Machine Learning (CS101) Unit 1: Introduction to Linear Discriminants and

Computer Science and Engineering (Jawaharlal Nehru Technological University, Hyderabad)

UNIT

1

Introduction and Linear Discriminants

SYLLABUS

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

LEARNING OBJECTIVES

- ✓ Concepts of Learning and Machine Learning
- ✓ Various Types of Machine Learning
- ✓ Overview of Supervised Learning including Regression and Classification
- ✓ Overview of Brain and the Neuron
- ✓ Design Issues of Learning System
- ✓ Perspectives and Issues in Machine Learning
- ✓ Illustration of Learning Task and Concept Learning as Search
- ✓ Concepts of Version Spaces and Candidate Elimination Algorithm
- ✓ Definition of Perceptron and Use of Perceptron Learning Algorithm
- ✓ Importance of Linear Separability
- ✓ Perceptron Convergence Theorem
- ✓ Functioning of Linear Regression.

INTRODUCTION

The concept of Learning, particularly Machine Learning focuses on acquiring knowledge from data. Machine learning aims to improve computer performance by adapting activities to become more accurate over time. Types of Machine Learning involve Supervised, Unsupervised, Reinforcement, and Evolutionary.

Major part of learning involves obtaining of general concepts from particular training examples. Every concept can be viewed as description of some subset of either objects or events which are defined over larger set.

The problem of automatically inferring general definition of a concept, given examples named as members and nonmembers of concept. This task called as concept learning or approximating the boolean-valued function from examples of input and output.

Linear discriminants are used to classify data by finding a hyperplane that separates different classes. Unlike the Perceptron, which focuses on linear separability, linear discriminants extend to linear regression techniques for better classification performance.

1.1 LEARNING

Q1. Explain the concept of learning in machine learning, and how is it applied in the context of human behaviour and machine algorithms.

Answer:

Learning

The idea of learning, particularly in the field of machine learning, focuses on acquiring knowledge from data. This phenomenon shows the process of human learning by experience, in both human and animal intelligence by changing their actions based on past interactions. The basic criteria of learning, applicable to both humans and machines, contain the abilities to retain information, adapt to new environments and apply knowledge to a broader context. This criteria is as follows,

- ❖ Remembering recalls the past circumstances and outcomes.
- ❖ Adapting involves changing behavior depending on success and failures..
- ❖ Generalizing uses learnt knowledge to unique but similar situations, which enhances the adaptability and usefulness in various situations.

Machine Learning

Machine learning is a field that aims to enable computers by enhancing their performance through adjusting or adapting their activities to become more precise over time. For example, when a computer engages in a game of Scrabble against a person, it may initially experience defeat. However, after multiple iterations of game play, the computer acquires and applies various tactics, thereby achieving an uninterrupted sequence of victory. Once acquired, the computer universally applies this acquired information to compete successfully against unfamiliar competitors eliminating the need to start from scratch in every instance.

Machine learning is a multidisciplinary field that includes principles from neuroscience, biology, statistics, mathematics and physics. The human brain with its complex mix of water, electrical and chemical components, provides as a model of acquiring knowledge. Insights from neuroscience have guided the development of neural networks, which are based on fundamental biological principles. Furthermore, data mining involves extracting valuable information from large datasets, which significantly impacts on machine learning by highlighting the need of efficient algorithms.

The high computational demands of machine learning algorithms are a critical factor to consider, particularly when working with large datasets. The concept of complexity can be categorized into two main types: training complexity and application complexity. Infrequent training can lead toa more time-consuming process. However, the trained algorithm must have computational efficiency in order to process large amounts of test data quickly while making predictions or choices. The complexity of high-degree polynomials or worse may present difficulties when dealing with large datasets, highlighting the importance of algorithm efficiency in machine learning.

1.2 TYPES OF MACHINE LEARNING

Q2. Explain different types of machine learning.

Answer:

There are different types of machine learning. They are,

1. **Supervised Learning:** Supervised learning trains an algorithm with labeled dataset by associating each training sample with an output label. The algorithm's goal is to acquire knowledge of the association between inputs and outputs and apply it to unknown data.
   Example: Forecasting the software that a website visitor is likely to acquire by analyzing their previous purchases and other attributes.
   Use cases: Classification tasks (like spam detection) and regression tasks (like predicting the value of homes).
2. **Unsupervised Learning:** Unsupervised learning involves an algorithm that receives data without clear instructions on how to process it. Instead, it strives to recognize patterns and correlations within the data. This frequently entails the process of clustering or grouping several data elements that have similarities.
   Example: Categorizing website visitors according to their browsing activity without prior knowledge of the classifications.
   Use cases: customer segmentation, density estimation and anomaly detection.
3. **Reinforcement Learning:** Reinforcement learning trains an algorithm by providing feedback in the form of rewards or penalties for its behavior. The algorithm acquires knowledge through the process of investigating various actions and getting feedback. This enhances its performance and maximizes the total rewards obtained.
   Example: Educating a robot to travel through a maze by providing positive reinforcement when it successfully reaches the destination and applying negative consequences when it encounters obstacles.
   Use cases: Game theory, such as chess, Go, robotics and autonomous vehicles.

4. **Evolutionary Learning:** Evolutionary learning processes the acquired knowledge and skills through the gradual development and adaptation of individuals over time. Natural selection is the mechanism from which this strategy derives. Problem solutions are considered as distinct entities within a population, and they undergo a gradual transformation. Such transformations are performed through techniques including mutation, crossover and selection, which are determined by their "fitness". It isa measure of computing the efficiency of transformation techniques in handlingthe problem inhand.

   Example: Enhancing the design of an engineering structure by refining several design factors to achieve maximum strength and minimum expense.

   Use cases: Solving optimization issues and implementing genetic algorithms.

1.3 SUPERVISED LEARNING

Q3. Explain briefly about supervised learning and its problems.

Answer:

Supervised Learning

For answer refer Unit-I, Q2, Topic: Supervised Learning (First Paragraph Only)

It is also a training model because it uses a collection of input data and their matching target data. This data is usually represented as pairs '(x_i, t_i)', where 'x_i' represents the input vectors and 't_i' represents the target vectors. This approach is dependent on having a large dataset enough for the algorithm to make generalizations, allowing it to generate precise results for new and unknown inputs. Generalization improves the algorithm's ability to manage noise, which refers to small errors in the data. Machine learning algorithms possess the capability of acquiring patterns and generating predictions beyond the given instances. This illustrates their capacity to adjust the new data, in contrast to a basic look-up table.

1. **Regression:** Regression involves predicting a continuous output value based on input features. Given a set of data points (x, t), the task is to predict the value of 'y' for a new input 'x=0.44'. This requires identifying a mathematical function that accurately represents the relationship between given data points and can estimate values for new inputs.

   There are multiple approaches to estimate this function, including linear interpolation, polynomial fitting, and more complex methods such as machine learning algorithms (e.g., linear regression, decision trees and neural networks). An ideal function should generalize well to new, unseen data rather than only fitting the given data points closely.

   The challenge lies in selecting an appropriate model that fits the given data points effectively while generalizing well with the new inputs. This involves balancing the complexity of model to avoid over-fitting (where the model fits the training data too closely and performs poorly on new data) and under-fitting (where the model is too simple to capture the underlying pattern in the data).

2. **Classification:** The classification problem involves the task of discovering specific class, out of 'N' classes, to which an input vector belongs. This determination is made by analyzing the provided training data. It is a discrete procedure in which each example is categorized into a single class, considering the whole range of possible outputs. Nevertheless, it is important to acknowledge that certain samples may only partially fit into many classes, an instance that can be solved successfully by fuzzy classifiers. Novelty detection is necessary when the classifier is presented with inputs that are not included in its training set, such as identifying foreign coins in a vending machine that has only been trained on local currencies.

   In order to establish a coin classifier, parameters like diameter, weight, and shape are used as characteristics for the input vector. The selection of features is crucial as having too many inputs might make training more complex known as the curse of dimensionality, while having too few characteristics may result in unreliable class classification. Relevant attributes for coin classification could encompass color and diameter; however, attributes like shape may be irrelevant in situations where all coins have identical shapes.

   Various classification techniques seek to identify decision boundaries that can effectively divide classes based on input attributes. Borders can take the form of either simple linear lines or more intricate non-linear curves, with non-linear borders typically providing a more efficient categorization of inputs. Considering and choosing the appropriate features and decision boundaries are essential for efficient classification.

   The machine learning process includes the selection of suitable models and features, the training of algorithms, and the evaluation of their performance on new data, considering both classification and regression problems. This technique is crucial for developing systems that have ability to acquire knowledge and provide forecasts or choices by analyzing input data.

1.4 THE BRAIN AND THE NEURON

Q4. Give an overview of the brain and the neuron. Explain about Hebb's rule.

Answer:

The Brain and the Neuron

The human brain functions based on simple and understandable concepts that might serve as a source of inspiration for machine learning systems. It effectively analyses noisy, high-dimensional data to get most precise responses. The brain's resilience and cognitive abilities are exceptional, despite a steady decline of neurons.

Neurons serve as the fundamental processing units of the brain. They transfer electrical pulses through axons and synapses to communicate. Each neuron forms connections with thousands of other neurons, resulting in approximately '100' trillion synapses. When the neurons' membrane potential crossesa certain threshold, neurons fire followed by a refractory period before firing again. Therefor, the human brain can be considered as a highly parallel computer consisting of '100' billion processors.

Neurons act as basic computational units, making decisions about whether to generate electrical impulses. This implies that it is theoretically possible to simulate the brain on a computer and potentially attain or human-like intelligence. It is generally called as artificial intelligence. Learning in the brain is facilitated by plasticity, which refers to the ability of synaptic connections to adapt and strengthen, as well as the creation of new connections. In 1949, Donald Hebb proposed a method for synaptic adaptation, which is explained below.

Hebb's Rule

Hebb's rule states that the electrical synapses between neurons become stronger in direct proportion to their simultaneous firing. When two neurons exhibit typical firing, their connection becomes stronger; in contrast, if they never fire together, the link shrinks and finally vanishes. This principle states that neurons react to the same stimuli and must develop connections. For instance, the simultaneous activation of specific neurons occurs when an individual looks at their grandmother and receives chocolate. This simultaneous activation reinforces the connection between these neurons, causing the individual to identify their grandmother with chocolate.

The principle of classical conditioning is shown by Pavlov's experiments. The dogs in Pavlov's experiment acquired the ability to link the sound of a bell with the presence of food since the neurons responsible for sense of hearing and salivation responded simultaneously, thereby reinforcing their relationship. Ultimately, the sound of the bell alone triggered a state of salivation.

Hebb's rule, sometimes referred to as long-term potentiation and neural plasticity, clarifies the process by which neurons establish and enhance connections through simultaneous firing. Real brains have been seen to exhibit this phenomenon, which provides support for the idea that learning and memory formation require changes in synaptic strength.

Q5. Discuss briefly about McCulloch and Pitts Neurons. Also mention its limitations.

Answer:

McCulloch and Pitts Neurons

The McCulloch and Pitts model, proposed in 1943, comprises three essential elements:

- A collection of weighted inputs, denoted as 'w_j' represents synapses.
- An adder calculates the total of the input signals.
- An activation function determines whether the neuron fires based on the input sum.

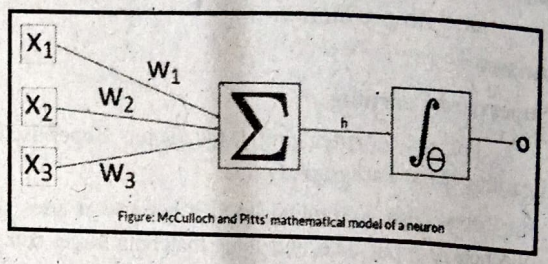


Figure: McCulloch and Pitts' mathematical model of a neuron

The model employs a collection of inputs (x_1, x_2, x_3), each multiplied by a matching synaptic weight (w_1, w_2, w_3). The inputs are added together, and if the sum exceeds a threshold 'θ', the neuron activates. The sum 'h' can be written as,

$$h = \sum_{i=0}^{m} w_i x_i$$

For instance, given a set of inputs $x_1 = 1, x_2 = 0, x_3 = 0.5$ and the corresponding synaptic weights $w_1 = 1, w_2 = -0.5, w_3 = -1$, the sum '$h$' of the inputs and weights is 0.5 (i.e., $h = 1 \times 1 + 0 \times -0.5 + 0.5 \times -1 = 1 + 0 + (-0.5) = 0.5$). Setting the threshold 'θ' to '0' causes the neuron to fire, as the value '0.5' exceeds '0', leading to an output of '1'. If the sum is less than the threshold, the output will be '0'. This model depicts a binary threshold device in which the neuron either activates or remains inactive depending on the sum of the input.

The activation function, '$g(h)$', defines the output of the neuron. It assigns a value of '1' if $h > \theta$, and a value of '0' if $h \leq \theta$.

$$o = g(h) = \begin{cases} 1 \text{ if } h > \theta \\ 0 \text{ if } h \leq \theta \end{cases}$$

Although it is simple, this model serves as the foundation for neural networks that can carry out any computation that a standard computer can perform, as long as the synaptic weights are accurately configured.

Limitations of the McCulloch and Pitts Neuronal Model

1. **Non-Linear Summations:** Non-linear summations occur in real neurons, when inputs are combined in a non-linear manner, in contrast to the linear summation observed in the model.
2. **Spike Train Output:** The output of real neurons is a sequence of pulses, known as a spike train output. It encodes information in a continuous and progressive fashion, compared to a single output response.
3. **Dynamic Threshold:** The firing threshold of actual neurons fluctuates throughout the time as a result of fluctuations in neurotransmitter levels and the present condition of the organism.
4. **Asynchronous Updates:** Actual neurons undergo arbitrary self-updates (asynchronously), in contrast to the clock-based updates found in numerous models.
5. **Fixed Synaptic Types:** In real neurons, excitatory or inhibitory synaptic connections consistently belong to the same category. However, in models, the weights associated with these connections can shift between positive and negative values.
6. **Feedback Loops:** Synapses connecting back to themselves are a common feature in real neurons, but they are frequently removed from models.
7. **Biological Complexity:** The model fails to account for complicated biological factors like chemical concentrations and refractory periods.

1.5 DESIGN A LEARNING SYSTEM

Q6. Explain in detail about the basic design issues. of learning system

Answer:

The basic design issues and approaches to machine learning can be determined by considering a program design to learn playing checkers. The purpose of it is to make it enter into world checkers tournament.

For remaining answer refer Unit-I, Q7, Q8, Q9, Q10.

Q7. Explain how training experience is selected.

Answer:

The success and failure of a learner completely depends on the type of training experience available. An important aspect is whether training experience offers direct or indirect feedback related to the choices of performance system. While learning to play checkers, the system may learn from direct training examples containing separate checker board states and their moves. As an alternative, their can be indirect data of move sequences and final result of different games. In this case, the data related to correctness of moves early in game need to be inferred indirectly from the fact that game is either lost or won.

Learner can face another problem related to credit assignment or determining the degree of moves in sequence. This degree deserves a credit or blame for final result. The credit assignment is a complex problem since the game might be lost even if early moves are optimal when poor moves follow them. Learning from indirect feedback is complex than learning from direct training feedback. Another aspect of training experience is the degree to which the learner can control sequence of training examples. The learner might depend upon teacher for selecting the informative board states and even to offer a correct move for each of them. As an alternative, the learner can propose board states that it requests teacher for correct move in case of confusion. The learner can have control on board states and training classification because this is done when it learns playing against itself with no teacher. The learner can select either experimenting with novel board states that remain unconsidered or honoring the skills by playing minor differences of lines of play which are promising.

The third aspect of training experience is the representation of examples distribution over which the final system performance p is measured. The learning is found to be reliable while the training examples are following a similar distribution to future test examples. In checkers learning, the performance metric p denotes the percent of games won by system in world tournament.

If the training experience contains the games which are played against itself, then there is a concern that the training experience might not be able to represent the distribution of situations upon which it is tested. For example, the learner may not encounter the crucial board states which can be played by human checkers champion. It is mandatory to learn from examples distribution which is different from those on which final system evaluates. These type of situations are problematic since mastery of examples distribution may not lead to strong performance over the other distributions.

The Checkers Learning Problem

Task T: Playing checkers

Performance Measure P: Percent of games won in world tournament

Training Experience E: The games played against itself

To complete the design of learning system, one among the following must be chosen,

1. The exact type of knowledge that is to be learned.
2. Representation for the target knowledge.
3. The learning mechanism.

Q8. Discuss briefly about the target function and the selection of its representation.

Answer:

Selecting a Target Function

Consider a checkers playing program that is capable of producing legal moves from any of the board states. The program requires to learn the selection of best move from the legal moves. Such type of learning task represents large set of tasks for which the legal moves (defining large search space) are called as priori. However, the best strategy is not known to them. This category includes different types of optimization problems like scheduling and controlling the manufacturing processes where the available manufacturing steps are well understood. In this setting, it is mandatory to learn about selection among legal moves. One necessary choice for information type to be learned is a program or function which selects best move for given board state. Assume it as Choose Move and use Choose Move: $B \rightarrow M$ for representing that the function accepts as input for any board from legal board states B. It generates a move from legal moves M.

Even though ChooseMove is a mandatory choice for target function, it is difficult to learn the case of indirect experience available to the system. Another target function is an evaluation function which is easier to learn and assigns numeric score to given board state. Let the target function be V which uses the notation $V : B \rightarrow A$ to represent that V maps legal board state from set B to a read value. Let this target function V assigns higher scores to improve board states. If the system can learn a target function V then it can use it for selecting the best move from present board position.

This can be implemented by producing the successor board state that is generated by legal move. Further more, it can usie V to select best successor state and best legal move. The value of target function V for any given board state can be any evaluation function which assigns higher scores to improve board states. Define the target value $V(b)$ for arbitrary board state b in B,

- If b is final board state which is won, then $V(b) = 100$.
- If b is final board state which is lost, then $V(b) = -100$.
- If b is final board state which is drawn, then $V(b) = 0$.
- If b is not final state in game, then $V(b) = V(b')$.

The recursive definition specifies the value of $V(b)$ for every board state b, but it is not much useful for checkers player since it is not computable efficiently. Defining the value of $V(b)$ for specific board state needs searching ahead for optimal line of play upto end of game except for trivial cases where games already ended. It is non operational definition since it is not computable by checkers playing program. The purpose behind the learning is to discover an operational description of V. The learning task is minimized to problem of discovering operational description of ideal target function V. It would be difficult to learn it perfectly.

Selecting Target Function Representation

The representation must be in such a way that the learning program uses it to describe the function $\hat{V}$ which is learnt. There are many options, for example the program can be allowed to represent $\hat{V}$ by using large table with different entries indicating the value for each distinct board state. It can be allowed to represent $\hat{V}$ by using a collection of rules which matches against features of board state or quadratic polynomial function of predefined board features or artificial neural network. This representation often involves crucial trade offs. Conisering an expressive representation for allowing the representation close to ideal target function. On the other hand the training data required by program for selecting alternative hypothesis depends upon the expressiveness of representation. Now select a simple representation for given board state, for which the function $\hat{V}$ is calculated as linear combination of below features,

→ x_1 : Number of black pieces on board

→ x_2 : Number of red pieces on board

→ x_3 : Number of black rings on board

→ x_4 : Number of red kings on board

→ x_5 : Number of black pieces threatened by red

→ x_6 : Number of red pieces threatened by black

The learning program will therefore represent $\hat{V}(b)$ as linear function of the given form,

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Partial Design of Checkers Learning Program

Task T: Playing checkers

Performance Measure P: Percent of games which are won in world tournament

Training Experience E: Games played against itself

Target Function V: Board → R

Target function representation.

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Q9. Explain about the approximation algorithm.

Answer:

A set of training examples are needed to learn the target function $\hat{V}$ where each training example determines particular board state b and training value $V_{train}(b)$ for b. Every training example can be defined as an ordered pair of form $<b, V_{train}(b)>$. For instance, the below example defines a board state b where black won the game and has the target function $V_{train}(b)$ as +100.

$$<<x_1=3, x_2=0, x_4=0, x_5=0, x_6=0>, +100>$$

Estimating Training Values

The learner is limited to only training information whether the game won or lost. The training examples that assign scores to specific board state is also required. Since values can be easily assigned to board states which correspond to end of the game, it is less obvious that the process of assigning training values to more intermediate board states occur before the end of the game. The success or failure of any game will not represent that every board state was good or bad along the game path. Since there is ambiguity in estimation of training values for intermediate board states, one simple method is found to be successful.

This method assigns the training value of $V_{train}(b)$ for intermediate board state b to be $\hat{V}$ *(Successor (b))*, where $\hat{V}$ is present approximation of learner to V and *successor(b)* is next board state that follows b. The rule of estimating training values is given as follows,

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

It might not be appropriate to use present version of $\hat{V}$ for estimating the training values which are used to refine the same function. It must be noted that estimates of value of *Successor (b)* are used for estimating the board state value b. This will definitely make some sense if $\hat{V}$ tends to be accurate for board states close to games end.

Adjusting the Weights

The task that is left over is to specify the learning algorithm for selecting w_i to best fit the set of training examples $\{\langle b, V_{train}(b)\rangle\}$.

The first step is to determine the best fit to training data. A method which is generally followed is to define best hypothesis, or set of weights that reduces the squared error E in between training values and the values which are predicted by hypothesis $\hat{V}$.

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

There are multiple algorithms to find the weights of a linear function which reduce defined value of E. In present scenario, an algorithm which incrementally refines the weights of new training examples is available. It is robust to errors in such estimated training values. One another algorithm is least main squares or LMS training rule. It adjusts the weights for every observed training in such a way that it minimizes the error in training example. The LMS algorithm is defined as follows,

LMS weight update rule :

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

Make use of present weights for calculating $\hat{V}(b)$

For every weight w_i update it as shown below,

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

The weights will not change when error $\left(V_{train}(b) - \hat{V}(b)\right)$ is zero. They get increased in proportion to value of the respective feature when $\left(V_{train}(b) - \hat{V}(b)\right)$ is positive. This leads to increase in $-\hat{V}(b)$ and decrease in error. If value of any feature x_i is zero then weight is not changed. Here the updated weights reflect to the features of training example board. This method in some settings proves to converge to least squared error approximation to V_{train} values.

Q10. Briefly explain about the final design of checkers learning system.

Answer:

The final design of checkers learning system is described naturally by four different program modules which indicate the central components in most of the learning systems. The four modules are depicted as follows,

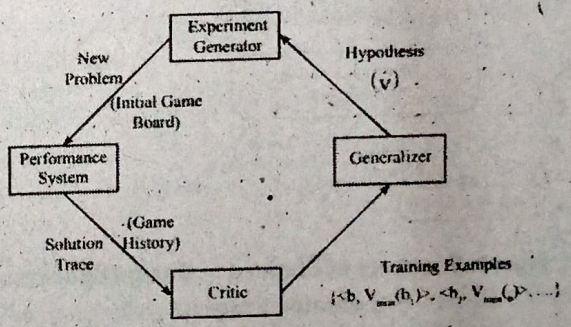


Figure: Final Design of Checkers Learning Program

1. **Performance System:** It solves the given performance task i.e., playing checkers through learned target functions (s). It accepts new problem instance as input and generates a copy of solution. In present scenario, the strategy for performance system is to determine the selection of next move at every step by learned $\hat{V}$ evaluation function. Performance is expected to get improvised since the evaluation function is accurate.

2. **Critic:** It accepts history or trace of game as input and generates set of training examples of target function as output. Every training example is related to some game state in trace with estimate V_{train} of target function value.
3. **Generalizer:** It accepts training examples as input and generates hypothesis which is the estimate of target function. It will generalize from particular training examples, hypothesizing general function. The generalizer relates to LMS algorithm. The output hypothesis is function $\hat{V}$ which is described by learned weights w_0,..... w_6.
4. **Experiment Generator:** It accepts current hypothesis as input and new problem as output for performance system in order to explore. The major role of it is to pick new practice problems which increase the learning rate of the system. The sequence of design choices that are made for checkers program is given as follows,

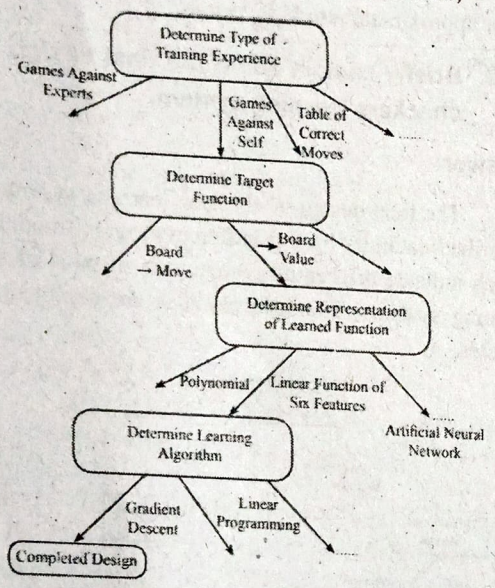


Figure: Summary of Choices to Design Checkers Learning Program

The design choices depicted above have constrained learning tasks in various ways. The type of knowledge is restricted, which can be acquired to single linear evaluation function. Later, it is constrained to depend upon six board features. If it is true, then the target function V can be represented by linear combination of specific features. The program can learn easily. Otherwise, it learns good approximation because program can never learn that it cannot represent. Finally, the design is base of decisions which will go into designing a learning method for particular class of tasks.

1.6 PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Q11. List the perspectives and issues in machine learning.

Answer:

A useful perspective on machine learning is that it involves searching of large space of possible hypothesis for determining the best fit for observed data and prior knowledge of learner. For example, consider space of hypothesis which will be generated by checkers learner. Such type of hypothesis space contains evaluation functions represented by certain choice of values for weights w_0 through w_6. The task of learner is to search the vast space in order to find hypothesis which is consistent along with already available training examples. The LMS algorithm for fitting the weights can achieve this by tuning the weights iteratively, while adding connection to every weight for prediction of values by hypothesized evaluation function. It works well only if hypothesis representation defines parameterized space of potential hypothesis continuously.

Issues in Machine Learning

There are several issues in machine learning which are depicted as follows,

1. What are the algorithms which are available for learning the general target functions from particular training examples? In which settings, the specific algorithms will converge to desired function provided sufficient training data? Which type of algorithms perform the best for different types of problems and representations?
2. How much training is required? What type of bounds can be found related to the confidence in learned hypothesis to amount of training experience and character of learner's hypothesis space?
3. How and when the prior knowledge of learner will guide the generalization from examples? Can the knowledge be useful when it is approximately correct?
4. What is the best strategy for selecting useful training experience? How the choice of strategy alters learning problem complexity?
5. In which way the learning task can be reduced to one or more function approximation problems? What are the specific functions that need to be learnt by the system? Can this process be automated?
6. How can the learner automatically modifies the representation to improve the representation and learns the target function?

1.7 CONCEPT LEARNING TASK

Q12. Give an explanation on the concept learning and concept learning task.

Answer:

Concept Learning

Major part of learning involves obtaining general concepts from particular training examples. For example, people will learn the general concepts or categories like "bird", "bike" etc., continuously. Every concept can be viewed as description of some subset of either objects or events which are defined over larger set. As an alternative, every concept can be considered as Boolean valued function which is defined over larger set.

The problem of automatically inferring general definition of a concept, given examples named as members and nonmembers of concept. This task is called as concept learning or approximating the Boolean-valued function from examples of input and output.

Concept Learning Task

Consider the example task of learning target concept "days which a friend enjoys favorite sport". The below table depicts set of example days which are represented by set of attributes.

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

In the above table, the attribute EnjoySport represents whether the friend enjoys favorite sport. The task is learned by prediction of value EnjoySport for any arbitrary day depending upon the other attribute values. Consider a simple representation where hypothesis contains conjunction of constraints on instance attributes.

Particularly, every hypothesis denotes a vector six constraints by specifying values of six attributes Sky, Humidity, Wind, Water, Forecast and AirTemp. For all the attributes, the hypothesis will

- Represent by a "?" such that any value is acceptable for this attribute
- Represent a single value for attribute
- Represent by " ϕ" such that no value is acceptable.

If an instance satisfies all the constraints of hypothesis h, then it can classify x as positive example ($h(x) = 1$). The hypothesis that friend enjoys the sport on cold days with high humidity is represented by expression.

$\langle ?, Cold, High, ?, ?, ? \rangle$

A general hypothesis – that all the days are positive examples – is represented as shown below,

$\langle ?, ?, ?, ?, ?, ? \rangle$

The specific possible hypothesis – that no day is positive example is represented as shown below,

$\langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

The EnjoySport concept learning task needs learning of a set of days for which EnjoySport = yes, by describing the set by conjunction of constraints over instance attributes. Any concept learning task can be defined by, set of instances over which target function is defined, the target function, set of candidate hypothesis as considered by learner and set of training examples which are available. The definition of EnjoySport concept learning task is given as follows,

Let,

Instances X: Possible days described by attributes

→ Sky

→ Air Temp

→ Humidity

→ Wind

→ Water

→ Forecast

Hypothesis H: Each hypothesis described by conjunction of constraints on attributes Sky, Airtemp, Humidity, Wind, Water, Forecast. The constraints may be "?", " ϕ" or any value.

Target Concept C: EnjoySport : $X \rightarrow \{0, 1\}$

Training Examples D: Positive and Negative examples of target function.

Determine: A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

Notation

The set of items over which the concept is defined is called as set of instances denoted by X. The X is set of possible days represented by attributes Sky, AirTemp, Humidity, Wind, Water and Forecast. The concept that is to be learnt is called target concept denoted by C. The target relates to value of attribute EnjoySport. While learning target concept, the learner is provided with a set of training examples containing an instance x from X with target concept value $C(x)$. The symbol D is used to represent set of available training examples.

Inductive Learning Hypothesis

The learning task determines hypothesis h similar to target concept Cover with complete set of instances X. The data available about C is value over training examples. Hence inductive learning algorithms can assure that hypothesis can best fit target concept over training data. The best hypothesis related to unseen instances is hypothesis which best fits observed training data. This is assumption of inductive learning.

1.8 CONCEPT LEARNING AS SEARCH

Q13. Briefly discuss about concept learning as search.

Answer:

Concept learning is a task that can be considered as searching a large space of hypothesis which are defined implicitly by hypothesis representation. The major aim of search is to locate the best fitting hypothesis for training examples. The designer of learning algorithm will define that the space of hypothesis run by program cannot represent or even learn. This can be done by designer by selecting hypothesis representation. For example, consider the instances X and hypothesis H in EnjoySport learning task. Given that,

Sky ← three possible values

Air Temp, Humidity, Wind, water and fore cast ← Two possible values

Instance space X ← 3.2.2.2.2.2 distinct instances

Another similar computation depicts that there are 5.4.4.4.4.4 syntactically distinct hypothesis in H. The number of semantically distinct hypothesis is 1 + (4.3.3.3.3.3) = 973.

General-to-Specific Ordering of Hypothesis

Most of the algorithms for concept learning conduct search on hypothesis space based on structure that is available for any concept learning problem. This means general-to-specific ordering of hypothesis with this advantage, learning algorithms can be designed that can search infinite hypothesis spaces without the need of enumerating each hypothesis. The general-to-specific ordering can be illustrated by considering two hypothesis,

$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$

$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$

Consider sets of instances which are classified as positive by h_1 and h_2. The h_2 can classify more number of instances as positive since it imposes only less number of constraints on instance. An instance which is classified by h_1 has a chance to get classified by h_2 . Thus, h_2 is said to be more general than h_1. For any instance x in X and hypothesis h in H, it is said that x can satisfy h when h(x) = 1. Consider two hypothesis h_j and h_k, h_j is said to be more general than or equal to h_k when any instance which satisfies h_k also satisfies h_j. If both h_j and h_k are boolean valued functions then h_j is said to be more general than or equal to h_k only when,

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

It is significant for considering the cases in which one of the hypothesis is more general than other. Thus, it can be said that h_j is more general than h_k only when $(h_j \geq_g h_k) \wedge (h_k \not\geq_g h_j)$. Atlast an inverse can be learned to be useful and in such case h_j is said to be more_specific_than h_k when h_k is more_general_than h_j . These definitions can be better explained by considering an example of three hypothesis h_1, h_2 and h_3. They are related by $\geq_g$ relation.

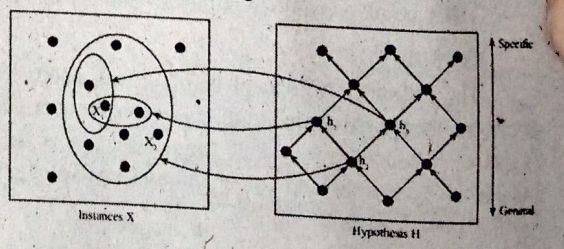


Figure: Instances, Hypothesis and More_general_than Relation

In the above figure, the box at left side depicts set X of all the instances such that

$x_1 = \langle Sunny, Warm, High, Strong, Cool, Same \rangle$

$x_2 = \langle Sunny, Warm, High, Light, Warm, Same \rangle$

The box at right side depicts set 4 of all the hypothesis such that

$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$

$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$

$h_3 = \langle Sunny, ?, ?, ?, Cool, ? \rangle$

The hypothesis h_2 is said to be more general than h_1 because the instances that satisfy h_1 will also satisfy h_2. In the same way h_2 is more general than h_3. But h_1 or h_3 are not more general than other and the relations $\geq_g$ and $>_g$ relations are defined as unrelated to target concept. The relation $\geq_g$ illustrates a partial order over H. It even offers a significant structure over hypothesis space H for any concept learning problem.

1.9 FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

Q14. Explain in detail about the finding a maximally specific hypothesis.

Answer:

The more_general_than partial ordering can be used to arrange the search for a hypothesis which is compatible with training examples by initiating with a possible hypothesis in H. Generalize it for every failure in order to enclose the observed positive training example. Consider the below algorithm.

1. Initialize h with a hypothesis in H.
2. For each positive training instance x
   - 2.1 For each attribute constraint ai in h.
     - 2.1.1 If x satisfies constraint ai then do nothing.
     - 2.1.2 Else replace ai in h with more general constraint that x can satisfy.
3. Output hypothesis h.

The above given is FIND-S algorithm. It can be better explained by considering a sequence of training examples from above algorithm for the task EnjoySport. The first step of it would be to assign a hypothesis H to h.

$h \leftarrow \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

The " ϕ" constraints in h are replaced with more general constraint because they are not satisfied. Let the attribute values be,

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

Here h seems to be still more specific. It states that all of the instances are negative other than for single positive training example.

The second training example derives algorithm to generalize h by submitting a "?" in place of an attribute value in h which is not satisfied with new example. In such case, the refined hypothesis is,

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

In third training example, the algorithm will not make any change to h. This is a negative example and such examples are ignored by FIND-S algorithm. The fourth example guides to further generalization of h.

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

The FIND-S algorithm determines an approach that allows to use more_general_than partial ordering for organizing the search for acceptable hypothesis. The search is conducted on every hypothesis along the chain of partial ordering.

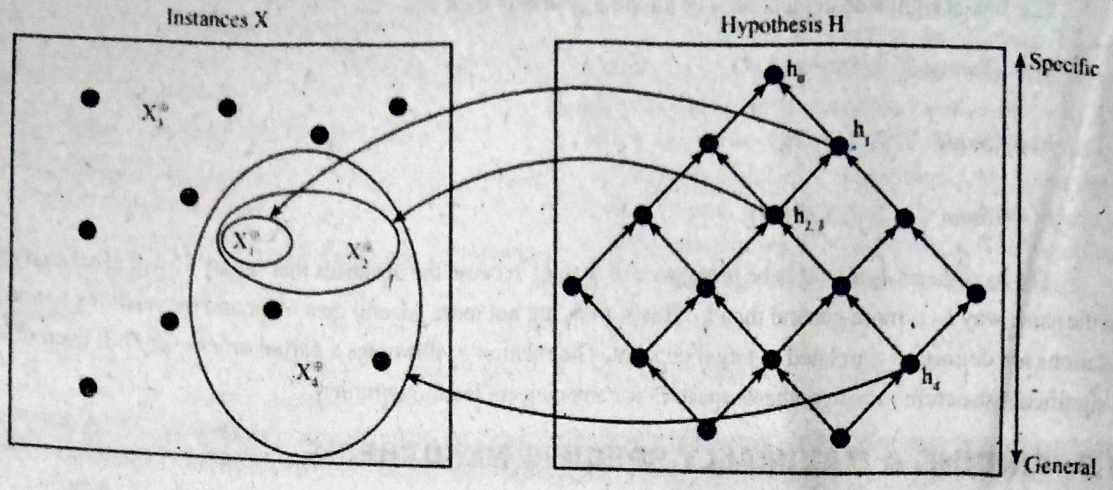


Figure: Hypothesis Space Search Performed by FIND-S

The above diagram depicts the search in terms of instance and hypothesis spaces. The box on left represents the set X of all instances such that,

x_1=<Sunny Warm Normal Strong Warm Same>, +

x_2=<Sunny Warm High Strong Warm Same>, +

x_3=<Rainy Cold High Strong Warm Change>, -

x_4=<Sunny Warm High Strong Cool Change>, +

The box at right represents set H of all hypothesis such that

h_0=<ϕ, ϕ, ϕ, ϕ, ϕ, ϕ >

h_1=<Sunny Warm Normal Strong Warm Same>

h_2=<Sunny Warm ? Strong Warm Same>

h_3=<Sunny Warm ? Strong Warm Same>

h_4=<Sunny Warm ? Strong ? ?>

In every step, the hypothesis is generalized as required to cover the new positive example. Moreover, hypothesis is specific and consistent along with the training examples. The FIND-S algorithm generates most specific hypothesis in H which is consistent along with positive training examples. The final hypothesis is also found to be consistent along with negative examples provided that the H holds correct target and training examples are also correct.

1.10 VERSION SPACES AND THE CANDIDATE ELIMINATION ALGORITHM

Q15. Discuss in detail about the version spaces and the candidate eliminating algorithm.

Answer:

The purpose of CANDIDATE-ELIMINATION algorithm is to generate a description for set of hypothesis compatible with training examples. It evaluates description of this set without computing its members. This is established through a more general hypothesis than partial ordering compatible with old compact hypothesis representation. This refines the representation of training example occurrences. Let the hypothesis be compatible with training examples if examples are correctly classified. It is said to be compatible only when $h(x) = c(x)$ for every example $<x, c(x)>$ in D.

$$Consistent(h, D) = (\forall < x, c(x)> \in D)\ h(x) = c(x)$$

The CANDIDATE-ELIMINATION algorithm represents set of all hypothesis compatible with observed training examples. Such type of subset hypothesis is known as version space in terms of hypothesis space H and training examples D since it holds all the possible versions of target concept.

$$VS_{H,0} \equiv \{h \in H | Compatible(h, D)\}$$

Version space can be represented by listing all of its members. This gives rise to a simple algorithm called LIST-THEN-ELIMINATE algorithm. It begins the version space to hold the hypothesis in H and then remove the hypothesis found incompatible with training example. It may decrease with an increase in examples until one hypothesis is found which is compatible with all examples.

1. Version space ← List of hypothesis in H.
2. For every training example, $<x, c(x)>$ eliminate any hypothesis h for which $h(x) \neq c(x)$ from version space.
3. Generate list of hypothesis in version space.

The CANDIDATE-ELIMINATION algorithm evaluates version space holding hypothesis from H compatible with observed sequence of training examples. It starts by initializing version space to set of hypothesis in H. This means, initializing G bounding set to hold most general hypothesis in H,,

$G_0 \leftarrow \{ < ?, ?, ?, ?, ?, ?> \}$

And initializing S boundary set to hold most specific hypothesis,

$S_0 \leftarrow \{ < \phi,\phi,\phi,\phi,\phi,\phi > \}$

The above boundary sets determine the complete hypothesis space since every other hypothesis in H is general than S_0 as well as specific than G_0. The S and G boundary sets can be generalized and specialized respectively for removing from version space and the incompatible hypothesis.

The computed version space only holds the hypothesis which are compatible. The general algorithm is as follows,

Algorithm

Assign set of maximally general hypothesis in H to G

Assign set of maximally specific hypothesis in H to S

For each training example x do,

→ If x is a positive example

→ Eliminate incompatible hypothesis with x from G

→ For each hypothesis s in S which is incompatible with x

→ Eliminate s from S

→ Add minimal generalizations h of s to S such that

→ h is compatible with x and some member of G is more general than h

→Eliminate hypothesis that is more general than other hypothesis in s from S

→If x is a negative example

→Eliminate hypothesis incompatible with x from S

→For each hypothesis g in G which is incompatible with x

→Eliminate g from G

→Add minimal specializations h of g to G such that

→h is compatible with x and some other member of S more specific than h

→Eliminate hypothesis that is less general than other hypothesis in G

The CANDIDATE-ELIMINATION algorithm is specified in terms of operations like evaluation of minimal generalizations and specializations of hypothesis and there by identifying nominal and non-maximal hypothesis. The algorithm is applicable to any concept learning task and hypothesis space with well defined operations.

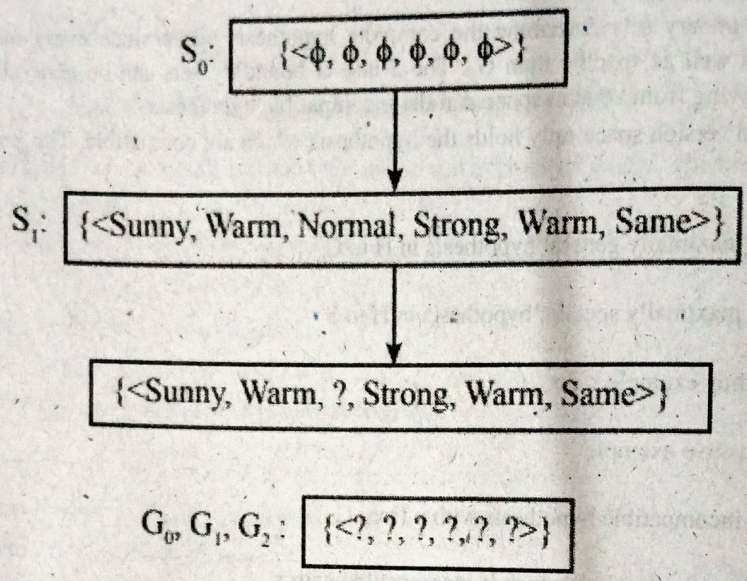
Q16. Provide an illustration to explain the mechanism of the candidate elimination procedure.

Answer:

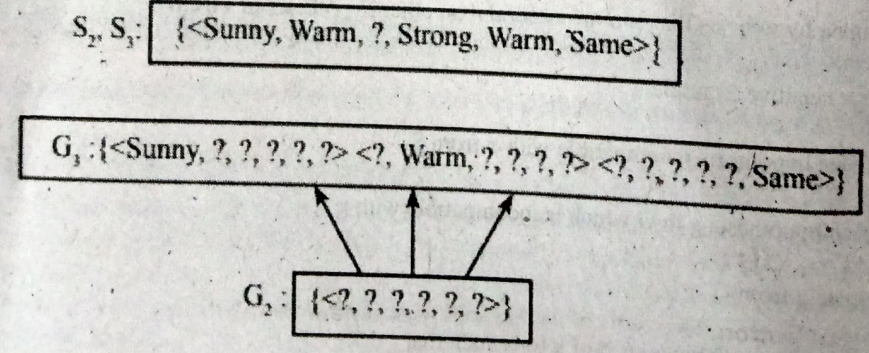
Consider the below example.

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	Cold	High	Strong	Warm	Change	Yes

The below figure traces the candidate elimination algorithm.

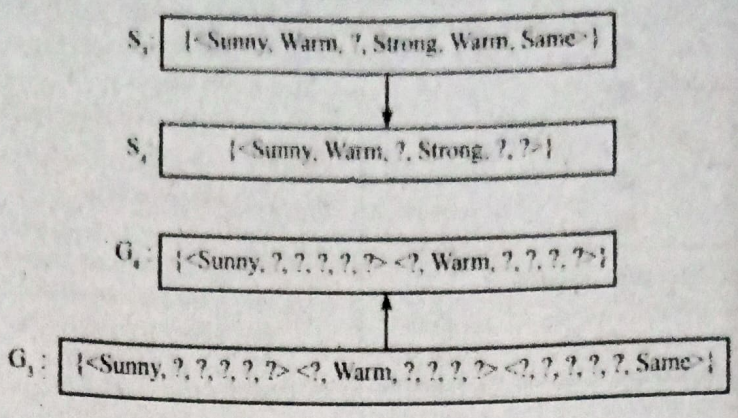


The boundary sets are initialized to G_0 and S_0. For the training example <Sunny, Warm, Normal, Strong, Warm, Same>, EnjoySport = Yes, the candidate elimination algorithm checks S boundary and then finds that it is specific and cannot cover the positive example. The boundary is updated by moving it to least general hypothesis which covers new example. The second training example <Sunny, Warm, High, Strong, Warm, Same>, EnjoySport = Yes is similar to effect of generalizing S followed by S_2 without updating G. Consider the third training example,



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

This is a negative example that represents that G boundary of version space is general. The hypothesis in G will not predict accurately that new example is positive. The hypothesis in G boundary needs to be specialized until it classifies new negative example accurately. A more specific hypothesis in above figure are members of new G_3 boundary set. Consider the fourth training example,

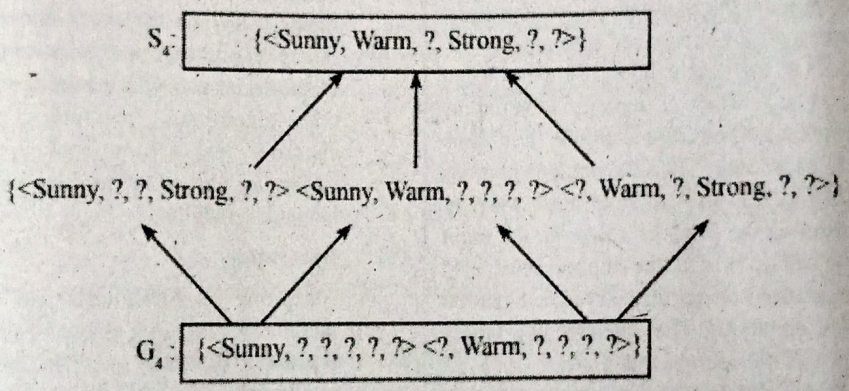


Training Example

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

This fourth example will generalize the *S* boundary of version space. It removes a member of *G* boundary since it cannot cover new positive example. The hypothesis needs to be dropped from *G* boundary by deleting the branch of partial ordering from version space of hypothesis. The boundary sets S_4 and G_4 after processing the four examples will delimit the version space of hypotheses that are consistent with set of incrementally observed training examples.

This is depicted in below figure,



The learned version space in above figure is not dependent on sequences based on which training examples are represented. The *S* and *G* boundaries are moved closer by delimiting a smaller version space of candidate hypothesis.

1.11 LINEAR DISCRIMINANTS

1.11.1 Perceptron

Q17. Define Perceptron. Explain its functionality in a neural network.

Answer:

Perceptron

A perceptron is a specific kind of artificial neural network that was created by Frank Rosenblatt in 1950s. Input nodes, weights, and neurons (sometimes called output nodes) make up its composition. Only binary classification tasks utilize the most basic form of a neural network.

Structure of Perceptron

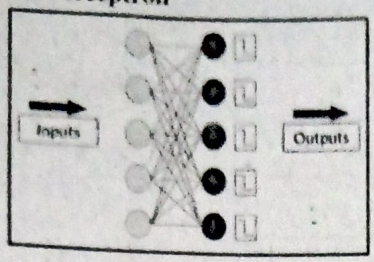


Figure: Perceptron Network

The image above depicts the perceptron network, with input nodes shown on its left side, distinguished from the neurons that are displayed on the opposite side being tinted in light grey. These input nodes provide values to the network; however, they are not actual neurons. Every input node is associated with an element within the input vector, usually portrayed as circles. This could potentially lead to confusion, as neurons can also appear as circles.

Each neuron in a Perceptron functions independently, deciding whether to activate by calculating the product of its input weights, adding them, and then comparing the outcome to a predetermined threshold. The neurons are independent of each other and only share input values. The quantity of inputs 'm' and neurons 'n' can differ, with inputs being defined by the data and outputs by the requirement to replicate a certain goal pattern during supervised learning.

The values of the weights in the Perceptron can be represented as 'w_{ij}', where 'i' represents the inputs and 'j' represents the neurons. For example, 'w_{32}' links input node '3' to neuron '2'. The implementations commonly utilize a '2D' array to store these weights. In order to determine the firing of a neuron, the input values are adjusted to match the input vector, and additional computations are performed for each neuron using designated equations. The outcome is a binary vector that indicates the neurons that fired.

If a neuron produces an inaccurate output, its weights require modification. The error function, denoted as '$y_k - t_k$', is used to evaluate if the weights are excessively large or little. In order to change the weights, the error value is multiplied by the appropriate input value and the learning rate 'η'. The weight update rule is expressed as '$w_{ij} = w_{ij} - \eta(y_j - t_j) \cdot x_i$'. The learning process involves presenting several training instances until the network's performance stabilizes or reaches a certain number of iterations, denoted as 'T'.

The threshold values of neurons are of utmost importance. If the input is '0', weight changes have no impact so the threshold must be adjusted. The learning rate, denoted as 'η', is a crucial parameter that significantly affects the rate of learning. Repeated training drives the network learning process until it accurately identifies all inputs or reaches the maximum number of iterations identified or until the maximum number of iterations are reached.

Q18. Briefly explain the flowing terms

(a) The learning rate

(b) Bias input

(c) Perceptron learning algorithm.

Answer:

(a) The Learning Rate

The learning rate, represented by the symbol 'η', is an essential parameter in the weight update rule of the Perceptron algorithm. This parameter determines the extent of changes in weight values throughout the learning process. Excluding the learning rate, which is equal to limiting it to '1', might result in substantial alterations in weights in accordance with errors, resulting in network instability and hindering the convergence process. On the other hand, a low learning rate necessitates that the network analyzes inputs regularly in order to make significant weight modifications, thereby extending the learning phase duration. However, this method improves the network reliability as well as its ability to withstand noise and errors in the information being transmitted. An optimal learning rate, usually ranging from '0.1' to '0.4', is suggested based on the anticipated inaccuracy in the inputs. Although the exact amount of learning rate is not of great importance for the Perceptron algorithm, it is a vital parameter for numerous other algorithms. Therefore, it is critical to choose a suitable learning rate in order to strike a balance between the learning velocity and the network reliability.

(b) Bias Input

According to McCulloch and Pitt's neuron model, every neuron is equipped with a firing threshold 'θ' which specifies the precise value at which the neuron will start firing. The ability to change this threshold is necessary in order to precisely regulate the firing of neurons. If all inputs to a neuron are '0', the weights become insignificant (since multiplying '0' by any amount of weight results in '0'). In such circumstances, modifying the threshold solely regulates the neuron's firing. Neurons lacking changeable thresholds would exhibit identical behavior, regardless of the weights, when their thresholds are exactly the same and their inputs are '0'.

Modifying the threshold adds an additional parameter that increases the coding complexity and weight changes. But there exists a sophisticated resolution that is, setting the threshold to '0' and including an additional input weight with a constant input value (often '-1', although any non-zero value is suitable).

The update process adjusts this extra weight to ensure accurate neuron firing, even in the absence of other inputs. Weights expressed with '0' subscript like 'w_{0j}', connect the bias node, often referred to as a special input, to the 'j^{th}' neuron. This strategy speeds up the implementation and preserves the necessary adaptability for the neurons to activate correctly in response to the inputs.

(c) Perceptron Learning Algorithm

It consists of two phases, namely, a training phase and a recall phase.

Initialization

- Assign small random values, both positive and negative, to each of the weights 'w_{ij}'.

Training

- During 'T' iterations or till all the outcomes are accurate, perform the following steps for every input vector:
  - Determine the activation of each individual neuron 'j' using the activation function 'g':

$$\sum_{i=0}^{m} w_{ij} x_i) = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_{ij} x_i > 0 \\ 0 \text{ if } \sum_{i=0}^{n} w_{ij} x_i \leq 0 \end{cases}$$

  - Use of following method to modify the weights for each individual:

$w_{ij} = w_{ij} - \eta(y_j - t_j) \cdot x_i$

Recall

- Determine the activation of every single neuron 'j' using the following:

$$y_j = \sum_{i=0}^{m} w_{ij} x_i) = \begin{cases} 1 \text{ if } w_{ij} x_i > 0 \\ 0 \text{ if } w_{ij} x_i \leq 0 \end{cases}$$

The recall phase is employed subsequent to training and is expected to be expedient in its utilization, given its frequency of usage exceeding the frequency of the training phase. The training phase uses the recall equation to identify the neuron activations before calculating the error and training the weights.

Determining the computational complexity for this algorithm is simple. The recall phase iterates through the neurons, and during that iteration, it iterates through the inputs, resulting in a complexity of '$O(mn)$'.

The training phase offers identical operation; but repeats it for 'T' iterations, resulting in a time complexity of '$O(Tmn)$'.

When the user encounters an algorithm in this format for the first time, it might be difficult to imagine how to translate it into code. Similarly, it can be difficult to understand how such an easy algorithm can learn. The sole method to rectify these issues is to manually execute the algorithm on one or two instances and attempt to compose the code, thereafter verifying it produces the anticipated outcome. Following that, the user will proceed to accomplish each of these tasks.

Q19. How can perceptron learning be applied to understand and implement logic functions.

Answer:

Network Configuration

. The structure of the perceptron neural network consists of the various nodes, namely:

- Two input nodes and a bias input are included.
- One output node.

The figure below shows the input and output values of the OR function along with its graphical representation and the corresponding neural network structure.

In_1	In_2	Out
0	0	0
0	1	1
1	0	1
1	1	1

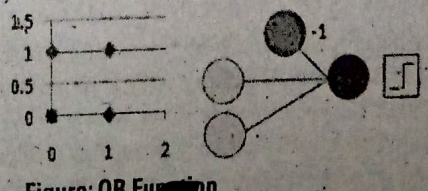


Figure: OR Function

Initialization

Begin by setting up the three weights with small random values: w_0=–0.05, w_1=–0.02, w_2=0.02.

Training Process

- **For input process (0, 0)**
  - The constant value assigned to the bias weight is consistently -1.
  - Determining the neuron's input: (–0.05×–1) + (–0.02×0) + (0.02×0) = 0.05.
  - Since the value 0.05 is greater than 0, the neuron fires and generates an output of 1, which is an invalid result.
  - Using the update rule with a learning rate of η=0.25, the updated values for the weights are as follows:
    - w_0 : –0.05–0.25 × (1–0) × –1 = 0.2.
    - w_1: ¬0.02–0.25 × (1–0) × 0 = –0.
    - w_2: 0.02–0.25 × (1–0) × 0 = 0.
- **For input process (0, 1)**
  - The constant value assigned to the bias weight is consistently -1.
  - Determining the neuron's input: (0.20×–1) + (–0.02×0) + (0.02×1) = –0.18.
  - Since the value -0.18 is less than 0, the neuron fires and generates an output of 0, which is an invalid result.
  - Implementing the updating algorithm:
    - w_0 : 0.2–0.25 × (0–1) × –1 = –0.05.
    - w_1: –0.02–0.25 × (0–1) × 0 = –0.
    - w_2: 0.02–0.25 × (0–1) × 1 = 0.27.
- **For input process (1, 0)**
  - The constant value assigned to the bias weight is consistently -1.
  - Determining the neuron's input: (–0.05×–1) + (–0.02×1) + (0.27×0) = 0.03.
  - Since the neuron generates the accurate output, there is no requirement for weight adjustment.
- **For input process (1, 1)**
  - The constant value assigned to the bias weight is consistently -1.
  - Determining the neuron's input: (–0.05×–1) + (–0.02×1) + (0.27×1) = 0.30.
  - Since the neuron generates the accurate output, there is no requirement for weight adjustment.

Iterations

When the evaluation of all inputs completed, it is possible that not all outputs will be accurate. The algorithm has to navigate across the inputs till the weights reach a stable state and refuse to change. In typical instances, it is normal for weights to never completely stabilize. Therefore, the algorithm is executed for a specific amount of iterations, denoted as '*T*'.

Convergence

Through iterative execution of the process, either individually or with computer code, users gradually attain weight values that approach a state of stability. Currently, the Perceptron has successfully acquired the ability to accurately perform the OR function. Various sets of weights determined by factors such as, the learning rate 'η', inputs and beginning weight values can yield the right outputs. The objective is to discover a collection of weights that effectively and broadly applies to other inputs instead of concentrating on the particular values.

1.11.2 Linear Separability

Q20. Give an explanation on the linear separability.

Answer:

The perceptron is a basic idea in machine learning that is utilized to categorize data by seeking a linear separation among diverse classes. When dealing with a single output neuron similar to the OR function, the perceptron aims to differentiate between situations that the warrant activation (fire) and those that do not. A line (in 2D, 3D, or a hyperplane in higher dimensions) represents this differentiation by differentiating one category from another. It is commonly known as the decision boundary or discriminant function as shown in the below figure.

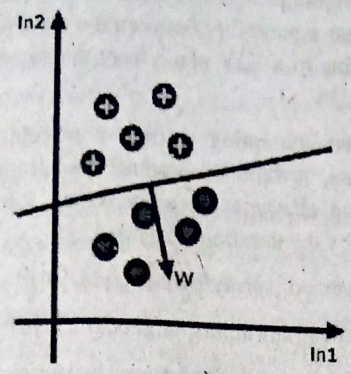


Figure: A Decision Boundary

In order to understand the functioning of the Perceptron, it is essential to examine the matrix notation applied to its implementation, with particular attention to a unique input vector '*x*'. The Perceptron is triggered or activated when '$x \cdot w^T \geq 0$'. Here, '*w*' is the row vector derived from the weight matrix '*W*' that links the inputs to a specific neuron. When employing the OR functions, there is a single neuron, resulting in a uniform value of '*w*' for all inputs. The transpose of vector '*w*', denoted as 'w^T', to transform 'w' and '*x*' into column vectors for matrix multiplication.

The symbol '*a.b*' indicates the inner or scalar product of two vectors. The computation is performed by multiplying each element of the first vector with the corresponding element of the second vector and then summing the results. The dot product, mathematically represented as $a.b = ||a||||b||\cos\theta$, calculates the angle '$\theta$' between two vectors '*a*' and '*b*' by multiplying their magnitudes ('$||a||$' and '$||b||$') with the cosine of 'θ'. The inner product uses the magnitudes of the vectors and the cosine of the angle between them to compute a scalar value. The dot product in Python can be calculated using the np.inner() method from the NumPy package.

When two input vectors, 'x_1' and 'x_2', in the context of the Perceptron, such that $x_1 \cdot w^T = 0$ and $x_2 \cdot w^T = 0$, then combine these equations to analyze the consequences for the decision boundary:

$$x_1 \cdot w^T = x_2 \cdot w^T \; (x_1 - x_2).w^T = 0$$

From the above equation, for the dot product to be equal to '0', either the magnitudes '$||a||$' or '$||b||$' must be '0'; hence, $\cos\theta$ = '0'. Here, 'θ' must be '$\pi/2$' or '$-\pi/2$' radians, indicating that the difference between 'x_1' and 'x_2' is perpendicular to '*w*'. Geometrically, the expression '$x_1 - x_2$' denotes a straight line connecting two points that are located on the decision boundary of the Perceptron. The weight vector 'w^T' must be perpendicular to this line, ensuring that it determines the orientation of the decision border that separates distinct classes.

The perceptron algorithm aims to identify a linear boundary that can separate the input data points based on their goal outputs. This algorithm performs well if it finds a line to separate the data, indicating that the problem is linearly separable. However, if the data does not exhibit linear separability, this approach faces difficulties. When there are several output neurons, each neuron establishes an individual decision boundary. Together, they divide the space successfully in order to classify the data.

In simple terms, the perceptron algorithm calculates a linear decision boundary to differentiate between various classes using the input data. This method is effective when dealing with such scenarios where the data can be separated in a straight line, but it necessitates more detailed solutions when the data cannot be separated in a straight line.

Q21. Explain in detail about the perceptron convergence theorem.

Answer:

Linear Separability and Convergence

Rosenblatt's proof demonstrates that if the dataset is linearly separable, meaning that there exists a hyper plane that can separate all data points belonging to various classes, then the Perceptron algorithm will converge to a solution. This implies that it will identify a collection of weights that accurately categorize all training examples within a limited number of iterations.

Bound on Number of Iterations

The proof establishes an upper limit on the total number of iterations needed to achieve convergence. More precisely, the distance 'γ' between the separating hyperplane and closest datapoint near it limits the maximum number of iterations '*t*'. This distance 'γ' is also referred to as the margin.

Proof Outline

The proof employs fundamental algebraic manipulations and uses the Perceptron algorithm properties:

- The process begins by assuming that there is a weight vector w^* that can effectively separate the data.
- The perceptron algorithm repeatedly changes the weight vector *w* in order to reduce the classification error.
- Every update incrementally enhances the inner product $w^* \cdot w$, assuring advancement towards convergence.
- The update rule is $w^{(t)} = w^{(t-1)} + yx$. With each iteration '*y*' represents the correct output and '*x*' represents the input vector, resulting in a reduction in errors.

Convergence Criteria

During each iteration, the algorithm evaluates both the inner product '$w*.w$' and the magnitude '$\|w\|$' of the weight vector:

- The inner product grows by a minimum of γ with each weight update, ensuring that '$w * .w \geq t_\gamma$'.
- Following the 't' steps, the weight vector's length is $\|w^{(t)}\|^2 \leq \|w^{(t-1)}\|^2+1$. Where the final line appears since the network encountered an error, $y^2 = 1$, $\|x\| \leq 1$, and hence 'x' and '$w^{(t-1)}$' are orthogonal to one another. This indicates that $\|w^{(t)}\|^2 \leq k$ after 't' steps. Combining both of these inequalities yields the following: $t_\gamma \leq \| w^{(t-1)} \| \leq \sqrt{t}$ and $t \leq \frac{1}{\gamma^2}$.

Final Remarks

The perceptron stops learning as soon as all training data points are correctly classified. It does not guarantee that the largest margin (maximum separation between classes) is identified. It only finds a separating hyperplane if one exists. This proof does not guarantee the behavior of the perceptron algorithm for datasets that are not linearly separable.

Q22. Discuss in detail how XOR function can be trained with the perceptron convergence theorem.

Answer:

The Exclusive OR (XOR) Function

The XOR function is used as an ideal instance to demonstrate the constraints of a single-layer perceptron when attempting to solve problems that are not linearly separable. Within a two-dimensional space, it is not possible to divide the XOR function using a straight line. As a result, the perceptron is unable to reach the proper solution.

XOR Function and Perceptron Failure

The XOR (exclusive OR) function is a digital logic gate that only produces a true or '1' output if the two binary bit inputs are different. The truth table for the XOR function is shown below:

In_1	In_2	Out
0	0	0
0	1	1
1	0	1
1	1	0

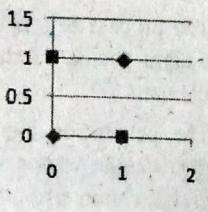


Figure: Ex-OR Function Table with Graph

- When both inputs are valued at '0', the resulting output will also be '0'.
- Setting one input to '1' and another to '0', results in an output of '1'.
- With both inputs set to '1', the output is '0'.

Examining the above figure, it is clear that the lack of linear separability between the classes prevents a straight line in a '2D' plane from distinguishing the XOR function.

When attempting to train a perceptron using XOR inputs, it fails to produce the proper result. The training alternates between wrong solutions, as evidenced by the iterations provided:

```
targets = np.array([[0], [1], [1], [0]])
pcn.pcntrain(inputs, targets, 0.25, 15)
# Outputs oscillating between two wrong solutions.
```

However in '3D', it is possible to resolve the XOR function using a linear function by adding a third input rather than using '2D'. The figure below displays the table and the graph for the XOR function.

In_1	In_2	In_3	Out
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	1

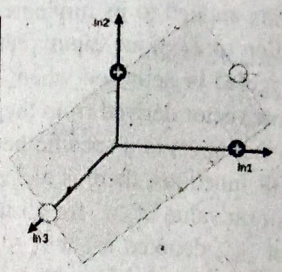


Figure: A Decision Boundary Solving the XOR Problem in 3D

By incorporating a third dimension, the issue becomes linearly separable. The concept involves incorporating an extra input that enables a plane (the '2D' equivalent of a line) to accurately divide the points in '3D' space.

For instance, the following code snippet facilitates XOR in '3D':

```
inputs = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0],
                   [1, 1, 0]])
pcn.pcntrain(inputs, targets, 0.25, 15)
# Successfully separates the classes
```

Output

```
Iteration: 14
[[-0.27757663]
 [-0.21083089]
 [-0.23124407]
 [-0.53808657]]
```

The final outputs are as follows:

```
[[0]
 [1]
 [1]
 [0]]
```

The above example demonstrates that a linear function effectively separates the two classes. Many methods exist within the classes to produce the data into a correct set of dimensions, which are referred to as kernel classifiers. Support vector machines are the best examples for the kernel classifiers.

It has been demonstrated that a linear Perceptron can be expanded to address non-linear situations by including non-linear variables. By introducing a non-linear transformation of the original variables 'x_1' and 'x_2' to include 'x_1', 'x_2' and '$x_1 \times x_2$', it becomes feasible to segregate the data using a plane. This shows how non-linear transformations can assist in accomplishing classification tasks.

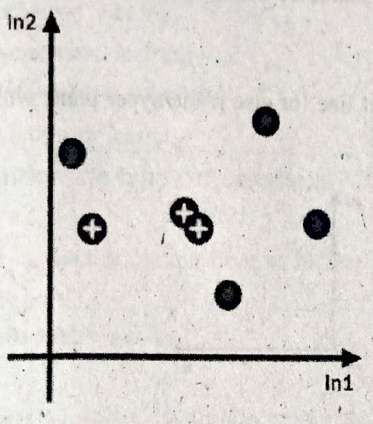


Figure: Non-separable 2D Dataset

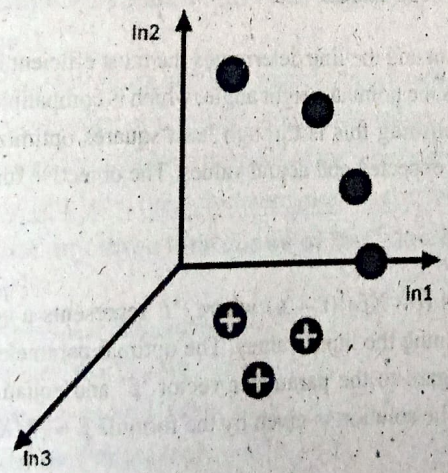


Figure: Separable 3D Dataset

Before the introduction of computers, people used to employ statistical approaches for categorization and regression. These methods rely on linear approaches and have provided valuable insights into learning processes. By integrating statistical techniques with computer science methodologies, users can attain a full comprehension of learning and data analysis.

Q23. Elaborate on the data preparation of preprocessing system.

Answer:

Preprocessing-Data Preparation

Preprocessing the inputs and targets before training enhances the performance of machine learning algorithms. For example, if the goal values are not already normalized to a range of '0' to '1', they should be adjusted to this range in order to avoid excessive weight growth. Conventional approaches such as normalizing to zero mean and unit variance, or scaling between '–1' and '1' are advantageous for scaling the inputs.

Normalization is a beneficial process that helps to prevent outliers from having a dominant impact on the data. While it is not necessary for every method, it is generally desirable. Normalizing data in NumPy is a simple process that involves using the `np.mean()` and `np.var()` functions while considering the specific axis along which these statistical measures. As an illustration, the code required to standardize input variables in a dataset might be:

```
data = (data - data.mean(axis=0))
/ data.var(axis=0)
```

Prior to dividing the dataset into training and testing sets, it is crucial to standardize the entire dataset in order to ensure uniformity.

In addition, applying preprocessing techniques appropriate to the domain can enhance the outcomes even more. In the Pima dataset, all patients are female. Column 0 corresponds to the number of pregnancies, while column 7 corresponds to age. Restricting the number of pregnancies to a maximum of 8 and categorizing ages into specific ranges (such as 21–30 and 31–40) can be beneficial. This can be accomplished via the `np.where` function, as demonstrated:

```
pima[np.where(pima[:, 0] > 8), 0] = 8
pima[np.where(pima[:, 7] <= 30), 7] = 1
pima[np.where((pima[:, 7] > 30) & (pima[:, 7]
<= 40)), 7] = 2
```

Ultimately, a primitive method of feature selection can be executed by training the classifier using subsets of the inputs. Through a systematic process of excluding various features and analyzing their effect on results, it is possible to identify and remove features that do not have a positive impact on the model. This approach facilitates the identification of correlations between features and the output, thereby improving the model's performance.

1.11.3 Linear Regression

Q24. Give an explanation on linear regression.

Answer:

Linear Regression

Regression involves the task of finding the best-fitting line that can be utilized in predicting an unknown value 'y', based on known values 'x_i'. In classification, the goal is to identify a boundary that distinguishes between different classes. There are two ways to make this change: first, add a variable called the indicator that shows class membership, which turns the problem into estimating this variable as a regression problem; second, do multiple regressions, one for each class, setting the indicator value to '1' for examples that belong to the class and '0' for examples that do not belong to the class.

The Perceptron algorithm differs from conventional statistical approaches primarily in terms of problem formulation. Regression involves making projections about the unidentified value 'y' by calculating a function based on the known values 'x_i'. The relationship is commonly represented by a linear function denoted as,

$$y = \sum_{i=0}^{M} \beta_i x_i$$

Where 'β_i' are constant parameters that determine a straight line (or else plane/hyper plane with greater dimensions) passing through the data points.

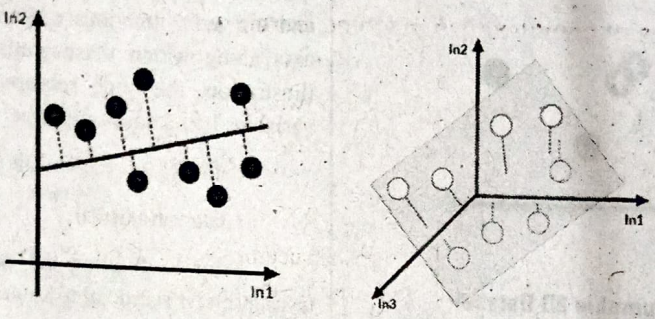


Figure: Linear regression in 2D and 3D

Minimizing the Euclidean distance among each data point and the line determines the most efficient line of best fit. The shortest distance takes place when the line intersects the point at a right angle, which is compatible with Pythagoras' theorem. A commonly used approach for accomplishing this is through least-squares optimization, which aims to minimize the sum of squared errors between the expected and actual values. The objective function to be minimized is the sum of squared errors, given by:

$$\sum_{i=0}^{N} \left(t_j - \sum_{i=0}^{M} \beta_i x_{ij}\right)^2.$$

The error function can be expressed in matrix form as $(t - X\beta)^T(t - X)$ where, 't' represents a column vector indicating the target values and 'X' is the matrix containing the input values. The optimal parameters can be identified by taking the derivative of this function with regard to the parameter vector 'β' and equating the derivative to zero. This leads to the equation $X^T(t - X\beta) = 0$. The solution is given by the formula $\beta = (X^TX)^{-1}X^Tt$ under the assumption that the matrix 'X^TX' is invertible.

Performing this calculation in Python is simple and efficient with the help of NumPy library. The np.linalg.inv() function computes the inverse of a matrix, thereby implementing the entire regression computation in the following code snippet:

```
def linreg(inputs, targets):
inputs  = np.concatenate((inputs, -np.ones((np.shape(inputs)[0], 1))), axis=1)
beta    = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(inputs), inputs)), np.transpose(inputs)), targets)
outputs = np.dot(inputs, beta)
```

This function appends a bias input to each of the inputs, calculates the regression coefficients 'β', and finally computes the output.

VERY SHORT QUESTIONS WITH SOLUTIONS (VSQS)

Q25. What are the basic criteria of learning?

Answer :

The basic criteria of learning, applicable to both humans and machines, contain the abilities to retain information, adapt to new environments, and apply knowledge to a broader context.

Q26. Mention the primary goal of machine learning.

Answer :

Machine learning is a field that aims to enable computers by enhancing their performance through adjusting or adapting their activities to become more precise over time.

Q27. List the types of machine learning.

Answer :

There are different types of machine learning, which are mentioned below:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning
4. Evolutionary learning

Q28. Mention the types of complexity in machine learning.

Answer :

The concept of complexity can be categorized into two main types: training complexity and application complexity.

Q29. State Hebb's rule.

Answer :

Hebb's rule states that the electrical synapses between neurons become stronger in direct proportion to their simultaneous firing. When two neurons exhibit typical firing, their connection becomes stronger; in contrast, if they never fire together, the link shrinks and finally vanishes.

Q30. State McCulloch and Pitts Neuron.

Answer :

The McCulloch and Pitts model comprises three essential elements:

- A collection of weighted inputs, denoted as 'wj' represents synapses
- An adder calculates the total of the input signals
- An activation function determines whether the neuron fires based on the input sum.

Q31. List any three limitations of McCulloch and Pitts Neuronal model.

Answer :

1. Non-linear summations
2. Spike Train Output
3. Asynchronous Update.

Q32. Define Perceptron.

Answer :

A perceptron is a specific kind of artificial neural network that was created by Frank Rosenblatt in 1950s. Input nodes, weights, and neurons (sometimes called output nodes) make up its composition. Only binary classification tasks utilize the most basic form of a neural network.

Q33. Write a short note on linear separability and convergence.

Answer :

Rosenblatt's proof demonstrates that if the dataset is linearly separable, meaning that there exists a hyper plane that can separate all data points belonging to various classes, then the Perceptron algorithm will converge to a solution. This implies that it will identify a collection of weights that accurately categorize all training examples within a limited number of iterations.

IMPORTANT QUESTIONS

Q1. Explain the concept of learning in machine learning, and how is it applied in the context of human behaviour and machine algorithms.

Answer : Important Question

For answer refer Unit-I, Q1.

Q2. Explain briefly about supervised learning and its problems.

Answer : Important Question

For answer refer Unit-I, Q3.

Q3. Discuss briefly about McCulloch and Pitts Neurons. Also mention its limitations.

Answer : Important Question

For answer refer Unit-I, Q5.

Q4. Explain in detail about the basic design issues of learning system.

Answer : Important Question

For answer refer Unit-I, Q6.

Q5. List the perspectives and issues in machine learning.

Answer : Important Question

For answer refer Unit-I, Q11.

Q6. Give an explanation on the concept learning and concept learning task.

Answer : Important Question

For answer refer Unit-I, Q12.

Q7. Discuss in detail about the version spaces and the candidate eliminating algorithm.

Answer : Important Question

For answer refer Unit-I, Q15.

Q8. Define Perceptron. Explain its functionality in a neural network.

Answer : Important Question

For answer refer Unit-I, Q17.

Q9. Give an explanation on the linear separability.

Answer : Important Question

For answer refer Unit-I, Q20.

Q10. Give an explanation on linear regression.

Answer : Important Question

For answer refer Unit-I, Q24.