UNIT – II

Chapter I:

DECISION TREE

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.

- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

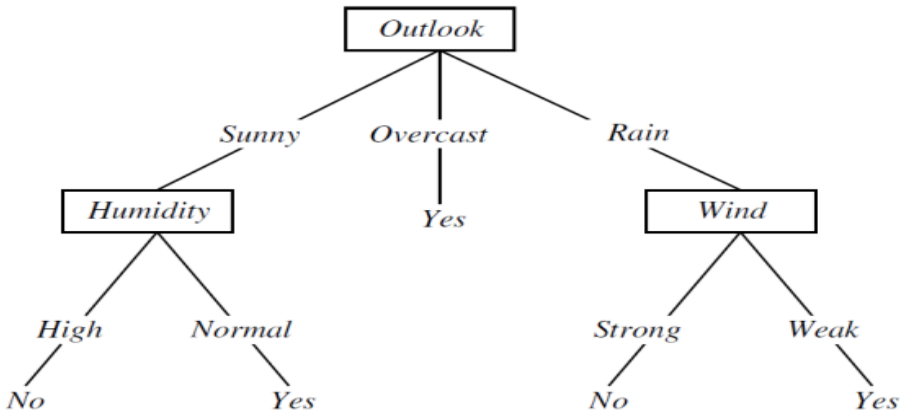


Figure 1: A decision tree for the concept PlayTennis.

An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf.

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions For example, the decision tree shown in above figure

corresponds to the expression (Outlook = Sunny ∧ Humidity = Normal) ∨ (Outlook = Overcast) ∨ (Outlook = Rain ∧ Wind = Weak)

Appropriate Problems for Decision Tree Learning:

Decision tree learning is generally best suited to problems with the following characteristics:

1. Instances are represented by attribute-value pairs – Instances are described by a fixed set of attributes and their values.

2. The target function has discrete output values – The decision tree assigns a Boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.

3. Disjunctive descriptions may be required.

4. The training data may contain errors – Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.

5. The training data may contain missing attribute values – Decision tree methods can be used even when some training examples have unknown values.

The Basic Decision Tree Learning Algorithm:

The basic algorithm is ID3 which learns decision trees by constructing them top-down

---

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

- Otherwise Begin
  - A ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = v_i
    - Let *Examples* $_{vi}$, be the subset of Examples that have value v_i for *A*
    - If *Examples* $_{vi}$, is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
      - Else below this new branch add the subtree
        ID3(*Examples* $_{vi}$, Targe_tattribute, Attributes – {A}))
- End
- Return Root

---

* The best attribute is the one with highest information gain

Summary of the ID3 algorithm specialized to learning Boolean-valued functions. ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used.

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- A statistical property called information gain that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses information gain measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES To define information gain, we begin by defining a measure called entropy. Entropy measures the impurity of a collection of examples. Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes

D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example:

To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.

Here the target attribute PlayTennis, which can have values yes or no for different days.

Consider the first step through the algorithm, in which the topmost node of the decision tree is created. Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is:

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$
$$= 0.940$$

The entropy is 0 if all members of S belong to the same class

- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Information Gain Measures the Expected Reduction in Entropy:

- Information gain, is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is example: Information Gain:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Example: Information gain

Let, Values S(Wind) = {Weak, Strong}

= [9+, 5−]

S = [6+, 2−]

Weak

S = [3+, 3−]

Strong

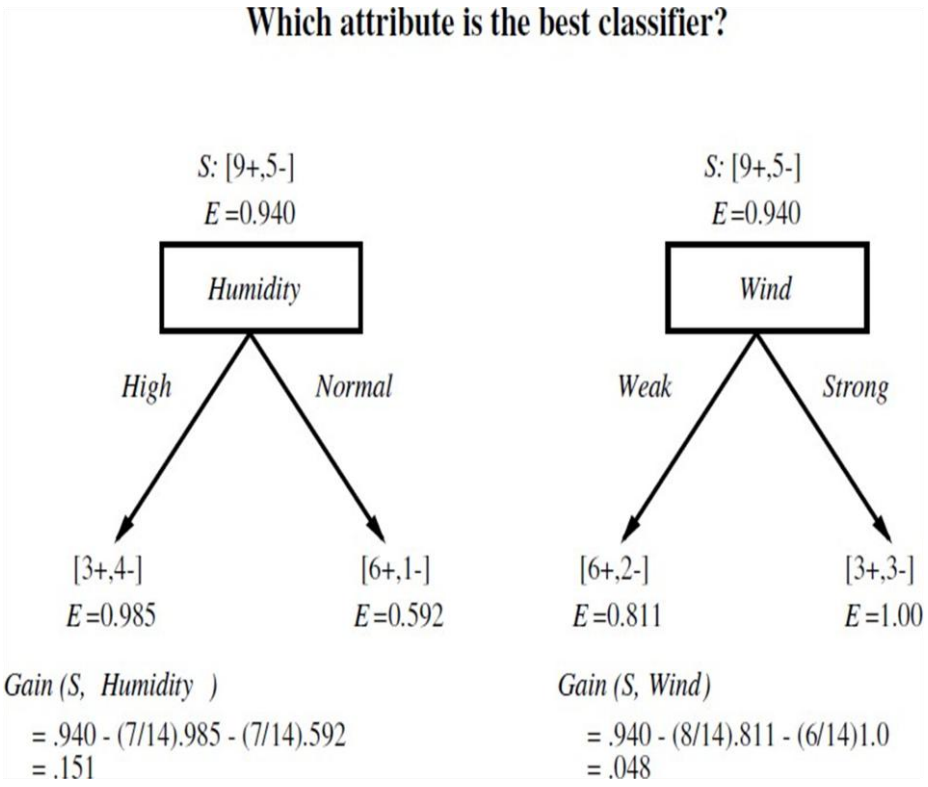
Information gain of attribute Wind:

Gain(S, Wind) = Entropy(S) − 8/14 Entropy (SWeak) − 6/14 Entropy (SStrong)

= 0.94 – (8/14)* 0.811 – (6/14) *1.00

= 0.048

ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain



The information gain values for all four attributes are:

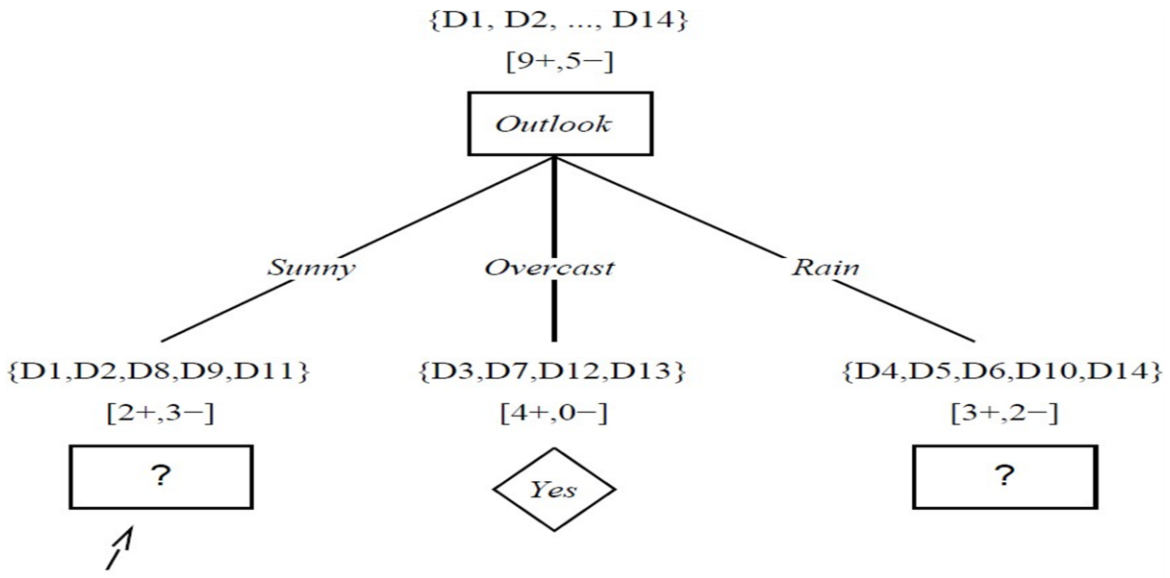
Gain(S, Outlook) = 0.246

Gain(S, Humidity) = 0.151

Gain(S, Wind) = 0.048

Gain(S, Temperature) = 0.029

According to the information gain measure, the Outlook attribute provides the best prediction of the target attribute, Play Tennis, over the training examples. Therefore, Outlook is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.



S_{sunny} = {D1,D2,D8,D9,D11}

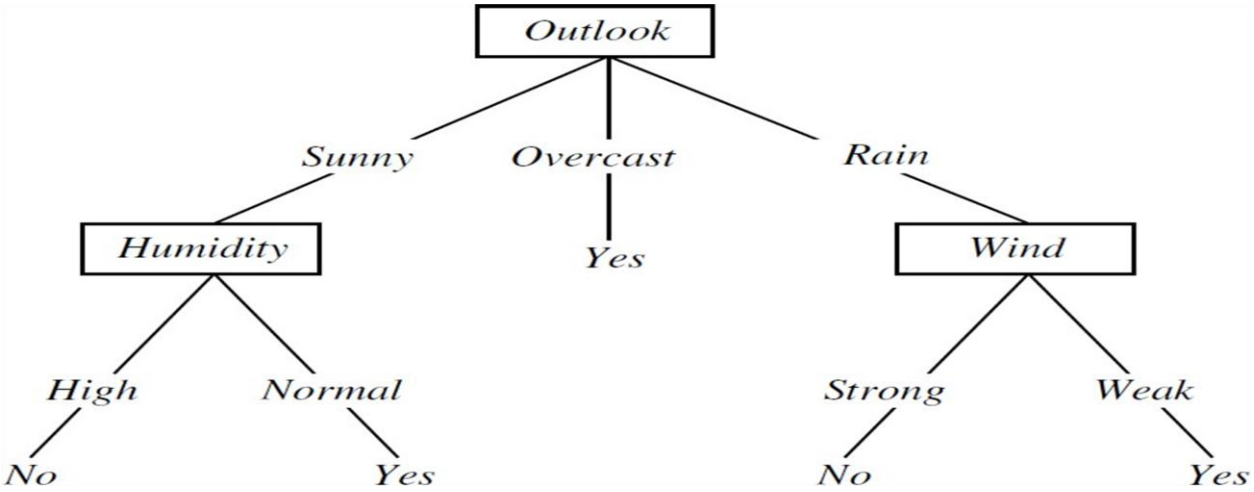
$Gain\ (S_{sunny}, Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$

$Gain\ (S_{sunny}, Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$

$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

S_{Rain} = { D4, D5, D6, D10, D14}

Gain (S_{Rain}, *Humidity)* = 0.970 – (2/5)1.0 – (3/5)0.917 = 0.019
Gain (S_{Rain}, *Temperature)* =0.970 – (0/5)0.0 – (3/5)0.918 – (2/5)1.0 = 0.019
Gain (S_{Rain}, *Wind)* =0.970 – (3/5)0.0 – (2/5)0.0 = 0.970



Chapter II

ARTIFICIAL NEURAL NETWORKS

INTRODUCTION

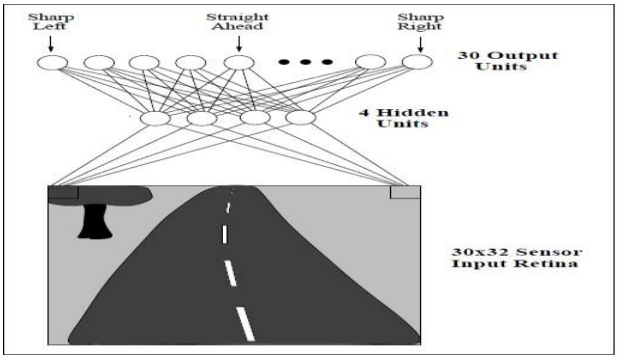
Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued target functions.

Biological Motivation

- The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected ***Neurons***
- Human information processing system consists of brain ***neuron***: basic building block cell that communicates information to and from various parts of body

NEURAL NETWORK REPRESENTATIONS

- A prototypical example of ANN learning is provided by Pomerleau's system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways
- The input to the neural network is a 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network output is the direction in which the vehicle is steered



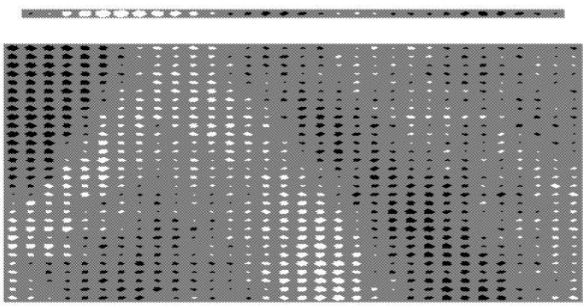


Figure: Neural network learning to steer an autonomous vehicle.

Figure illustrates the neural network representation.

- The network is shown on the left side of the figure, with the input camera image depicted below it.
- Each node (i.e., circle) in the network diagram corresponds to the output of a single network unit, and the lines entering the node from below are its inputs.
- There are four units that receive inputs directly from all of the 30 x 32 pixels in the image. These are called "hidden" units because their output is available only within the network and is not available as part of the global network output. Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs
- These hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.
- The diagrams on the right side of the figure depict the learned weight values associated with one of the four hidden units in this ANN.
- The large matrix of black and white boxes on the lower right depicts the weights from the 30 x 32 pixel inputs into the hidden unit. Here, a white box indicates a positive weight, a black box a negative weight, and the size of the box indicates the weight magnitude.
- The smaller rectangular diagram directly above the large matrix shows the weights from this hidden unit to each of the 30 output units.

APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

ANN is appropriate for problems with the following characteristics:

1. Instances are represented by many attribute-value pairs.
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 of humans to understand the learned target function is not important

PERCEPTRON

- One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.

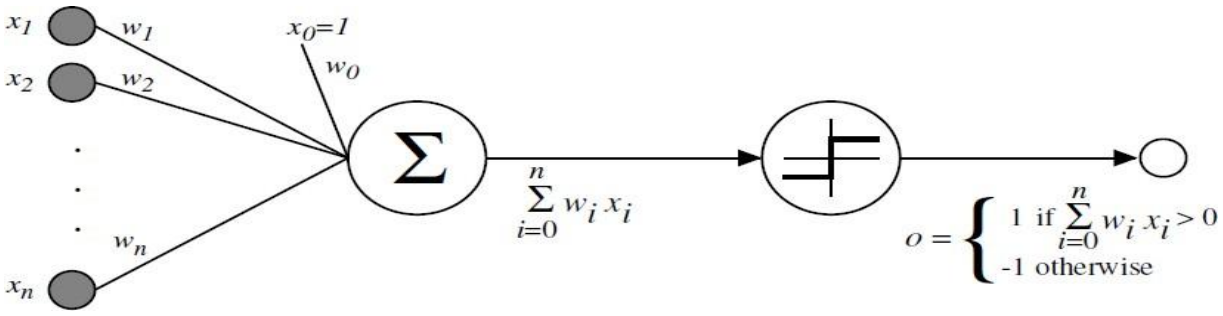


Figure: A perceptron

- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- Given inputs ***x*** through ***x,*** the output **O*(x1,* . . . , *xn)*** computed by the perceptron is
- Where, each w_i is a real-valued constant, or weight, that determines the

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

contribution of input x_i to the perceptron output.

- $-w_0$ is a threshold that the weighted combination of inputs $w_1x_1 + \ldots + w_nx_n$ must

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

Where,

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

surpass in order for the perceptron to output a 1.

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

Sometimes, the perceptron function is written as,

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

The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight wi associated with input xi according to the rule.

$$w_i \leftarrow w_i + \Delta w_i$$

Where,

$$\Delta w_i = \eta(t - o)x_i$$

Here,
t is the target output for the current training example
o is the output generated by the perceptron
η is a positive constant called the ***learning rate***

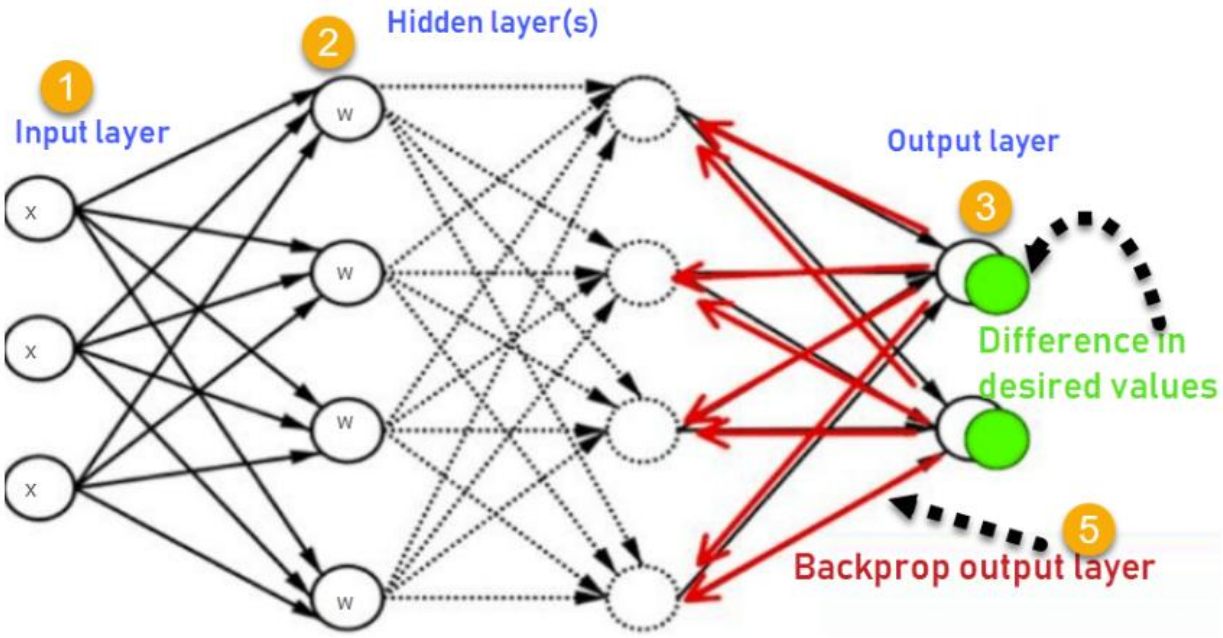
- The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases

Drawback:
The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.

The BACKPROPAGATION Algorithm

Backpropagation in neural network is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks.

The BACKPROPAGATION Algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.



1. nputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

In BACKPROPAGATION algorithm, we consider networks with multiple output units rather than single units as before, so we redefine E to sum the errors over all of the network output units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \quad \text{.....equ. (1)}$$

where,

- ***outputs*** - is the set of output units in the network
- ***tkd*** and ***Okd*** - the target and output values associated with the ***kth*** output unit
- ***d*** - training example

Algorithm:

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

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

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

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

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

Propagate the input forward through the network:

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

Propagate the errors backward through the network:

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

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

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

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

4. Update each network weight w_{ji}

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

where

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

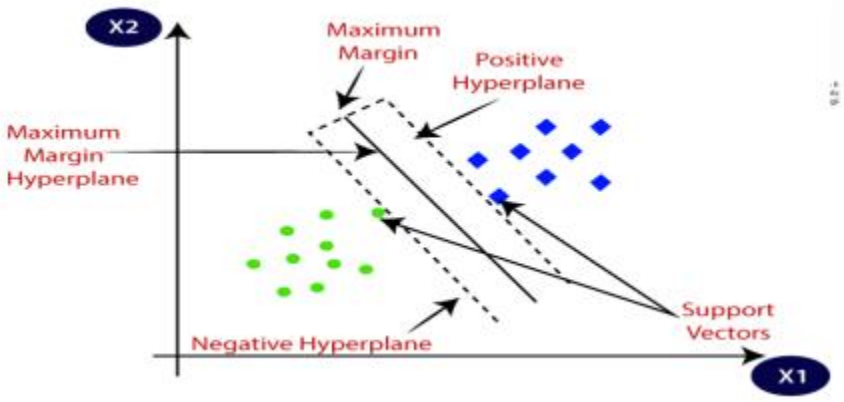
TABLE 4.2
The stochastic gradient descent version of the BACKPROPAGATION algorithm for feedforward networks containing two layers of sigmoid units.

Chapter III

SUPPORT VECTOR MACHINE

INTRODUCTION

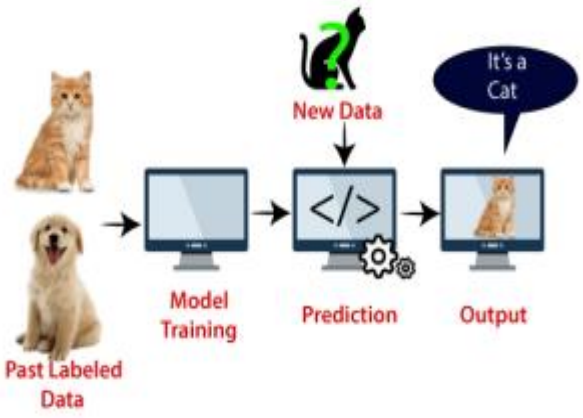
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case

of cat and dog. On the basis of the support vectors, it will classify it as a cat.

Consider the below diagram:



SVM algorithm can be used for Face detection, image classification, text categorization, etc. Types of SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier. O

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

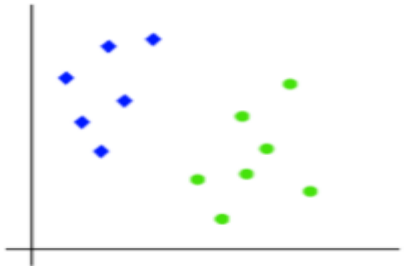
Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in ndimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM. The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Support Vectors: The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

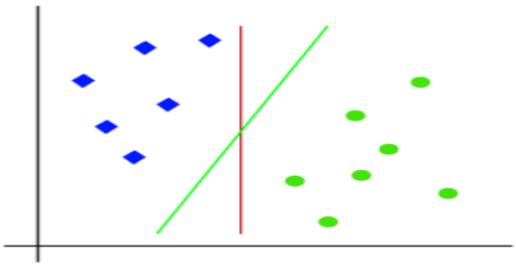
How does SVM works?

2.4.1.

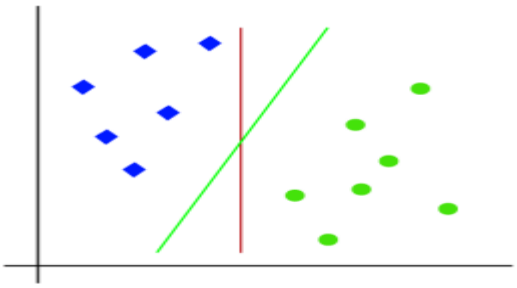
Linear SVM: The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



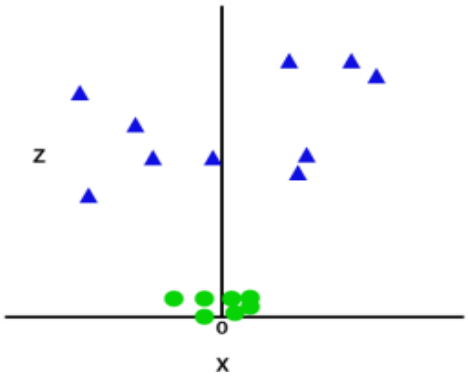
So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image



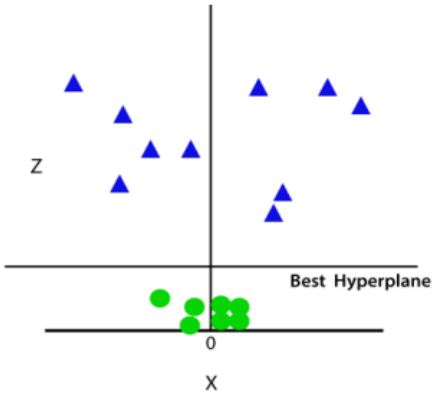
Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane. 2.4.2. Non-Linear SVM: If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



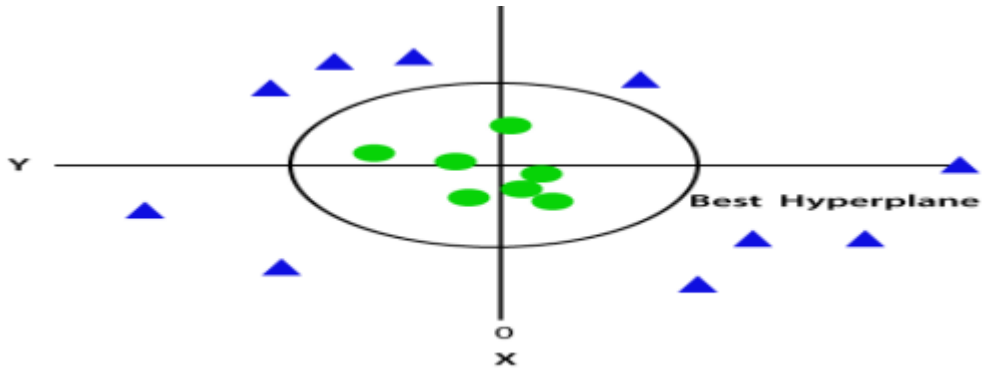
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as: z=x 2 +y2 By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Advantages of support vector machine:

- Support vector machine works comparably well when there is an understandable margin of dissociation between classes.
- It is more productive in high-dimensional spaces.
- It is effective in instances where the number of dimensions is larger than the number of specimens.
- Support vector machine is comparably memory systematic.Support Vector Machine (SVM) is a powerful supervised machine learning algorithm with several advantages. Some of the main advantages of SVM include:
- Handling high-dimensional data: SVMs are effective in handling high-dimensional data, which is common in many applications such as image and text classification.
- Handling small datasets: SVMs can perform well with small datasets, as they only require a small number of support vectors to define the boundary.
- Modeling non-linear decision boundaries: SVMs can model non-linear decision boundaries by using the kernel trick, which maps the data into a higher-dimensional space where the data becomes linearly separable.
- Robustness to noise: SVMs are robust to noise in the data, as the decision boundary is determined by the support vectors, which are the closest data points to the boundary.
- Generalization: SVMs have good generalization performance, which means that they are able to classify new, unseen data well.

- Versatility: SVMs can be used for both classification and regression tasks, and it can be applied to a wide range of applications such as natural language processing, computer vision and bioinformatics.
- Sparse solution: SVMs have sparse solutions, which means that they only use a subset of the training data to make predictions. This makes the algorithm more efficient and less prone to overfitting.
- Regularization: SVMs can be regularized, which means that the algorithm can be modified to avoid overfitting.

Disadvantages of support vector machine:

- Support vector machine algorithm is not acceptable for large data sets.
- It does not execute very well when the data set has more sound i.e. target classes are overlapping.
- In cases where the number of properties for each data point outstrips the number of training data specimens, the support vector machine will underperform.
- As the support vector classifier works by placing data points, above and below the classifying hyperplane there is no probabilistic clarification for the classification.
- Support Vector Machine (SVM) is a powerful supervised machine learning algorithm, but it also has some limitations and disadvantages. Some of the main disadvantages of SVM include:
- Computationally expensive: SVMs can be computationally expensive for large datasets, as the algorithm requires solving a quadratic optimization problem.
- Choice of kernel: The choice of kernel can greatly affect the performance of an SVM, and it can be difficult to determine the best kernel for a given dataset.
- Sensitivity to the choice of parameters: SVMs can be sensitive to the choice of parameters, such as the regularization parameter, and it can be difficult to determine the optimal parameter values for a given dataset.
- Memory-intensive: SVMs can be memory-intensive, as the algorithm requires storing the kernel matrix, which can be large for large datasets.
- Limited to two-class problems: SVMs are primarily used for two-class problems, although multi-class problems can be solved by using one-versus-one or one-versus-all strategies.

- Lack of probabilistic interpretation: SVMs do not provide a probabilistic interpretation of the decision boundary, which can be a disadvantage in some applications.
- Not suitable for large datasets with many features: SVMs can be very slow and can consume a lot of memory when the dataset has many features.
- Not suitable for datasets with missing values: SVMs requires complete datasets, with no missing values, it cannot handle missing values.