DIGITAL NOTES
ON
Machine Learning
(R20D5803)

M.Tech., II YEAR – I SEM
(2021-2022)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SYLLABUS

II Year M. Tech. CSE – I Sem **L/T/P/ C**

3 / - / - 3

(R20D5803) Machine Learning

Objectives:

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

UNIT - I

Introduction Well-posed learning problems, designing a learning system Perspectives and issues in machine learning Concept learning and the general to specific ordering Introduction,A concept learning task, concept learning as search, Find-S: Finding a Maximally Specific Hypothesis, Version Spaces and the Candidate Elimination algorithm, Remarks on Version Spaces and Candidate Elimination, Inductive Bias. Decision Tree Learning-Introduction, Decision Tree Representation, Appropriate Problems for Decision Tree Learning, The Basic Decision Tree Learning Algorithm Hypothesis Space Search in Decision Tree Learning, Inductive Bias in Decision Tree Learning, Issues in Decision Tree Learning.

UNIT - II

Artificial Neural Networks -Introduction, Neural Network Representation, Appropriate Problems for Neural Network Learning, Perceptions, Multilayer Networks and the Back propagation Algorithm. Discussion on the Back Propagation Algorithm, An illustrative Example: Face Recognition

UNIT - III

Bayesian learning-Introduction, Byes Theorem, Bayes Theorem and Concept Learning Maximum Likelihood and Least Squared Error Hypotheses, Maximum Likelihood Hypotheses for Predicting Probabilities, Minimum Description Length Principle, Bayes Optimal Classifier, Gibs Algorithm, Naïve Bayes Classifier, An Example: Learning to Classify Text, Bayesian Belief Networks, EM Algorithm.

Instance-Based Learning-Introduction, k-Nearest Neighbor Learning, Locally Weighted Regression, Radial Basis Functions, Case-Based Reasoning, Remarks on Lazy and Eager Learning.

UNIT -IV

Pattern Comparison Techniques-Temporal patterns, Dynamic Time Warping Methods,Clustering, Introduction to clustering, K-means clustering, K-Mode Clustering. Codebook Generation, Vector Quantization.

UNIT - V

Genetic Algorithms: Different search methods for induction - Explanation-based Learning: using prior knowledge to reduce sample complexity. Dimensionality reduction: feature selection, principal component analysis, linear discriminate analysis, factor analysis, independent component analysis, multidimensional scaling, and manifold learning.

Textbooks:

1. Machine Learning – Tom M. Mitchell, -MGH
2. Fundamentals of Speech Recognition By Lawrence Rabiner and Biing – Hwang Juang .Ethem Alpaydin, "Introduction to Machine Learning", MIT Press, Prentice Hall of India, 3 rd Edition2014.
3. Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar " Foundations of Machine Learning",MIT Press,2012

References:

1. Machine Learning : An Algorithmic Perspective, Stephen Marsland, Taylor & Francis .

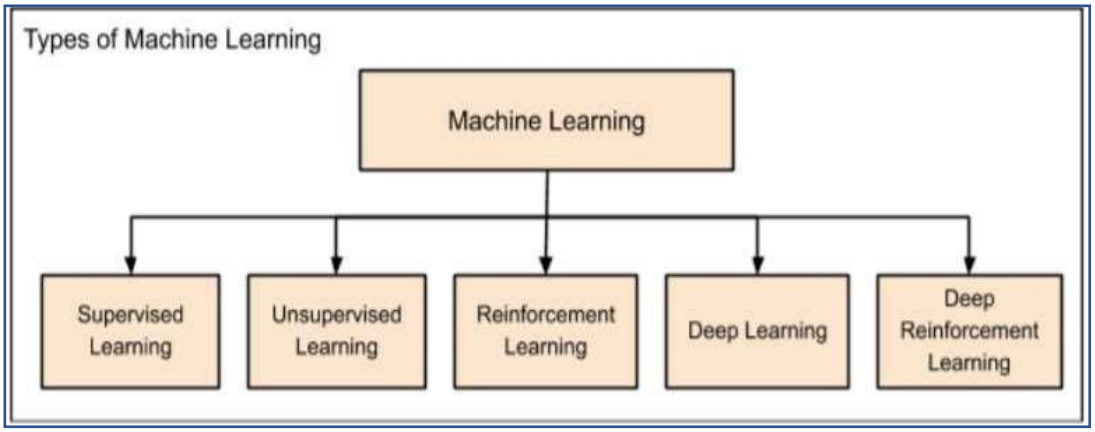
INDEX

UNIT-I

Machine Learning

is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: ***The ability to learn***. Machine learning is actively being used today, perhaps in many more places than one would expect.

Machine Learning is broadly categorized under the following headings:



Machine learning evolved from left to right as shown in the above diagram.

• Initially, researchers started out with Supervised Learning. This is the case of housing price prediction discussed earlier

. • This was followed by unsupervised learning, where the machine is made to learn on its own without any supervision.

• Scientists discovered further that it may be a good idea to reward the machine when it does the job the expected way and there came the Reinforcement Learning.

• Very soon, the data that is available these days has become so humongous that the conventional techniques developed so far failed to analyse the big data and provide us the predictions.

- Thus, came the deep learning where the human brain is simulated in the Artificial Neural Networks (ANN) created in our binary computers.
- The machine now learns on its own using the high computing power and huge memory resources that are available today.
- It is now observed that Deep Learning has solved many of the previously unsolvable problems.
- The technique is now further advanced by giving incentives to Deep Learning networks as awards and there finally comes Deep Reinforcement Learning.

Let us now study each of these categories in more details

Supervised Learning:

Supervised learning is analogous to training a child to walk. You will hold the child's hand, show him how to take his foot forward, walk yourself for a demonstration and so on, until the child learns to walk on his own.

Regression:

Similarly, in the case of supervised learning, you give concrete known examples to the computer. You say that for given feature value x1 the output is y1, for x2 it is y2, for x3 it is y3, and so on. Based on this data, you let the computer figure out an empirical relationship between x and y. Once the machine is trained in this way with a sufficient number of data points, now you would ask the machine to predict Y for a given X. Assuming that you know the real value of Y for this given X, you will be able to deduce whether the machine's prediction is correct. Thus, you will test whether the machine has learned by using the known test data. Once you are satisfied that the machine is able to do the predictions with a desired level of accuracy (say 80 to 90%) you can stop further training the machine. Now, you can safely use the machine to do the predictions on unknown data points, or ask the machine to predict Y for a given X for which you do not know the real value of Y. This training comes under the regression that we talked about earlier.

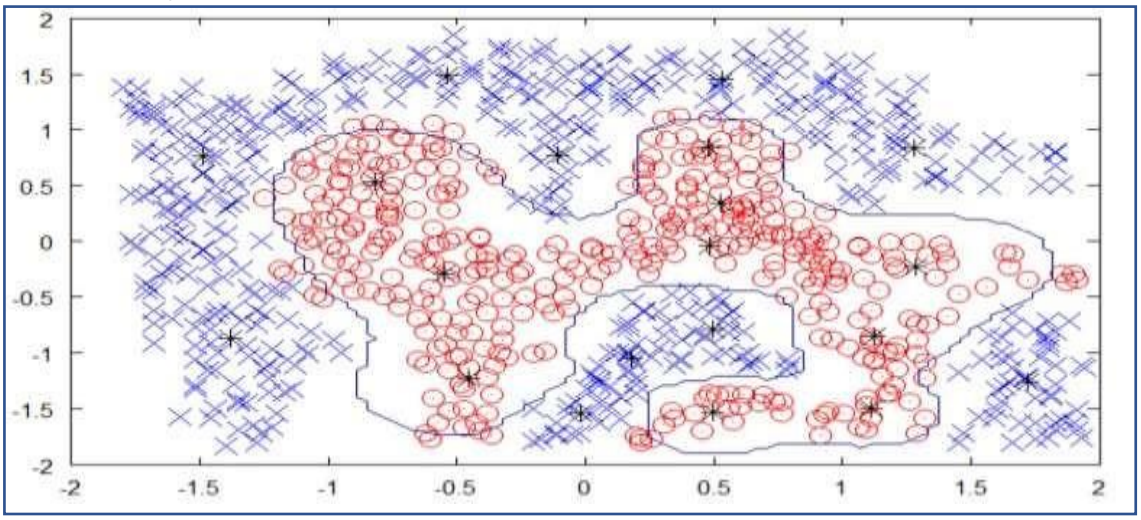
Classification:

You may also use machine learning techniques for classification problems. In classification problems, you classify objects of similar nature into a single group. For example, in a set of 100 students say, you may like to group them into three groups based on their heights - short, medium and long. Measuring the height of each student, you will place them in a proper group. Now, when a new student comes in, you will put him in an appropriate group by measuring his height. By following the principles in regression training, you will train the machine to classify a student based on his feature – the height. When the machine learns how the groups are formed, it will be able to classify any unknown new student correctly. Once again, you would use the test data to verify that the machine has learned your technique of classification before putting the developed model in production. Supervised Learning is where the AI really began its journey. This technique was applied successfully in several cases. You have used this model while doing the hand-written recognition on your machine. Several algorithms have been developed for supervised learning. You will learn about them in the following chapters.

Unsupervised Learning:

In unsupervised learning, we do not specify a target variable to the machine, rather we ask machine "What can you tell me about X?". More specifically, we may ask questions such as given a huge data set X, "What are the five best groups we can make out of X?" or "What features occur together most frequently in X?". To arrive at the answers to such questions, you can understand that the number of data points that the machine would require to deduce a strategy would be very large. In case of supervised learning, the machine can be trained with even about few thousands of data points. However, in case of unsupervised learning, the number of data points that is reasonably accepted for learning starts in a few millions. These days, the data is generally abundantly available. The data ideally requires curating. However, the amount of data that is continuously flowing in a social area network, in most cases data curation is an impossible task. The following figure shows the boundary between the yellow and red dots as determined by unsupervised machine learning. You can see it clearly that the machine

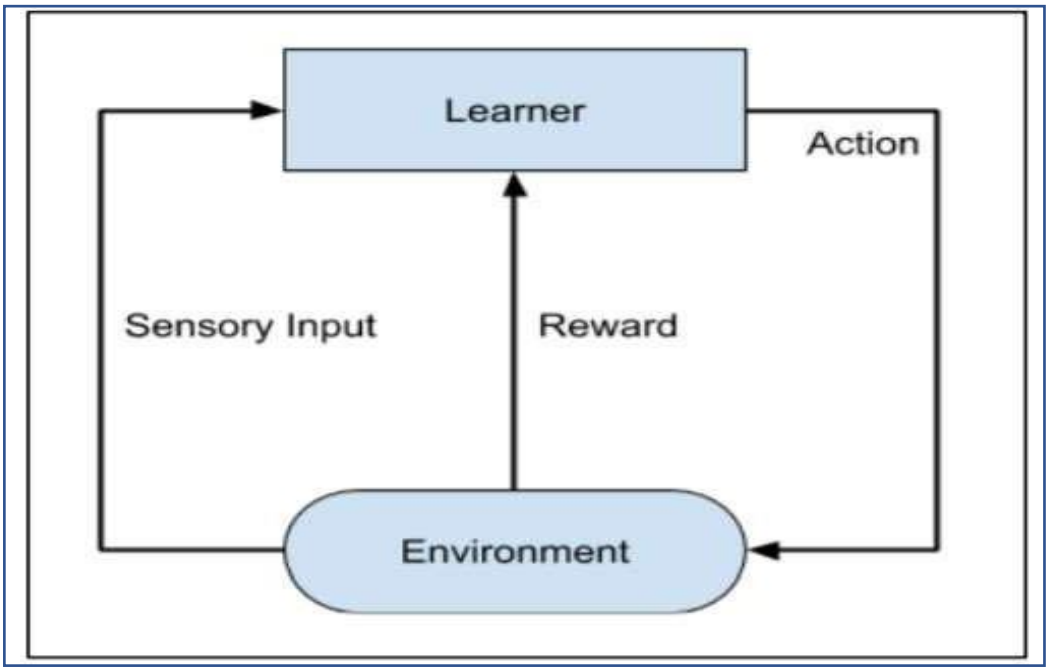
would be able to determine the class of each of the black dots with a fairly good accuracy.



Reinforcement Learning:

Consider training a pet dog, we train our pet to bring a ball to us. We throw the ball at a certain distance and ask the dog to fetch it back to us. Every time the dog does this right, we reward the dog. Slowly, the dog learns that doing the job rightly gives him a reward and then the dog starts doing the job right way every time in future. Exactly, this concept is applied in "Reinforcement" type of learning. The technique was initially developed for machines to play games. The machine is given an algorithm to analyse all possible moves at each stage of the game. The machine may select one of the moves at random. If the move is right, the machine is rewarded, otherwise it may be penalized. Slowly, the machine will start differentiating between right and wrong moves and after several iterations would learn to solve the game puzzle with a better accuracy. The accuracy of winning the game would improve as the machine plays more and more games.

The entire process may be depicted in the following diagram:



Deep Learning:

The deep learning is a model based on Artificial Neural Networks (ANN), more specifically Convolutional Neural Networks (CNN)s. There are several architectures used in deep learning such as deep neural networks, deep belief networks, recurrent neural networks, and convolutional neural networks. These networks have been successfully applied in solving the problems of computer vision, speech recognition, natural language processing, bioinformatics, drug design, medical image analysis, and games. There are several other fields in which deep learning is proactively applied. The deep learning requires huge processing power and humongous data, which is generally easily available these days. We will talk about deep learning more in detail in the coming chapters.

Deep Reinforcement Learning:

The Deep Reinforcement Learning (DRL) combines the techniques of both deep and reinforcement learning. The reinforcement learning algorithms like Q learning are now combined with deep learning to create a powerful DRL model. The technique has been with a great success in the fields of robotics, video games, finance and healthcare. Many previously unsolvable problems are now solved by creating DRL models. There is lots of research going on in this area and this is very actively pursued by the industries. So far, you

have got a brief introduction to various machine learning models, now let us explore slightly deeper into various algorithms that are available under these models.

Well posed learning problems:

A computer program is said to learn from experience E in context to some task T and some performance measure P, if its performance on T, as was measured by P, upgrades with experience E.

Any problem can be segregated as well-posed learning problem if it has three traits –

- Task
- Performance Measure
- Experience

Certain example that efficiently defines the well-posed learning problems are:

1. **To better filter emails as spam or not**
   - Task – Classifying emails as spam or not
   - Performance Measure – The fraction of emails accurately classified as spam or not spam
   - Experience – Observing you label emails as spam or not spam
2. **A checkers learning problem**
   - Task – Playing checkers game
   - Performance Measure – percent of games won against opposer
   - Experience **–** playing implementation games against itself
3. **Handwriting Recognition Problem**
   - Task – Acknowledging handwritten words within portrayal
   - Performance Measure – percent of words accurately classified
   - Experience – a directory of handwritten words with given classifications
4. **A Robot Driving Problem**
   - Task – driving on public four-lane highways using sight scanners
   - Performance Measure – average distance progressed before a fallacy
   - Experience – order of images and steering instructions noted down while observing a human driver
5. **Fruit Prediction Problem**

- Task – forecasting different fruits for recognition
- Performance Measure – able to predict maximum variety of fruits
- Experience – training machine with the largest datasets of fruits images

6. Face Recognition Problem

- Task – predicting different types of faces
- Performance Measure – able to predict maximum types of faces
- Experience – training machine with maximum amount of datasets of different face images

7. Automatic Translation of documents

- Task – translating one type of language used in a document to other language
- Performance Measure – able to convert one language to other efficiently
- Experience – training machine with a large dataset of different types of languages

Design of a learning system:

Just now we looked into the learning process and also understood the goal of the learning. When we want to design a learning system that follows the learning process, we need to consider a few design choices. The design choices will be to decide the following key components:

1. Type of training experience
2. Choosing the Target Function
3. Choosing a representation for the Target Function
4. Choosing an approximation algorithm for the Target Function
5. The final Design

We will look into the game - checkers learning problem and apply the above design choices. For a checkers learning problem, the three elements will be,

- Task T: To play checkers
- Performance measure P: Total present of the game won in the tournament.
- Training experience E: A set of games played against itself.

Type of training experience:

During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

Direct or Indirect training experience:

In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, lose or draw) are given for a number of games. How to assign credit or blame to individual moves is the credit assignment problem.

1. Teacher or Not:

- Supervised:

  The training experience will be labelled, which means, all the board states will be labelled with the correct move. So the learning takes place in the presence of a supervisor or a teacher.

- Un-Supervised:

  The training experience will be unlabelled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervision or teacher involvement.

- Semi-supervised:

  Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing.

2. Is the training experience good:

- Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.
- The checker player learns by playing against oneself. Its experience is indirect. It may not encounter moves that are common in human expert play. Once the proper training experience is available, the next design step will be choosing the Target Function.

Choosing the Target Function:

When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different possibilities. You think and apply the learning that you have gained from the experience. Here the learning is, for a specific board, you move a checker such that your board

state tends towards the winning situation. Now the same learning has to be defined in terms of the target function.

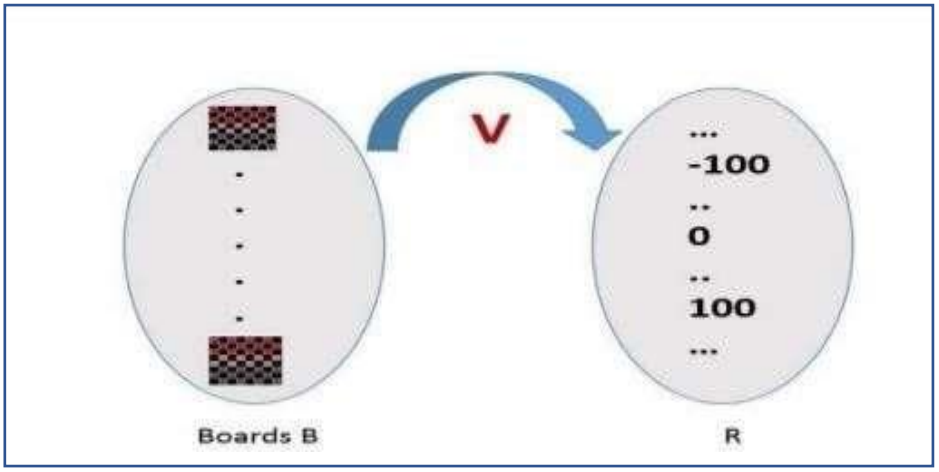
Here there are 2 considerations — direct and indirect experience.

- **During the direct experience** the checkers learning system, it needs only to learn how to choose the best move among some large search space. We need to find a target function that will help us choose the best move among alternatives.

Let us call this function Choose Move and use the notation Choose Move: B →M to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

- **When there is an indirect experience it** becomes difficult to learn such function. How about assigning a real score to the board state.

So the function be V: B →R indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states



If the system can successfully learn such a target function V, then it can easily use it to select the best move from any board position.

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

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

The (4) is a recursive definition and to determine the value of V(b) for a particular board state, it performs the search ahead for the optimal line of play, all the way to the end of the game. So this definition is not efficiently computable by our checkers playing program, we say that it is a non-operational definition.

Choosing a representation for the Target Function:

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function ^V that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial neural network. In general, this choice of representation involves a crucial trade off. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.

On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation: for any given board state, the function ^V will be calculated as a linear combination of the following board features:

- x1(b) — number of black pieces on board b
- x2(b) — number of red pieces on b
- x3(b) — number of black kings on b

- x4(b) — number of red kings on b
- x5(b) — number of red pieces threatened by black • x6(b) — number of black pieces threatened by red

^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b)

Where w0 through w6 are numerical coefficients or weights to be obtained by a learning algorithm. Weights w1 to w6 will determine the relative importance of different board features.

Specification of the Machine Learning Problem at this time: Till now we worked on choosing the type of training experience, choosing the target function and its representation. The checkers learning task can be summarized as below.

- Task T: Play Checkers
- Performance Measure: % of games won in world tournament
- Training Experience E: opportunity to play against itself
- Target Function: V: Board → R
- Target Function Representation: ^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b)

The first three items above correspond to the specification of the learning task, where as the final two items constitute design choices for the implementation of the learning program.

Choosing an approximation algorithm for the Target Function:

Generating training data — To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b. Each training example is an ordered pair <b,v_train(b)>.

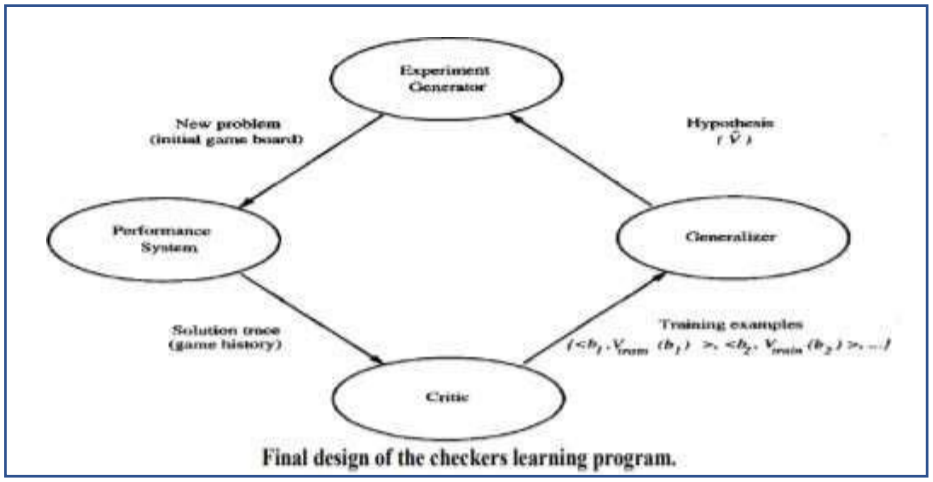
Temporal difference (TD) learning is a concept central to reinforcement learning, in which learning happens through the iterative correction of your estimated returns towards a more accurate target return.

✞ **V_train(b) ← ^V(Successor(b))**

Final Design for Checkers Learning system:

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

1. The performance System: Takes a new board as input and outputs a trace of the game it played against itself.
2. The Critic: Takes the trace of a game as an input and outputs a set of training examples of the target function.
3. The Generalizer: Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
4. The Experiment Generator: Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers learning program.

Issues in Machine Learning:

Our checkers example raises a number of generic questions about machine learning. The field of machine learning, and much of this book, is concerned with answering questions such as the following:

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

CONCEPT LEARNING:

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

A Formal Definition for Concept Learning:

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

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

A Concept Learning Task: Enjoy Sport Training Examples

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

ATTRIBUTES CONCEPT

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

Concept Learning as Search:

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.
- By selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.

FIND-S:

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

FIND-S Algorithm:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x For each attribute constraint a, in h

   If the constraint a, is satisfied by x

   Then do nothing

   3. Else replace a, in h by the next more general constraint that is satisfied by x 4. Output hypothesis h

FIND-S Algorithm – Example:

Important-Representation:

1. **?** indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. **Φ** indicates that no value is acceptable.
4. The most **general hypothesis** is represented by: **{?, ?, ?, ?, ?, ?}**
5. The most **specific hypothesis** is represented by: **{ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

Steps Involved in Find-S:

1. Start with the most specific hypothesis. **h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

2. Take the next example and if it is negative, then no changes occur to the hypothesis.
3. If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.
4. Keep repeating the above steps till all the training examples are complete.
5. After we have completed all the training examples we will have the final hypothesis when can use to classify the new examples. **Example:** Consider the following data set having the data about which particular seeds are poisonous.

EXAMPLE	COLOR	TOUGHNESS	FUNGUS	APPEARANCE	POISONOUS
1.	GREEN	HARD	NO	WRINKELD	YES
2.	GREEN	HARD	YES	SMOOTH	NO
3.	BROWN	SOFT	NO	WRINKLED	NO
4.	ORANGE	HARD	NO	WRINKLED	YES
5.	GREEN	SOFT	YES	SMOOTH	YES
6.	GREEN	HARD	YES	WRINKLED	YES
7.	ORANGE	HARD	NO	WRINKLED	YES

ATTRIBUTES ON WHICH THE CONCEPT DEPENDS ON — CONCEPT

First, we consider the hypothesis to be a more specific hypothesis. Hence, our hypothesis would be: h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}

Consider example 1:

The data in example 1 is {GREEN, HARD, NO, WRINKLED}. We see that our initial hypothesis is more specific and we have to generalize it for this example.

Hence, the hypothesis becomes:

h = {GREEN, HARD, NO, WRINKLED}

Consider example 2:

Here we see that this example has a negative outcome. Hence we neglect this example and our hypothesis remains the same. h = {GREEN, HARD, NO, WRINKLED}

Consider example 3:

Here we see that this example has a negative outcome. hence we neglect this example and our hypothesis remains the same. h = {GREEN, HARD, NO, WRINKLED}

Consider example 4:

The data present in example 4 is {ORANGE, HARD, NO, WRINKLED}. We
compare every single attribute with the initial data and if any mismatch is found we replace that particular attribute with a general case (" ?"). After doing the process the hypothesis becomes: h = {?, HARD, NO, WRINKLED }

Consider example 5:

The data present in example 5 is {GREEN, SOFT, YES, SMOOTH}. We compare every single attribute with the initial data and if any mismatch is found we replace that particular attribute with a general case ("?"). After doing the process the hypothesis becomes:
h = {?, ?, ?, ? }
Since we have reached a point where all the attributes in our hypothesis have the general condition, example 6 and example 7 would result in the same hypothesizes with all general attributes. h = {?, ?, ?, ? }

Hence, for the given data the final hypothesis would be:
Final Hypothesis: h = { ?, ?, ?, ? }.

Version Spaces

Definition(Version space). A concept is complete if it covers all positive examples.

A concept is consistent if it covers none of the negative examples. The version space is the set of all complete and consistent concepts. This set is convex and is fully defined by its least and most general elements.

Candidate-Elimination Learning Algorithm

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

Initialize G to the set of maximally general hypotheses in H Initialize S to the set of maximally specific hypotheses in H For each training example d, do

- If d is a positive example
- Remove from G any hypothesis inconsistent with d
- For each hypothesis s in S that is not consistent with d
- Remove s from S • Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h
- Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
- Remove from S any hypothesis inconsistent with d
- For each hypothesis g in G that is not consistent with d
- Remove g from G 18\
- Add to G all minimal specializations h of g such that
- h is consistent with d, and some member of S is more specific than h
- Remove from G any hypothesis that is less general than another hypothesis in G.

CANDIDATE- ELIMINTION algorithm using version spaces An Illustrative Example:

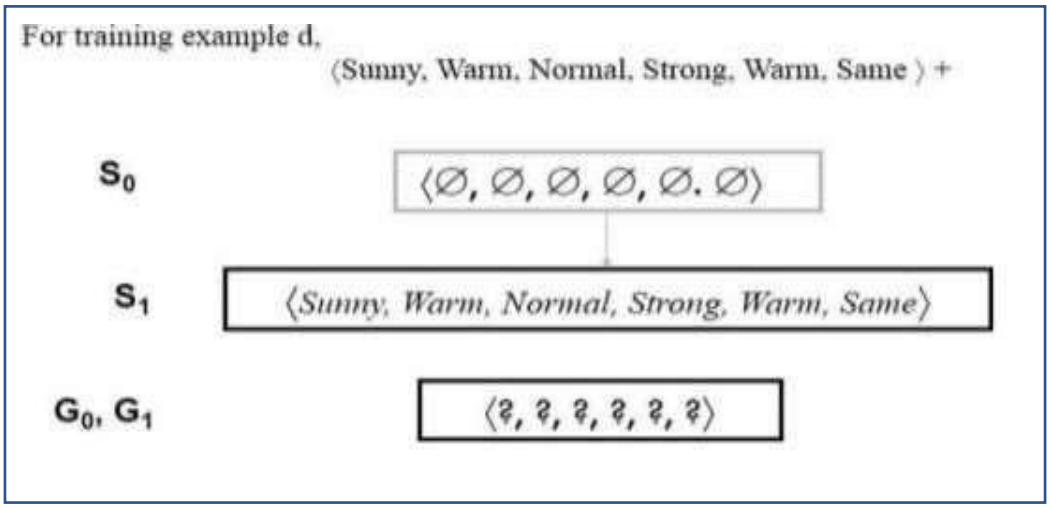
Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

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

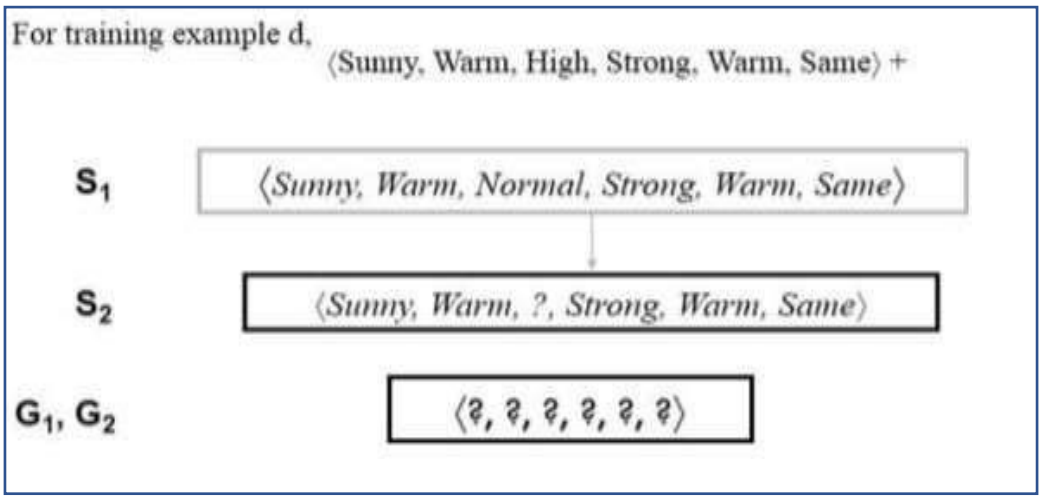
boundary set to contain the most general hypothesis in H, G0 ?, ?, ?, ?, ?,

When the first training example is presented, the CANDIDATEELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.

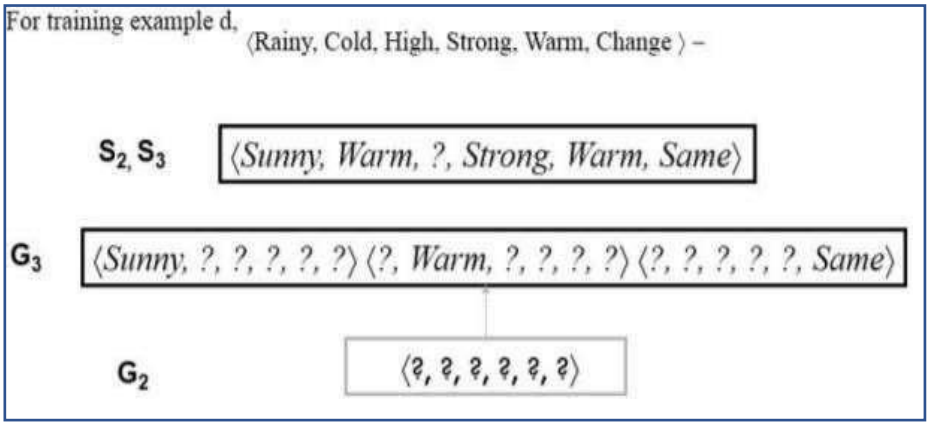
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example.
- No update of the G boundary is needed in response to this training example because Go correctly covers this example.



- When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 =G0

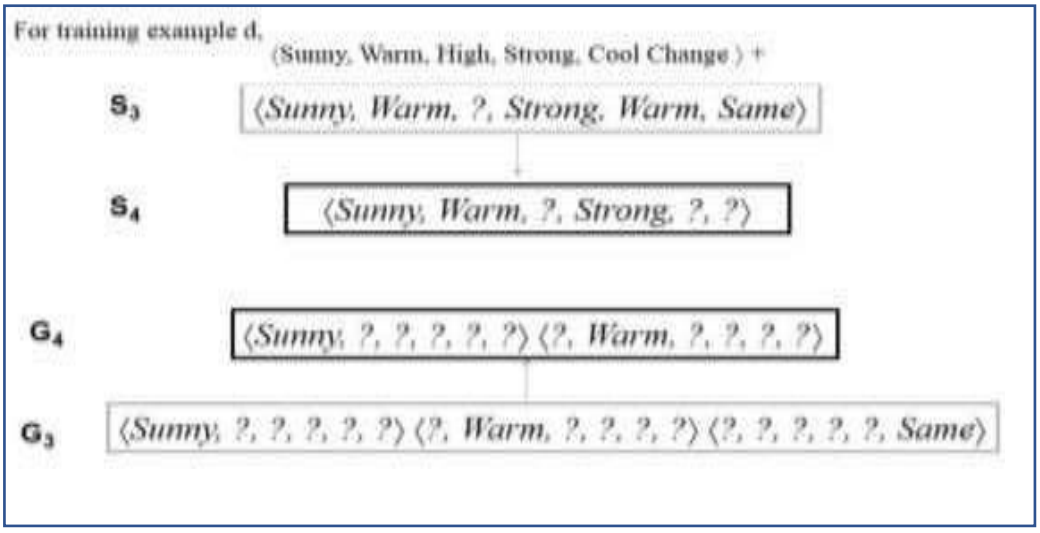


- Consider the third training example. This negative example reveals that the boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example.

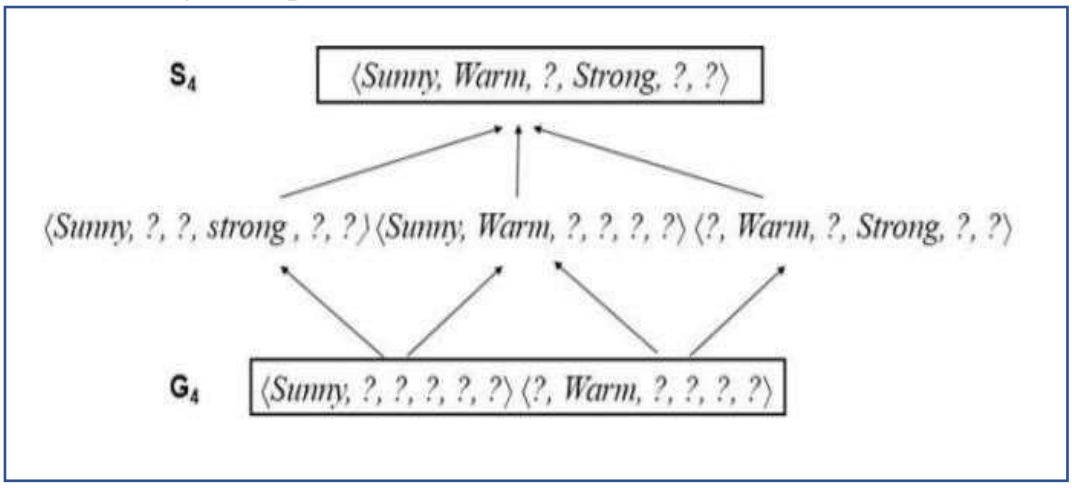


Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?

For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples. Consider the fourth training example.



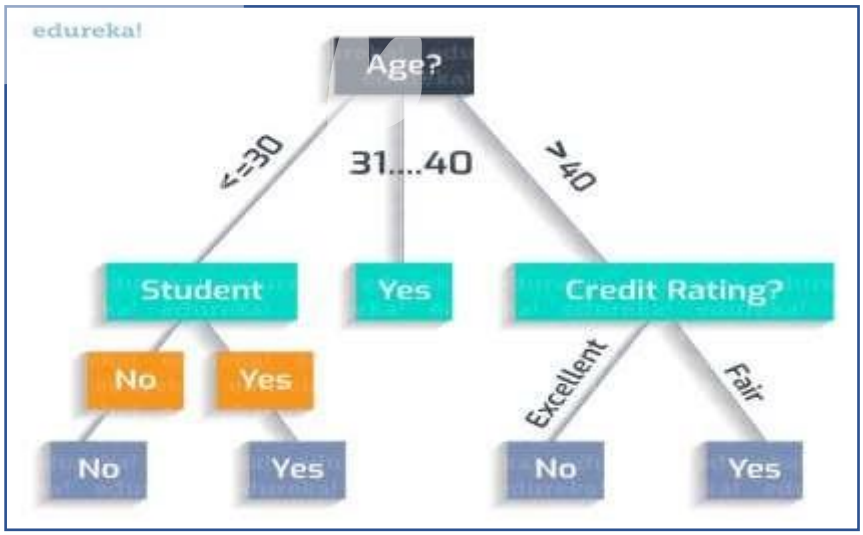
- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.
- After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



Inductive bias:

Decision Tre:e Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where th e data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

Decision Tree Representation:



An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', and 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem). There are two main types of Decision Trees:

1. Classification trees (Yes/No types):

What we have seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is Categorical.

Inductive bias refers to the restrictions that are imposed by the assumptions

Here the decision or the outcome variable is Continuous, e.g. a number like 123. Working Now that we know what a Decision Tree is, we'll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as ID3 Algorithm. ID3 Stands for Iterative Dichotomiser3.

Before discussing the ID3 algorithm, we'll go through few definitions. Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data.

Appropriate Problems for Decision Tree Learning:

- Instances are represented by attribute-value pair
- The target function has discrete output values
- Disjunctive descriptions may be required
- The training data may contain errors
- The training data may contain missing attribute values.
- Suitable for classifications.

Hypothesis Space Search:

The set of possible decision tree, Simple to complex, hill climbing search.

Capability:

- Hypothesis space of all decision trees is a complete space of finite discrete valued functions.
- ID3 maintains only a single current hypothesis.
- Cannot determine how many alternative decision trees are consistent with the available training data.

- ID3 uses all training example at each step to make statistically based decisions regarding how to refine its current hypothesis.

- The resulting search is much less sensitive to errors in individual training examples.

Inductive Bias in Decision Tree Learning: Note H is the power set of instances X

- Inductive Bias in ID3 – Approximate inductive bias of ID3

- Shorter trees are preferred over larger tress

- BFS-ID3

Difference between (ID3 & C-E) && Restriction bias and Preference bias

ID3	**Candidate-Elimination**
Searches a complete hypothesis space incompletely	Searches an incomplete hypothesis space completely
Inductive bias is solely a consequence of the ordering of hypotheses by its search strategy	Inductive bias is solely a consequence of the expressive power of its hypothesis representation

sss

Restriction bias	**Preference bias**
Candidate-Elimination	ID3
Categorical restriction on the set of hypotheses considered	Preference for certain hypotheses over others

Possibility of excluding the unknown target function	Work within a complete hypothesis space

Issues in Decision Tree Learning:

- Determine how deeply to grow the decision tree
- Handling continuous attributes
- Choosing an appropriate attribute selection measure
- Handling training data with missing attribute values
- Handling attributes with differing costs
- Improving computational efficiency

UNIT-II

Artificial Neural Networks

Introduction:

Artificial Neural Networks (ANN) are algorithms based on brain function and are used to model complicated patterns and forecast issues. The Artificial Neural Network (ANN) is a deep learning method that arose from the concept of the human brain Biological Neural Networks. The development of ANN was the result of an attempt to replicate the workings of the human brain. The workings of ANN are extremely similar to those of biological neural networks, although they are not identical. ANN algorithm accepts only numeric and structured data.

The ANN applications:

Classification, the aim is to predict the class of an input vector

- Pattern matching, the aim is to produce a pattern best associated with a given input vector.
- Pattern completion, the aim is to complete the missing parts of a given input vector.
- Optimization, the aim is to find the optimal values of parameters in an optimization problem.
- Control, an appropriate action is suggested based on given an input vectors
- Function approximation/times series modelling, the aim is to learn the functional relationships between input and desired output vectors.
- Data mining, with the aim of discovering hidden patterns from data (knowledge discovery). **ANN architectures**

- Neural Networks are known to be universal function approximators
- Various architectures are available to approximate any nonlinear function
- Different architectures allow for generation of functions of different complexity and power

- Feed forward networks
- Feedback networks
- Lateral networks

Advantages of Artificial Neural Networks

Attribute-value pairs are used to represent problems in ANN.

1. The output of ANNs can be discrete-valued, real-valued, or a vector of multiple real or discrete-valued characteristics, while the target function can be discrete-valued, real-valued, or a vector of numerous real or discrete-valued attributes.
2. Noise in the training data is not a problem for ANN learning techniques. There may be mistakes in the training samples, but they will not affect the final result.
3. It's utilized when a quick assessment of the taught target function is necessary.
4. The number of weights in the network.
5. the number of training instances evaluated, and the settings of different learning algorithm parameters can all contribute to extended training periods for ANNs.

Disadvantages of Artificial Neural Networks

1. Hardware Dependence:

- The construction of Artificial Neural Networks necessitates the use of parallel processors.
- As a result, the equipment's realization is contingent.

2. Understanding the network's operation:

- This is the most serious issue with ANN.
- When ANN provides a probing answer, it does not explain why or how it was chosen.
- As a result, the network's confidence is eroded.

3. Assured network structure:

- Any precise rule does not determine the structure of artificial neural networks.
- Experience and trial and error are used to develop a suitable network structure.

4. Difficulty in presenting the issue to the network:

- ANNs are capable of working with numerical data.
- Before being introduced to ANN, problems must be converted into numerical values.
- The display method that is chosen will have a direct impact on the network's performance.
- The user's skill is a factor here.

5. The network's lifetime is unknown: • When the network's error on the sample is decreased to a specific amount, the training is complete.
    - The value does not produce the best outcomes.

Appropriate Problems for Neural Network Learning:

1. Instances are represented by many attribute-value pairs (e.g., the pixels of a picture. ALVINN [Mitchell, p. 84]).
2. The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
3. The training examples may contain errors.
4. Long training times are acceptable.
5. Fast evaluation of the learned target function may be required.
6. The ability for humans to understand the learned target function is not important.

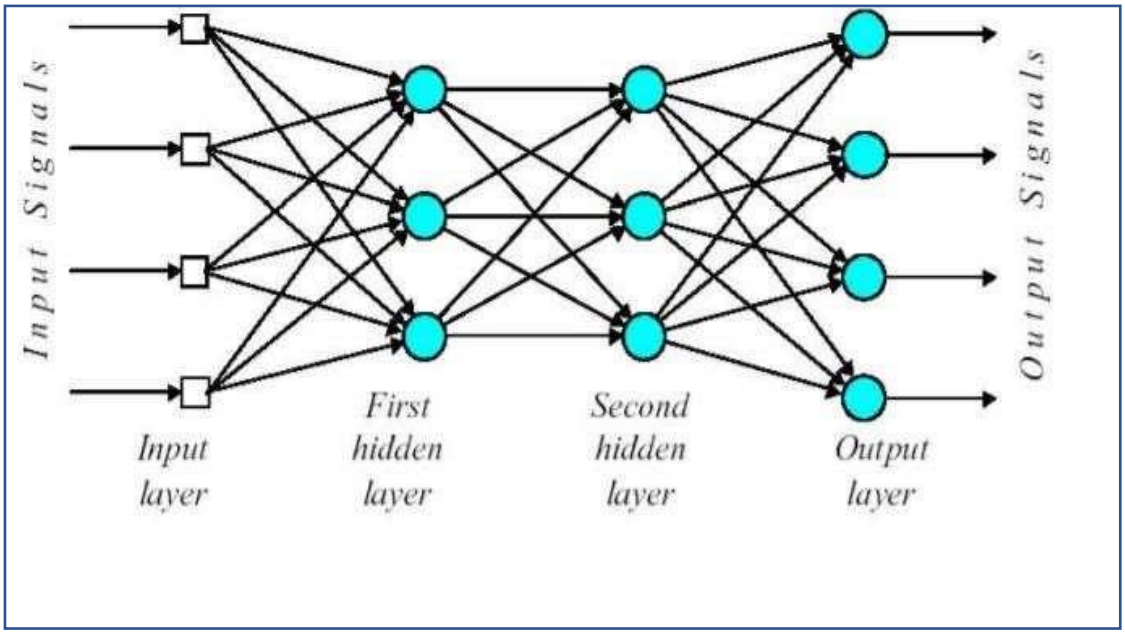
History of Neural Networks:

1. 1943: McCulloch and Pitts proposed a model of a neuron Perceptron (read [Mitchell, section 4.4])
2. 1960s: Widrow and Hoff explored Perceptron networks (which they called "Adelines") and the delta rule.
3. 1962: Rosenblatt proved the convergence of the perceptron training rule.

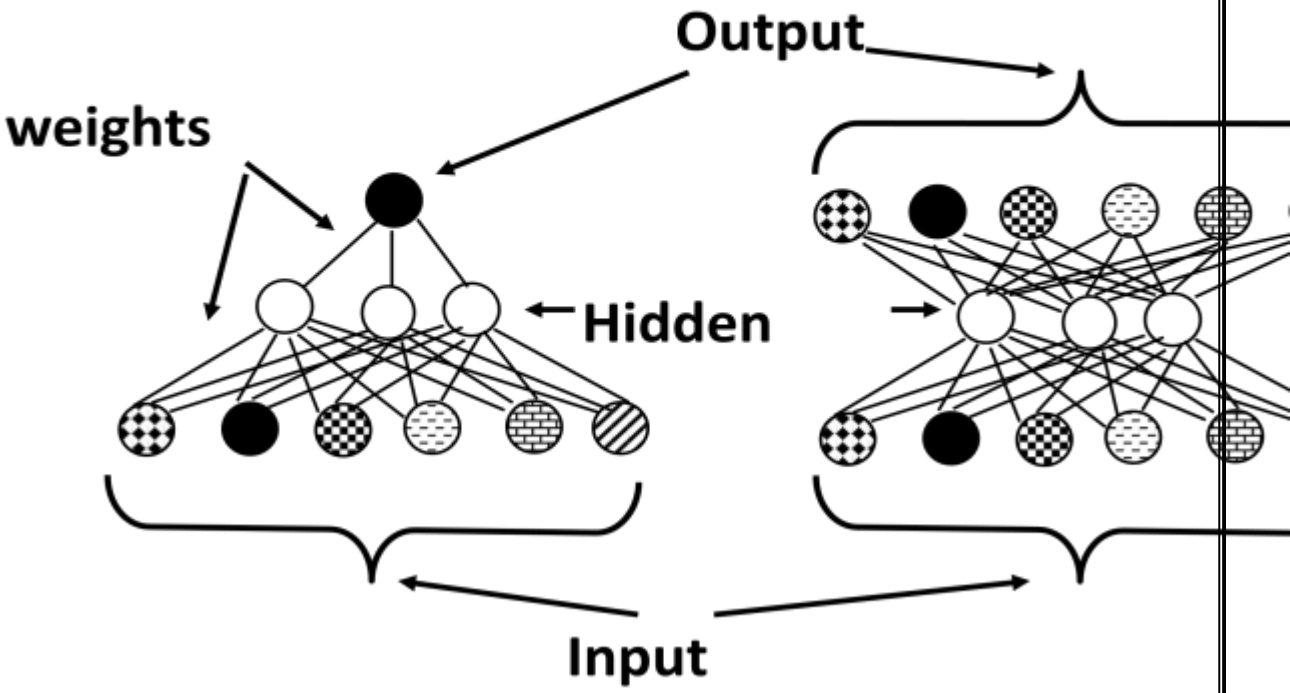
4. 1969: Minsky and Papert showed that the Perceptron cannot deal with nonlinearly-separable data sets---even those that represent simple function such as X-OR.
5. 1970-1985: Very little research on Neural Nets
6. 1986: Invention of Backpropagation Rumelhart and McClelland, but also Parker and earlier on: Werbos which can learn from nonlinearly-separable data sets.
7. Since 1985: A lot of research in Neural Nets!

Multilayer Neural Network:

- A multiplayer perceptron is a feed forward neural network with one or more hidden layers
- The network consists of an input layer of source neurons, at least one hidden layer of computational neurons, and an output layer of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.
- Neurons in the hidden layer cannot be observed through input/output behaviour of the network.
- There is no obvious way to know what the desired output of the hidden layer should be.


Input Signals
Output Signals
Input layer
First hidden layer
Second hidden layer
Output layer

Multilayer Neural Network Representation:



Learning in Multilayer Neural Networks:

- Learning consists of searching through the space of all possible matrices of weight values for a combination of weights that satisfies a database of positive and negative examples (multi-class as well as regression problems are possible).
- Note that a Neural Network model with a set of adjustable weights defines a restricted hypothesis space corresponding to a family of functions. The size of this hypothesis space can be increased or decreased by increasing or decreasing the number of hidden units present in the network.

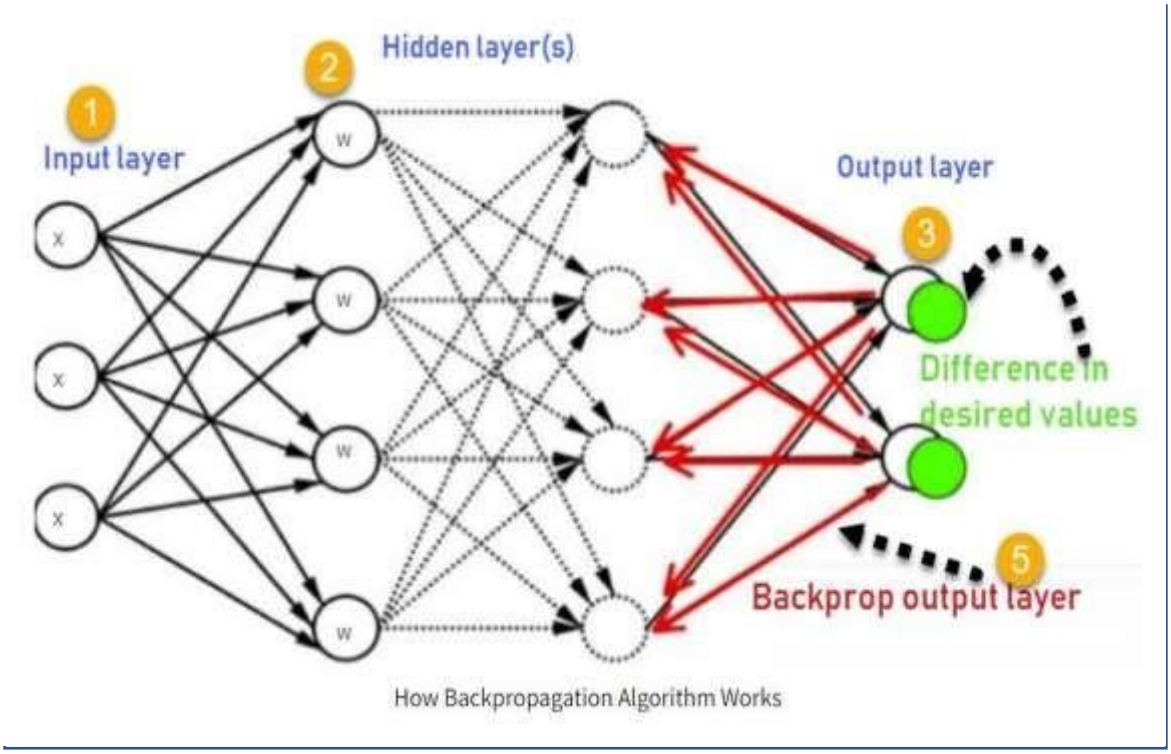
Back propagation: Overview

- Back propagation works by applying the *gradient descent* rule to a feed forward network.
- The algorithm is composed of two parts that get repeated over and over until a pre-set maximal number of *epochs, EP max*.
- Part I, the *feed forward* pass: the activation values of the hidden and then output units are computed.
- Part II, the *back propagation* pass: the weights of the network are updated-starting with the hidden to output weights and followed by the input to hidden weights--with respect to the sum of squares error and through a series of weight update rules called the *Delta Rule*.

Definition:

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



How Backpropagation Algorithm Works

- Inputs X, arrive through the preconnected path
- Input is modelled using real weights W. The weights are usually randomly selected.
- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- Calculate the error in the outputs

$$Error_B = \text{Actual Output} - \text{Desired Output}$$

- Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
- Keep repeating the process until the desired output is achieved

Why We Need Back propagation?

- Most prominent advantages of Back propagation are:
- Back propagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Types of Back propagation Networks

Two Types of Back propagation Networks are:

- Static Back-propagation
- Recurrent Back propagation **Static back-propagation**:

It is one kind of back propagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Back propagation:

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

Disadvantages of using Back propagation

- The actual performance of back propagation on a specific problem is dependent on the input data.
- Back propagation algorithm in data mining can be quite sensitive to noisy data
- You need to use the matrix-based approach for back propagation instead of mini-batch.

Back propagation: The Algorithm

- Initialize the weights to small random values; create a random pool of all the training patterns; set *EP*, the number of epochs of training to 0.
- 2. Pick a training pattern □ from the remaining pool of patterns and propagate it forward through the network.
- 3. Compute the deltas, □k for the output layer.
- 4. Compute the deltas, □j for the hidden layer by propagating the error backward.
- Update all the connections such that
- $W^{New}ji = wji^{old} + \square wji$ and $w^{New}kj = wkj^{Old} + \square wkj$

- If any pattern remains in the pool, then go back to Step 2. If all the training patterns in the pool have been used, then set EP = EP+1, and if EP EPMax, then create a random pool of patterns and go to Step 2. If EP = EPMax, then stop.

Back propagation: The Momentum:

- To this point, Back propagation has the disadvantage of being too slow if is small and it can oscillate too widely if is large.
- To solve this problem, we can add a *momentum* to give each connection some inertia, forcing it to change in the direction of the downhill "force".
- New Delta Rule:

$$\Delta w_{pq}(t+1) = -\eta \, \partial E/\partial w_{pq} + \alpha \, \Delta w_{pq}(t)$$

- Where p and q are any input and hidden, or, hidden and output units; t is a time step or epoch; and is the momentum parameter which regulates the amount of inertia of the weights.

UNIT - III

Introduction to Bayesian Learning

Imagine a situation where your friend gives you a new coin and asks you the fairness of the coin (or the probability of observing heads) without even flipping the coin once. In fact, you are also aware that your friend has not made the coin biased. In general, you have seen that coins are fair, thus you expect the probability of observing heads is 0.50.5. In the absence of any such observations, you assert the fairness of the coin only using your past experiences or observations with coins.

Suppose that you are allowed to flip the coin 1010 times in order to determine the fairness of the coin. Your observations from the experiment will fall under one of the following cases:

- **Case 1**: observing 55 heads and 55 tails.
- **Case 2**: observing hh heads and 10−h10−h tails, where h≠10−hh≠10−h.

If case 1 is observed, you are now more certain that the coin is a fair coin, and you will decide that the probability of observing heads is 0.50.5 with more confidence. If case 2 is observed you can either:

1. Neglect your prior beliefs since now you have new data, decide the probability of observing heads is h/10h/10 by solely depending on recent observations.
2. Adjust your belief accordingly to the value of hh that you have just observed, and decide the probability of observing heads using your recent observations.

The first method suggests that we use the frequentist method, where we omit our beliefs when making decisions. However, the second method seems to be more convenient because 1010 coins are insufficient to determine the fairness of a coin. Therefore, we can make better decisions by combining our recent observations and beliefs that we have gained through our past experiences. It is this thinking model which uses our most recent observations together with our beliefs or inclination for critical thinking that is known as Bayesian thinking.

Moreover, assume that your friend allows you to conduct another 1010 coin flips. Then we can use these new observations to further update our beliefs. As we gain more data, we can incrementally update our beliefs increasing the certainty of our conclusions. This is known as incremental learning, where you update your knowledge incrementally with new evidence. Bayesian learning comes into play on such occasions, where we are unable to use *frequentist statistics* due to the drawbacks that we have discussed above. We can use Bayesian learning to address all these drawbacks and even with additional capabilities (such as incremental updates of the posterior) when testing a hypothesis to estimate unknown parameters of a machine learning models. Bayesian learning uses Bayes' theorem to determine the conditional probability of a hypotheses given some evidence or observations.

The Famous Coin Flip Experiment

When we flip a coin, there are two possible outcomes - heads or tails. Of course, there is a third rare possibility where the coin balances on its edge without falling onto either side, which we assume is not a possible outcome of the coin flip for our discussion. We conduct a series of coin flips and record our observations i.e. the number of the heads (or tails) observed for a certain number of coin flips. In this experiment, we are trying to determine the fairness of the coin, using the number of heads (or tails) that we observe.

Frequentist Statistics

Let us think about how we can determine the fairness of the coin using our observations in the above mentioned experiment. Once we have conducted a sufficient number of coin flip trials, we can determine the frequency or the probability of observing the heads (or tails). If we observed heads and tails with equal frequencies or the probability of observing heads (or tails) is 0.50.5, then it can be established that the coin is a fair coin. Failing that, it is a biased coin. Let's denote pp as the probability of observing the heads. Consequently, as the quantity that pp deviates from 0.50.5 indicates how biased the coin is, pp can be considered as the degree-of-fairness of the coin.

Testing whether a hypothesis is true or false by calculating the probability of an event in a prolonged experiment is known as *frequentist statistics*. As such, determining the fairness of a coin by using the probability of observing the heads is an example of *frequentist statistics* (a.k.a. *frequentist approach*).

Let us now further investigate the coin flip example using the *frequentist approach*. Since we have not intentionally altered the coin, it is reasonable to assume that we are using an unbiased coin for the experiment. When we flip the coin 1010 times, we observe the heads 66 times. Therefore, the pp is 0.60.6 (note that pp is the number of heads observed over the number of total coin flips). Hence, according to frequencies statistics, the coin is a biased coin — which opposes our assumption of a fair coin. Perhaps one of your friends who is more skeptical than you extends this experiment to 100100 trails using the same coin. Then she observes heads 5555 times, which results in a different pp with 0.550.55. Even though the new value for pp does not change our previous conclusion (i.e. that the coin is biased), this observation raises several questions:

- How confident are we of pp being 0.60.6?
- How confident are of pp being 0.550.55?
- Which of these values is the accurate estimation of pp?

Will pp continue to change when we further increase the number of coin flip trails?

We cannot find out the exact answers to the first three questions using *frequentist statistics*. We may assume that true value of pp is closer to 0.550.55 than 0.60.6 because the former is computed using observations from a considerable number of trials compared to what we used to compute the latter. Yet there is no way of confirming that hypothesis. However, if we further increase the number of trials, we may get a different probability from both of the above values for observing the heads and eventually, we may even discover that the coin is a fair coin.

Number of coin flips	Number of heads	Probability of observing heads
10	6	0.6
50	29	0.58
100	55	0.55
200	94	0.47
500	245	0.49

Table 1 - Coin flip experiment results when increasing the number of trials

Table 1 presents some of the possible outcomes of a hypothetical coin flip experiment when we are increasing the number of trials. The fairness (pp) of the coin changes when increasing the number of coin-flips in this experiment. Our confidence of estimated pp may also increase when increasing the number of coin-flips, yet the frequentist statistic does not facilitate any indication of the confidence of the estimated pp value. We can attempt to understand the importance of such a confident measure by studying the following cases:

- An experiment with an infinite number of trials guarantees pp with absolute accuracy (100% confidence). Yet, it is not practical to conduct an experiment with an infinite number of trials and we should stop the experiment after a sufficiently large number of trials. However, deciding the value of this sufficient number of trials is a challenge when using *frequentist statistics*.
  If we can determine the confidence of the estimated pp value or the inferred conclusion, in a situation where the number of trials is limited, this will allow

us to decide whether to accept the conclusion or to extend the experiment with more trials until it achieves sufficient confidence.

Moreover, we may have valuable insights or prior beliefs (for example, coins are usually fair and the coin used is not made biased intentionally, therefore p≈0.5p≈0.5) that describes the value of pp. Embedding that information can significantly improve the accuracy of the final conclusion. Such beliefs play a significant role in shaping the outcome of a hypothesis test especially when we have limited data. However, with *frequentist statistics*, it is not possible to incorporate such beliefs or past experience to increase the accuracy of the hypothesis test.

Some Terms to Understand

Before delving into Bayesian learning, it is essential to understand the definition of some terminologies used. I will not provide lengthy explanations of the mathematical definition since there is a lot of widely available content that you can use to understand these concepts.

- Random variable (Stochastic variable) - In statistics, the random variable is a variable whose possible values are a result of a random event. Therefore, each possible value of a random variable has some probability attached to it to represent the likelihood of those values.
- Probability distribution - The function that defines the probability of different outcomes/values of a random variable. The continuous probability distributions are described using probability density functions whereas discrete probability distributions can be represented using probability mass functions.

  Conditional probability - This is a measure of probability P(A|B)P(A|B) of an event A given that another event B has occurred.
- Joint probability distribution

Bayes' Theorem

Bayes' theorem describes how the conditional probability of an event or a hypothesis can be computed using evidence and prior knowledge. It is similar to concluding that our code has no bugs given the evidence that it has passed

all the test cases, including our prior belief that we have rarely observed any bugs in our code. However, this intuition goes beyond that simple hypothesis test where there are multiple events or hypotheses involved (let us not worry about this for the moment).

The Bayes' theorem is given by:

$$P(\theta|X)=P(X|\theta)P(\theta)P(X)P(\theta|X)=P(X|\theta)P(\theta)P(X)$$

I will now explain each term in Bayes' theorem using the above example. Consider the hypothesis that there are no bugs in our code. θθ and XX denote that our code is bug free and passes all the test cases respectively.

- $P(\theta)P(\theta)$ - Prior Probability is the probability of the hypothesis θθ being true before applying the Bayes' theorem. Prior represents the beliefs that we have gained through past experience, which refers to either common sense or an outcome of Bayes' theorem for some past observations. For the example given, prior probability denotes the probability of observing no bugs in our code. However, since this is the first time we are applying Bayes' theorem, we have to decide the priors using other means
  (Otherwise we could use the previous posterior as the new prior). Let us assume that it is very unlikely to find bugs in our code because rarely have we observed bugs in our code in the past. With our past experience of observing fewer bugs in our code, we can assign our prior $P(\theta)P(\theta)$ with a higher probability. However, for now, let us assume that $P(\theta)=pP(\theta)$

  This term depends on the test coverage of the test cases. Even though we do not know the value of this term without proper measurements, in order to continue this discussion let us assume that $P(X|\neg\theta)=0.5P(X|\neg\theta)=0.5$. Accordingly,
  $P(X)=1\times p+0.5\times(1-p)=0.5(1+p)P(X)=1\times p+0.5\times(1-p)=0.5(1+p)$

- $P(\theta|X)P(\theta|X)$ - Posteriori probability denotes the conditional probability of the hypothesis θθ after observing the evidence XX. This is the probability of observing no bugs in our code given that it passes all the test cases. Since we

now know the values for the other three terms in the Bayes' theorem, we can calculate the posterior probability using the following formula:

$P(\theta|X)=1\times p0.5(1+p)P(\theta|X)=1\times p0.5(1+p)$

We can also calculate the probability of observing a bug, given that our code passes all the test cases $P(\neg\theta|X)P(\neg\theta|X)$.

$P(\neg\theta|X)=P(X|\neg\theta).P(\neg\theta)P(X)=0.5\times(1-p)0.5\times(1+p)=(1-p)(1+p)P(\neg\theta|X)=P(X|\neg\theta).P(\neg\theta)$

$P(X)=0.5\times(1-p)0.5\times(1+p)=(1-p)(1+p)$

We now know both conditional probabilities of observing a bug in the code and not observing the bug in the code. Yet how are we going to confirm the valid hypothesis using these posterior probabilities?

Maximum a Posteriori (MAP)

We can use MAP to determine the valid hypothesis from a set of hypotheses. According to MAP, the hypothesis that has the maximum posterior probability is considered as the valid hypothesis. Therefore, we can express the hypothesis $\theta MAP\theta MAP$ that is concluded using MAP as follows: $\theta MAP=argmax\theta P(\theta i|X)=argmax\theta(P(X|\theta i)P(\theta i)P(X))\theta MAP=argmax\theta P(\theta i|X)=argmax\theta(P(X|\theta\ i)P(\theta i)P(X))$

The $argmax\theta argmax\theta$ operator estimates the event or hypothesis $\theta i\theta i$ that maximizes the posterior probability $P(\theta i|X)P(\theta i|X)$. Let us apply MAP to the above example in order to determine the true hypothesis:

$\theta MAP=argmax\theta\{\theta:P(\theta|X)=p0.5(1+p),\neg\theta:P(\neg\theta|X)=(1-p)(1+p)\}\theta MAP=argmax\theta\{\theta:P(\theta|X)=p0.5(1+p),\neg\theta:P(\neg\theta|X)=(1-p)(1+p)\}$

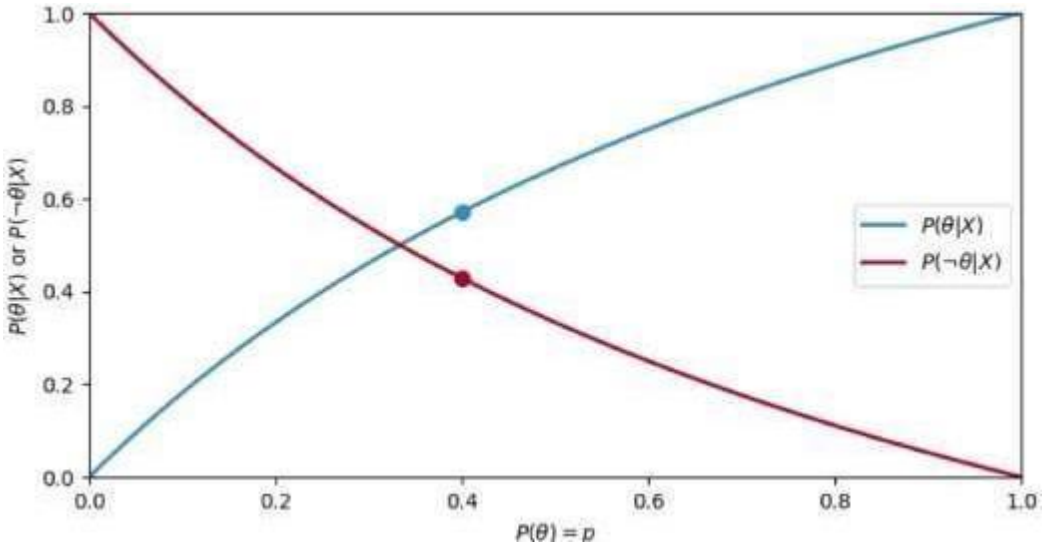


Figure 1 - P(θ|X)P(θ|X) and P(¬θ|X)P(¬θ|X) when changing the P(θ)=pP(θ)=p Figure 1 illustrates how the posterior probabilities of possible hypotheses change with the value of prior probability. Unlike *frequentist statistics* where our belief or past experience had no influence on the concluded hypothesis, Bayesian learning is capable of incorporating our belief to improve the accuracy of predictions. Assuming that we have fairly good programmers and therefore the probability of observing a bug is P(θ)=0.4P(θ)=0.4 , then we find the θMAPθMAP:

MAP=argmaxθ{θ:P(|X)=0.40.5(1+0.4),¬θ:P(¬θ|X)=0.5(1−0.4)0.5(1+0.4)}=argmaxθ{θ:P(θ|X)=0.57,¬θ:P(¬θ|X)=0.43}=θ⟹No bugs present in our codeMAP=argmaxθ{θ:P(|X)=0.40.5(1+0.4),¬θ:P(¬θ|X)=0.5(1−0.4)0.5(1+0.4)}=argmaxθ{θ:P(θ|X)=0.57,¬θ:P(¬θ|X)=0.43}=θ⟹No bugs present in our code

However, P(X)P(X) is independent of θθ, and thus P(X)P(X) is same for all the events or hypotheses. Therefore, we can simplify the θMAPθMAP estimation, without the denominator of each posterior computation as shown below: θMAP=argmaxθ(P(X|θi)P(θi))θMAP=argmaxθ(P(X|θi)P(θi))

Notice that MAP estimation algorithms do not compute posterior probability of each hypothesis to decide which is the most probable hypothesis. Assuming that our hypothesis space is continuous (i.e. fairness of the coin encoded as probability of observing heads, coefficient of a regression model, etc.), where endless possible hypotheses are present even in the smallest range that the human mind can think of, or for even a discrete hypothesis space with a large number of possible outcomes for an event, we do not need to find the posterior of each hypothesis in order to decide which is the most probable hypothesis. Therefore, the practical implementation of MAP estimation algorithms use approximation techniques, which are capable of finding the most probable hypothesis without computing posteriors or only by computing some of them.

Using the Bayesian theorem, we can now incorporate our belief as the prior probability, which was not possible when we used *frequentist statistics*. However, we still have the problem of deciding a sufficiently large number of trials or attaching a confidence to the concluded hypothesis. This is because the above example was solely designed to introduce the Bayesian theorem and each of its terms. Let us now gain a better understanding of
Bayesian learning to learn about the full potential of Bayes' theorem.

Binomial Likelihood

The likelihood for the coin flip experiment is given by the probability of observing heads out of all the coin flips given the fairness of the coin. As we have defined the fairness of the coins (θθ) using the probability of observing heads for each coin flip, we can define the probability of observing heads or

tails given the fairness of the coin P(y|θ)P(y|θ) where y=1y=1 for observing heads and y=0y=0 for observing tails. Accordingly:

P(y=1|θ)=θP(y=0|θ)=(1−θ)P(y=1|θ)=θP(y=0|θ)=(1−θ)

Now that we have defined two conditional probabilities for each outcome above, let us now try to find the P(Y=y|θ)P(Y=y|θ) joint probability of observing heads or tails:

P(Y=y|θ)={θ, if y=11−θ, otherwise P(Y=y|θ)={θ, if y=11−θ, otherwise

Note that yy can only take either 00 or 11, and θθ will lie within the range of [0,1][0,1]. We can rewrite the above expression in a single expression as follows:

P(Y=y|θ)=θy×(1−θ)1−yP(Y=y|θ)=θy×(1−θ)1−y

The above equation represents the likelihood of a single test coin flip experiment.

Interestingly, the likelihood function of the single coin flip experiment is similar to the Bernoulli probability distribution. The Bernoulli distribution is the probability distribution of a single trial experiment with only two opposite outcomes. As the Bernoulli probability distribution is the simplification of Binomial probability distribution for a single trail, we can represent the likelihood of a coin flip experiment that we observe kk number of heads out of NN number of trials as a Binomial probability distribution as shown below:

P(k,N|θ)=(Nk)θk(1−θ)N−k

Maximum likelihood estimation method (MLE)

The likelihood function indicates how likely the observed sample is as a function of possible parameter values. Therefore, maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE is usually recommended for large samples because it is versatile, applicable to most models and different types of data, and produces the most precise estimates.

Least squares estimation method (LSE)

Least squares estimates are calculated by fitting a regression line to the points from a data set that has the minimal sum of the deviations squared (least square error). In reliability analysis, the line and the data are plotted on a probability plot.

Bayes Optimal Classifier

The Bayes optimal classifier is a probabilistic model that makes the most probable prediction for a new example, given the training dataset.

This model is also referred to as the Bayes optimal learner, the Bayes classifier, Bayes optimal decision boundary, or the Bayes optimal discriminant function.

Gibbs Sampling Algorithm

We start off by selecting an initial value for the random variables ***X*** & ***Y***. Then, we sample from the conditional probability distribution of X given Y = Y^0 denoted $p(X|Y^0)$. In the next step, we sample a new value of Y conditional on X^1, which we just computed. We repeat the procedure for an additional ***n - 1*** iterations, alternating between drawing a new sample from the conditional probability distribution of X and the conditional probability distribution of Y, given the current value of the other random variable.

Let's take a look at an example. Suppose we had the following posterior and conditional probability distributions.

initialize Y^0, X^0
for j = 1, 2, 3,... do
    sample $X^j \sim p(X|Y^{j-1})$
    sample $Y^j \sim p(Y|X^j)$
end for

$$p(x, y) = \frac{1}{C} e^{-\frac{x^2y^2 + x^2 + y^2 - 8x - 8y}{2}}$$

$$p(x|y) = g(y) e^{-(x - \frac{4}{1+y^2})^2 (\frac{1+y^2}{2})}$$

$$p(y|x) = g(x) e^{-(y - \frac{4}{1+x^2})^2 (\frac{1+x^2}{2})}$$

$$\mathcal{N}(\mu, \sigma)$$

$$y|x \sim \mathcal{N}(\frac{4}{1+x^2}, \sqrt{\frac{1}{1+x^2}})$$

$$x|y \sim \mathcal{N}(\frac{4}{1+y^2}, \sqrt{\frac{1}{1+y^2}})$$

Naive Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.

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

EXAMPLE

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

Outlook		**Play**
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes

7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

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

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

P(Sunny)= 0.35

P(Yes)=0.71

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

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

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

P(No)= 0.29

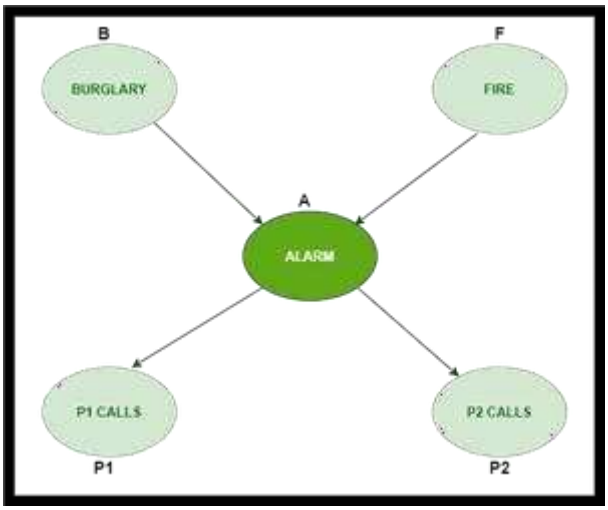
P(Sunny)= 0.35

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

Bayesian Belief Network:

It is a graphical representation of different probabilistic relationships among random variables in a particular set. It is a classifier with no dependency on attributes i.e it is condition independent. Due to its feature of joint probability, the probability in Bayesian Belief Network is derived, based on a condition — P(attribute/parent) i.e probability of an attribute, true over parent attribute.

Consider this example:



- In the above figure, we have an alarm 'A' – a node, say installed in a house of a person 'gfg', which rings upon two probabilities i.e burglary 'B' and fire

'F', which are – parent nodes of the alarm node. The alarm is the parent node of two probabilities P1 calls 'P1' & P2 calls 'P2' person nodes.

- Upon the instance of burglary and fire, 'P1' and 'P2' call person 'gfg', respectively. But, there are few drawbacks in this case, as sometimes 'P1' may forget to call the person 'gfg', even after hearing the alarm, as he has a tendency to forget things, quick. Similarly, 'P2', sometimes fails to call the person 'gfg', as he is only able to hear the alarm, from a certain distance.

Expectation-Maximization Algorithm

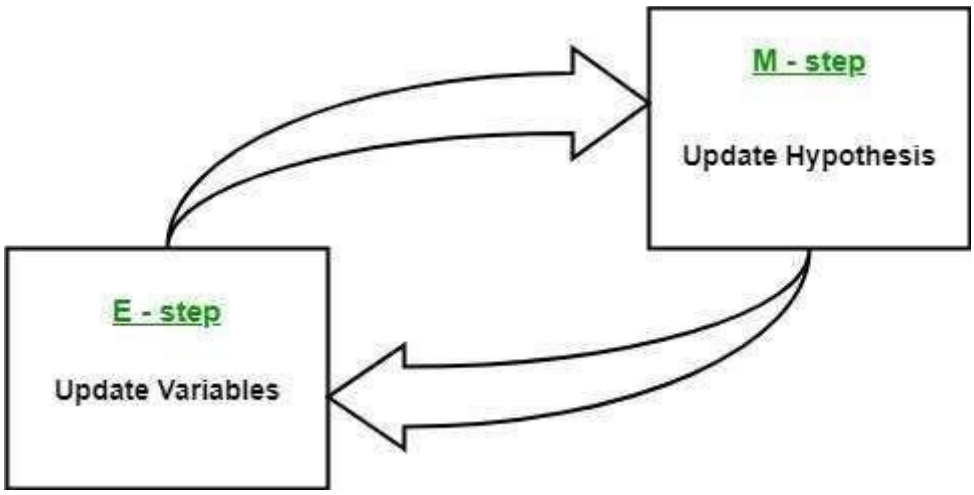
In the real-world applications of machine learning, it is very common that there are many relevant features available for learning but only a small subset of them are observable. So, for the variables which are sometimes observable and sometimes not, then we can use the instances when that variable is visible is observed for the purpose of learning and then predict its value in the instances when it is not observable.

On the other hand, ***Expectation-Maximization algorithm*** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

Algorithm:

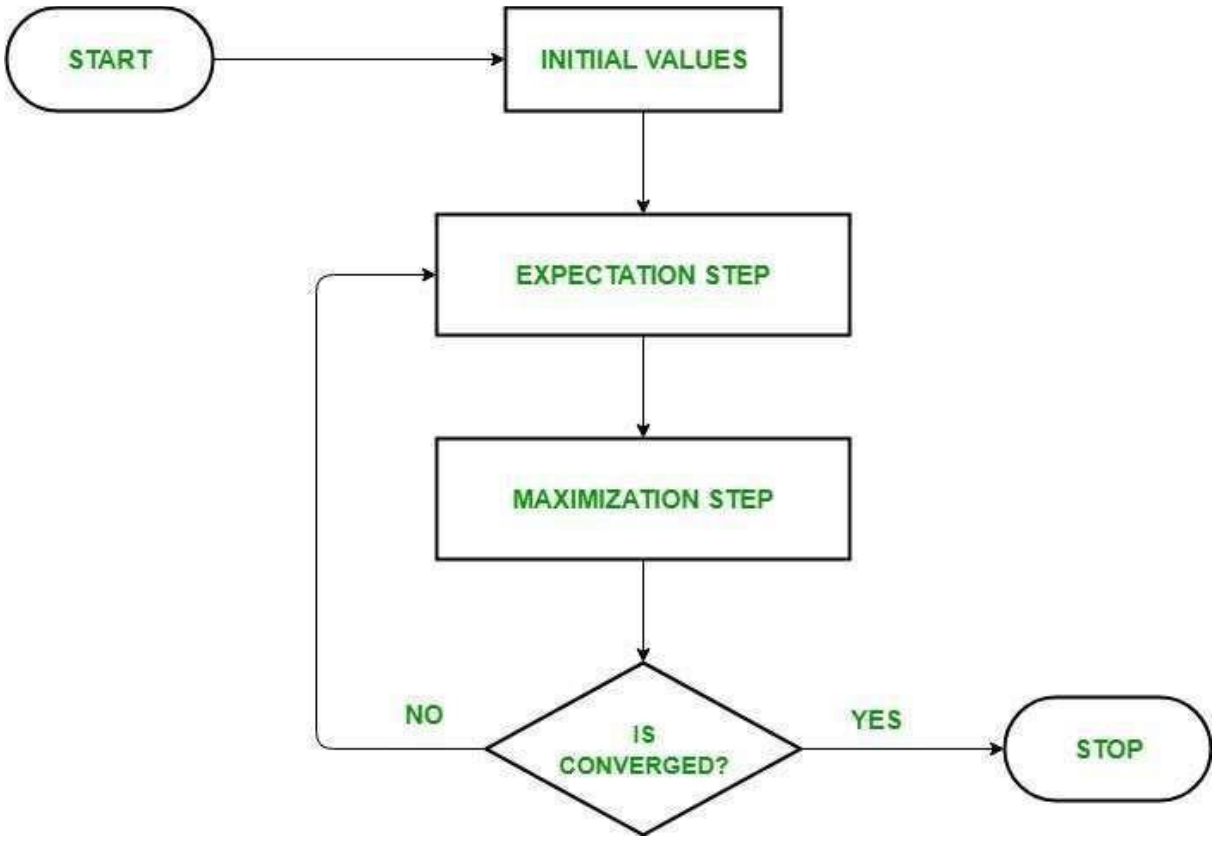
1. Given a set of incomplete data, consider a set of starting parameters.
2. **Expectation step (E – step):** Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. **Maximization step (M – step):** Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.



The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

- Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.
- The next step is known as "Expectation" – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.
- The next step is known as "Maximization"-step or *M-step*. In this step, we use the complete data generated in the preceding "Expectation" – step in order to update the values of the parameters. It is basically used to update the hypothesis.
- Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. "Expectation" – step and
  "Maximization" – step until the convergence occurs.

Flow chart for EM algorithm



Usage of EM algorithm

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

Instance-based learning

The Machine Learning systems which are categorized as **instance-based learning** are the systems that learn the training examples by heart and then generalizes to new instances based on some similarity measure. It is called instance-based because it builds the hypotheses from the training instances. It is also known as **memory-based learning** or **lazy-learning.** The time complexity of this algorithm depends upon the size of training data. The worst-case time complexity of this algorithm is **O (n)**, where n is the number of training instances.

For example, If we were to create a spam filter with an instance-based learning algorithm, instead of just flagging emails that are already marked as spam emails, our spam filter would be programmed to also flag emails that are very similar to them. This requires a measure of resemblance between two emails. A similarity measure between two emails could be the same sender or the repetitive use of the same keywords or something else.

Advantages:

1. Instead of estimating for the entire instance set, local approximations can be made to the target function.
2. This algorithm can adapt to new data easily, one which is collected as we go.

Disadvantages:

1. Classification costs are high
2. Large amount of memory required to store the data, and each query involves starting the identification of a local model from scratch.

Some of the instance-based learning algorithms are :

1. K Nearest Neighbor (KNN)
2. Self-Organizing Map (SOM)
3. Learning Vector Quantization (LVQ)
4. Locally Weighted Learning (LWL)

K-Nearest Neighbor(KNN) Algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Working of KNN Algorithm

K-nearest neighbours (KNN) algorithm uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

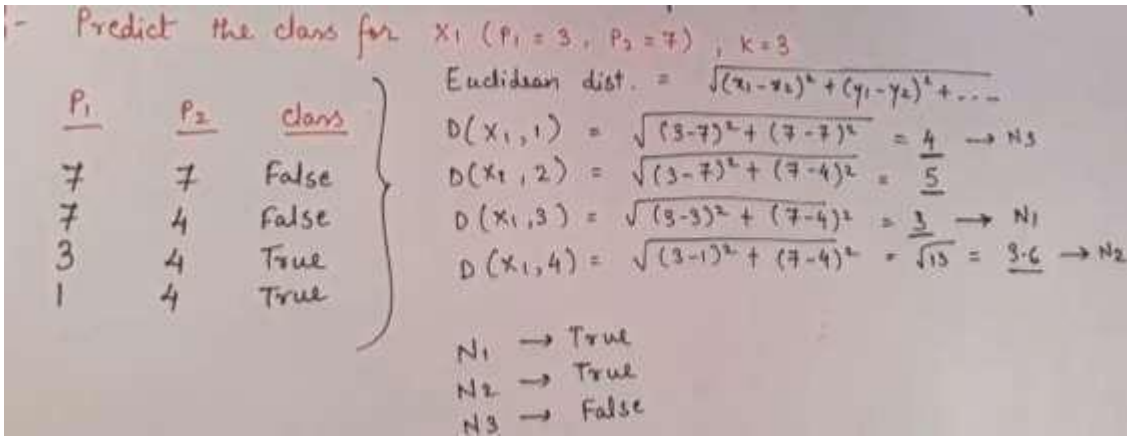
Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** − Now, based on the distance value, sort them in ascending order.
- **3.3** − Next, it will choose the top K rows from the sorted array.
- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

EXAMPLE :



Case Based Reasoning

As we know **Nearest Neighbour classifiers** stores training tuples as points in Euclidean space. But **Case-Based Reasoning classifiers (CBR)** use a database of problem solutions to solve new problems. It stores the tuples or cases for problem-solving as complex symbolic descriptions.

How CBR works?

When a new case arrises to classify, a Case-based Reasoner(CBR) will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the CBR will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbours of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The CBR tries to combine the solutions of the neighbouring training cases to propose a solution for the new case. If compatibilities arise with the individual solutions, then backtracking to search for other solutions

may be necessary. The CBR may employ background knowledge and problem-solving strategies to propose a feasible solution.

Applications of CBR includes:

1. Problem resolution for customer service help desks, where cases describe product-related diagnostic problems.
2. It is also applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively.
3. Medical educations, where patient case histories and treatments are used to help diagnose and treat new patients.

Challenges with CBR

- Finding a good similarity metric (eg for matching subgraphs) and suitable methods for combining solutions.
- Selecting salient features for indexing training cases and the development of efficient indexing techniques.

CBR becomes more intelligent as the number of the trade-off between accuracy and efficiency evolves as the number of stored cases becomes very large. But after a certain point, the system's efficiency will suffer as the time required to search for and process relevant cases increases.

Some differences on eager and lazy learning

- Eager learning methods construct general, explicit description of the target function based on the provided training examples.
- Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.
- Lazy learning methods can construct a different approximation to the target function for each encountered query instance.

Lazy learning is very suitable for complex and incomplete problem domains, where a complex target function can be represented by a collection of less complex local approximations.

Eager learning methods use the same approximation to the target function, which must be learned based on training examples and before input queries are observed.

UNIT - IV

PATTERN COMPARISON TECHNIQUES

Pattern recognition is a **process of finding regularities** and similarities in data using machine learning data. Now, these similarities can be found based on statistical analysis, historical data, or the already gained knowledge by the machine itself. A pattern is a regularity in the world or in abstract notions. If we discuss sports, a description of a type would be a pattern. If a person keeps watching videos related to cricket, YouTube wouldn't recommend them chess tutorials videos.

Examples: Speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis.

Before searching for a pattern there are some certain steps and the first one is to collect the data from the real world. The collected data needs to be filtered and preprocessed so that its system can extract the features from the data. Then based on the type of the data system will choose the appropriate algorithm among Classification, Regression, and Regression to recognize the pattern.

- **Classification.** In classification, the algorithm assigns labels to data based on the predefined features. This is an example of supervised learning.
- **Clustering.** An algorithm splits data into a number of clusters based on the similarity of features. This is an example of unsupervised learning.
- **Regression.** Regression algorithms try to find a relationship between variables and predict unknown dependent variables based on known data. It is based on supervised learning. [2]
- **Features** can be represented as continuous, discrete, or discrete binary variables. A feature is basically a function of one or more measurements, computed to quantify the significant characteristics of the object. The feature is one of the most important components in the Pattern Recognition system. **Example:** consider a football, shape, size and color, etc. are features of the football.

A feature vector is a set of features that are taken together.

Example: In the above example of football, if all the features (shape, size, color etc.) taken together then the sequence is feature vector ([shape, size, color]). The feature vector is the sequence of features represented as an n-dimensional column vector. In the case of speech, MFCC (Mel-frequency Cepstral Coefficient) is the spectral features of the speech. The sequence of the first 13 features forms a feature vector.

Temporal patterns

Temporal patterns are one of the pattern comparison techniques that is defined as **a segment of signals that recurs frequently in the whole temporal signal sequence**. For example, the temporal signal sequences could be the movements of head, hand, and body, a piece of music, and so on.

Temporal abstraction and data mining are two research fields that have tried to synthesis time oriented data and bring out an understanding on the hidden relationships that may exist between time oriented events. In clinical settings, having the ability to know the hidden relationships on patient data as they unfold could help save a life by aiding in detection of conditions that are not obvious to clinicians and healthcare workers. Understanding the hidden patterns is a huge challenge due to the exponential search space unique to time-series data. In this paper, we propose a temporal pattern recognition model based on dimension reduction and similarity measures thereby maintaining the temporal nature of the raw data

INTRODUCTION

Temporal pattern processing is important for various intelligent behaviours, including hearing, vision, speech, music and motor control. Because we live in an ever-changing environment, an intelligent system, whether it be a human or a robot, must encode patterns over time, recognize and generate temporal patterns. Time is embodied in a temporal pattern in two different ways: • Temporal order. It refers to the ordering among the components of a sequence. For example, the sequence N-E-T is different from T-E-N. Temporal order may

also refer to a syntactic structure, such as subject-verb-object, where each component may be any of a category of possible symbols

• Time duration. Duration can play a critical role for temporal processing. In speech recognition, for example, we want rate invariance while distinguishing relative durations of the vowel /i:/ (as in beet) and /i/ (as in bit)

TEMPORAL PATTERN RECOGNITION

The shared goal of all STM models is to make input history available simultaneously when recognition takes place. With a STM model in place, recognition is not much different from the recognition of static patterns.

Template Matching Using Hebbian Learning

The architecture for this type of recognition is simply a two-layer network: the input layer that incorporates STM, and the sequence recognition layer where each unit encodes an individual sequence. The recognition scheme is essentially template matching, where templates are formed through following Hebbian learning

$$W_{ij}(t) = W_{ij}(t-1) + C\, s_i(t)[x_j(t) - W_{ij}(t-1)]$$

where Wij is the connection weight from unit xj in the input layer to sequence recognizer si in the recognition layer. Parameter C controls learning rate. Hebbian learning is applied after the presentation of the entire sequence is completed. The templates thus formed can be used to recognize specific input sequences. The recognition layer typically includes recurrent connections for selecting a winner by self-organization (e.g. winner-take-all) during training or recognition.

Associative Memory Approach

The dynamics of the Hopfield associative memory model can be characterized as evolving towards the memory state most similar to the current input pattern.

If one views each memory state as a category, the Hopfield net performs pattern recognition: the recalled category is the recognized pattern. This process of dynamic evolution can also be viewed as an optimization process, which minimizes a cost function until equilibrium is reached.

With normalized exponential kernel STM, Tank and Hopfield (1987) described a recognition network based on associative memory dynamics. A layer of sequence recognizers receives inputs from the STM model. Each recognizer encodes a different template sequence by its unique weight vector acting upon the inputs in STM. In addition, recognizers form a competitive network. The recognition process uses the current input sequence (evidence) to bias a minimization process so that the most similar template wins the competition, thus activating its corresponding recognizer. Due to the exponential kernels, they demonstrated that recognition is fairly robust to time warping, distortions in duration. A similar architecture is later applied to speakerindependent spoken digit recognition.

Multilayer Perceptrons

A popular approach to temporal pattern learning is multilayer perceptrons (MLP). MLPs have been demonstrated to be effective for static pattern recognition. It is natural to combine MLP with an STM model to do temporal pattern recognition. For example, using delay line STM Waibel et al. (1989) reported an architecture called Time Delay Neural Networks (TDNN) for spoken phoneme recognition. Besides the input layer, TDNN uses 2 hidden layers and an output layer where each unit encodes one phoneme. The feed forward connections converge from the input layer to each successive layer so that each unit in a specific layer receives inputs within a limited time window from the previous layer. They demonstrated good recognition performance: for the three stop consonants /b/, /d/, and /g/, the accuracy of speaker dependent recognition reached 98.5%.

DYNAMIC TIME WARPING

Sounds like time traveling or some kind of future technic, however, it is not. Dynamic Time Warping is used to compare the similarity or calculate the

distance between two arrays or time series with different length. Suppose we want to calculate the distance of two equal-length arrays:

a = [1, 2,
3] b = [3,
2, 2]

How to do that? One obvious way is to match up a and b in 1-to-1 fashion and sum up the total distance of each component. This sounds easy, but what if a and b have different lengths?

a = [1, 2, 3] b
= [2, 2, 2, 3,
4]

How to match them up? Which should map to which? To solve the problem, there comes dynamic time warping. Just as its name indicates, to warp the series so that they can match up.

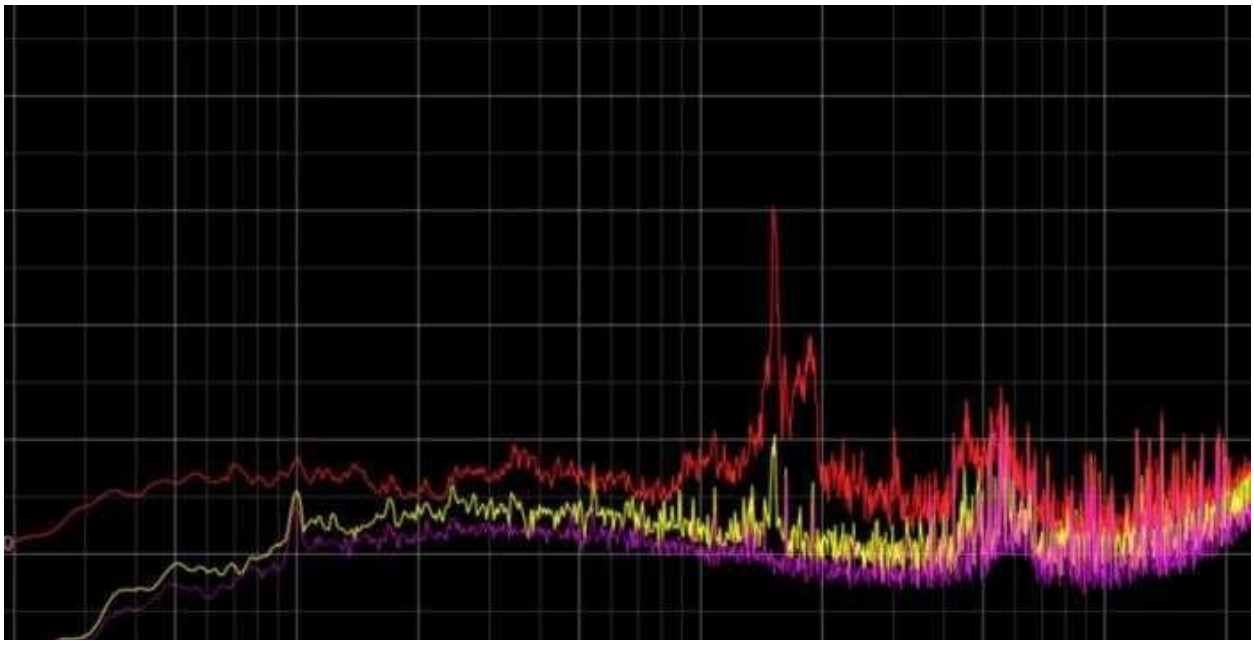
Use Cases

Before digging into the algorithm, you might have the question that is it useful? Do we really need to compare the distance between two unequal-length time series?

Yes, in a lot of scenarios DTW is playing a key role.

Sound Pattern Recognition

One use case is to detect the sound pattern of the same kind. Suppose we want to recognise the voice of a person by analysing his sound track, and we are able to collect his sound track of saying **Hello** in one scenario. However, people speak in the same word in different ways, what if he speaks hello in a much slower pace like **Heeeeeeellooooooo** , we will need an algorithm to match up the sound track of different lengths and be able to identify they come from the same person.



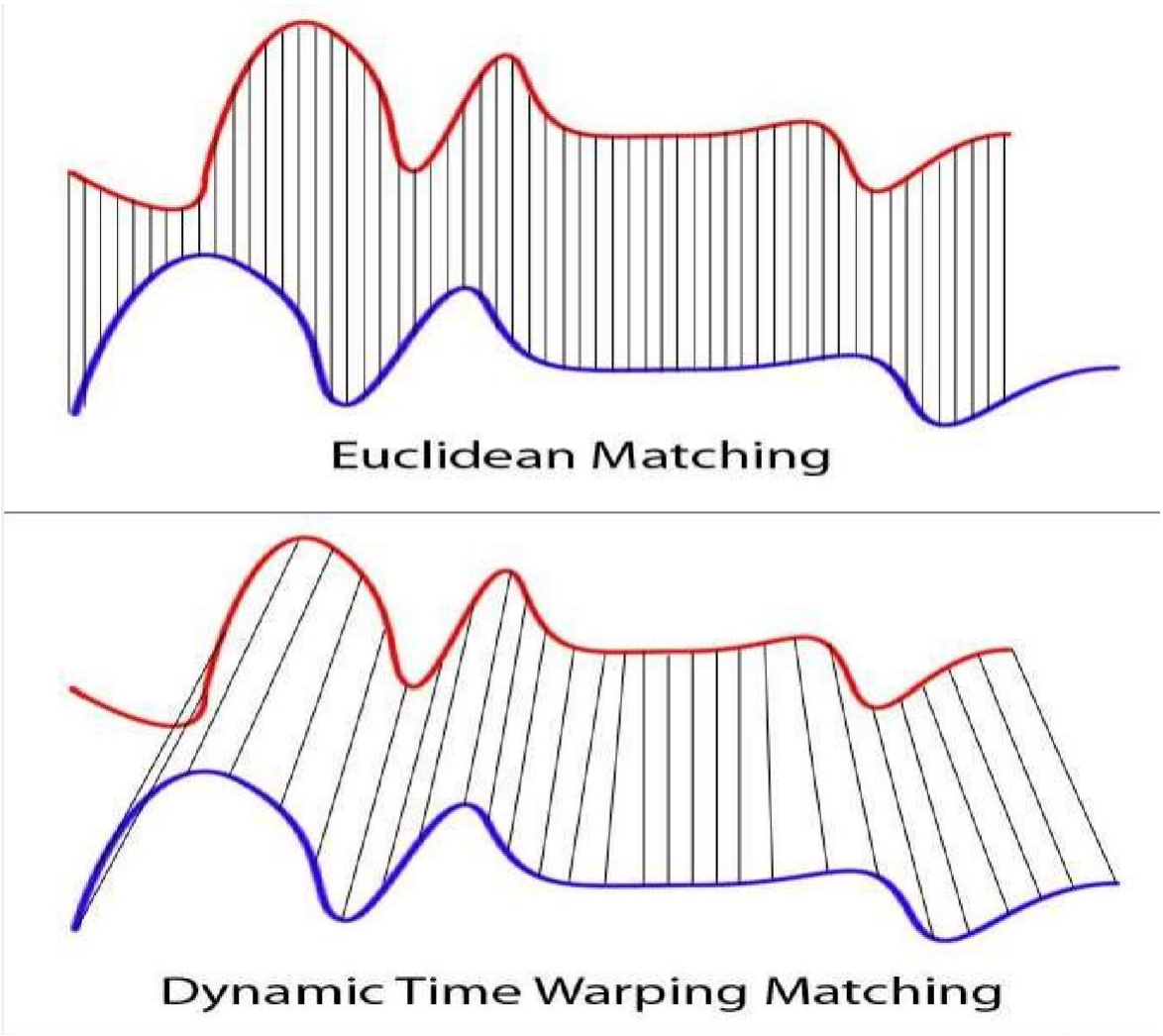
Stock Market

In a stock market, people always hope to be able to predict the future, however using general machine learning algorithms can be exhaustive, as most prediction task requires test and training set to have the same dimension of features. However, if you ever speculate in the stock market, you will know that even the same pattern of a stock can have very different length reflection on klines and indicators.

In time series analysis, dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. DTW has been applied to temporal sequences of video, audio, and graphics data — indeed, any data that can be turned into a linear sequence can be analysed with DTW.

The idea to compare arrays with different length is to build one-to-many and many-to-one matches so that the total distance can be minimised between the two.

Suppose we have two different arrays red and blue with different length:



Clearly these two series follow the same pattern, but the blue curve is longer than the red. If we apply the one-to-one match, shown in the top, the mapping is not perfectly synced up and the tail of the blue curve is being left out.

DTW overcomes the issue by developing a one-to-many match so that the troughs and peaks with the same pattern are perfectly matched, and there is no left out for both curves(shown in the bottom top).

Rules

In general, DTW is a method that calculates an optimal match between two given sequences (e.g. time series) with certain restriction and rules(comes from wiki):

- Every index from the first sequence must be matched with one or more indices from the other sequence and vice versa
- The first index from the first sequence must be matched with the first index from the other sequence (but it does not have to be its only match)
- The last index from the first sequence must be matched with the last index from the other sequence (but it does not have to be its only match)
- The mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing, and vice versa, i.e. if $j > i$ are indices from the first sequence, then

  there must not be two indices $l > k$ in the other sequence, such

  that index i is matched with index l and index j is matched with index k, and vice versa.

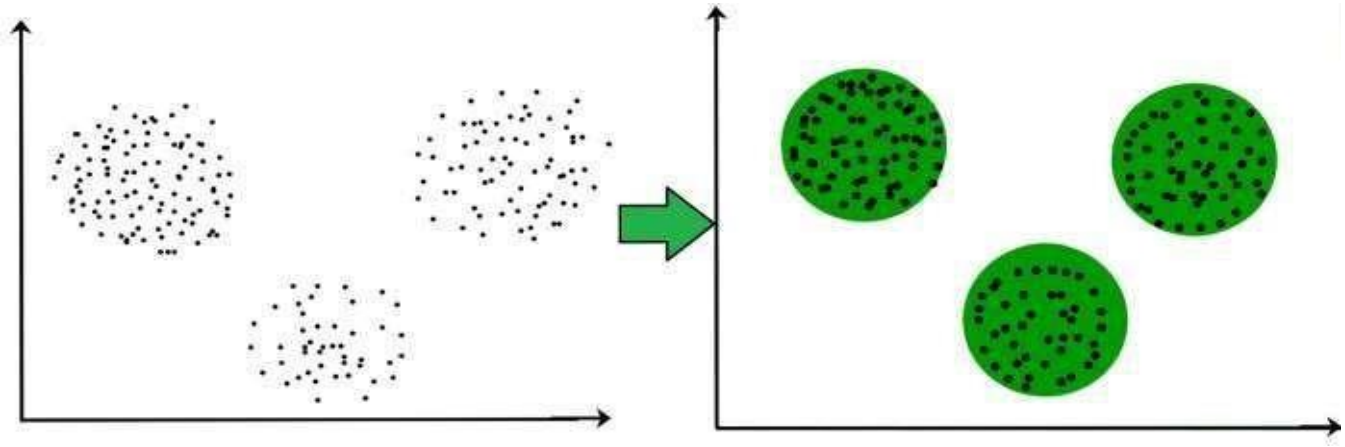
The optimal match is denoted by the match that satisfies all the restrictions and the rules and that has the minimal cost, where the cost is computed as the sum of absolute differences, for each matched pair of indices, between their values.

Introduction to Clustering:

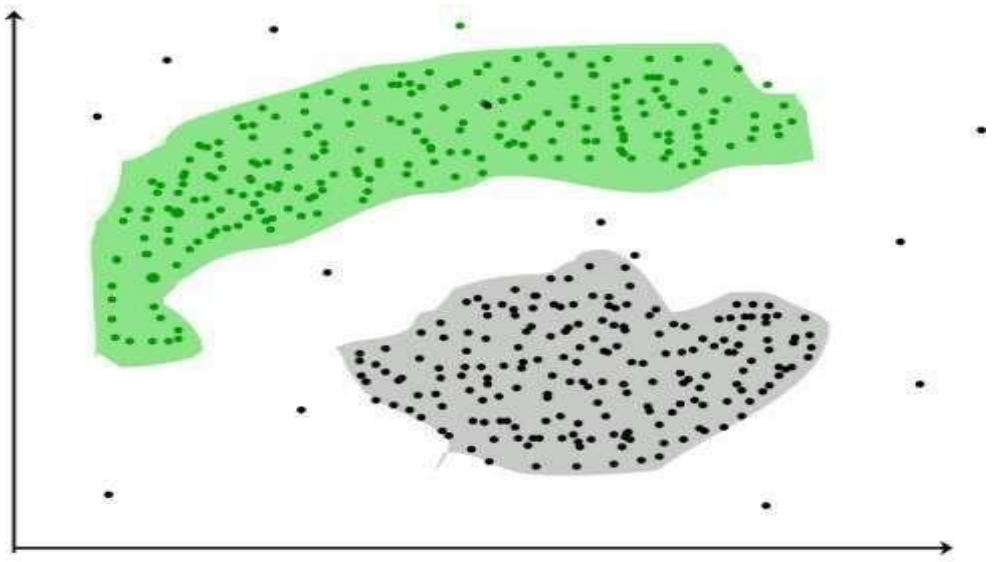
It is basically a type of *unsupervised learning method*. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



It is not necessary for clusters to be spherical. Such as:



DBSCAN: Density-based Spatial Clustering of Applications with Noise

These data points are clustered by using the basic concept that the data point lies within the given constraint from the cluster center. Various distance methods and techniques are used for the calculation of the outliers.

Why Clustering?

Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding "natural clusters" and describe their unknown properties ("natural" data types), in finding useful and suitable groupings ("useful" data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods :

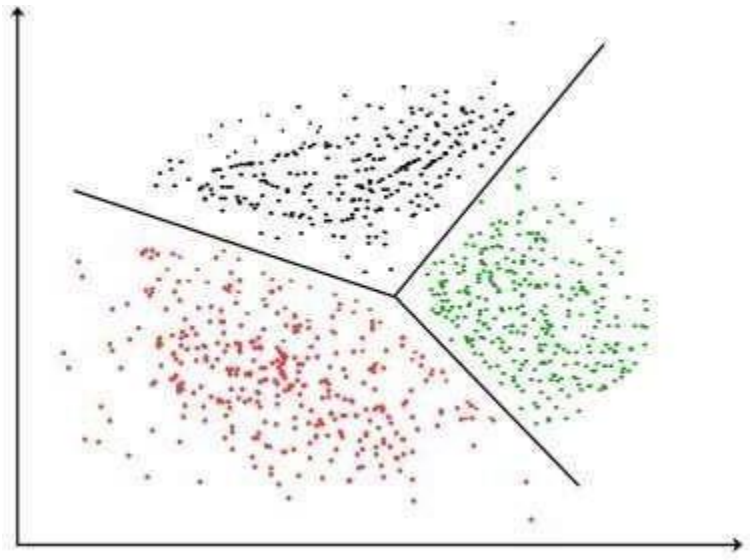
- **Density-Based Methods:** These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example *DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure)*, etc.
- **Hierarchical Based Methods:** The clusters formed in this method form a treetype structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
- **Agglomerative** (bottom-up *approach*)
- **Divisive** (top-down *approach*)

examples *CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies)*, etc.

- **Partitioning Methods:** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means, CLARANS (Clustering Large Applications based upon Randomized Search)*, etc.
- **Grid-based Methods:** In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects example *STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest)*, etc.

K means Clustering:

It is the simplest unsupervised learning algorithm that solves clustering problem.K-means algorithm partitions n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster.



Applications of Clustering in different fields

- **Marketing:** It can be used to characterize & discover customer segments for marketing purposes.
- **Biology:** It can be used for classification among different species of plants and animals.
- **Libraries:** It is used in clustering different books on the basis of topics and information.
- **Insurance:** It is used to acknowledge the customers, their policies and identifying the frauds.
- **City Planning:** It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

- **Earthquake studies:** By learning the earthquake-affected areas we can determine the dangerous zones.

  The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement. The algorithm works as follows:

1. First, we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

   The "points" mentioned above are called means because they hold the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature *x* the items have values in [0,3], we will initialize the means with values for *x* at [0,3]).

The above algorithm in pseudocode:

```
Initialize k means with random values

For a given number of iterations:
    Iterate through items:
        Find the mean closest to the item
        Assign item to mean
        Update mean
```

K-MODE CLUSTERING

KModes clustering is one of the unsupervised Machine Learning algorithms that is used to cluster categorical variables.

How does the KModes algorithm work?

1. Pick K observations at random and use them as leaders/clusters
2. Calculate the dissimilarities and assign each observation to its closest cluster
3. Define new modes for the clusters

4. Repeat 2–3 steps until there are is no re-assignment required

Example: Imagine we have a dataset that has the information about hair color, eye color, and skin color of persons. We aim to group them based on the available information(maybe we want to suggest some styling ideas)

Hair color, eye color, and skin color are all categorical variables. Below is how our dataset looks like.

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Alright, we have the sample data now. Let us proceed by defining the number of clusters(K)=3

Step 1: Pick K observations at random and use them as leaders/clusters

I am choosing P1, P7, P8 as leaders/clusters

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Leaders and Observations

Step 2: Calculate the dissimilarities(no. of mismatches) and assign each observation to its closest cluster

Iteratively compare the cluster data points to each of the observations. Similar data points give 0, dissimilar data points give 1.

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing leader/Cluster P1 to the observation P1 gives 0 dissimilarities

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing leader/cluster P1 to the observation P2 gives 3(1+1+1)

dissimilarities. Likewise, calculate all the dissimilarities and put them in a matrix as shown below and assign the observations to their closest cluster (cluster that has the least dissimilarity)

	Cluster 1 (P1)	Cluster 2 (P7)	Cluster 3 (P8)	Cluster
P1	0 ✓	2	2	Cluster 1
P2	3 ✓	3	3	Cluster 1
P3	3	1 ✓	3	Cluster 2
P4	3	3	1 ✓	Cluster 3
P5	1 ✓	2	2	Cluster 1
P6	3	3	2 ✓	Cluster 3
P7	2	0 ✓	2	Cluster 2
P8	2	2	0 ✓	Cluster 3

Dissimilarity matrix (Image by Author)

After step 2, the observations P1, P2, P5 are assigned to cluster 1; P3, P7 are assigned to Cluster 2; and P4, P6, P8 are assigned to cluster 3.

Step 3: Define new modes for the clusters

Mode is simply the most observed value. Mark the observations according to the cluster they belong to. Observations of Cluster 1 are marked in Yellow, Cluster 2 are marked in Brick red, and Cluster 3 are marked in Purple.

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Considering one cluster at a time, for each feature, look for the Mode and update the new leaders.

Explanation: Cluster 1 observations(P1, P2, P5) has brunette as the most observed hair color, amber as the most observed eye color, and fair as the most observed skin color.
Below are our new leaders after the update.

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

Repeat steps 2–4 : After obtaining the new leaders, again calculate the dissimilarities between the observations and the newly obtained leaders.

New Leaders			
	hair color	**eye color**	**skin color**
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	**hair color**	**eye color**	**skin color**
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing Cluster 1 to the observation P1 gives 1 dissimilarity.

New Leaders			
	hair color	**eye color**	**skin color**
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	**hair color**	**eye color**	**skin color**
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Comparing Cluster 1 to the observation P2 gives 2 dissimilarities.

Likewise, calculate all the dissimilarities and put them in a matrix. Assign each observation to its closest cluster.

	Cluster 1	Cluster 2	Cluster 3	Cluster
P1	1 ✓	2	3	Cluster 1
P2	2 ✓	3	2	Cluster 1
P3	3	1 ✓	2	Cluster 2
P4	3	3	0 ✓	Cluster 3
P5	0 ✓	2	3	Cluster 1
P6	3	3	1 ✓	Cluster 3
P7	2	0 ✓	3	Cluster 2
P8	2	2	1 ✓	Cluster 3

The observations P1, P2, P5 are assigned to Cluster 1; P3, P7 are assigned to Cluster 2; and P4, P6, P8 are assigned to Cluster 3.

We stop here as we see there is no change in the assignment of observations.

Implementation of KModes in Python:

Begin with Importing necessary libraries

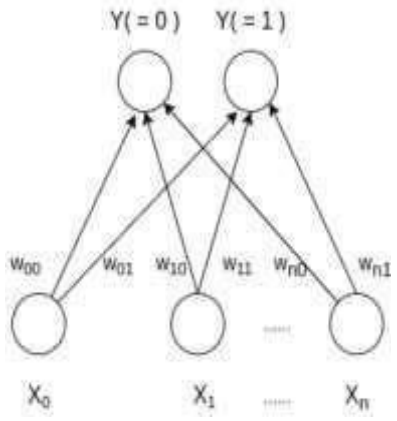
```
# importing necessary libraries
import pandas as pd
import numpy as np
# !pip install kmodes
from kmodes.kmodes import KModes
import matplotlib.pyplot as plt
%matplotlib inline
```

Creating toy dataset

```
# Create toy dataset
hair_color = np.array(['blonde', 'brunette', 'red', 'black', 'brunette', 'black', 'red', 'black'
eye_color = np.array(['amber', 'gray', 'green', 'hazel', 'amber', 'gray', 'green', 'hazel'])
skin_color = np.array(['fair', 'brown', 'brown', 'brown', 'fair', 'brown', 'fair', 'fair'])
person = ['P1','P2','P3','P4','P5','P6','P7','P8']
data = pd.DataFrame({'person':person, 'hair_color':hair_color, 'eye_color':eye_color, 'skin_col
data = data.set_index('person')
data
```

Vector Quantization

Learning Vector Quantization (or LVQ) is a type of Artificial Neural Network which also inspired by biological models of neural systems. It is based on prototype supervised learning classification algorithm and trained its network through a competitive learning algorithm similar to Self Organizing Map. It can also deal with the multiclass classification problem. LVQ has two layers, one is the Input layer and the other one is the Output layer. The architecture of the Learning Vector Quantization with the number of classes in an input data and n number of input features for any sample is given below:



Let say an input data of size (m, n) where m is number of training example and n is the number of features in each example and a label vector of size (m, 1). First, it initializes the weights of size (n, c) from the first c number of training samples with different labels and should be discarded from all training samples. Here, c is the number of classes. Then iterate over the remaining input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by :

$w_{ij} = w_{ij}(old) - alpha(t) * (x_i^k - w_{ij}(old))$

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the i^{th} feature of training example and k denotes the k^{th} training example from the input data. After training the LVQ network, trained weights are used for classifying new examples. A new example labeled with the class of winning vector.

Algorithm

Steps involved are :

- Weight initialization
- For 1 to N number of epochs
- Select a training example
- Compute the winning vector
- Update the winning vector
- Repeat steps 3, 4, 5 for all training example.
- Classify test sample

UNIT- V

Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Different search methods for induction

In the field of machine learning, an induction algorithm represents an example of using mathematical principles for the development of sophisticated computing systems. Machine learning systems go beyond a simple "rote input/output" function, and evolve the results that they supply with continued use. Induction algorithms can help with the real-time handling of sophisticated data sets, or more long-term efforts.

The induction algorithm is something that applies to systems that show complex results depending on what they are set up for. One of the most fundamental ways that engineers use an induction algorithm is to enhance knowledge acquisition in a given system. In other words, with the algorithm in place, the set of "knowledge data" that end users get is somehow improved, whether that's regarding the quantity of data, the filtering of noise and undesirable results, or the refinement of some data points.

Although the technical descriptions of induction algorithms are largely the territory of mathematical and scientific journals, one of the basic ideas about using the induction algorithm is that it can organize "classification rules" according to the induction principle and separate corollary results from different kinds of

system noise or exceptions. Filtering out noise from a domain is a prominent use of the induction algorithm in general. There is the idea that in real-world data filtering, induction algorithms can compose different sets of rules for both the legitimate results and the system noise, in order to distinguish one from the other.

By setting up induction algorithms according to certain training examples, stakeholders are looking for the ability of these systems to identify and assess consistent rules and data that represents exceptions to these rules. In a sense, the use of an induction algorithm uses the induction principle to "prove" certain results that can aid knowledge, because they provide more marked delineations in a data set (or multiple data sets) – distinctions that can drive all sorts of end user capabilities.

Like other kinds of machine learning software, induction algorithms are often thought of as a form of "decision support."

"We consider the principal task of a real-world induction system to be assisting the expert in expressing his or her expertise," write the authors of a Turing Institute paper on induction in machine learning back in the 1980s. "Consequently, we require that the induced rules are highly predictive and are easily comprehensible to the expert."

With this in mind, induction algorithms can be part of many kinds of software products that seek to refine data and produce evolving results for human users. In general, machine learning and the use of visual dashboards is generating new tools through which users can more rapidly develop in-depth knowledge about any given system, whether it's related to marine research, medical diagnosis, e-commerce, or any other kind of data-rich system.

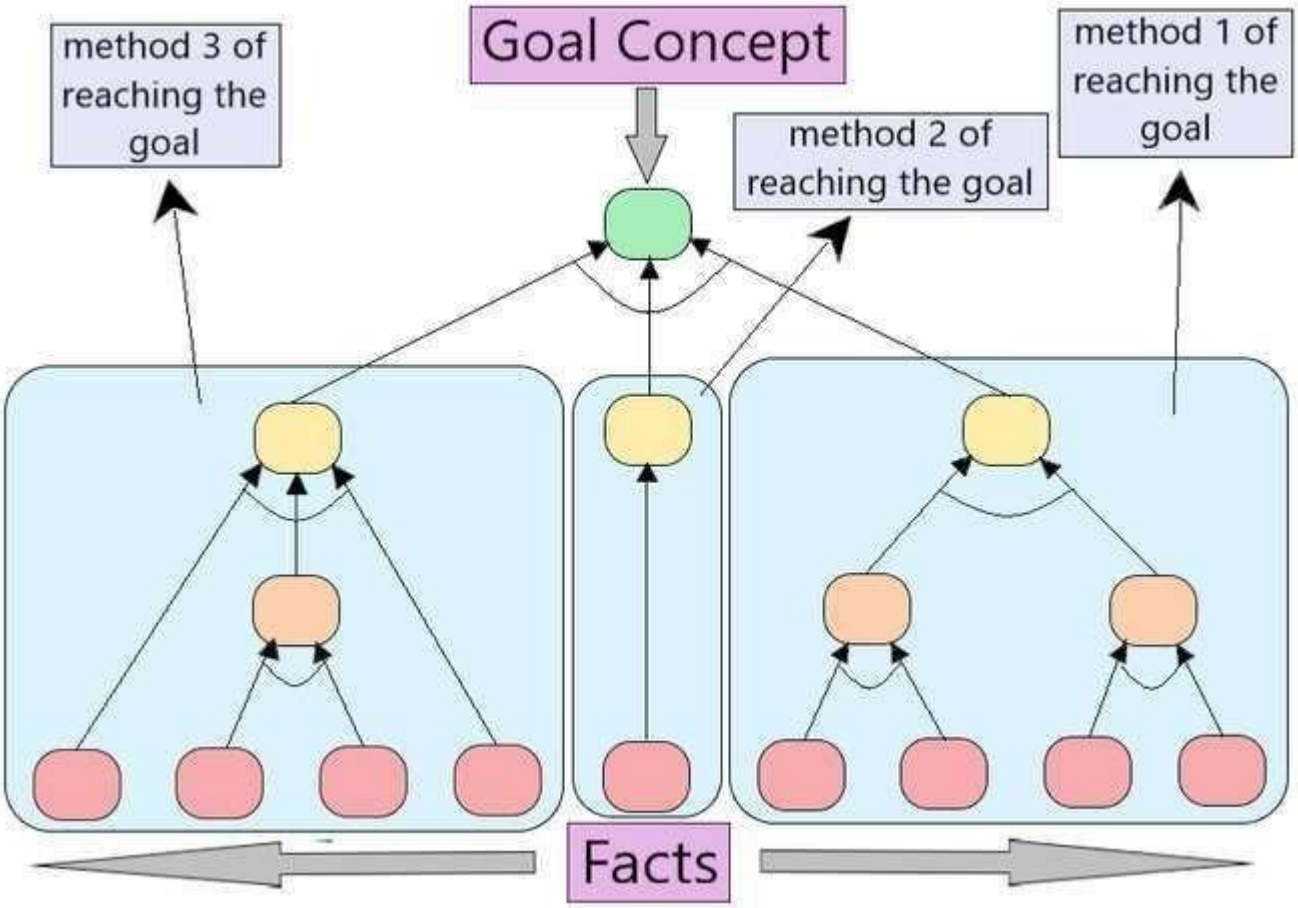
Explanation-Based Learning (EBL)

In simple terms, it is the ability to gain basic problem-solving techniques by observing and analysing solutions to specific problems. In terms of Machine Learning, it is an algorithm that aims to understand why an example is a part of a particular concept to make generalizations or form concepts from training examples. For example, EBL uses a domain theory and creates a program that learns to play chess. EBL involves 2 steps:

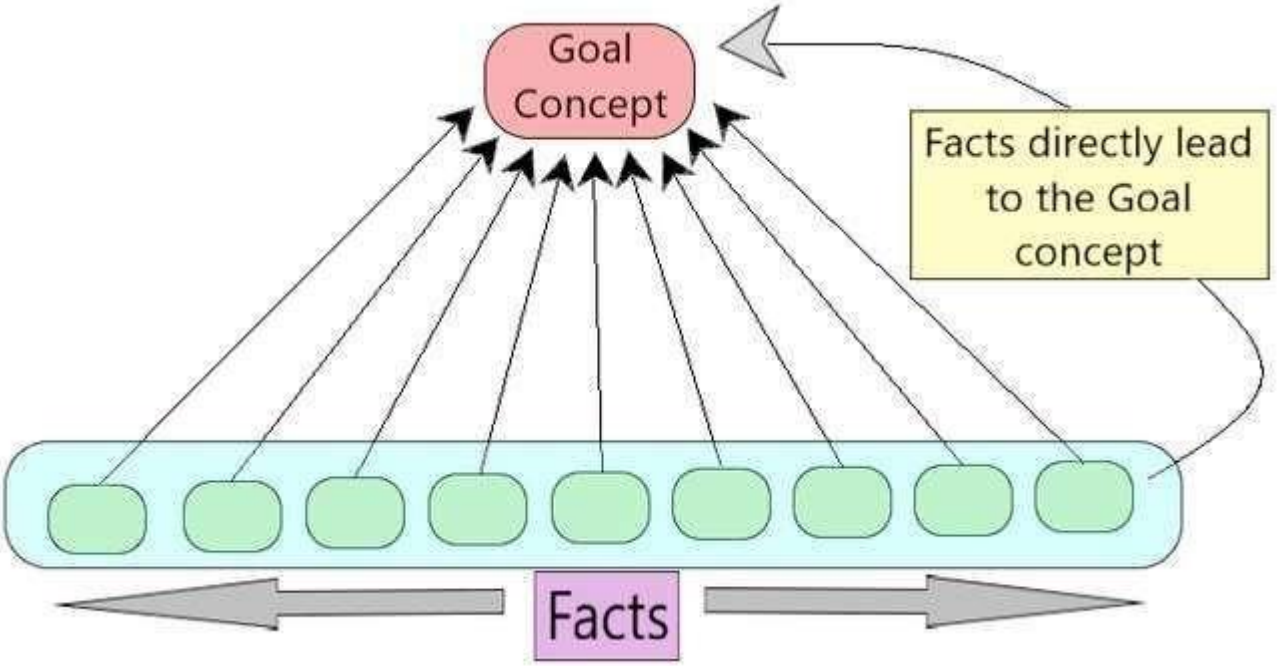
1. **Explanation —** The domain theory is used to eliminate all the unimportant training example while retaining the important ones that best describe the goal concept.
2. **Generalization —** The explanation of the goal concept is made as general and widely applicable as possible. This ensures that all cases are covered, not just certain specific ones.

EBL Architecture:

- **EBL model during training**
- During training, the model generalizes the training example in such a way that all scenarios lead to the Goal Concept, not just in specific cases. **(As shown in Fig 1)**

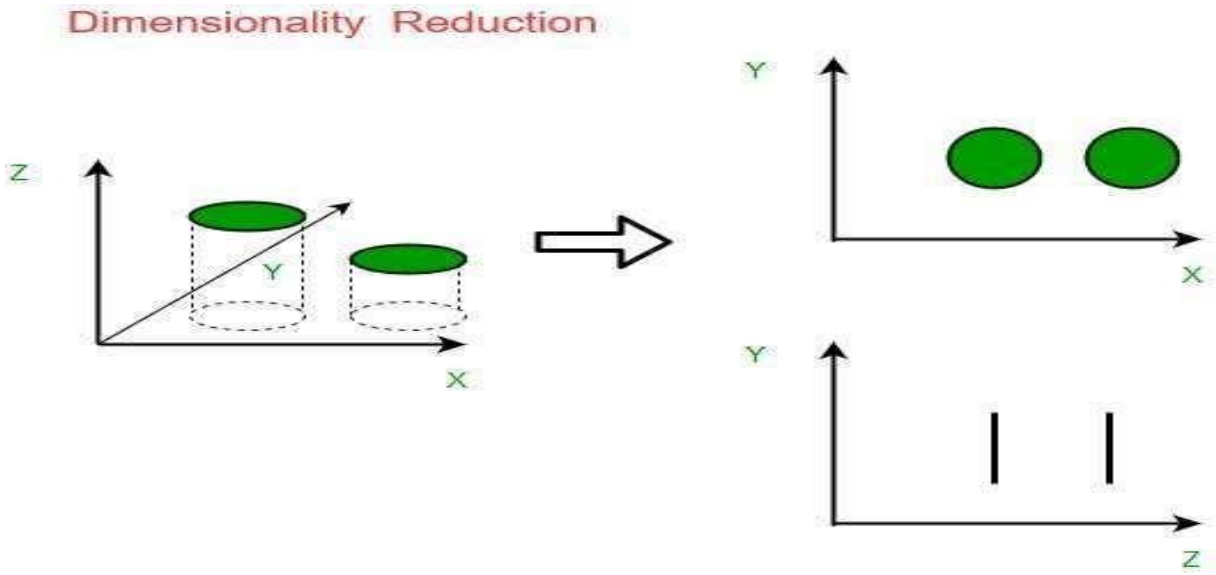


- **EBL model after training**
- Post training, EBL model tends to directly reach the hypothesis space involving the goal concept. **(As shown in Fig 2)**



Dimensionality Reduction

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:

1. Filter
2. Wrapper
3. Embedded

- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

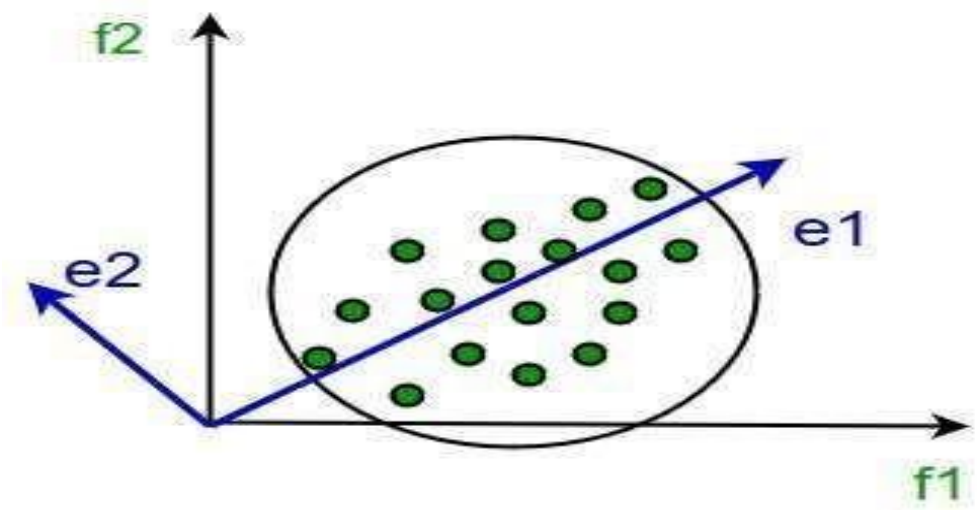
Methods of Dimensionality Reduction The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

  Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

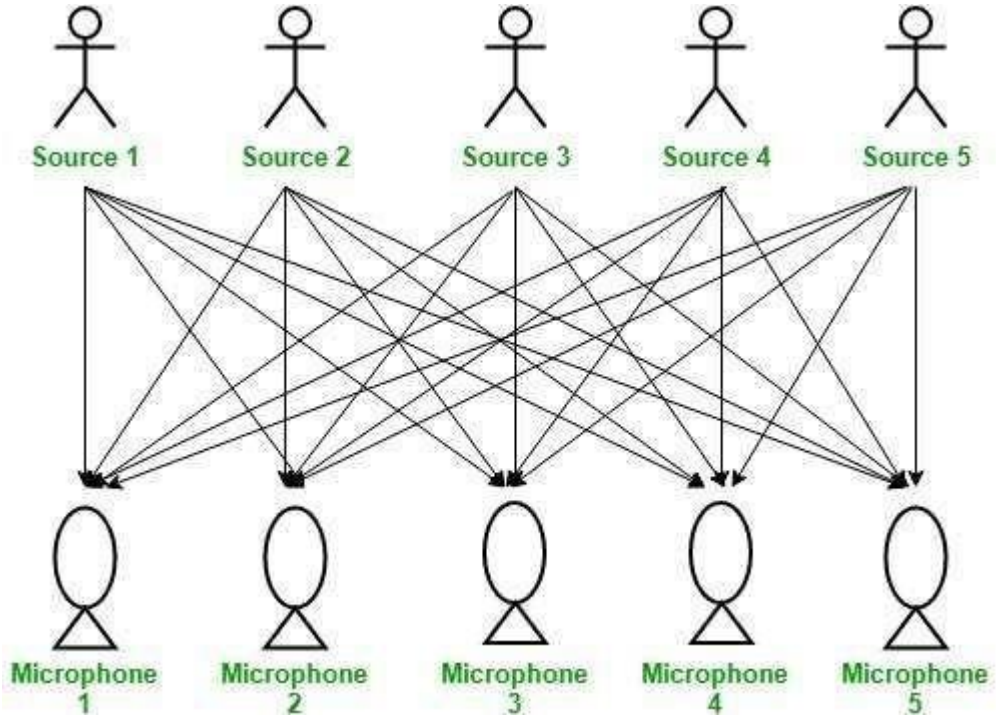
Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any. **Disadvantages of Dimensionality Reduction •** It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.

- We may not know how many principal components to keep- in practice, some thumb rules are applied.

Factor analysis.

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of observed variables called **factors**. For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors plus "error" terms, hence factor analysis can be thought of as a special case of errors-invariables models.



Here,There is a party going into a room full of people. There is ‘n’ number of speakers in that room and they are speaking simultaneously at the party. In the same room, there are also ‘n’ number of microphones placed at different

distances from the speakers which are recording 'n' speakers' voice signals. Hence, the number of speakers is equal to the number must of microphones in the room.

Now, using these microphones' recordings, we want to separate all the 'n' speakers' voice signals in the room given each microphone recorded the voice signals coming from each speaker of different intensity due to the difference in distances between them. Decomposing the mixed signal of each microphone's recording into independent source's speech signal can be done by using the machine learning technique, independent component analysis.

[X1, X2, ..., Xn] => [Y1, Y2, ..., Yn]

where, X1, X2, …, Xn are the original signals present in the mixed signal and Y1, Y2, …, Yn are the new features and are independent components which are independent of each other.

Restrictions on ICA

1. The independent components generated by the ICA are assumed to be statistically independent of each other.
2. The independent components generated by the ICA must have non-gaussian distribution.
3. The number of independent components generated by the ICA is equal to the number of observed mixtures.

Multidimensional scaling

Multidimensional scaling is a **visual representation of distances or dissimilarities between sets of objects**.

"Objects" can be colors, faces, map coordinates, political persuasion, or any kind of real or conceptual stimuli

(Kruskal and Wish, 1978). Objects that are more similar (or have shorter distances) are closer together on the graph than objects that are less similar (or have longer distances). As well as interpreting dissimilarities as distances on a

graph, MDS can also serve as a dimension reduction technique for high-dimensional data (Buja et. al, 2007).

The term **scaling** comes from psychometrics, where abstract concepts ("objects") are assigned numbers according to a rule (Trochim, 2006). For example, you may want to quantify a person's attitude to global warming. You could assign a "1" to "doesn't believe in global warming", a 10 to "firmly believes in global warming" and a scale of 2 to 9 for attitudes in between. You can also think of "scaling" as the fact that you're essentially *scaling down the data* (i.e.
making it simpler by creating lower-dimensional data). Data that is scaled down in dimension keeps similar properties. For example, two data points that are close together in high-dimensional space will also be close together in low-dimensional space (Martinez, 2005). The "**multidimensional**" part is due to the fact that you aren't limited to two dimensional graphs or data. Three-dimensional, four-dimensional and higher plots are possible.

MDS is now used over a wide variety of disciplines. It's use isn't limited to a specific matrix or set of data; In fact, just about any matrix can be analyzed with the technique as long as the matrix contains some type of relational data (Young, 2013). Examples of relational data include correlations, distances, multiple rating scales or similarities.

Manifold learning

What is a manifold?

A two-dimensional manifold is any 2-D shape that can be made to fit in a higher dimensional space by twisting or bending it, loosely speaking.

What is the Manifold Hypothesis?

"The Manifold Hypothesis states that real-world high-dimensional data lie on low dimensional manifolds embedded within the high-dimensional space."

In simpler terms, it means that higher-dimensional data most of the time lies on a much closer lower-dimensional manifold. The process of modelling the manifold on which training instances lie is called *Manifold Learning*.

Locally Linear Embedding (LLE)

Locally Linear Embedding (LLE) is a Manifold Learning technique that is used for non-linear dimensionality reduction. It is an unsupervised learning algorithm that produces low-dimensional embeddings of high-dimensional inputs, relating each training instance to its closest neighbor.

How does LLE work?

For each training instance $x^{(i)}$, the algorithm first finds its k nearest neighbors and then tries to express $x^{(i)}$ as a linear function of them. In general, if there are m training instances in total, then it tries to find the set of weights w which minimizes the squared distance between $x^{(i)}$ and its linear representation.

So, the cost function is given by

$$\sum_{i=1}^{m}(x^{(i)} - \sum_{j=1}^{m} w_{i,j}x^{(j)})^2$$

where $w_{i,j}$ =0, if j is not included in the k closest neighbors of i.

Also, it normalizes the weights for each training instance $x^{(i)}$,

$$\sum_{j=1}^{m} w_{i,j} = 1$$

Finally, each high-dimensional training instance $x^{(i)}$ is mapped to a low-dimensional (say, d dimensions) vector $y^{(i)}$ while preserving the neighborhood relationships. This is done by choosing d-dimensional coordinates which minimize the cost function,

$$\sum_{i=1}^{m} (y^{(i)} - \sum_{j=1}^{m} w_{i,j} y^{(j)})^2$$

Here the weights $w_{i,j}$ are kept fixed while we try to find the optimum coordinates $y^{(i)}$