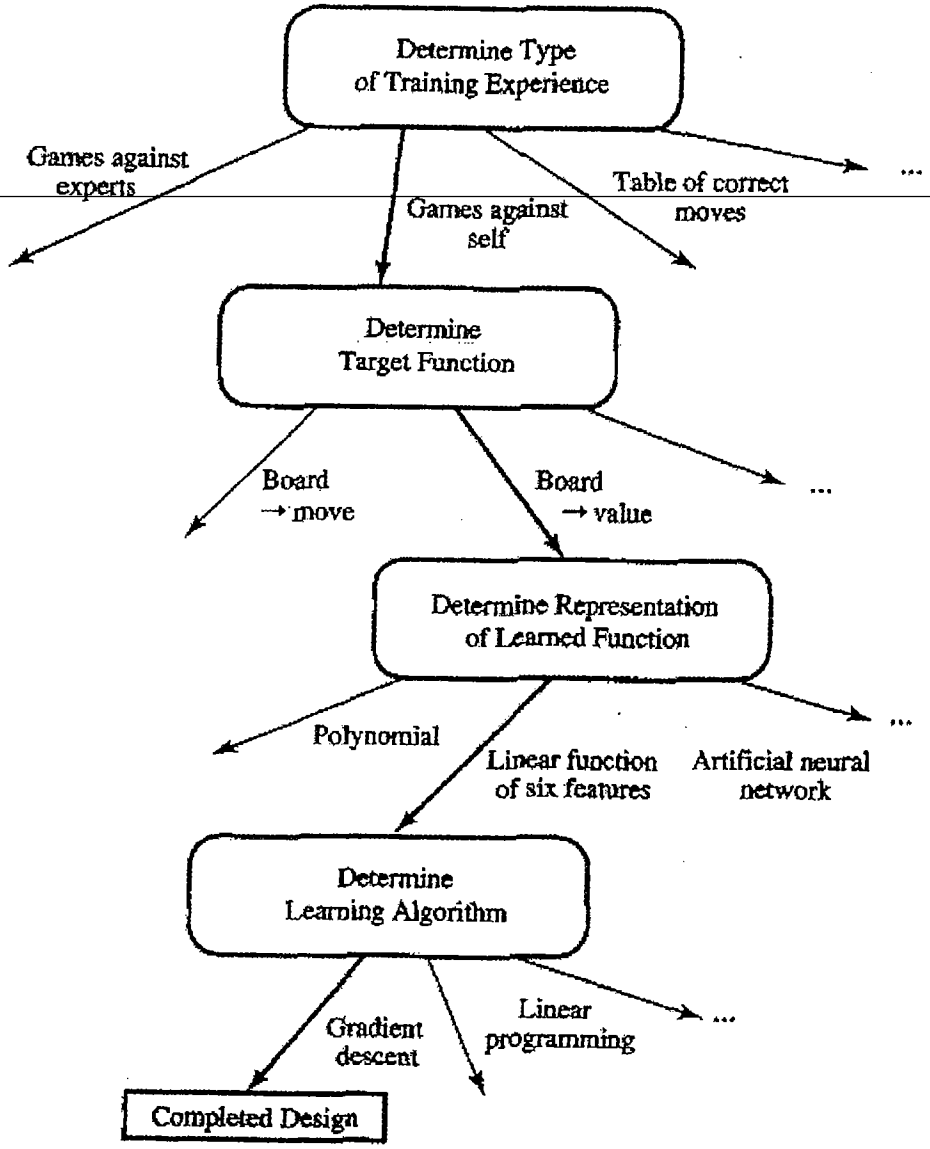
§ ***The Performance Module****: Takes as input a new board and outputs a trace of the game it played against itself.*
Uses the learned target function to solve the problem at hand.

§ ***The Critic****: Takes as input the trace of a game and outputs a set of training examples of the target function*
Produces training examples after an experiment

§ ***The Generalizer****: Takes as input training examples and outputs a hypothesis which estimates the target function. Good generalization to new cases is crucial. (we are using this in our syllabus)*
Updates the learned target function using the critic's training examples

§ ***The Experiment Generator:*** *Takes as input the current hypothesis (currently learned function) and outputs a new problem (an initial board state) for the performance system to explore*
Creates an experiment to start the whole process over again

Summary of Design Choices



General Perspectives and Issues in Machine Learning

- In general, a learning algorithm searches a large (potentially infinite) set of **hypotheses**, and tries to find one that **fits best** the data.
- The various techniques of Machine Learning are in most cases different ways of **hypothesis representation.**
- Different representations suite better different tasks. The class will review representations and explain how they exploit the underlying structure for different problems.

§ What algorithms exist for learning general target functions from specific training examples? Which algorithms perform best for which types of problems and representations?

§ What algorithms are available for learning a concept? How well do they perform?

§ How much training data is sufficient to learn a concept with high confidence? Or

How much training data is needed to achieve a level of 'confidence' to the learned hypothesis? Are there any theoretical and general bounds?

§ When is it useful to use prior knowledge?

§ Are some training examples more useful than others? Or

How should we chose the training experiences? What is the relationship between the strategy and the complexity of the learning problem.

§ What are best tasks for a system to learn?

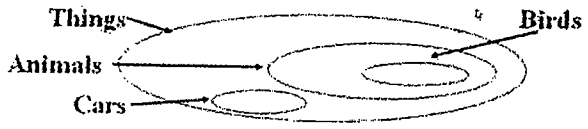
§ What is the best way for a system to represent its knowledge?

§ Is it possible to make automatic the process of picking target functions for learning?

§ Is it possible to have a system that doesn't have a fixed hypothesis representation but keeps changing it in response to the performance?

Concept Learning

A concept is a subset of objects or events defined over a larger set. For example, we refer to the set of everything (i.e., all objects) as the set of things. Animals are a subset of things, and birds are a subset of animals.



In more technical terms, a concept is a Boolean-valued function defined over this larger set. For example, a function defined over all animals whose value is true for birds and false for every other animal.

[But Car is not a subset of Animals.
If instead of things if we consider living things, then car will not appear here for discussion itself, rather humans can include.]

What is Concept Learning?

Given a set of examples labelled as members or non-members of a concept,
concept-learning consists of automatically inferring the general definition of this concept.

In other words, ***Concept Learning is the process of inferring a boolean-valued function from training examples of its input and output.***

Concept Learning can be formulated as a problem of searching through a predefined space of potential hypothesis for the hypothesis that best fits the training examples.

Example of a concept learning task

- ***Target Concept:*** Good days for watersports (values: Yes, No)
  "Days on which my friend Aldo enjoy his favourite water sport"

- ***Target Attribute:*** **EnjoySport**
  "indicates whether Aldo enjoys his favorite water sport or not on this day"

- ***Task:*** To learn to predict the value of EnjoySport for an arbitrary day, based on the values of its other attributes.

- ***Attributes / Features with values for EnjoySport***
  List of possible values for each of the attributes

Sky	Sunny, Cloudy, Rainy
Temp	Warm, Cold
Humid	Normal, High
Wind	Strong, Weak
Water	Warm, Cool
Forecast	Same, Change

- ***Training Examples for EnjoySport*** (both positive and negative training examples)

Sky	Temp	Humid	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

This table consists a set of days, each represented by a set of attributes.

The attribute EnjoySport indicates whether or not Aldo enjoys his favorite water sport on this day.

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

- Example of Training Point :
  < Sunny, Warm, High, Strong, Warm, Same, Yes >

Chosen Hypothesis Representation:

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

Selecting a hypothesis representation is an important step since it restricts (or biases) the space that can be searched. [For example, the hypothesis " If the air temperature is cold or the humidity high then it is a good day for water sports" cannot be expressed in our chosen representation.]

What hypothesis representation shall we provide to the learner in this case?

➢ ***Hypothesis Representation***

- Let us begin by considering a simple representation in which each hypothesis consists of a Conjunction of constraints on the instance attributes.
  Here, h is a conjunction of constraints on attributes.
- Let each hypothesis be a vector of 6 constraints, specifying the values of the six attributes *Sky, AirTemp, Humidity, Wind, Water,* and *Forecast.*
- For each attribute the hypothesis will either "?", single value (e.g. *Warm*) , or "Ø"
  - exact value if we want that specific value (e.g., Water = W arm)
  - ? means "***any value is acceptable***" for this attribute or don't care(eg,Water=?)
  - "Ø" means "no value is acceptable" (e.g., W ater = Ø).

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

○ Most General Hypothesis: Everyday is a good day for water sports
< ?, ?, ?, ?, ?, ? >
That is ***everyday*** is a positive example

○ Most Specific Hypothesis: No day is a god day for water sports
<Ø, Ø, Ø, Ø, Ø, Ø>
That is ***no day*** is a positive example.

- Hypotheses are represented as vectors, e.g.,

Sky	AirTemp	Humid	Wind	Water	Forecast
<Sunny	?	?	Strong	?	Same>

➢ Goal : To infer the "best" concept- description from the set of all possible hypothesis ("best" means generalizes to all (known or unknown) elements of the instance space)

➢ ***Example of a hypothesis :***

Many possible representations.

Hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity, independent of the values of the other attributes, is represented by the expression

< ?, Cold, High, ?, ?, ? >

Above hypothesis representation means that If the air temperature is cold and the humidity high then it is a good day for water sports.

What do u say, whether the statement is true or not

- If we look into the training example table, the only day when AirTemp is cold is day three. But EnjoySport of Day 3 is No, so it says hypothesis is false.

Learning Task Notation

Given

- ***Set of Instances X:*** Set of items over which the concept is defined.
- ***Target concept c:*** Concept or function to be learned

In general c can be any Boolean valued function defined over the instances X,

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

- ***Training examples (positive/negative):*** $< x, c(x) >$

Training examples consists of the instances x along with their target concept value c(x).

- ***Training set D:*** available training examples

Eg: X is set of all Possible days, each described by the attributes.
Sky, AirTemp, Humidity, Wind, Water, Forecast

The target concept corresponds to the value of the attribute EnjoySport

$$c(x)=1 \text{ if } EnjoySport = yes, \text{ and}$$

$$c(x) = 0 \text{ if } EnjoySport = No$$

Instances for which c(x)=1are called positive examples or members of the target concept.

Instances for which c(x)=0 are called negative examples or nonmembers of the target concept.

Sky	Sunny, Rainy, Cloudy
Airtemp	Warm, Cold
Humidity	Normal, High
Wind	Strong, Weak
Water	Warm, Cool
Forecast	Same, Change

The function EnjoySport() which is the concept to be learnt can be denoted as a boolean function as:

C(x) => { Yes, No }

Where C(x) is the EnjoySport values

Dave defines a function called Enjoy Sport which is a Boolean function which takes the attributes as inputs and outputs the result as yes or no based on the day when water sports are enjoyed.

Determine

- ***Hypotheses H : Set of all possible hypothesis. It is determined by the human designer's choice of a hypothesis representation.***

Conjunction of constraints on attributes.

Given a set of training examples of the target concept ***c,*** the problem faced by the learner is to hypothesize, or estimate, ***c.***

- ***Goal of concept- learning : to find a hypothesis (h)***

In general, each hypothesis ***h*** in H represents a boolean-valued function defined over X.

A hypothesis h in H, h: X -> {0,1} such that h(x) = c(x) for all x in X.

(Hypothesis – h- single hypothesis, Hypotheses - H – multiple hypothesis)

- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "$\emptyset$" (no value is acceptable), or a specific value.
  - Target concept c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples D: Positive and negative examples of the target function (see Table 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

Inductive Learning Hypothesis

- **Learning task** is to determine h identical to c over the *entire* set of instances X.
- **But** the only information about c is its value over D.
- **Inductive learning algorithms** can *at best* guarantee that the induced h fits c over D.
- **Assumption** is that the best h regarding *unseen* instances is the h that best fits the *observed* data in D.
- **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.

  (Any good hypothesis over a sufficiently large set of training examples will also approximate the target function. well over *unseen* examples.)

Concept Learning as Search

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

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

Search

- Find a hypothesis that best fits training examples
- Efficient search in hypothesis space (finite/infinite)

Search space in *EnjoySport*

Consider, for example, the instances X and hypotheses H in the ***EnjoySport*** learning task.

- Given that the attribute ***Sky*** has three possible values, and that ***AirTemp, Humidity, Wind, Water,*** and ***Forecast*** each have two possible values, the instance space X contains exactly

  3*2*2*2*2*2 = 96 distinct instances

  (eg. 3 is Sky={Sunny, Cloudy, Rainy} 2 is AirTemp={warm, cold})

- 5*4*4*4*4*4 = 5120 *syntactically distinct* hypotheses within *H* (considering 0 and ? in addition)

- 1+4*3*3*3*3*3 = 973 *semantically distinct* hypotheses (count just one 0 for each attribute since every hypo having one or more 0 symbols is empty)

General-to-Specific Ordering of Hypotheses

To illustrate the general-to-specific ordering, consider the 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.

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

We now define the ***more-general_than_or_equal_to*** relation in terms of the sets of instances that satisfy

Given two hypotheses: h_j and h_k,
h_j is more-general-than-or- equal-to h_k ,
if and only if any instance that satisfies h_k also satisfies h_j.

Definition: Let h_j and h_k be boolean-valued functions defined over X. Then h_j is **more_general_than_or_equal_to** h_k (written $h_j \geq_g h_k$) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

More-General-Than Ordering – Formal Definition

- Let h_j and h_k be boolean-valued functions defined over X.
- h_j is more-general-than-or-equal-to h_k (written $h_j \geq_g h_k$) iff
$$(\forall x \in X)(h_k(x) = 1 \rightarrow h_j(x) = 1).$$
- h_j is (strictly) more-general-than h_k (written $h_j >_g h_k$) iff
$$h_j \geq_g h_k \wedge h_k \not\geq_g h_j.$$
- h_k is more-specific-than h_j iff h_j is more-general-than h_k.
- $\geq_g$ is a partial ordering over H,
i.e., it is reflexive, antisymmetric, transitive and some pairs may not be ordered.

Instances, Hypotheses, and the More-General-Than Ordering

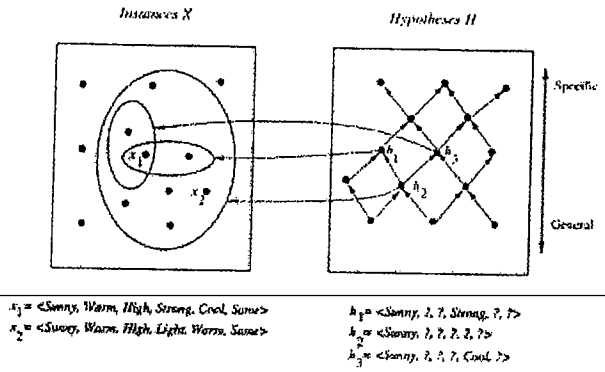


FIGURE 2.1
Instances, hypotheses, and the *more_general_than* relation. The box on the left represents the set X of all instances, the box on the right the set H of all hypotheses. Each hypothesis corresponds to some subset of X—the subset of instances that it classifies positive. The arrows connecting hypotheses represent the *more_general_than* relation, with the arrow pointing toward the less general hypothesis. Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1, hence h_2 is *more_general_than* h_1.

To illustrate these definitions, consider the three hypotheses h_1, h_2, and h_3 from our *EnjoySport* example, shown in Figure 2.1. How are these three hypotheses related by the $\geq_g$ relation? As noted earlier, hypothesis h_2 is more general than h_1 because every instance that satisfies h_1 also satisfies h_2. Similarly, h_2 is more general than h_3. Note that neither h_1 nor h_3 is more general than the other; although the instances satisfied by these two hypotheses intersect, neither set subsumes the other. Notice also that the $\geq_g$ and $>_g$ relations are defined independent of the target concept. They depend only on which instances satisfy the two hypotheses and not on the classification of those instances according to the target concept. Formally, the $\geq_g$ relation defines a partial order over the hypothesis space H (the relation is reflexive, antisymmetric, and transitive). Informally, when we say the structure is a partial (as opposed to total) order, we mean there may be pairs of hypotheses such as h_1 and h_3, such that $h_1 \not\geq_g h_3$ and $h_3 \not\geq_g h_1$.

The $\geq_g$ relation is important because it provides a useful structure over the hypothesis space H for *any* concept learning problem. The following sections present concept learning algorithms that take advantage of this partial order to efficiently organize the search for hypotheses that fit the training data.

FIND-S Algorithm

1. Initialize *h* to the most specific hypothesis in *H*
2. For each positive training example/ Instance *x* *do:*

   For each attribute constraint *ai* in *h do:*

   If the constraint *ai in h* is satisfied by *x*

   Then do nothing

   Else replace *ai* in *h* by the next more general constraint that is satisfied by *x*
3. Output hypothesis *h*

The first step of FINDS is to initialize *h* to the most specific hypothesis in H.

$$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

Upon observing the first training example, which happens to be a positive example, it becomes clear that our hypothesis is too specific.
In particular, none of the "∅" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example; namely, the attribute values for this training example.

$$h \leftarrow \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$$

This *h* is still very specific.
The second training example (also positive in this case) forces the algorithm to further generalize *h,* this time substituting a "?' in place of any attribute value in h that is not satisfied by the new example.

The refined hypothesis in this case is

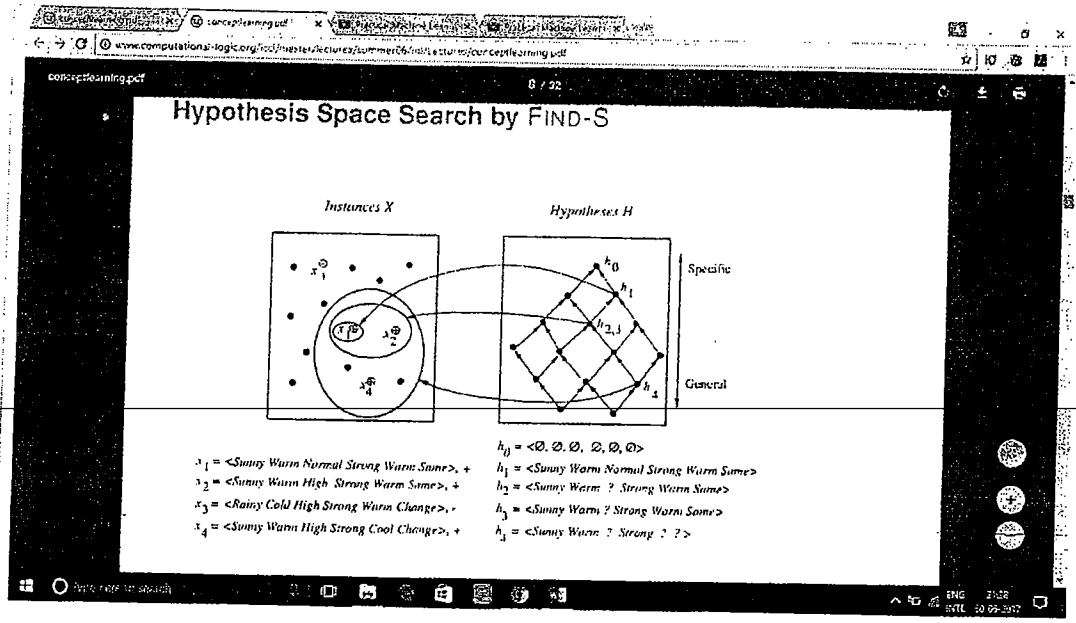
$$h \leftarrow \langle Sunny, Warm, ?, Strong, Warm, Same \rangle$$

In this case, FIND- S algorithm simply ***ignores all negative example.***

$$h \leftarrow \langle \textbf{Sunny, Warm, ?, Strong, ?, ?} \rangle$$

Drawbacks of FIND-S

- FIND-S is very sensitive to incorrect data i.e., noise, that may be in training examples.
- The only contributors to the hypothesis are positive examples in training data.
- Although it will find hypothesis consistent with the data, there is no way to determine that this hypothesis is the only target concept consistent with the data.
- There is no way to determine how many consistent hypothesis exists within the hypothesis class.

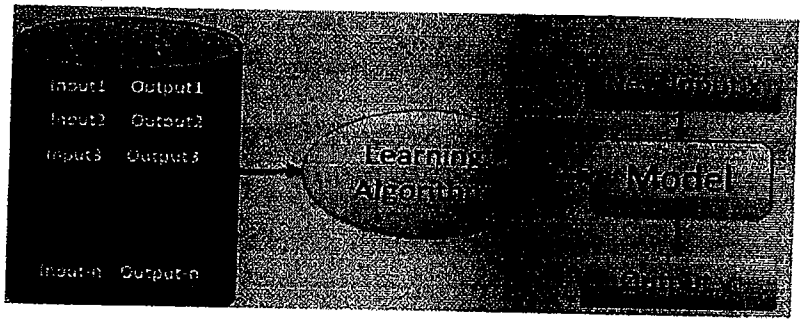

conceptlearning.pdf
6 / 32
Hypothesis Space Search by FIND-S
Instances X
Hypotheses H
Specific
General
x_1 = <Sunny Warm Normal Strong Warm Same>, +
x_2 = <Sunny Warm High Strong Warm Same>, +
x_3 = <Rainy Cold High Strong Warm Change>, -
x_4 = <Sunny Warm High Strong Cool Change>, +
h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny Warm Normal Strong Warm Same>
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ?>

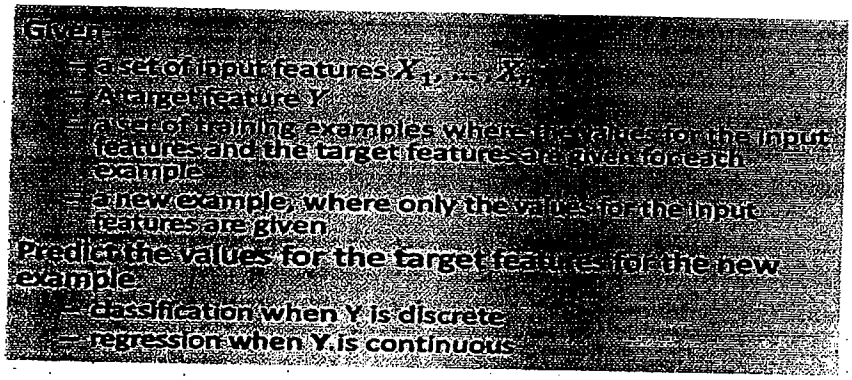
Learning can happen in 3 methods

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

UNIT-2,3

Supervised Learning





The output feature Y can be discrete or continuous.
Discrete – classification

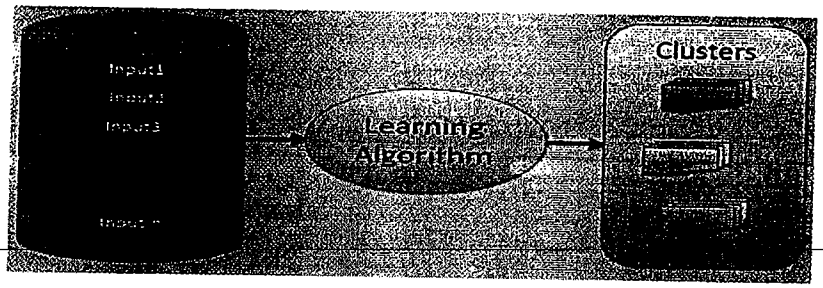
1. Tomorrow rains or not
2. By giving different symptoms of patient and predicts whether the patient has particular disease or not

Continuous- regression

1. Predicting price of a house, which is a real valued number, by givining different features like location, built in area, no. of bedrooms etc.

Credit – scoring

Unsupervised Learning



In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor. In unsupervised learning, there is no such supervisor and we only have input data. The aim is to find the regularities in the input.

In Clustering , training will be done without any output.
And grouping will be done with out any predefined labels

Reinforcement Learning

The machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. Such learning methods are called *reinforcement learning* algorithms.

A good example is *game playing* where a single move by itself is not that important; it is the sequence of right moves that is good. A move is good if it is part of a good game playing policy. Game playing is an important research area in both artificial intelligence and machine learning. This is because games are easy to describe and at the same time, they are quite difficult to play well. A game like chess has a small number of rules but it is very complex because of the large number of possible moves at each state and the large number of moves that a game contains. Once we have good algorithms that can learn to play games well, we can also apply them to applications with more evident economic utility.

A robot navigating in an environment in search of a goal location is another application area of reinforcement learning. At any time, the robot can move in one of a number of directions. After a number of trial runs, it should learn the correct sequence of actions to reach to the goal state from an initial state, doing this as quickly as possible and without hitting any of the obstacles.

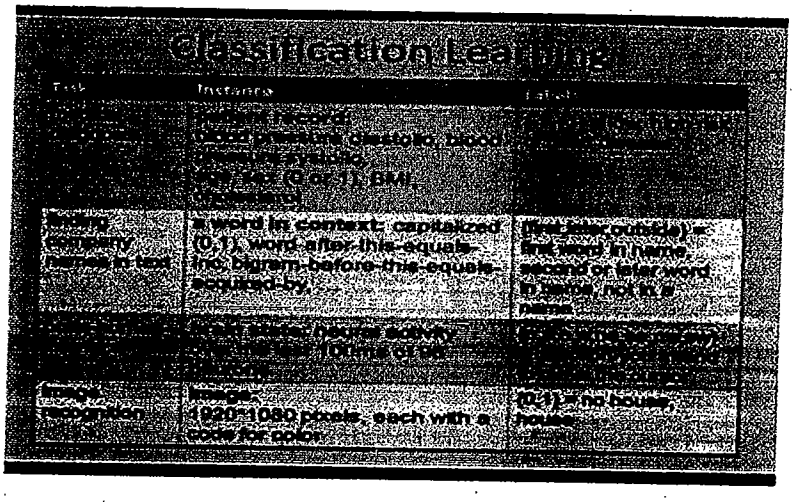
One factor that makes reinforcement learning harder is when the system has unreliable and partial sensory information. For example, a robot equipped with a video camera has incomplete information and thus at any time is in a *partially observable state* and should decide taking into account this uncertainty; for example, it may not know its exact location in a room but only that there is a wall to its left. A task may also require a concurrent operation of *multiple agents* that should interact and cooperate to accomplish a common goal. An example is a team of robots playing soccer.

Classification

It falls under supervised learning. It is used for prediction or estimating depending on predefined labels.

Eg. - Given email labeled as spam/not spam, learn a spam filter.

- Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.



- A credit is an amount of money loaned by a financial institution, for example, a bank, to be paid back with interest, generally in instalments.

It is important for the bank to be able to predict in advance the risk associated with a loan, which is the probability that the customer will default and not pay the whole amount back. This is both to make sure that the bank will make a profit and also to not inconvenience a customer with a loan over his or her financial capacity.

Some of the attributes are income, savings, profession, age, past financial history, and so forth. The bank has a record of past loans containing such customer data and whether the loan was paid back or not. From this data of particular applications, the aim is to infer a general rule coding the association between a customer's attributes and his risk.

This classification is an example of a *classification* problem where there are two classes: low-risk and high-risk customers.

Input to the classifier : customer information
Task: to assign the input to one of the two classes.

After training with the past data, a classification rule learned may be of the form

IF income> $\theta 1$ AND savings> $\theta 2$ THEN low-risk ELSE high-risk

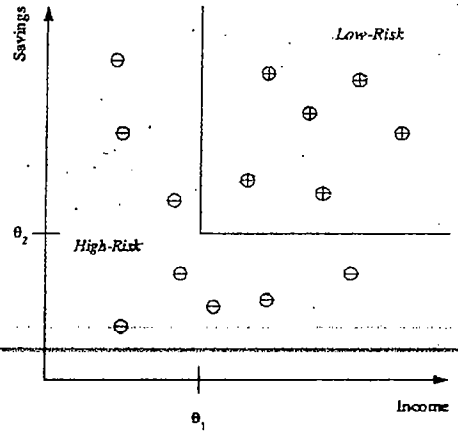
for suitable values of $\theta 1$ and $\theta 2$.

This is an example of a *discriminant*; it is a function that separates the examples of different classes.
Having a rule like this, the main application is *prediction*: Once we have a rule that fits the past data, if the future is similar to the past, then we can make correct predictions for novel instances. Given a new application with a certain income and savings, we can easily decide whether it is lowrisk or high-risk.

In some cases, instead of making a 0/1 (low-risk/high-risk) type decision, we may want to calculate a probability, namely, $P(Y|X)$,

where X are the customer attributes and Y is 0 or 1 respectively for low-risk and high-risk. From this perspective, we can see classification as learning an association from X to Y. Then for a given $X = x$, if we have $P(Y = 1|X = x) = 0.8$, we say that the customer has an 80 percent probability of being high-risk, or equivalently a 20 percent probability of being low-risk. We then decide whether to accept or refuse the loan depending on the possible gain and loss.

There pattern are many applications of machine learning in *pattern recognition*.
One is *optical character recognition*, which is recognizing character codes from their images. This is an example where there are multiple classes, as many as there are characters we would like to recognize. Especially interesting is the case when the characters are handwritten—for example, to read zip codes on envelopes or amounts on checks. People have different handwriting styles; characters may be written small or large, slanted, with a pen or pencil, and there are many possible images corresponding to the same character. Though writing is a human invention, we do not have any system that is as accurate as a human reader. We do not have a formal description of 'A' that covers all 'A's and none of the non-'A's. Not having it, we take samples from writers and learn a definition of A-ness from these examples. But though we do not know what it is that makes an image an 'A', we are certain that all those distinct 'A's have something in common, which is what we want to extract from the examples. We know that a character image is not just a collection of random dots; it is a collection of strokes and has a regularity that we can capture by a learning program.



Regression

Let us say we want to have a system that can predict the price of a used car.
Inputs are the car attributes—brand, year, engine capacity, mileage, and other information—that we believe affect a car's worth.

The output is the price of the car. Such problems where the output is a number are *regression* problems.

Let X denote the car attributes and Y be the price of the car. Again surveying the past transactions, we can collect a training data and the machine learning program fits a function to this data to learn Y as a function of X. The fitted function is of the form

$$y = wx + w0$$

for suitable values of w and $w0$.

Linear regression is an approach for modelling the relationship between a scalar dependent variable Y and one or more explanatory variables (independent) denoted by X.
If the equation has only one independent variable, it is called simple linear regression.

Application

1. If the goal is prediction or forecasting or error reduction, linear regression can be used **to fit a predictive model** to an observed data set Y and X values.
2. Linear regression can be applied to quantify the strength of the relationship between Y and X.

Both regression and classification are *supervised learning* problems where there is an input, X, an output, Y, and the task is to learn the mapping from the input to the output. The approach in machine learning is that we assume a model defined up to a set of parameters:
$y = g(x|\theta)$

where $g(\cdot)$ is the model and θ are its parameters. Y is a number in regression and is a class code (e.g., 0/1) in the case of classification. $g(\cdot)$ is the regression function or in classification, it is the discriminant function separating the instances of different classes. The machine learning program optimizes the parameters, θ, such that the approximation error is minimized, that is, our estimates are as close as possible to the correct values given in the training set.

Another example of regression is navigation of a mobile robot, for example, an autonomous car, where the output is the angle by which the steering wheel should be turned at each time, to advance without hitting obstacles and deviating from the route. Inputs in such a case are provided by sensors on the car—for example, a video camera, GPS, and so forth. Training data can be collected by monitoring and recording the actions of a human driver.

Clustering where the aim is to find clusters or groupings of input. In the case of a company with a data of past customers, the customer data contains the demographic information as well as the past transactions with the company, and the company may want to see the distribution of the profile of its customers, to see what type of customers frequently occur. In such a case, a clustering model allocates customers similar in their attributes to the same group, providing the company with natural groupings of its customers; this is called *customer segmentation*. Once such groups are found, the company may decide strategies, for example, services and products, specific to different groups; this is known as *customer relationship management*. Such a grouping also allows identifying those who are outliers, namely, those who are different from other customers, which may imply a niche in the market that can be further exploited by the company.

An interesting application of clustering is in *image compression*. In this case, the input instances are image pixels represented as RGB values. A clustering program groups pixels with similar colors in the same group, and such groups correspond to the colors occurring frequently in the image. If in an image, there are only shades of a small number of colors, and if we code those belonging to the same group with one color, for example, their average, then the image is quantized.

Clustering, the aim is to group similar documents. For example, news reports can be subdivided as those related to politics, sports, fashion, arts, and so on.

Association

Determines which items are frequently purchased together with in same transactions.

Eg. For basket analysis in a supermarket chain,

some customers who purchase 'x', typically purchase 'y'. So, we can identify the potential customers who are purchasing only 'x' but not 'y'. If we believe he is a potential customer, we can target for "cross-selling".

⇒ $P(Y/x)$ – y is the product that we brought on purchase of x.

$P(\text{chips}/\text{coke}) = 0.7$

ie, 70% customers who buy coke will purchase chips ie, nothing but confidence.

→ Antivirus / Laptop , Jam / Bread

So we can find (estimate) what is the Probability (chance) that if we (customer) buy X Product, he will buy 'Y' Product too.

⇒ we need distinction among different customers.

$P(Y/x, D)$ where D = set of customer attributes like gender, age, marital status etc.

A Book seller want to check how many student customers are purchasing book & how many faculty customers are purchasing.

$P(\text{R Program}/\text{MachineLearning}, \text{faculty})$

How many faculties who are purchasing ML textbook are purchasing R Program text book?

Apriori algorithm is used for mining frequent itemsets for Boolean associations

It follows a two-step process, consisting of Join & prune actions

1) The Join step: Here a set of candidate itemsets are generated

2) The prune step: Here the itemsets with $\geq$ minimum support is selected & the other itemsets are discarded.

Association

Association is useful for discovering interesting relationships hidden in large datasets.

The strength of the Association Rule can be measured in terms of its 'support' and 'confidence.

Support - determines how often a rule is applicable to a given dataset.

$$S(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

confidence - determines how frequently items Y appears in transactions that contains X.

$$C(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

The first association rule mining algorithm is APRIORI algorithm. The keyidea of the algorithm is to begin generating frequent itemset.

Eg. from next page dataset

$\{Milk, Egg\} \rightarrow \{chip\}$

$$\text{Support}(\{Milk, Egg\} \rightarrow \{chip\}) = \frac{\sigma(Milk, Egg, chip)}{N} = \frac{4}{9} = 0.45$$

$$\text{confidence}(\{Milk, Egg\} \rightarrow \{chip\}) = \frac{\sigma(Milk, Egg, chip)}{\sigma(Milk, Egg)} = \frac{4}{4} = 1$$

100% confidence that whoever purchase Milk, Egg, they Purchase chips too.

Consider the foll. transaction database

Customer	Items
C_1	Milk, Egg, Bread, chip
C_2	Egg, Popcorn, chip, coke
C_3	Egg, Bread, chip
C_4	Milk, Egg, Bread, Popcorn, chip, coke
C_5	Milk, Bread, coke
C_6	Egg, Bread, coke
C_7	Milk, Bread, chip
C_8	Milk, Egg, Bread, Butter, chip
C_9	Milk, Egg, Butter, chip

Min sup = 30% = 3 → Means 3 records in this example

C_1

Itemset	sup. count
{Milk}	6
{Egg}	7
{Bread}	7
{Popcorn}	2
{Butter}	2
{Chip}	7
{Coke}	4

scan D for count of each candidate →

Compare candidate support count with min. support count →

Itemset	sup. count
{Milk}	6
{Egg}	7
{Bread}	7
{Chip}	7
{Coke}	4

→ Scan D for count of each candidate

Generate C_2 candidates →

Itemset	Sup. count
Milk, egg	4
Milk, Bread	5
Milk, chip	5
Milk, coke	(2) X
Egg, Bread	5
Egg, chip	6
Egg, coke	3
Bread, chip	5
Bread, coke	3
chip, coke	(2) X

removed

Compare sup. count with min. sup. count →

Itemset	Sup. count
Milk, egg	4
Milk, Bread	5
Milk, chip	5
Egg, Bread	5
Egg, chip	6
Egg, coke	3
Bread, chip	5
Bread, Coke	3

Generate candidate C_3 from →

Itemset	Sup. count ← Scan D for each candidate count
Milk, Egg, Bread	3
Milk, Egg, chip	4
Milk, Egg, coke	(1) ×
Milk, Bread, chip	4
Milk, Bread, coke	(2) ×
Egg, Bread, chip	4
Egg, Bread, coke	(2) ×
Egg, chip, coke	(2) ×
Bread, coke, chip	(1) ×

removed

compare sup. count with min. sup. count →

Itemset	sup. count
Milk, Egg, Bread	3
Milk, Egg, chip	4
Milk, Bread, chip	4
Egg, Bread, chip	4

Generate candidate C_4 →

Itemset	sup. count ← Scan from dataset
Milk, Egg, Bread, chip	4

So {Milk, Egg, Bread, chip} is the itemset which sold frequently

Dimensionality Reduction

Introduction

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, means noise in data. Noisy or redundant data makes it more difficult to discover meaningful patterns. If the data set is high-dimensional, most data mining algorithms require a much larger training data set.

This becomes even more important when the number of features (attributes) are very large. You need not use every feature for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. For example, you might have a dataset with 500 columns that describe the characteristics of customers; however, if the data in some of the columns is very sparse you would gain very little benefit from adding them to the model, and if some of the columns duplicate each other, using both columns could affect the model.

You not only reduce the training time and the evaluation time, you also have less things to worry about!

In an application, whether it is classification or regression, observation data will be considered as inputs and fed to the system for decision making. Ideally, no need of feature selection or extraction as a separate process; the classifier (or regressor) should be able to use whichever features are necessary, discarding the irrelevant.

However, there are several reasons why we are interested in reducing dimensionality as a separate preprocessing step:

In most learning algorithms, the complexity depends on the number of input dimensions, d, as well as on the size of the data sample, N.

Advantages of Reducing dimensionality without loss of information are

- Improves the quality of the model
- Process of modelling will be more efficient .
- Use less memory space during training process
- Needs less time for computation time
- Decrease the cost of extracting input
- Make Visualisation easier
- Can be plotted easily
- Outliers can be identified easily
- Get a better idea about the process
- A simpler model is simpler to understand and explain
- It reduces overfitting [If more instances had been given for training, then scenarios will be more , then system will by hard the things instead of running algorithm and predicting]

Methods for reducing dimensionality:

Feature selection

Feature extraction

What is Feature Selection

Feature selection is also called variable selection or attribute selection.

Feature selection is an important part of machine learning. Feature selection refers to the process of reducing the inputs for processing and analysis, or of finding the most meaningful inputs.

Feature selection... is the process of selecting a subset of relevant features for use in model construction

It is the automatic selection of attributes in your data (such as columns in tabular data) that are most relevant to the predictive modelling problem you are working on.

In feature selection, we are interested in finding k of the d dimensions that give us the most information and we discard the other (d – k) dimensions.

Subset selection is a feature selection method.

Fewer attributes is desirable because

- It enables the machine learning algorithm to train faster.
- It reduces the complexity of a model and makes it easier to interpret.
- It improves the accuracy of a model if the right subset is chosen.
- Feature selection methods can be used to identify and remove unneeded, irrelevant and redundant attributes from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model.
- The objective of variable selection is three-fold: improving the prediction performance of the predictors, providing faster and more cost-effective predictors, and providing a better understanding of the underlying process that generated the data.

There are three general classes of feature selection algorithms:

filter methods, wrapper methods and embedded methods.

What is Feature Extraction

In feature extraction, we are interested in finding a new set of k dimensions that are combinations of the original d dimensions.

The best known and most widely used feature extraction methods are

Principal Components Analysis (PCA) - unsupervised and
Linear Discriminant Analysis (LDA) - supervised

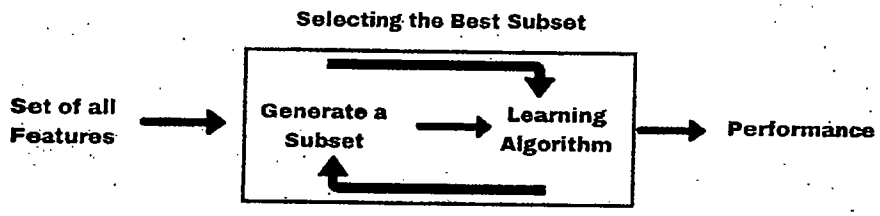
- Both methods seek to reduce the number of attributes in the dataset, but a feature extraction method do so by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them.
- These methods may be supervised or unsupervised depending on whether or not they use the output information.

The best subset contains the least number of dimensions that most contribute to accuracy. The remaining attributes will be discarded.

If there are d variables, then there will be 2^d possible subsets but we cannot test for all of them unless d is small and we employ heuristics to get a reasonable (but not optimal) solution in reasonable (polynomial) time. Your goal in feature selection should be to identify the minimum number of columns from the data source that are significant in building a model.

One of the feature selection method is Wrapper method.

Wrapper Method



In wrapper methods, we try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from your subset. The problem is essentially reduced to a search problem. These methods are usually computationally very expensive.

Two famous approaches in wrapper method are

- ✓ Forward Selection
- ✓ Backward Selection

- **Forward Selection:** Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.
- **Backward Elimination:** In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.

In ***forward selection***, we start with no variables and add them one by one, at each step adding the one that decreases the error the most, until any further addition does not decrease the error.

In ***backward selection***, we start with all variables and remove them one by one, at each step removing the one that decreases the error the most (or increases it only slightly), until any further removal increases the error significantly.

In either case, checking the error should be done on a validation set distinct from the training set because we want to test the generalization accuracy.
With more features, generally we have lower training error, but not necessarily lower validation error.

Let us denote by F, a feature set of input dimensions, xi, i = 1, . . . , d.
E(F) denotes the error incurred on the validation sample when only the inputs in F are used.

Depending on the application, the error is either the mean square error or misclassification error.

In sequential forward selection, we start with no features: F = ∅.
At each step, for all possible xi , we train our model on the training set and

calculate E(F ∪ xi) on the validation set.

Then, we choose that input xj that causes the least error

$$j = \arg\min_i E(F \cup x_i)$$

and we

add x_j to F if $E(F \cup x_j) < E(F)$

We stop if adding any feature does not decrease E or too small change in E.

There is a drawback with this method, does not guarantee finding the optimal subset, namely, the minimal subset causing the smallest error. For example, *xi* and *xj* by themselves may not be good but together may decrease the error a lot, but because this algorithm is greedy and adds attributes one by one, it may not be able to detect this.

In ***sequential backward*** *selection*, we start with F containing all features and do a similar process except that we remove one attribute from F as opposed to adding to it, and we remove the one that causes the least error

$$j = \arg\min_i E(F - x_i)$$

and we

remove x_j from F if $E(F - x_j) < E(F)$

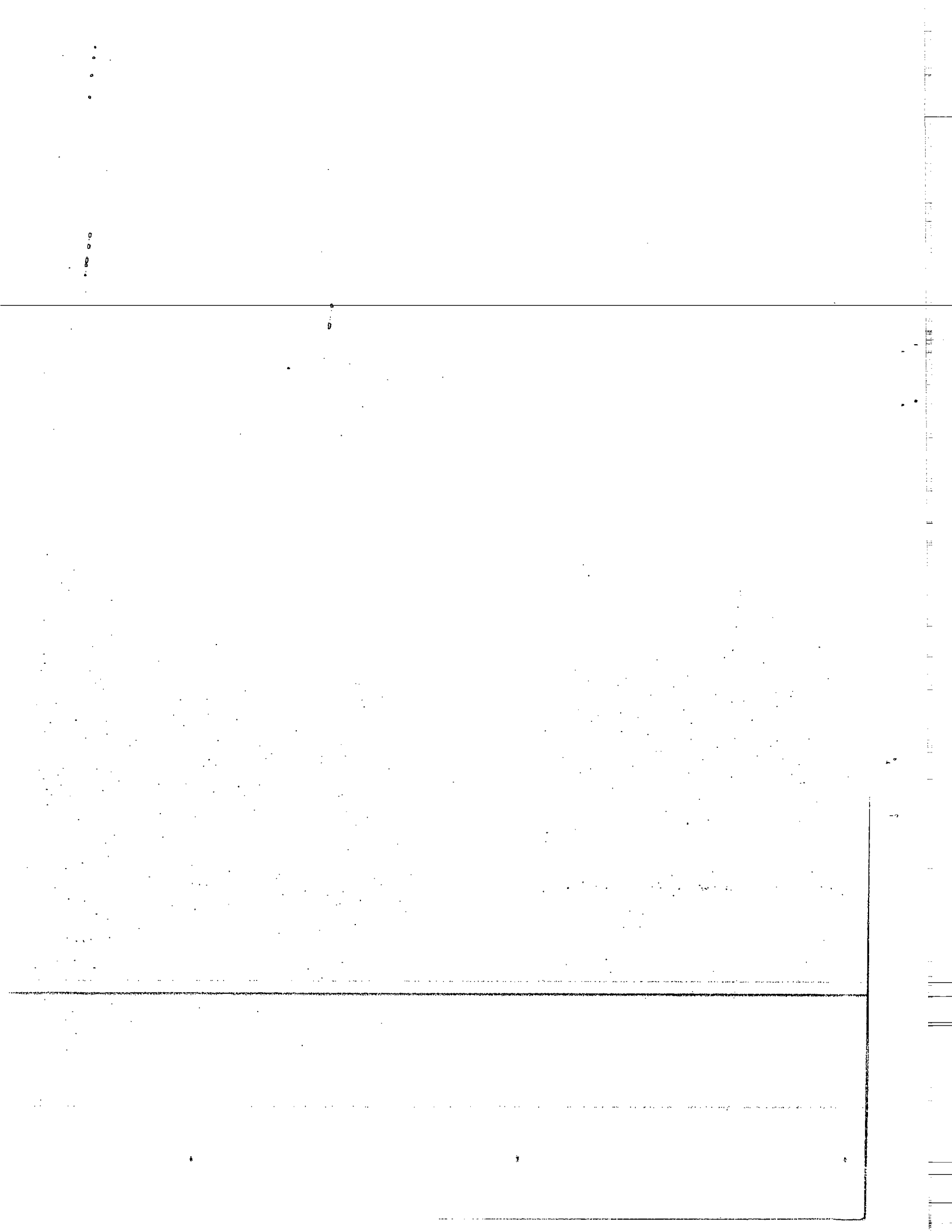
We stop if removing a feature does not decrease the error. To decrease complexity, we may decide to remove a feature if its removal causes only a slight increase in error.

Comparing Forward and Backward selection

All the variants possible for forward search are also possible for backward search. The complexity of backward search has the same order of complexity as forward search, except that training a system with more features is more costly than training a system with fewer features, and forward search may be preferable especially if we expect many useless features.

Comparing Feature selection and Feature Extraction

In an application like face recognition, feature selection is not a good method for dimensionality reduction because individual pixels by themselves do not carry much discriminative information; it is the combination of values of several pixels together that carry information about the face identity. This is done by feature extraction methods.



Principal component Analysis (PCA) Module 2

⇒ PCA is one of the method used for feature extraction.

⇒ Feature extraction means deriving subset of variables from n variables.

⇒ General objective of PCA is
- dimension reduction without much loss of information.
(through finding max. variance & orthogonal).

Suppose a brand shops there with 2 brands of bread, 3 brands of Rice.

Shop	X_1 Bread Brand P	X_2 Bread Brand Q	X_3 Rice Brand A	X_4 Rice Brand B	X_5 Rice Brand C
1	15	20	45	55	75
2	14	19	46	58	72
=					
=					
100					

→ for understanding purpose

⇒

Shop	Average of Bread	Average of Rice
1	17.5	58.3
2	16.5	58.6
=		
=		
100		

From Average price we can't get the info. of individual Brands.
means we are loosing information.
we are not support to calculate / compute & make as one attribute to reduce no. of variables.
Instead we should try to derive into a new attribute from multiple attributes, without loosing actual information.

⇒ Analysis of PCA serve as intermediate steps or much

Input to
- multiple regression
- factor analysis
- discriminant analysis

⇒ A Principal component analysis (PCA) is concerned with explaining the variance-covariance structure of a set of variables through a few linear combinations of these variables.

we have → set of variables $(x_1, x_2, \dots x_n)$ → represents in form of Matrix

from → Linear combination $(a_1x_1 + a_2x_2 + \dots + a_px_p)$

trying to explain → variance - covariance structure (covariance matrix)

To find covariance matrix we find
- Eigen values
- Eigen vectors
- Maximum variance (deviation)
- direction of PC 1
- Orthogonal to PC_1 — ie, PC_2

→ for understanding purpose what definition of PCA means

Eigen values - finds direction of variance of data, determines the radius of ellipse.

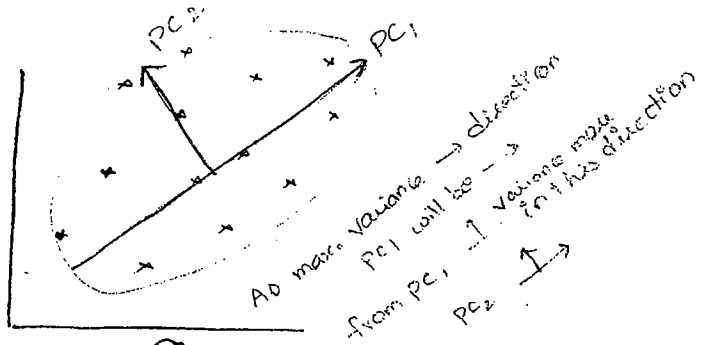
⇒ Eigen vectors — axis of maximum variability.

⇒ PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables.

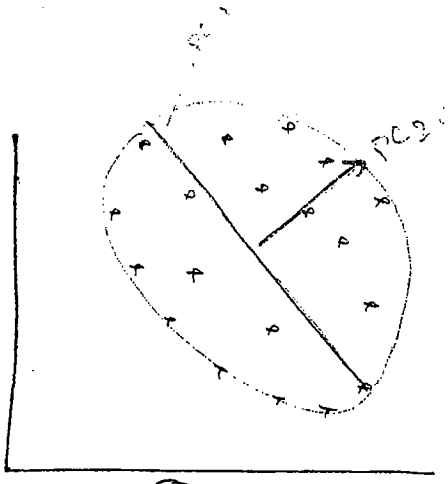
⇒ PCA with the help of the Eigen vector, transform the features into a new space.

⇒ PCA finds the linear combination of variables by rotating data distribution such that about axis of maximum variance.

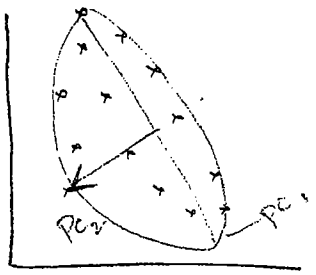
→ This becomes new horizontal axis.
called as Varimax rotation.



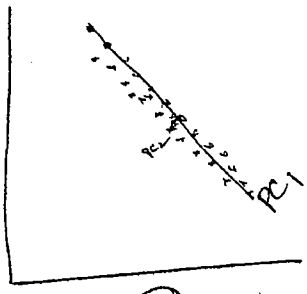
(A)



(B)



(C)

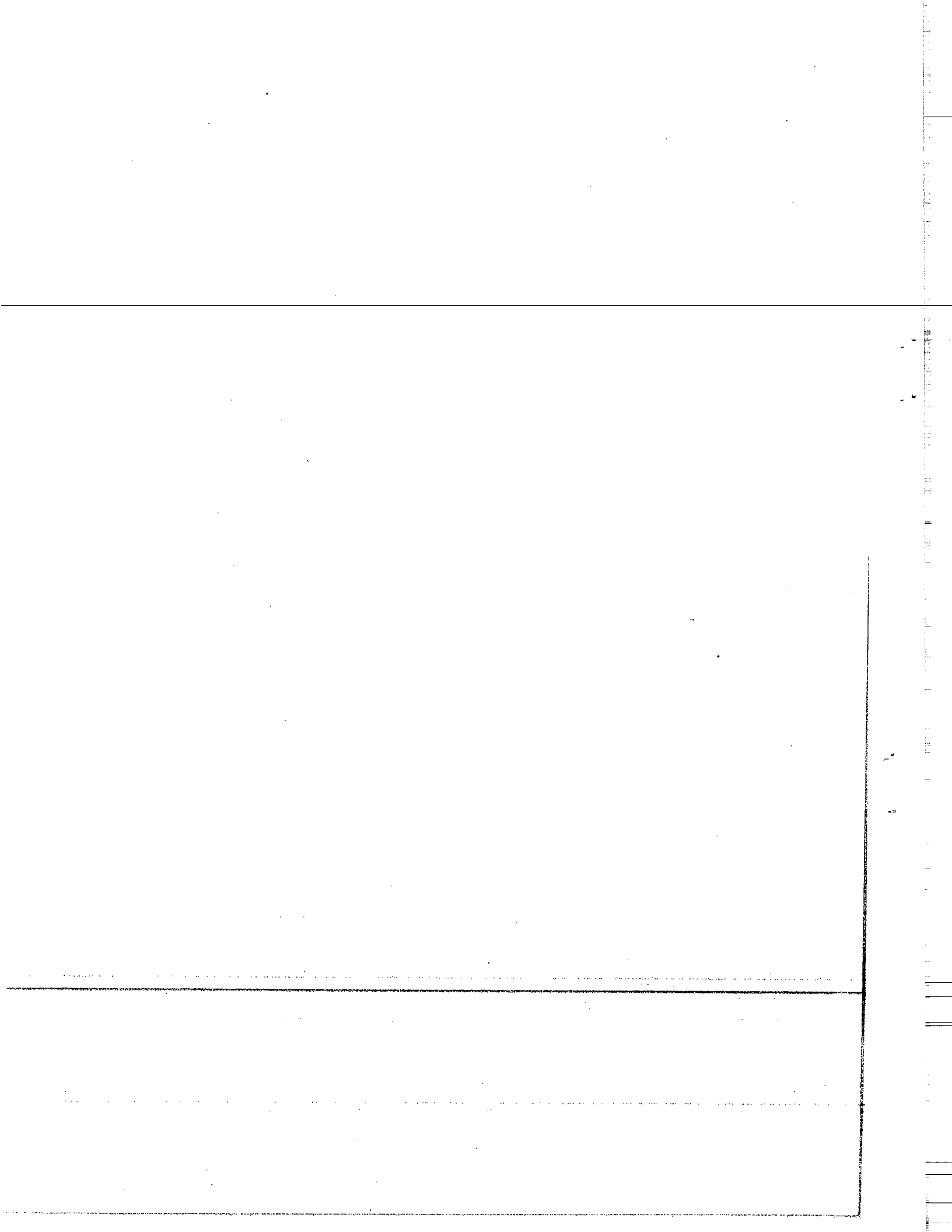


(D)

PC_2 doesnot capture much variability, likely to drop PC_2 in set D.
So in this way we try to reduce dimensions to 1.

High redundancy in dataset D.
Low redundancy in dataset A & B.

Dimension redundancy is highly successful, if there is high redundancy in dataset.



Given

x_1	y_1
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2.0	1.6
1.0	1.1
1.5	1.6
1.1	0.9

$n = 10$ samples / instances

$P = 2$ variables

Step 1
- Consider the data in a matrix format if it has not given
- Draw the scatter plot - we can identify PC1, PC2 directions here.
- find mean of x_1, y_1, to shift the centroid the centroid or origin of axes.

Step 2
- Subtract the means from the corresponding data component to re-center the dataset.
- Re-construct the scatter plot
- Note that the adjusted data set will have means zero

i.e., $\bar{X} = 0$, $\bar{Y} = 0$

center = mean matrix at zero.

Step 3 compute the sample variance - covariance matrix C.

$$C = \frac{1}{N-1} (X - 1\bar{X}')' (X - 1\bar{X}')$$

mean matrix

As $\bar{X} = 0$

* will not be there.

$$= \frac{1}{N-1} (X)' X$$

If the data is standardized, then C is the correlation matrix

$$C = \frac{1}{N-1} \; 2.2.$$

Step 4 Compute the eigen values λ_1, λ_2 and (unit or normalised) eigen vectors e_1 of C, order the correspondig pairs from the highest to the lowest eigen values.

Eigen values determines the radius of the ellipse.

Step 5 Eigen vectors & Normalise the matrix.

Step 6 Analyse the PC_1, PC_2 from eigen values & direction of eigen vectors

Step 7 calculate Transformed matrix $Y = X V$

X - original data matrix
V - transformation matrix

covariance - to converge into one line

Normalisation - to bond all the features & rotate

Given

x_1	y_1	X $x_1-\bar{x}_1$	Y $y_1-\bar{y}_1$
2.5	2.4	0.69	0.49
0.5	0.7	-1.31	-1.21
2.2	2.9	0.39	0.99
1.9	2.2	0.09	0.21
3.1	3.0	1.29	1.09
2.3	2.7	0.49	0.79
2.0	1.6	0.19	-0.31
1.0	1.1	-0.81	-0.81
1.5	1.6	-0.31	-0.31
1.1	0.9	-0.71	-1.01

$\bar{x}_1 = \frac{18.1}{10} = 1.81$ $\quad \bar{y}_1 = \frac{19.1}{10} = 1.91$

step1

$\bar{x} = 0$ $\quad \bar{y} = 0$

step2

$X = [x \; y]_{10\times 2}$

$$C = \frac{1}{N-1} X_c' X_c$$

Finding covariance matrix

step3

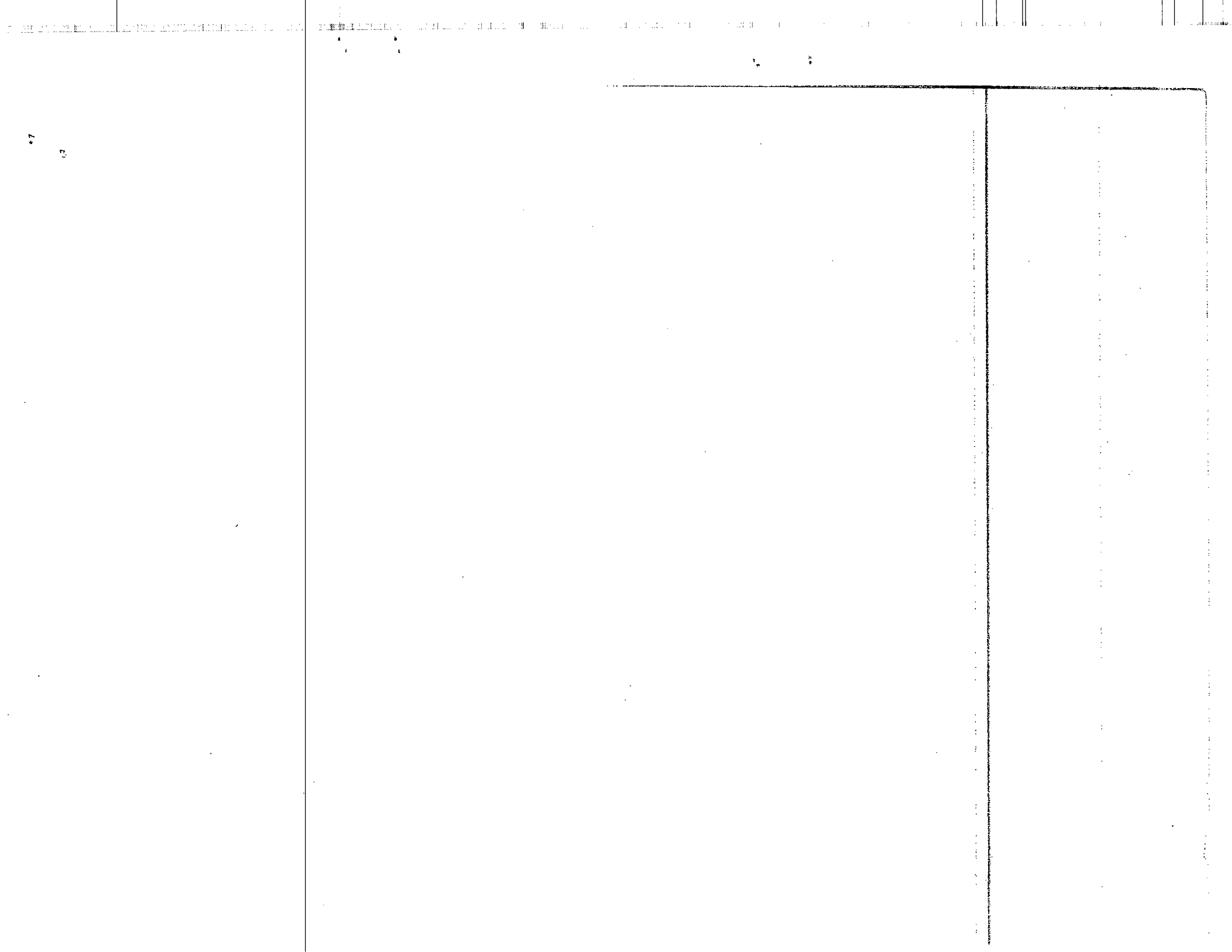
$$C = \frac{1}{10-1} \begin{bmatrix} 0.69 & -1.31 & 0.39 & 0.09 & 1.29 & 0.49 & 0.19 & -0.81 & -0.31 & -0.71 \\ 0.49 & -1.21 & 0.99 & 0.21 & 1.09 & 0.79 & -0.31 & -0.81 & -0.31 & -1.01 \end{bmatrix}$$

$$\begin{bmatrix} 0.69 & 0.49 \\ -1.31 & -1.21 \\ 0.39 & 0.99 \\ 0.09 & 0.21 \\ 1.29 & 1.09 \\ 0.49 & 0.79 \\ 0.19 & -0.31 \\ -0.81 & -0.81 \\ -0.31 & -0.31 \\ -0.71 & -1.01 \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

$[\;]_{2\times 10} \cdot [\;]_{10\times 2} = [\;]_{2\times 2}$

$$= \frac{1}{9} \begin{bmatrix} (0.69)^2+(-1.31)^2+(0.39)^2+(0.09)^2+ & (-0.69)(0.49)+(-1.31)(-1.21)+ \\ (1.29)^2+(0.49)^2+(0.19)^2+(-0.81)^2+(-0.31)^2+(-0.71)^2 & (0.39)(0.99)+(0.09)(0.21)+ \\ & (1.29)(1.09)+(0.49)(0.79)+ \\ & (0.19)(-0.31)+(-0.81)^2+ \\ & (-0.31)^2+(-0.71)(-1.01) \\ (0.49)(0.69)+(-1.21)(-1.31)+(0.99)(0.39)+ & (0.49)^2+(-1.21)^2+(0.99)^2+ \\ (0.21)(0.09)+(0.99)(0.49)+(0.79)(0.69)+ & (0.21)^2+(1.09)^2+(0.79)^2+ \\ (-0.31)(0.19)+(-0.81)(-0.81)+(-0.31)^2+ & (-0.31)^2+(-0.81)^2+(-0.31)^2+ \\ (-1.01)(-0.71) & (-1.01)^2 \end{bmatrix}$$



$= \frac{1}{9}\Big[0.2401 + 0.0361 + 0.6561 + 0.0961 + 0.5041 \quad 1.4061 + \Big]$

$=$

$$C = \begin{bmatrix} 0.6166 & 0.6154 \\ 0.6154 & 0.7166 \end{bmatrix}_{2\times 2}$$

This is the important matrix on which we want to work on. with the help of Eigen values & Eigen vectors, we will break down this matrix to understand the direction of the axis of maximum variance and also of $\perp^{ar}$ axis of maximum variance.

steps finding Eigen values & Eigen vectors

Eigen values determines the radius of the ellipse

$$\begin{vmatrix} 0.6166 - \lambda & 0.6154 \\ 0.6154 & 0.7166 - \lambda \end{vmatrix} = 0$$

$$(0.6166 - \lambda)(0.7166 - \lambda) - (0.6154)(0.6154) = 0$$

$$0.6418 - 0.6166\lambda - 0.7166\lambda + \lambda^2 - 0.3787 = 0$$

$$\lambda^2 - 1.3332\lambda + 0.0631 = 0$$

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1.3332) \pm \sqrt{(1.3332)^2 - 4 \times 0.0631}}{2 \times 1}$$

$$= \frac{1.3332 \pm \sqrt{1.774 - 0.2524}}{2} \quad \frac{1.3332 \pm \sqrt{1.5216}}{2}$$

$$= 1.3332 \pm \frac{1.2335}{2}$$

$\lambda_1 = 1.2834$

$\lambda_2 = 0.049$

Eigenvectors

steps when $\lambda_1 = 1.2841$

$$(A - \lambda I)x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.6166 - 1.2841 & 0.6154 \\ 0.6154 & 0.7166 \end{bmatrix} \Rightarrow \begin{bmatrix} -0.6675 & 0.6154 \\ 0.6154 & -0.5675 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-0.6675x + 0.6154y = 0 \quad -(1)$$

$$0.6154x - 0.5675 = 0 \quad -(2)$$

from (1) $y = \frac{0.6675}{0.6154} x = 1.084x$

$$X_1 = K_1 \begin{bmatrix} 1 \\ 1.084 \end{bmatrix} \begin{matrix} x_1 \\ y_1 \end{matrix}$$

take $K_1 = 1$

when $\lambda_2 = 0.0490$

$$\begin{bmatrix} 0.6166 - 0.0490 & 0.6154 \\ 0.6154 & 0.6166 - 0.0490 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 0.5676 & 0.6154 \\ 0.6154 & 0.6676 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$0.5676x + 0.6154y = 0 \Rightarrow y = \frac{-0.5675}{0.6154} x = -0.9221x$

$$X_2 = K_2 \begin{bmatrix} 1 \\ -0.9221 \end{bmatrix} \begin{matrix} x_2 \\ y_2 \end{matrix}$$

Eigenvector

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = \begin{bmatrix} 1 & 1.084 \\ 1 & -0.9221 \end{bmatrix}$$

Normalize the matrix

$$\begin{bmatrix} \frac{x_1}{\sqrt{x_1^2+y_1^2}} & \frac{y_1}{\sqrt{x_1^2+y_1^2}} \\ \frac{x_2}{\sqrt{x_2^2+y_2^2}} & \frac{y_2}{\sqrt{x_2^2+y_2^2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{1+(1.084)^2}} & \frac{1.084}{\sqrt{1+(1.084)^2}} \\ \frac{1}{\sqrt{1+(-0.9221)^2}} & \frac{-0.9221}{\sqrt{1+(-0.9221)^2}} \end{bmatrix}$$

$$= \begin{bmatrix} 0.678 & 0.735 \\ 0.735 & -0.678 \end{bmatrix}$$

$(1.084)^2 = 1.175$

$\sqrt{1+(1.084)^2} = \sqrt{2.175} = 1.4747$

$\frac{1}{1.4747} = 0.678$

$\frac{1.084}{1.4747} = 0.735$

$(0.9221)^2 = 0.8502$

$\sqrt{1+(0.9221)^2} = \sqrt{1.8502} = 1.3602$

$\frac{1}{1.3602} = 0.735$

$-\frac{0.9221}{1.3602} = -0.678$

	variable	EV_1	EV_2
Eigen vector $V =$	X	0.678	0.735
	Y	0.735	-0.678
Eigen values		$\lambda_1 = 1.2840$ (Bigger Eigenvalue)	$\lambda_2 = 0.0490$
% of total variance		$\frac{\lambda_1}{\lambda_1+\lambda_2}$	$\frac{\lambda_2}{\lambda_1+\lambda_2}$
		96.3%	3.7%

Bigger Eigen value (λ) is $\lambda_1 = 1.2840$

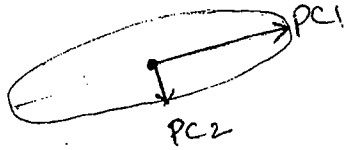
so this axis is max. variance. (PC1)

λ_2 with smaller value is PC_2

Direction — from origin

PC1 EV_1 – both are +ve: X, rightside

Y upside



PC_2 EV_2 – X +ve → rightside

Y -ve → downside

Step 1 Retain 2pcs

X - Original data matrix 10×2

V - transformation matrix 2×2

Y - transformed matrix 10×2

Transformed matrix $Y = X V$

$$= \begin{bmatrix} x_1 & x_2 \\ 2.5 & 2.4 \\ 0.5 & 0.7 \\ 2.2 & 2.9 \\ 1.9 & 2.2 \\ 3.1 & 3.0 \\ 2.3 & 2.7 \\ 2.0 & 1.6 \\ 1.0 & 1.1 \\ 1.5 & 1.6 \\ 1.1 & 0.9 \end{bmatrix}_{10 \times 2} \quad \begin{bmatrix} 0.678 & 0.735 \\ 0.735 & -0.678 \end{bmatrix}_{2 \times 2}$$

$\longrightarrow$ $y = 0.678 x_1 + 0.735 x_2$

$y^* = 0.735 x_1 - 0.678 x_2$

$$\begin{bmatrix} y & y^* \end{bmatrix} = \begin{bmatrix} 3.459 & 0.211 \\ 0.853 & -0.107 \\ 3.624 & -0.348 \\ 2.905 & -0.094 \\ 4.306 & 0.245 \\ 3.544 & -0.139 \\ 2.532 & 0.386 \\ 1.487 & -0.104 \\ 2.193 & 0.018 \\ 1.407 & 0.199 \end{bmatrix}_{10 \times 2}$$

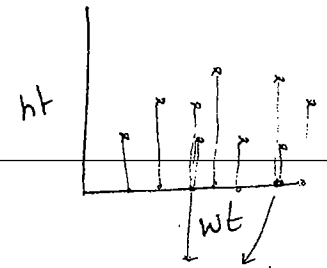
Only $x_1 = 1.284$

$$\begin{bmatrix} 3.459 \\ 0.853 \\ 3.624 \\ 2.905 \\ 4.306 \\ 3.544 \\ 2.532 \\ 1.487 \\ 2.193 \\ 1.407 \end{bmatrix}$$

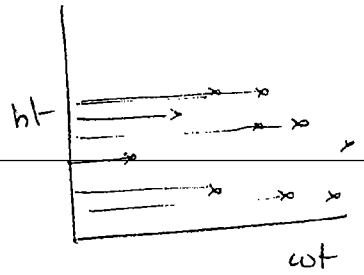
(for understanding) Linear Discriminant Analysis (LDA)

Both LDA & PCA are used for dimensionality reduction, which comes under feature extraction.

Same eg. if we consider, Prediction of gender by consider height & weight



These 2 points are overlapping if we are trying to consider only wt. by Projecting on to x-axis.
For the ~~different~~ heights, same wt. is not practical.

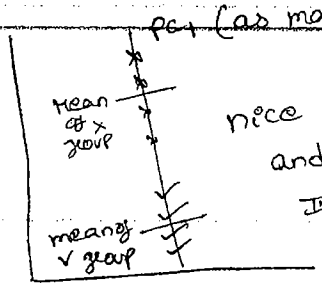


Similarly if we are projecting on to y-axis, here also same issue rises. we are loosing some info.

~~For~~ solve this issue, we are taking the help of PCA concept.
In PCA we ~~first~~ find where the more variance exists, that line we consider as PC1, ~~for~~ to PC1 as PC_2.
But if we want to classify, PCA will not help us.
As PCA is unsupervised, we consider all points as one group.
If we want to classify a particular point under which class it present, we need the dataset with labels (supervised), then we should find a line (Projection line) in such a way where
- the distance between groups is maximum (mean distance among groups)
- minimum distance with in the group.

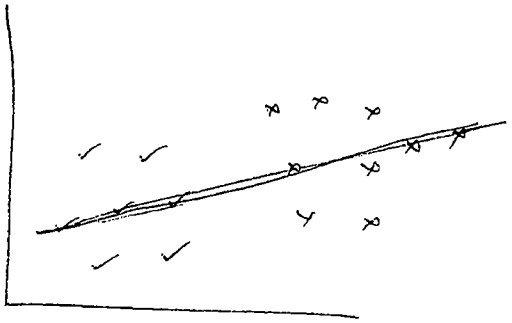
Can PCA line act like as LDA line?

(1)



nice separation there btn. groups, and max. distance btn. groups.
If a new point given, easily we can classify to which group it belong.
So in this case PCA can be LDA also.

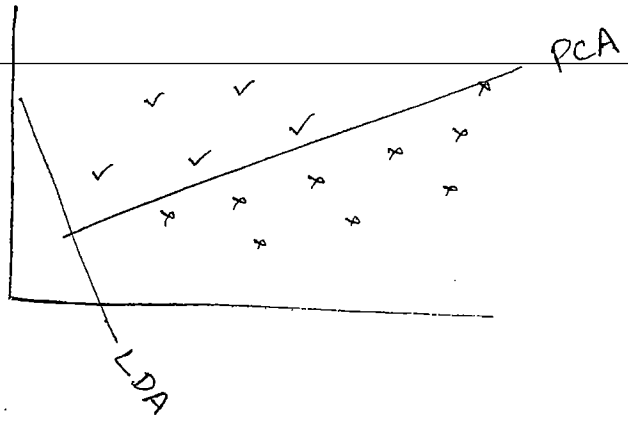
(2)

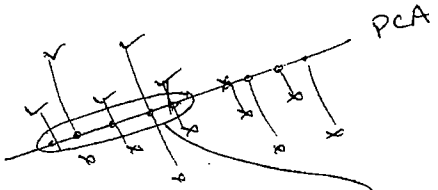


Same line can be LDA also.

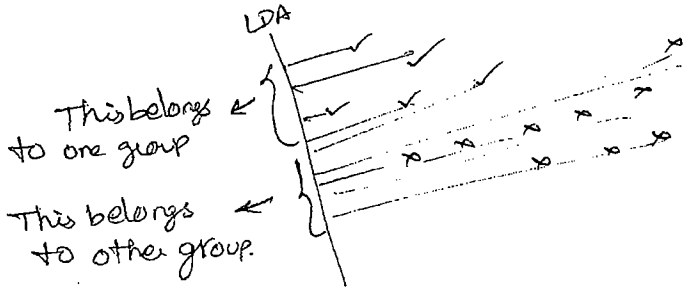
(3)

Here PCA & LDA are different

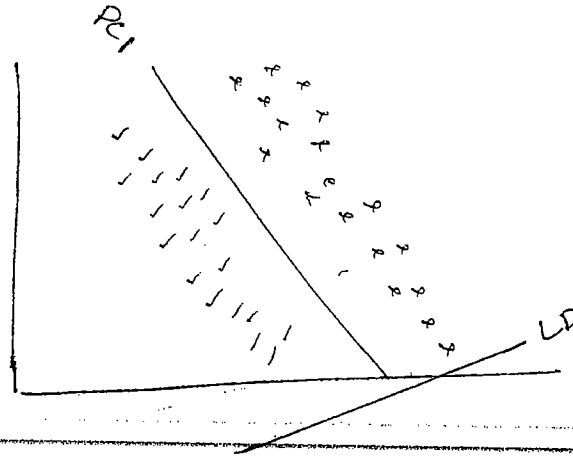




If we Project on to PCA in this region both classes are overlapping.



(4)



If we Project points on to PC1 all Points are overlap.

On to LDA, we can classify easily.

LDA

LDA is used for feature extraction (dimensionality reduction).

LDA is a supervised method.

LDA focuses on the maximizing the separability among known categories.

The aim here is to find a direction "w" such that, it separates the classes well.

→ classes

Y_i = ✓, ×

X_i = all points (all instances)

Let $\{(x_i, y_i)\ i = 1, \dots n\}$ be the labelled classes.

where x_i is the input data

y_i is the corresponding label for x_i.

Let $y_i \in \{0, 1\}$ 2-class data

Let C_0 & C_1 denote the 2 classes.

If $y_i = 0$, then $x_i \in C_0$ &

if $y_i = 1$, then $x_i \in C_1$

some instances belong to C_0 = no. of instances in $C_0 = n_0$

some instances belong to another class C_1

no. of instances in $C_1 = n_1$

Let total no. of instances be n.

n_0 & n_1 denote the number of examples of each class.

$$n = n_0 + n_1$$

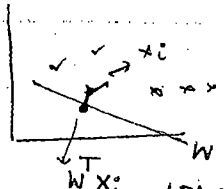
n_0 – number of examples belong to C_0

n_1 – number of examples belong to C_1

For any point w, let $Z_i = w^T x_i$

~~where Z_i is the project of x_i on direction w.~~

w^T – Projected value of x on the direction w.



Let M_0 & M_1 be the means of the 2 classes.

$$M_0 = \frac{1}{n_0} \sum_{x_i \in C_0} x_i \qquad M_1 = \frac{1}{n_0} \sum_{x_i \in C_1} x_i$$

Let m_0 & m_1 be the projected mean values

ie, $m_0 = W^T M_0 \qquad M_1 = W^T M_1$

The difference $(m_0 - m_1)$ gives the separation between samples of the 2 classes, after projecting the data on the direction W.

We want a W that maximizes $(m_0 - m_1)^2$.

As we don't know which is greater m_0 or m_1 to avoid -ve sign square it

Variance of the 2 classes in the projected data

$$S_0^2 = \sum_{x_i \in C_0} (W^T x_i - m_0)^2$$

$$S_1^2 = \sum_{x_i \in C_1} (W^T x_i - m_1)^2$$

S_0, S_1 – Scatter

Scatter is spread of data around the mean, which should be minimum.

This gives us variance of the the 2 classes in the projected data.

Objective function that needs to be maximized

$$J(W) = \frac{(m_0 - m_1)^2}{S_0^2 + S_1^2} \qquad \frac{\text{ideally large}}{\text{ideally small}}$$

$$= \frac{(W^T M_0 - W^T M_1)^2}{S_0^2 + S_1^2}$$

As distance btn. means should be large, it is in numerator. $(m_0 - m_1)^2$

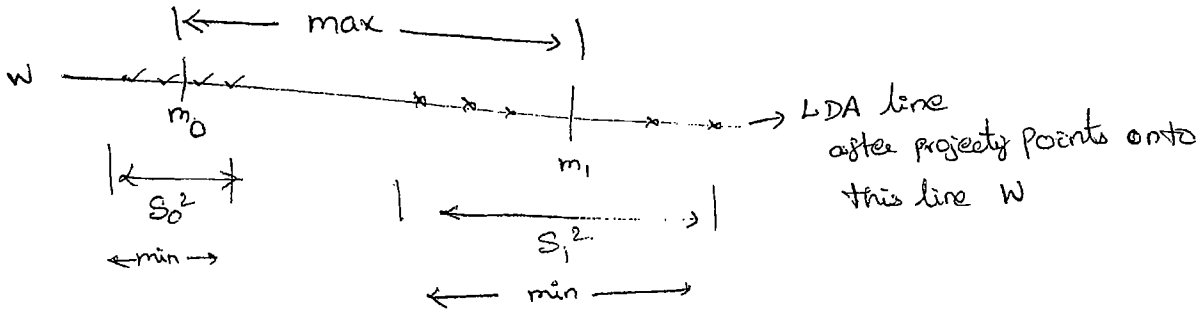
we want the distance with in group should be less, so it is in denominator. $S_0^2 + S_1^2$

How LDA creates a new axis

The new axis is created according to 2 Criteria.

1) maximize the distance between means

2) minimise the variation [In LDA, variance called as scatter & is represented by s^2] within each category



Difference between PCA | LDA

1) PCA is unsupervised (does not know anything about class labels)

   LDA is supervised.

2) PCA finds direction that maximises the variance of the data.
   most variation in the data is PC_1
   second variation is PC_2, which is ⊥ar to PC_1

   LDA finds the direction that maximises the difference between 2 classes.
   most variation between the categories $m_0 - m_1$
   less with in categories.
   Looks for a dimension that makes it easy to separate classes.

3) LDA discovers relationship between dependent & independent variables.

   PCA discovers relationship between independent variables.

4) LDA used for Prediction of classes.

5) PCA is used for reducing variables based on collinearity of independent variables.

   LDA is used for variable reduction based on strength of relationship between independent and dependent variables.

Decision Tree

Decision tree is one of the learning algorithm.

Decision Tree is **a supervised** learning algorithm.

Decision Tree builds classification or regression model in the form of a tree structure.

Means it is used for both classification and regression problems, so called

CART (ClassificationAnd Regression Tree).

But mostly used for classification.

Regression trees are used when dependent variables is continuous.

Classification trees are used when dependent variables is categorical.

Decision trees are built by splitting the training set into distinct nodes, where one node contains all or one category of the data. These categories can be called as subsets.

A decision tree is a tree like structure in which internal node represents test on an attribute, each branch represent outcome of test and each leaf node represent class label (decision taken after computing all attributes). A path from root to leaf represents classification rules.

Decision Tree Learning (DTL) is a form of inductive learning task, meaning it has the following objective: use a training set of examples to create a hypothesis that makes general conclusions.

Decision Tree is a classifier in form of a tree- which is a tree structure classifier.

The representation of decision tree is a non-linear function.

Tree models are popular in ML

Eg. Sensing device for Xbox game console, has decision tree classifier as its heart.

Eg. Based on length, width of a petal, we can classify to which species a flower fall into

Some Terminologies in Decision Tree

Node: A test for the values (data) of a certain attribute. Goal of the node is to split the dataset or an attribute
Root node: The beginning node that contains the entire dataset.
Decision node: specifies a choice or test, test may have more than one result, test usually will be done on the value of a feature or the attribute of an instance. It has 2 or more branches
Leaf node: Terminal node that predicts the outcome
It indicates the classification of an example or the value of an example
Attribute: a variable that we take into account in making a decision
Target attribute: the attribute that we want to take on a certain value, we'll decide based on it

Applications

- For an insurance company whether customer will pay premium or not (Yes/No)
- For a titanic survival, depends on Gender & age, what are the chances of survival
- To determine whether a person is male or female depending on the height & weight
- Price of a home based on number of rooms, floor size etc (regression problem)

Advantages

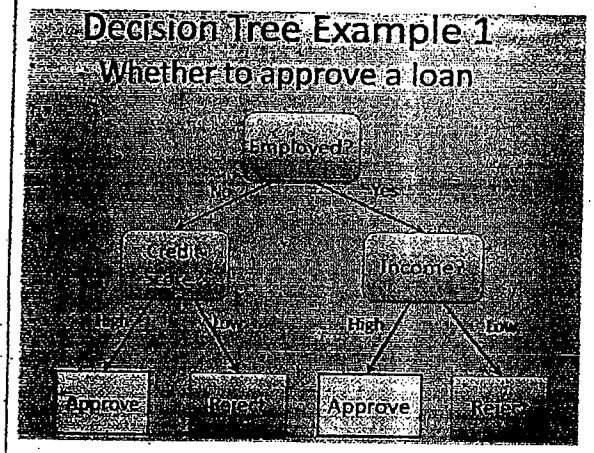
- Simple to understand, interpret, visualize
- Implicitly perform variable screening or feature selection
- Can handle both numeric or categorical data
- Little effort needed for the data preparation
- Non-linear relationship[between parameters does not affect the performance

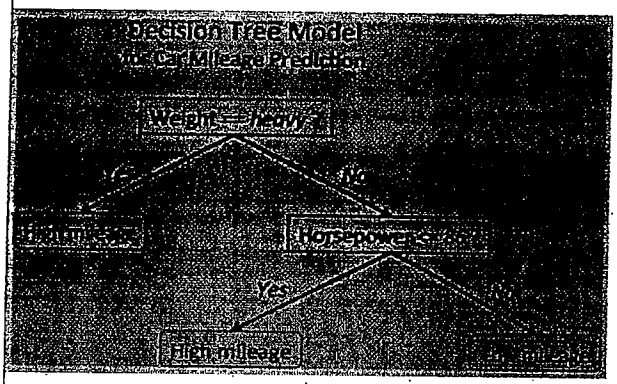
Disadvantages

- If we create over complex trees that do not generalize the data well, we call it as overfitting
- Decision trees can become unstable because small variations (variance) in the data might result in a completely different

Eg Decision tree to approve a loan

From the training set we can come up with a decision tree like this and for the new example we can come to a decision whether loan can be given or not.





Decision / Splitting Criteria

When to stop

- All examples have the same value
- There are no more attributes
- There are no more examples

At every step we have to decide whether we have to stop growing the tree at that node or proceed.
If we want to continue, we have to decide which attribute to continue.
We can stop the tree when that leaf has specific value
Or grow when that node has more than one possible value.

Which is the better attribute to split.

Measures for splitting

- Gini Index
- Chi- Square
- Information Gain
- Reduction in Variance

One best method is based on Entropy and Information Gain.

Basic Algorithm

Tree is constructed in a top-down recursive divide and conquer manner.

At start, all the training examples are at the root.

Attributes are categorical.

Input data is portioned recursively based on selected attributes.

Test attributes at each node are selected on the basis of a heuristic or statistical measure (eg information gain)

Entropy is a measure of disorder in a particular system.
Or Entropy is a measure of purity.
Or amount of information disorder.
Or amount of randomness in the data.
Or measure of impurity in a bunch of examples.

~~Entropy controls how and where the Decision Tree split the data.~~

Entropy is a measure of uncertainty, or measure of information content.

In a particular node, if all set of examples are +ve or –ve class, then that set is called as homogenous set of examples and entropy is low. Pure set of examples will have entropy as zero.

If we have 2 classes and half examples belong to one class and another half belong to another class, then entropy is high.

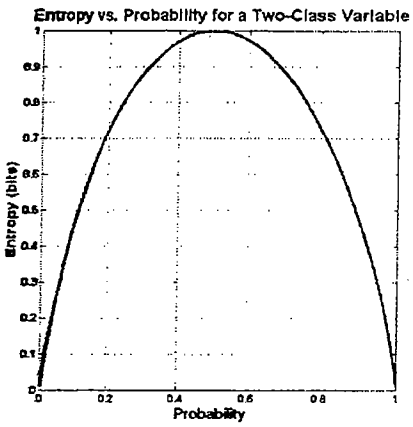
Leaf node is the one ideally where all examples belong to same class, means entropy is low.

We want to reach the leaf node quickly where entropy is zero.

Information gain will decided which is the most important attribute.

If all the examples are equally divided among the classes, and we are asked to predict by giving a random example, we have to guess randomly as we have no information, then high uncertainty, then entropy is high, then information gain is low.

If examples belong to same class, then no uncertainty, entropy is 0 (i.e., disorder in a system is 0), then information is very high.



Entropy is maximised when probability is 50%.

When probability is 0% or 100%, entropy is 0.

By adding information we can reduce entropy and gain certainty.

Information collected that can increase the level of certainty in a particular prediction.

As gain increases, Entropy decreases.

It is a measure of the decrease in disorder achieved by portioning the original dataset.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D_1	Rainy	Hot	High	weak	NO
D_2	Rainy	Hot	High	strong	NO
D_3	Overcast	Hot	High	weak	Yes
D_4	Sunny	Mild	High	weak	Yes
D_5	Sunny	Cool	Normal	weak	Yes
D_6	Sunny	Cool	Normal	strong	NO
D_7	Overcast	Cool	Normal	strong	Yes
D_8	Rainy	Mild	High	weak	NO
D_9	Rainy	Cool	Normal	weak	Yes
D_{10}	Sunny	Mild	Normal	weak	Yes
D_{11}	Rainy	Mild	Normal	strong	Yes
D_{12}	Overcast	Mild	High	Strong	Yes
D_{13}	Overcast	Hot	Normal	weak	Yes
D_{14}	Sunny	Mild	High	Strong	NO

$$\text{Entropy}(S) = \sum_{i=1}^{C} -P_i \log_2^{P_i}$$

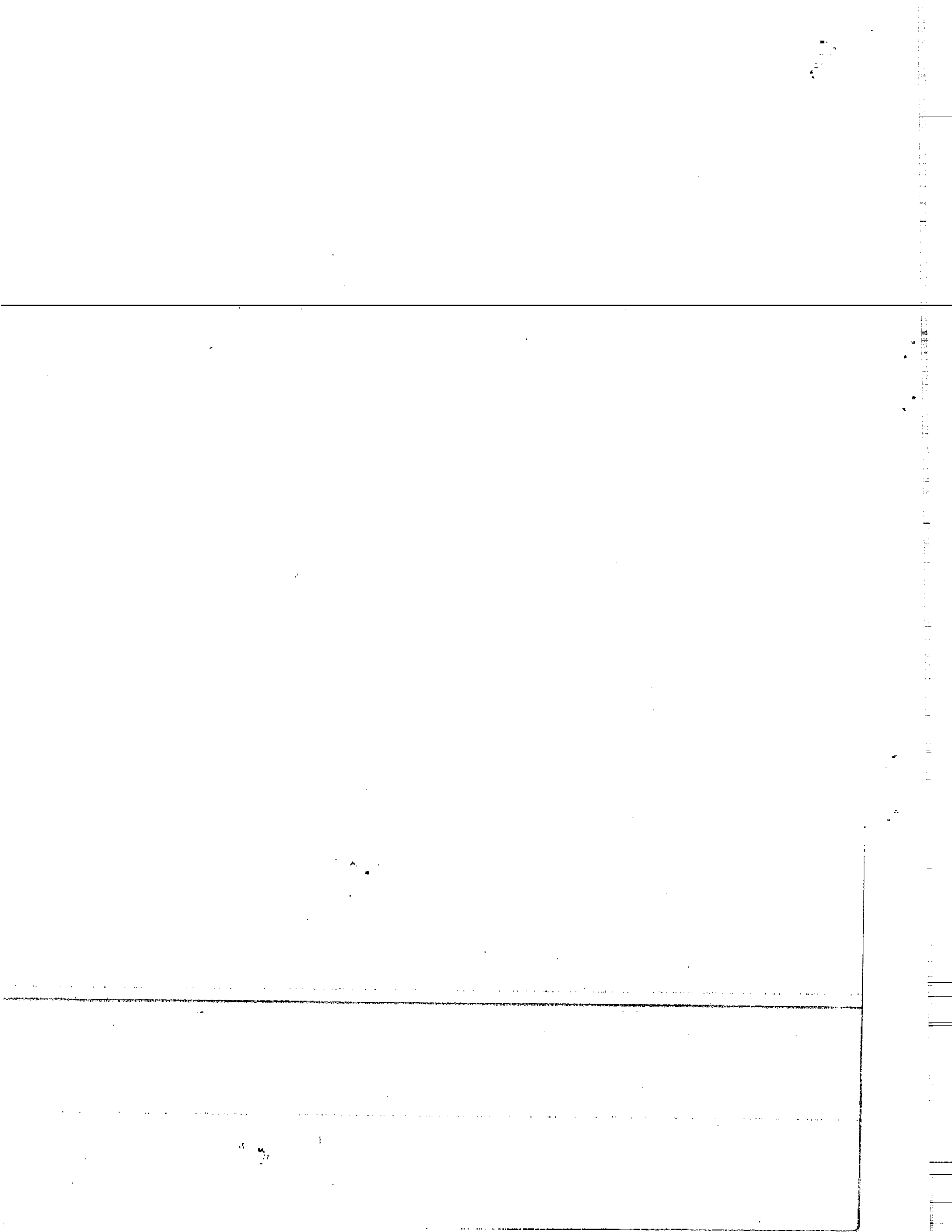
P_i = Probability
C = no. of classification

$$\text{Gain}(S,A) = \text{Entropy}(S) - \text{Entropy}(S,A)$$

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

values(A) is the set of all possible values for attribute A

S_v is the subset of S for which attribute A has value v.



① $E(S) = E(\text{PlayTennis}) = E[9+, 5-]$

+ YES
− NO

$= -\frac{9}{14}\log_2^{(9/14)} + \left(-\frac{5}{14}\log_2^{(5/14)}\right)$

S is a collection of 14 examples with 9 Yes & 5 NO sample

$= (-0.64 \times -0.64) + (-0.36 \times 1.47)$

$= 0.52 + 0.4 = 0.94$

② Now find which is root node

As we have 4 other attributes find Gain for those attributes

$\text{Gain}(\text{PlayTennis}, \text{outlook}) = E(\text{PlayTennis}) - \sum \frac{|S_v|}{|S|} \text{Entropy}(S_v)$

$\forall \in (\text{Sunny}, \text{overcast}, \text{Rainy})$

Sunny → 3+, 2− = 5

overcast → 4+, 0− = 4

Rainy → 2+, 3− = 5

14

$= E(PT) - \left[\frac{5}{14} E(\text{Sunny}) + \frac{4}{14} E(\text{overcast}) + \frac{5}{14} E(\text{Rainy})\right]$

$= E(PT) - \left[\frac{5}{14} E(3+,2-) + \frac{4}{14} E(4+,0-) + \frac{5}{14} E(2+,3-)\right]$

$= 0.94 - \left[\frac{5}{14}\left(-\frac{3}{5}\log_2^{3/5} - \frac{2}{5}\log_2^{2/5}\right) + \frac{4}{14}\left(-\frac{4}{4}\log_0^{4/4} - \frac{0}{4}\log_0^{0/4}\right) + \frac{5}{14}\left(-\frac{2}{5}\log_0^{2/5} - \frac{3}{5}\log_0^{3/5}\right)\right]$

$= 0.94 - \left[\frac{5}{14}\left(-\frac{3}{5}\times(-0.737) - \frac{2}{5}\times(-1.322)\right) + \frac{4}{14}\left((-1\times 0) - 0\right) + \frac{5}{14}\left(\frac{-2}{5}(1.322) - \frac{3}{5}(-0.737)\right)\right]$

$= 0.94 - \left[\frac{5}{14}\times 0.971 + \frac{4}{14}\times 0 + \frac{5}{14}\times 0.971\right]$

$= 0.94 - 0.693$

$= \underline{0.247}$

Gain(PlayTennis, Temperature)

Hot — 2+, 2− = 4
Mild — 4+, 2− = 6
Cold — 3+, 1− = 4

$$= E(PT) - \sum_{v \in (Hot, Mild, Cold)} \frac{|S_v|}{S} E(S_v)$$

$$= E(PT) - \left[\frac{4}{14} E(hot) + \frac{6}{14} E(mild) + \frac{4}{14} E(cold)\right]$$

$$= E(PT) - \left[\frac{4}{14} E(2,2) + \frac{6}{14} E(4,2) + \frac{4}{14} E(3,1)\right]$$

$$= E(PT) - \left[\frac{4}{14}\left(-\frac{2}{4}\log_2^{2/4} - \frac{2}{4}\log_2^{2/4}\right) + \frac{6}{14}\left(-\frac{4}{6}\log_2^{4/6} - \frac{2}{6}\log_2^{2/6}\right) + \frac{4}{14}\left(-\frac{3}{4}\log_2^{3/4} - \frac{1}{4}\log_2^{1/4}\right)\right]$$

$$= 0.94 - \left[\frac{4}{14}(0.5+0.5) + \frac{6}{14}\left(\frac{4}{6}\times 0.578 + \frac{2}{6}\times 1.56\right) + \frac{4}{14}\left(\frac{3}{4}\times 0.415 + \frac{1}{4}\times 2\right)\right]$$

$$= 0.94 - \left[\frac{4}{14} + \frac{6}{14}\times 0.905 + \frac{4}{14}\times 0.81\right]$$

$$= 0.94 - [0.286 + 0.388 + 0.232]$$

$$= 0.94 - 0.906$$

$$= 0.029$$

Gain (PlayTennis, Humidity)

High — 3+, 4- = 7
Normal — 6+, 1- = 7
14

$$= E(PT) - \left[\sum_{V \in (High, Normal)} \frac{|S_v|}{8} E(S_v) \right]$$

$$= E(PT) - \left[\frac{7}{14} E(high) + \frac{7}{14} E(Normal) \right]$$

$$= E(PT) - \left[\frac{7}{14} E(3+,4-) + \frac{7}{14} E(6+,1-) \right]$$

$$= E(PT) - \left[\frac{7}{14} \left(-\frac{3}{7} \log_2^{3/7} - \frac{4}{7} \log_2^{4/7} \right) + \frac{7}{14} \left(-\frac{6}{7} \log_2^{6/7} - \frac{1}{7} \log_2^{1/7} \right) \right]$$

$$= 0.94 - \left[\frac{7}{14} \left(\frac{3}{7} \times 1.22 + \frac{4}{7} \times 0.806 \right) + \frac{7}{14} \left(\frac{6}{7} \times 0.223 + \frac{1}{7} \times 2.806 \right) \right]$$

$$= 0.94 - (0.492 + 0.296)$$

$$= 0.94 - 0.79$$

$$= \underline{0.15}$$

weak — 6+, 2- = 8
strong — 3+, 3- = 6

Gain (PlayTennis, Windy)

$$= E(PT) - \sum_{V \in (weak, strong)} \frac{|S_v|}{8} E(S_v)$$

$$= E(PT) - \left[\frac{8}{14} \overset{E(weak)}{E(6+,2-)} + \frac{6}{14} \overset{E(strong}{E(3+,3-)} \right]$$

$$= E(PT) - \left[\frac{8}{14} \left(-\frac{6}{8} \log_2^{6/8} - \frac{2}{8} \log_2^{2/8} \right) + \frac{6}{14} \left(-\frac{3}{6} \log_2^{3/6} - \frac{3}{6} \log_2^{3/6} \right) \right]$$

$$= 0.94 - \left[\frac{8}{14} \times \frac{1}{8} \left[6 \times 0.415 + 2 \times 2 \right] + \frac{6}{14} \right]$$

$$= 0.94 - \left[\frac{1}{14} \times 6.49 + \frac{6}{14} \right]$$

$$= 0.94 - 0.892$$

$$= \underline{0.048}$$

DT Pb 1.3
DT (3)

Gain	outlook	Temperature	Humidity	windy
	0.247	0.029	0.152	0.048

Largest Information gain will become as ~~decision~~ root node.

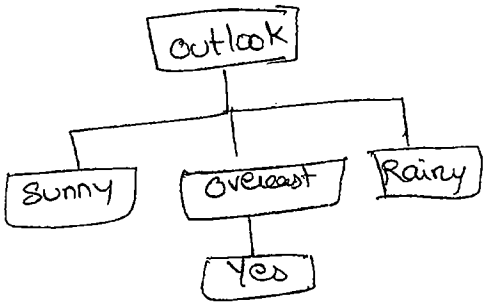
So here outlook is root node.

But in outlook we have 3 options sunny, overcast, Rainy

Check among these 3 any one is having all instances with same class (Yes/NO).

for overcast → all are yes.

E(overcast) = 0.



So to find out next decision node under sunny

consider subset of which are Sunny

	Temp	Humidity	wind	PlayTennis
D_1	Mild	High	Weak	Yes
D_5	Cool	Normal	Weak	Yes
D_6	Cool	Normal	Strong	NO
D_{10}	Mild	Normal	Weak	Yes
D_{14}	Mild	High	Strong	NO

$$E(\text{PlayTennis}) = E(S) = E(3+, 2-)$$

$$= -\frac{3}{5}\log_2^{3/5} \quad \frac{2}{5}\log_2^{2/5} = 0.971$$

$$G(PT, Temp) = E(PT) - E(S/A)$$

$$= E(PT) - \sum_{v \in (Mild, cool)} \frac{|S_v|}{S} E(S_v)$$

Mild – 2+, 1– = 3
Cool = 1+, 1– = 2
5

$$= E(PT) - \left[\frac{3}{5} E(Mild) + \frac{2}{5} E(cool)\right]$$

$$= E(PT) - \left[\frac{3}{5} E\left(\frac{2}{3}, \frac{1}{3}\right) + \frac{2}{5} E\left(\frac{1}{2}, \frac{1}{2}\right)\right]$$

$$G(PT, Humidity) =$$

$$E(PT) - \left[\frac{2}{5} E\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{3}{5} E\left(\frac{2}{3}, \frac{1}{3}\right)\right]$$

High → 1+, 1– = 2
Normal → 2+, 1– = 3
5

$$G(PT, Windy) =$$

$$= E(PT) - \left[\frac{3}{5} E\left(\frac{3}{3}, \frac{0}{3}\right) + \frac{2}{5} E\left(\frac{0}{2}, \frac{1}{2}\right)\right]$$

weak = 3+, 0–
Strong = 0+, 2–

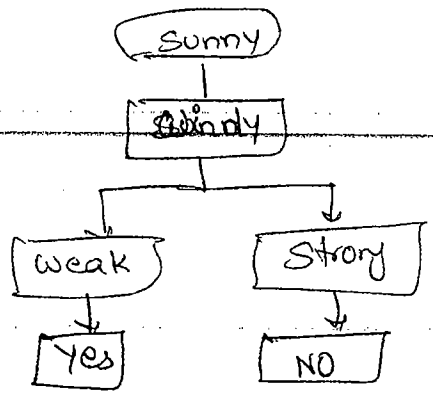
$$= E(PT) - \left[\frac{3}{5} E(1,0) + \frac{2}{5} E(0, \frac{1}{2})\right]$$

↓

From this I can say that Gain is high as both terms are '0'.

So next decision node is Windy.

In windy → If it is weak → all yes
If it is Strong → all NO



D1	Hot	High	Weak	No
D2	Hot	High	Strong	NO
D8	Mild	High	weak	NO
D9	cool	Normal	weak	Yes
D11	Mild	Normal	Strong	Yes

$E(\text{PlayTennis}) = E[2+, 3-]$

$= -\frac{2}{5}\log_2^{2/5} - \frac{3}{5}\log_2^{3/5} = 0.971$

G(PT, Temp)

Hot 0+, 2- = 2
cool 1+, 0- = 1
Mild 1+, 1- = $\frac{2}{5}$

G (PT, Humidity)

High 0+, 3-
Normal 2+, 0-

G (PT, windy)

weak 2+, 1-
Strong 1+, 1-

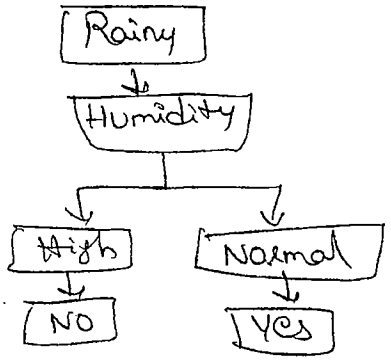
→ looking into this

for humidity → for both high & normal one value is 0,

So Gain will be high for this

In Humidity also,

for high → all NO
Normal → all yes



If you want to calculate & judge → next page

G(PT, Temperature)

$$= E(PT) - \left[\frac{2}{5}E(\text{Hot}) + \frac{1}{5}E(\text{cool}) + \frac{2}{5}E(\text{Mild})\right]$$

$$= E(PT) - \left[\frac{2}{5}E(0+,2-) + \frac{1}{5}E(1+,0-) + \frac{2}{5}E(1+,1-)\right]$$

$$= E(PT) - \left[0 + 0 + \frac{2}{5}\left(-\frac{1}{2}\log_2^{1/2} - \frac{1}{2}\log_2^{1/2}\right)\right]$$

=

G(PT, Humidity)

$$= E(PT) - \left[\frac{3}{5}E(\text{High}) + \frac{2}{5}E(\text{Normal})\right]$$

$$= E(PT) - \left[\frac{3}{5}E(0+,3-) + \frac{2}{5}E(2+,0-)\right]$$

$$= E(PT) - [0+0] =$$

$$= 0.971 - 0 = 0.971$$

As in Temp, windy a term is there to subtract from E(PT)

and in humidity no term.

Gain of humidity is more.

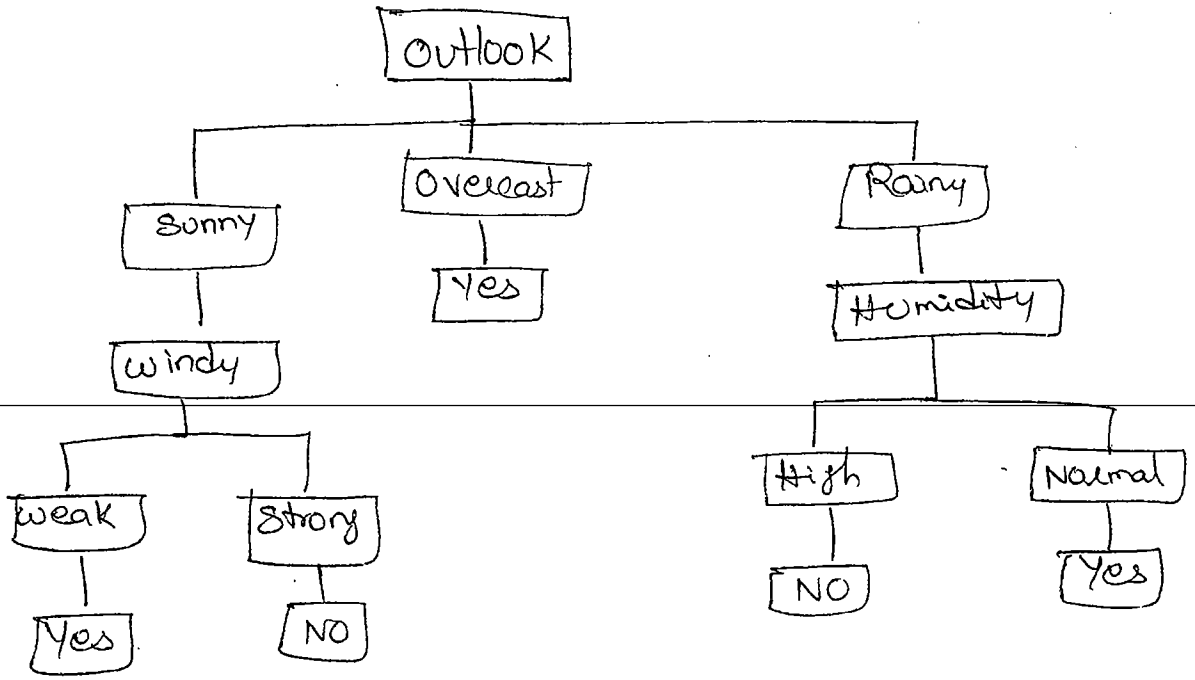
G(PT, windy)

$$= E(PT) - \left[\frac{3}{5}E(\text{weak}) + \frac{2}{5}E(\text{strong})\right]$$

$$= E(PT) - \left[\frac{3}{5}E(2+,1-) + \frac{2}{5}E(1+,1-)\right]$$

$$= E(PT) - \left[\frac{3}{5}\left(-\frac{2}{3}\log_2^{2/3} - \frac{1}{3}\log_2^{1/3}\right) + \frac{2}{5}\left(-\frac{1}{2}\log_2^{1/2} - \frac{1}{2}\log_2^{1/2}\right)\right]$$

=

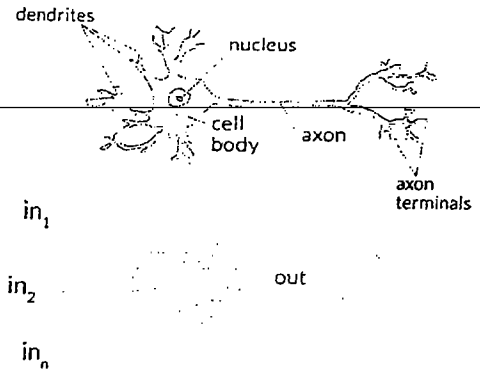

Outlook
Sunny
Overcast
Rainy
Yes
Windy
Humidity
weak
Strong
High
Normal
Yes
NO
NO
Yes

UNIT-4

MLA.

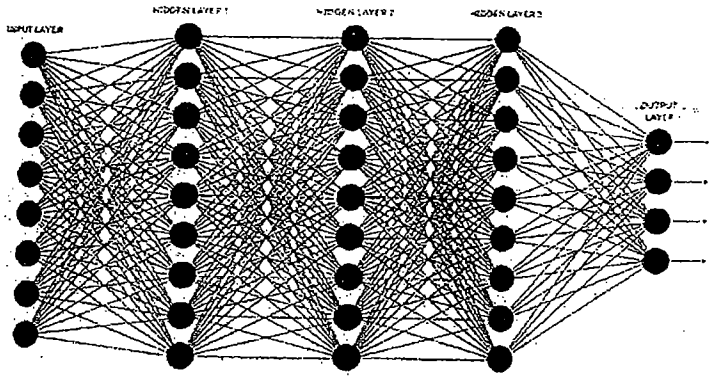
Brief overview of neural networks

. A neural network is a very powerful machine learning mechanism which basically mimics how a human brain learns. The brain receives the stimulus from the outside world, does the processing on the input, and then generates the output.



As the task gets complicated multiple neurons form a complex network, passing information among themselves.

Using a artificial neural network, we try to mimic a similar behavior. The network you see below is a neural network made of interconnected neurons.



The black circles in the picture above are neurons. Each neuron is characterized by its weight, bias and activation function. The input is fed to the input layer. The neurons do a

linear transformation on the input by the weights and biases. The non linear transformation is done by the activation function. The information moves from the input layer to the hidden layers. The hidden layers would do the processing and send the final output to the output layer. This is the forward movement of information known as the forward propagation. But what if the output generated is far away from the expected value? In a neural network, we would update the weights and biases of the neurons on the basis of the error. This process is known as back-propagation. Once the entire data has gone through this process, the final weights and biases are used for predictions.

What is an Activation Function?

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

The activation function is the non linear transformation that we do over the input signal. This transformed output is then sen to the next layer of neurons as input.

Can we do without an activation function?

When we do not have the activation function the weights and bias would simply do a linear transformation. A linear equation is simple to solve but is limited in its capacity to solve complex problems. A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. We would want our neural networks to work on complicated tasks like language translations and image classifications. Linear transformations would never be able to perform such tasks.

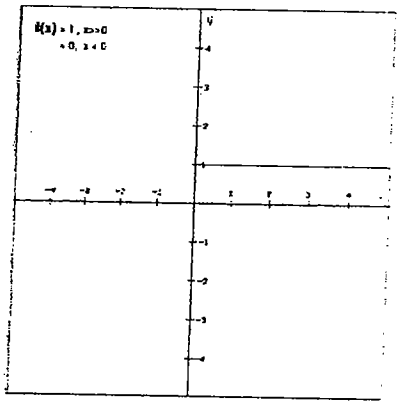
Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable non linear function, this would not be possible.

Popular types of activation functions and when to use them

Binary Step Function

The first thing that comes to our mind when we have an activation function would be a threshold based classifier i.e. whether or not the neuron should be activated. If the value Y is above a given threshold value then activate the neuron else leave it deactivated.

It is defined as f(x) =1, x>0, else 0

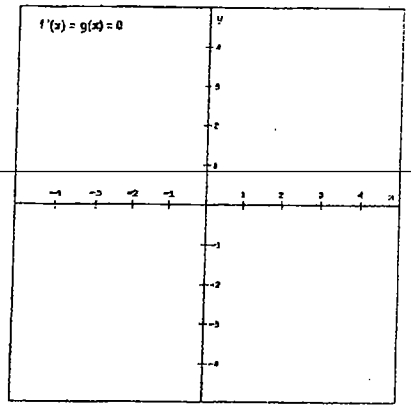


The binary function is extremely simple. It can be used while creating a binary classifier. When we simply need to say yes or no for a single class, step function would be the best choice, as it would either activate the neuron or leave it to zero.

The function is more theoretical than practical since in most cases we would be classifying the data into multiple classes than just a single class. The step function would not be able to do that.

Moreover, the gradient of the step function is zero. This makes the step function not so useful since during back-propagation when the gradients of the activation functions are sent for error calculations to improve and optimize the results. The gradient of the step function reduces it all to zero and improvement of the models doesn't really happen.

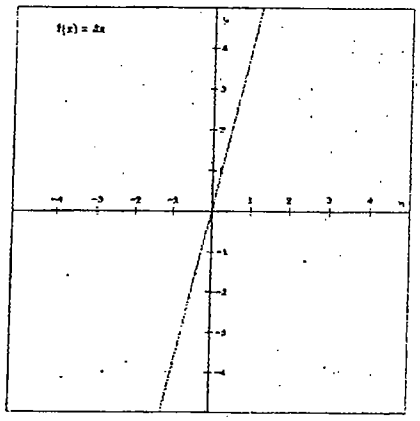
f '(x) = 0, for all x



Linear Function

We saw the problem with the step function, the gradient being zero, it was impossible to update gradient during the backpropagation. Instead of a simple step function, we can try using a linear function. We can define the function as-

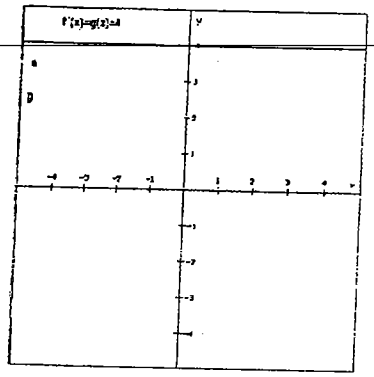
$$f(x)=ax$$



We have taken a as 4 in the figure above. Here the activation is proportional to the input. The input x, will be transformed to ax. This can be applied to various neurons and

multiple neurons can be activated at the same time. Now, when we have multiple classes, we can choose the one which has the maximum value. But we still have an issue here. Let's look at the derivative of this function.

```
f'(x) = a
```



The derivative of a linear function is constant i.e. it does not depend upon the input value x.

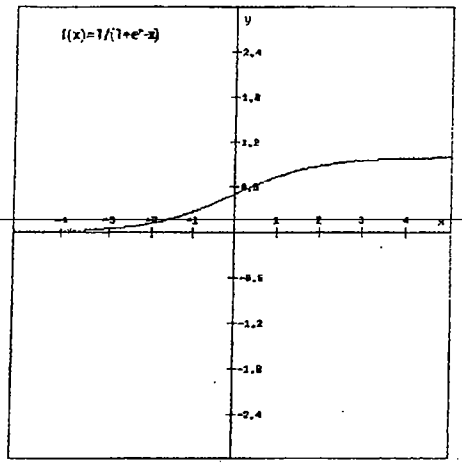
This means that every time we do a back propagation, the gradient would be the same. And this is a big problem, we are not really improving the error since the gradient is pretty much the same. And not just that suppose we are trying to perform a complicated task for which we need multiple layers in our network. Now if each layer has a linear transformation, no matter how many layers we have the final output is nothing but a linear transformation of the input. Hence, linear function might be ideal for simple tasks where interpretability is highly desired.

Sigmoid

Sigmoid is a widely used activation function. It is of the form-

```
f(x)=1/(1+e^-x)
```

Let's plot this function and take a look of it.



This is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear. This is an incredibly cool feature of the sigmoid function. This essentially means that when I have multiple neurons having sigmoid function as their activation function – the output is non linear as well. The function ranges from 0-1 having an S shape. Let's take a look at the shape of the curve. The gradient is very high between the values of -3 and 3 but gets much flatter in other regions. How is this of any use?

This means that in this range small changes in x would also bring about large changes in the value of Y. So the function essentially tries to push the Y values towards the extremes. This is a very desirable quality when we're trying to classify the values to a particular class.

It's smooth and is dependent on x. This means that during backpropagation we can easily use this function. The error can be backpropagated and the weights can be accordingly updated.

Sigmoids are widely used even today but we still have a problems that we need to address. As we saw previously – the function is pretty flat beyond the +3 and -3 region. This means that once the function falls in that region the gradients become very small.

This means that the gradient is approaching to zero and the network is not really learning.

Another problem that the sigmoid function suffers is that the values only range from 0 to 1. This meas that the sigmoid function is not symmetric around the origin and the values received are all positive. So not all times would we desire the values going to the next neuron to be all of the same sign. This can be addressed by scaling the sigmoid function. That's exactly what happens in the tanh function. let's read on.

Tanh

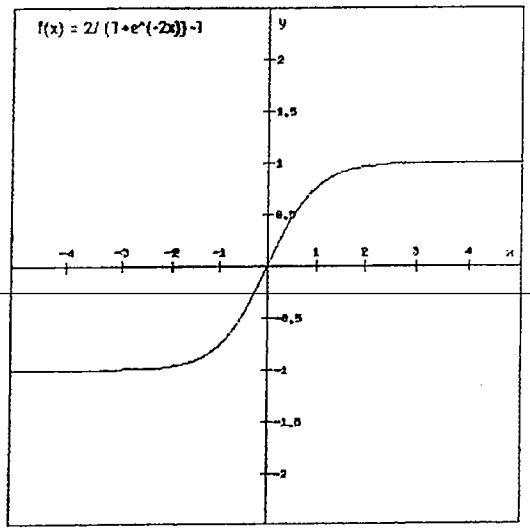
The tanh function is very similar to the sigmoid function. It is actually just a scaled version of the sigmoid function.

```
tanh(x)=2sigmoid(2x)-1
```

It can be directly written as –

```
tanh(x)=2/(1+e^(-2x)) -1
```

Tanh works similar to the sigmoid function but is symmetric over the origin. it ranges from -1 to 1.



It basically solves our problem of the values all being of the same sign. All other properties are the same as that of the sigmoid function. It is continuous and differentiable at all points. The function as you can see is non linear so we can easily backpropagate the errors.

The gradient of the tanh function is steeper as compared to the sigmoid function. Our choice of using sigmoid or tanh would basically depend on the requirement of gradient in the problem statement. But similar to the sigmoid function we still have the vanishing gradient problem. The graph of the tanh function is flat and the gradients are very low.

ReLU

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as-

```
f(x)=max(0,x)
```

It can be graphically represented as-

often used, such as the decision tree learning tasks discussed in Chapter 3. In these cases ANN and decision tree learning often produce results of comparable accuracy. See Shavlik et al. (1991) and Weiss and Kapouleas (1989) for experimental comparisons of decision tree and ANN learning. The BACKPROPAGATION algorithm is the most commonly used ANN learning technique. It is appropriate for problems with the following characteristics:

- *Instances are represented by many attribute-value pairs.* The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values in the ALVINN example. These input attributes may be highly correlated or independent of one another. Input values can be any real values.
- *The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.* For example, in the ALVINN system the output is a vector of 30 attributes, each corresponding to a recommendation regarding the steering direction. The value of each output is some real number between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.
- *The training examples may contain errors.* ANN learning methods are quite robust to noise in the training data.
- *Long training times are acceptable.* Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.
- *Fast evaluation of the learned target function may be required.* Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast. For example, ALVINN applies its neural network several times per second to continually update its steering command as the vehicle drives forward.
- *The ability of humans to understand the learned target function is not important.* The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules.

The rest of this chapter is organized as follows: We first consider several alternative designs for the primitive units that make up artificial neural networks (perceptrons, linear units, and sigmoid units), along with learning algorithms for training single units. We then present the BACKPROPAGATION algorithm for training

multilayer networks of such units and consider several general issues such as the representational capabilities of ANNs, nature of the hypothesis space search, overfitting problems, and alternatives to the BACKPROPAGATION algorithm. A detailed example is also presented applying BACKPROPAGATION to face recognition, and directions are provided for the reader to obtain the data and code to experiment further with this application.

4.4 PERCEPTRONS

One type of ANN system is based on a unit called a *perceptron*, illustrated in Figure 4.2. A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs x_1 through x_n, the output $o(x_1, \ldots, x_n)$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

where each w_i is a real-valued constant, or *weight*, that determines the contribution of input x_i to the perceptron output. Notice the quantity $(-w_0)$ is a threshold that the weighted combination of inputs $w_1 x_1 + \cdots + w_n x_n$ must surpass in order for the perceptron to output a 1.

To simplify notation, we imagine an additional constant input $x_0 = 1$, allowing us to write the above inequality as $\sum_{i=0}^{n} w_i x_i > 0$, or in vector form as $\vec{w} \cdot \vec{x} > 0$. For brevity, we will sometimes write the perceptron function as

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

Learning a perceptron involves choosing values for the weights $w_0, \ldots, w_n$. Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors.

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

4.4.1 Representational Power of Perceptrons

We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure 4.3. The equation for this decision hyperplane is $\vec{w} \cdot \vec{x} = 0$. Of course, some sets of positive and negative examples cannot be separated by any hyperplane. Those that can be separated are called *linearly separable* sets of examples.

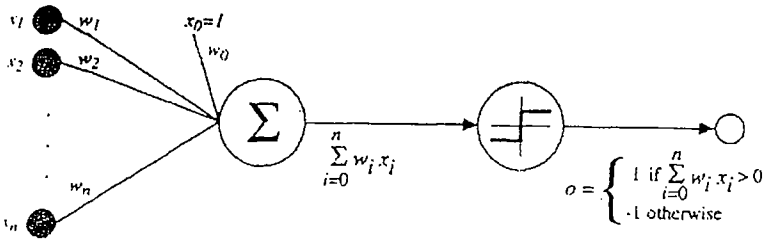


FIGURE 4.2
A perceptron.

A single perceptron can be used to represent many boolean functions. For

A single perceptron can be used to represent many boolean functions. For example, if we assume boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights $w_0 = -.8$, and $w_1 = w_2 = .5$. This perceptron can be made to represent the OR function instead by altering the threshold to $w_0 = -.3$. In fact, AND and OR can be viewed as special cases of m-of-n functions: that is, functions where at least m of the n inputs to the perceptron must be true. The OR function corresponds to $m = 1$ and the AND function to $m = n$. Any m-of-n function is easily represented using a perceptron by setting all input weights to the same value (e.g., 0.5) and then setting the threshold w_0 accordingly.

Perceptrons can represent all of the primitive boolean functions AND, OR, NAND ($\neg$ AND), and NOR ($\neg$ OR). Unfortunately, however, some boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if $x_1 \neq x_2$. Note the set of linearly nonseparable training examples shown in Figure 4.3(b) corresponds to this XOR function.

The ability of perceptrons to represent AND, OR, NAND, and NOR is important because *every* boolean function can be represented by some network of interconnected units based on these primitives. In fact, every boolean function can be represented by some network of perceptrons only two levels deep, in which

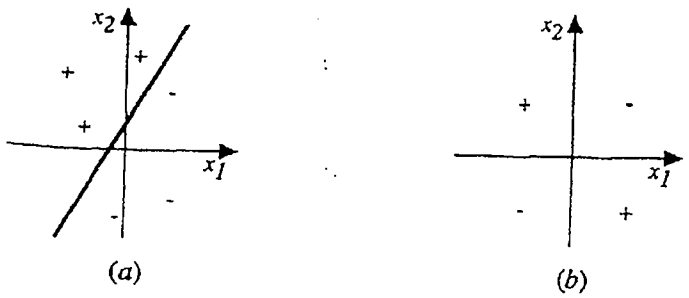


FIGURE 4.3
The decision surface represented by a two-input perceptron. (a) A set of training examples and the decision surface of a perceptron that classifies them correctly. (b) A set of training examples that is not linearly separable (i.e., that cannot be correctly classified by any straight line). x_1 and x_2 are the perceptron inputs. Positive examples are indicated by "+", negative by "−".

the inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage. One way is to represent the boolean function in disjunctive normal form (i.e., as the disjunction (OR) of a set of conjunctions (ANDs) of the inputs and their negations). Note that the input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight.

Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

4.4.2 The Perceptron Training Rule

Although we are interested in learning networks of many interconnected units, let us begin by understanding how to learn the weights for a single perceptron. Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct $+1$ output for each of the given training examples

training a single unit, gradient descent can be used to attempt to find a hypothesis to minimize E.

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $(\vec{x}, \vec{t})$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji} \tag{T4.5}$$

TABLE 4.2
The stochastic gradient descent version of the BACKPROPAGATION algorithm for feedforward networks containing two layers of sigmoid units.

One major difference in the case of multilayer networks is that the error surface can have multiple local minima, in contrast to the single-minimum parabolic error surface shown in Figure 4.4. Unfortunately, this means that gradient descent is guaranteed only to converge toward some local minimum, and not necessarily the global minimum error. Despite this obstacle, in practice BACKPROPAGATION has been found to produce excellent results in many real-world applications.

The BACKPROPAGATION algorithm is presented in Table 4.2. The algorithm as described here applies to layered feedforward networks containing two layers of sigmoid units, with units at each layer connected to all units from the preceding layer. This is the incremental, or stochastic, gradient descent version of BACKPROPAGATION. The notation used here is the same as that used in earlier sections

reward or penalty to indicate the desirability of the resulting state. For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states. The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward. This chapter focuses on an algorithm called Q learning that can acquire optimal control strategies from delayed rewards, even when the agent has no prior knowledge of the effects of its actions on the environment. Reinforcement learning algorithms are related to dynamic programming algorithms frequently used to solve optimization problems.

13.1 INTRODUCTION

Consider building a learning robot. The robot, or *agent*, has a set of sensors to observe the *state* of its environment, and a set of *actions* it can perform to alter this state. For example, a mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn." Its task is to learn a control strategy, or *policy*, for choosing actions that achieve its goals. For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

10 (

This chapter is concerned with how such agents can learn successful control policies by experimenting in their environment. We assume that the goals of the agent can be defined by a *reward* function that assigns a numerical value—an immediate payoff—to each distinct action the agent may take from each distinct state. For example, the goal of docking to the battery charger can be captured by assigning a positive reward (e.g., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition. This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot. The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy. The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent. This general setting for robot learning is summarized in Figure 13.1.

As is apparent from Figure 13.1, the problem of learning a control policy to maximize cumulative reward is very general and covers many problems beyond robot learning tasks. In general the problem is one of learning to control sequential processes. This includes, for example, manufacturing optimization problems in which a sequence of manufacturing actions must be chosen, and the reward to be maximized is the value of the goods produced minus the costs involved. It includes sequential scheduling problems such as choosing which taxis to send for passengers in a large city, where the reward to be maximized is a function of the wait time of the passengers and the total fuel costs of the taxi fleet. In general, we are interested in any type of agent that must learn to choose actions that alter the state of its environment and where a cumulative reward function is used to define the quality of any given action sequence. Within this class of problems we will consider specific settings, including settings in which the actions have deterministic or nondeterministic outcomes, and settings in which the agent

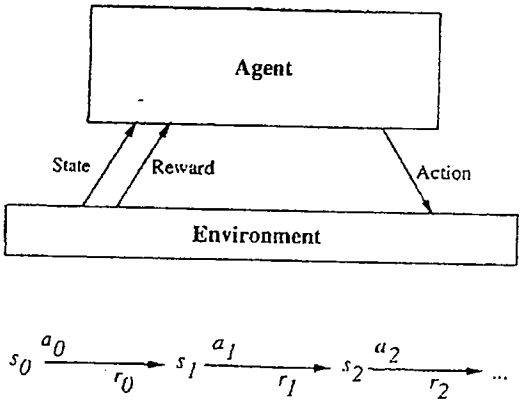


FIGURE 13.1
An agent interacting with its environment. The agent exists in an environment described by some set of possible states S. It can perform any of a set of possible actions A. Each time it performs an action a_t in some state s_t the agent receives a real-valued reward r_t that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure. The agent's task is to learn a control policy, $\pi : S \rightarrow A$, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

has or does not have prior knowledge about the effects of its actions on the environment.

Note we have touched on the problem of learning to control sequential processes earlier in this book. In Section 11.4 we discussed explanation-based learning of rules to control search during problem solving. There the problem is for the agent to choose among alternative actions at each step in its search for some goal state. The techniques discussed here differ from those of Section 11.4, in that here we consider problems where the actions may have nondeterministic outcomes and where the learner lacks a domain theory that describes the outcomes of its actions. In Chapter 1 we discussed the problem of learning to choose actions while playing the game of checkers. There we sketched the design of a learning method very similar to those discussed in this chapter. In fact, one highly successful application of the reinforcement learning algorithms of this chapter is to a similar game-playing problem. Tesauro (1995) describes the TD-GAMMON program, which has used reinforcement learning to become a world-class backgammon player. This program, after training on 1.5 million self-generated games, is now considered nearly equal to the best human players in the world and has played competitively against top-ranked players in international backgammon tournaments.

The problem of learning a control policy to choose actions is similar in some respects to the function approximation problems discussed in other chapters. The target function to be learned in this case is a control policy, $\pi : S \rightarrow A$, that outputs an appropriate action a from the set A, given the current state s from the set S. However, this reinforcement learning problem differs from other function approximation tasks in several important respects.

- *Delayed reward.* The task of the agent is to learn a target function π that maps from the current state s to the optimal action $a = \pi(s)$. In earlier chapters we have always assumed that when learning some target function such as π, each training example would be a pair of the form $(s, \pi(s))$. In reinforcement learning, however, training information is not available in this form. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of *temporal credit assignment*: determining which of the actions in its sequence are to be credited with

- *Delayed reward.* The task of the agent is to learn a target function π that maps from the current state s to the optimal action $a = \pi(s)$. In earlier chapters we have always assumed that when learning some target function such as π, each training example would be a pair of the form $\langle s, \pi(s) \rangle$. In reinforcement learning, however, training information is not available in this form. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of *temporal credit assignment*: determining which of the actions in its sequence are to be credited with producing the eventual rewards.
- *Exploration.* In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a tradeoff in choosing whether to favor *exploration* of unknown states and actions (to gather new information), or *exploitation* of states and actions that it has already learned will yield high reward (to maximize its cumulative reward).
- *Partially observable states.* Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. For example, a robot with a forward-pointing camera cannot see what is

behind it. In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.
- *Life-long learning.* Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

13.2 THE LEARNING TASK

In this section we formulate the problem of learning sequential control strategies more precisely. Note there are many ways to do so. For example, we might assume the agent's actions are deterministic or that they are nondeterministic. We might assume that the agent can predict the next state that will result from each action, or that it cannot. We might assume that the agent is trained by an expert who shows it examples of optimal action sequences, or that it must train itself by performing actions of its own choice. Here we define one quite general formulation of the problem, based on Markov decision processes. This formulation of the problem follows the problem illustrated in Figure 13.1.

In a Markov decision process (MDP) the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform. At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it. The environment responds by giving the agent a reward $r_t =$

13.3.2 An Algorithm for Learning Q

Learning the Q function corresponds to learning the optimal policy. How can Q be learned?

The key problem is finding a reliable way to estimate training values for Q, given only a sequence of immediate rewards r spread out over time. This can be accomplished through iterative approximation. To see how, notice the close relationship between Q and V^*,

$$V^*(s) = \max_{a'} Q(s, a')$$

which allows rewriting Equation (13.4) as

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \tag{13.6}$$

[redacted]

This recursive definition of Q provides the basis for algorithms that iteratively approximate Q (Watkins 1989). To describe the algorithm, we will use the symbol $\hat{Q}$ to refer to the learner's estimate, or hypothesis, of the actual Q function. In this algorithm the learner represents its hypothesis $\hat{Q}$ by a large table with a separate entry for each state-action pair. The table entry for the pair $\langle s, a \rangle$ stores the value for $\hat{Q}(s, a)$—the learner's current hypothesis about the actual but unknown value $Q(s, a)$. The table can be initially filled with random values (though it is easier to understand the algorithm if one assumes initial values of zero). The agent repeatedly observes its current state s, chooses some action a, executes this action, then observes the resulting reward $r = r(s, a)$ and the new state $s' = \delta(s, a)$. It then updates the table entry for $\hat{Q}(s, a)$ following each such transition, according to the rule:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') \tag{13.7}$$

Note this training rule uses the agent's current $\hat{Q}$ values for the new state s' to refine its estimate of $\hat{Q}(s, a)$ for the previous state s. This training rule is motivated by Equation (13.6), although the training rule concerns the agent's approximation $\hat{Q}$, whereas Equation (13.6) applies to the actual Q function. Note although Equation (13.6) describes Q in terms of the functions $\delta(s, a)$ and $r(s, a)$, the agent does not need to know these general functions to apply the training rule of Equation (13.7). Instead it executes the action in its environment and then observes the resulting new state s' and reward r. Thus, it can be viewed as sampling these functions at the current values of s and a.

The above Q learning algorithm for deterministic Markov decision processes is described more precisely in Table 13.1. Using this algorithm the agent's estimate $\hat{Q}$ converges in the limit to the actual Q function, provided the system can be modeled as a deterministic Markov decision process, the reward function r is

---

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

TABLE 13.1
Q learning algorithm, assuming deterministic rewards and actions. The discount factor γ may be any constant such that $0 \leq \gamma < 1$.

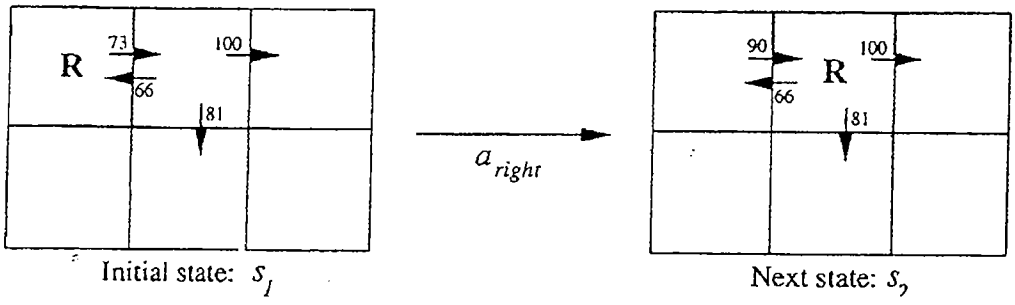
bounded, and actions are chosen so that every state-action pair is visited infinitely often.

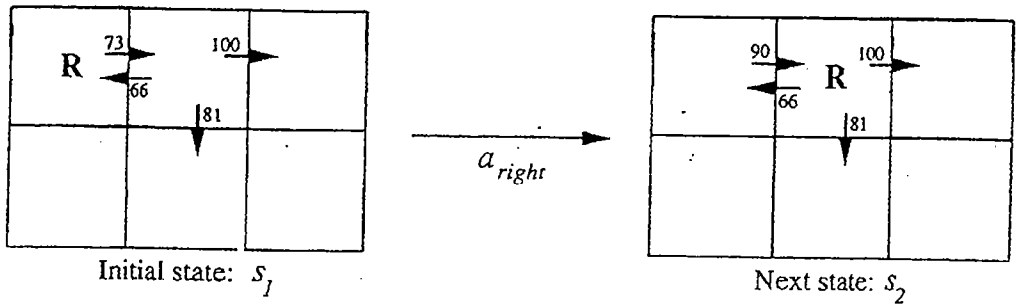
13.3.3 An Illustrative Example

To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to $\hat{Q}$ shown in Figure 13.3. In this example, the agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition. It then applies the training rule of Equation (13.7) to refine its estimate $\hat{Q}$ for the state-action transition it just executed. According to the training rule, the new $\hat{Q}$ estimate for this transition is the sum of the received reward (zero) and the highest $\hat{Q}$ value associated with the resulting state (100), discounted by γ (.9).

Each time the agent moves forward from an old state to a new one, Q learning propagates $\hat{Q}$ estimates *backward* from the new state to the old. At the same time, the immediate reward received by the agent for the transition is used to augment these propagated values of $\hat{Q}$.

Consider applying this algorithm to the grid world and reward function shown in Figure 13.2, for which the reward is zero everywhere, except when entering the goal state. Since this world contains an absorbing goal state, we will assume that training consists of a series of *episodes*. During each episode, the agent begins at some randomly chosen state and is allowed to execute actions until it reaches the absorbing goal state. When it does, the episode ends and





$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$
$$\leftarrow 90$$

FIGURE 13.3
The update to $\hat{Q}$ after executing a single action. The diagram on the left shows the initial state s_1 of the robot (R) and several relevant $\hat{Q}$ values in its initial hypothesis. For example, the value $\hat{Q}(s_1, a_{right}) = 72.9$, where a_{right} refers to the action that moves R to its right. When the robot executes the action a_{right}, it receives immediate reward $r = 0$ and transitions to state s_2. It then updates its estimate $\hat{Q}(s_1, a_{right})$ based on its $\hat{Q}$ estimates for the new state s_2. Here $\gamma = 0.9$.

the agent is transported to a new, randomly chosen, initial state for the next episode.

How will the values of $\hat{Q}$ evolve as the Q learning algorithm is applied in this case? With all the $\hat{Q}$ values initialized to zero, the agent will make no changes to any $\hat{Q}$ table entry until it happens to reach the goal state and receive a nonzero reward. This will result in refining the $\hat{Q}$ value for the single transition leading into the goal state. On the next episode, if the agent passes through this state adjacent to the goal state, its nonzero $\hat{Q}$ value will allow refining the value for some transition two steps from the goal, and so on. Given a sufficient number of training episodes, the information will propagate from the transitions with nonzero reward back through the entire state-action space available to the agent, resulting eventually in a $\hat{Q}$ table containing the Q values shown in Figure 13.2.

In the next section we prove that under certain assumptions the Q learning algorithm of Table 13.1 will converge to the correct Q function. First consider two general properties of this Q learning algorithm that hold for any deterministic MDP in which the rewards are non-negative, assuming we initialize all $\hat{Q}$ values to zero. The first property is that under these conditions the $\hat{Q}$ values never decrease during training. More formally, let $\hat{Q}_n(s, a)$ denote the learned $\hat{Q}(s, a)$ value after the nth iteration of the training procedure (i.e., after the nth state-action transition taken by the agent). Then

$$(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$$

A second general property that holds under these same conditions is that throughout the training process every $\hat{Q}$ value will remain in the interval between zero and its true Q value.

$$(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$