



MODULE-4

Rule based classification- 1R. Neural Networks-

Back propagation. Support Vector Machines,

Lazy Learners-K Nearest Neighbor Classifier.

Accuracy and error Measures evaluation.

Prediction:-Linear Regression and Non-Linear Regression.


- Rule Based Classifier



USING IF-THEN RULES FOR CLASSIFICATION

- In rule-based classifiers, the learned model is represented as a set of IF-THEN rules.
- Rules are a good way of representing information or bits of knowledge.
- A **rule-based classifier** uses a set of IF-THEN rules for classification.
- An **IF-THEN** rule is an expression of the form
$$\text{IF } \textit{condition} \text{ THEN } \textit{conclusion}$$
- *Example*

R1: IF *age* =*youth* AND *student* =*yes* THEN *buys_computer* = *yes*

- *IF* part is called - **rule antecedent** or **precondition**.
 - **THEN** part is called - **rule consequent**
- 

- $R1$ can also be written as

$$R1: (age = youth) \wedge (student = yes) \Rightarrow (buys_computer = yes).$$

- If the condition (i.e., all the attribute tests) in a rule antecedent holds true for a given tuple, we say that the rule antecedent is **satisfied** (or the rule is satisfied) and that the rule **covers** the tuple.
- A rule R can be assessed by its coverage and accuracy.

$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}.$$

- Where

X - given tuple

D – class labeled data set

n_{covers} be the number of tuples covered by R ;

$n_{correct}$ be the number of tuples correctly classified by R

$|D|$ be the number of tuples in D .



EXAMPLE

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

<i>RID</i>	<i>age</i>	<i>Income</i>	<i>student</i>	<i>credit_rating</i>	<i>Class: buys_computer</i>
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$R1: (age = youth) \wedge (student = yes) \Rightarrow (buys_computer = yes).$

○ For the rule $R1$, (tuple 9 and 11)

$coverage(R1) = 2/14 = 14.28\%$

$Accuracy(R1) = 2/2 = 100\%.$



HOW TO PREDICT THE CLASS LABEL ?

- If a rule is satisfied by X , the rule is said to be **triggered**.
- Suppose if

$X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit_rating} = \text{fair}).$

- X satisfies $R1$, which triggers the rule.
- If $R1$ is the only rule satisfied, then the rule **fires** by returning the class prediction for X .
- Note that triggering does not always mean firing because there may be more than one rule that is satisfied!
- **If more than one rule is triggered**- we have a potential problem- What if they each specify a different class? Or what if no rule is satisfied by X ?



- If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to **X** .
- There are many possible strategies.
- We look at two, namely
 1. *Size ordering*
 2. *Rule ordering*.



SIZE ORDERING

- The **size ordering** scheme assigns the highest priority to the triggering rule that has the “toughest” requirements, where toughness is measured by the rule antecedent *size*.
- That is, the triggering rule with the most attribute tests is fired.



RULE ORDERING

- The **rule ordering** scheme prioritizes the rules beforehand.
- The ordering may be
 1. *class-based* or
 2. *rule-based*.
- With **class-based ordering**, the classes are sorted in order of decreasing “importance” such as by decreasing *order of prevalence*.
- That is, all the rules for the most prevalent (or most frequent) class come first, the rules for the next prevalent class come next, and so on.
- Alternatively, they may be sorted based on the misclassification cost per class.
- Within each class, the rules are not ordered—they don’t have to be because they all predict the same



- With **rule-based ordering**, the rules are organized into one long priority list, according to some measure of rule **quality**- such as accuracy, coverage, or size (number of attribute tests in the rule antecedent), or based on advice from domain experts.
- When rule ordering is used, **the rule set is known as a decision list**.
- With rule ordering, **the triggering rule that appears earliest in the list has the highest priority, and so it gets to fire its class prediction.**
- Any other rule that satisfies X is ignored.
- Most rule-based classification systems use a class-based rule-ordering strategy.



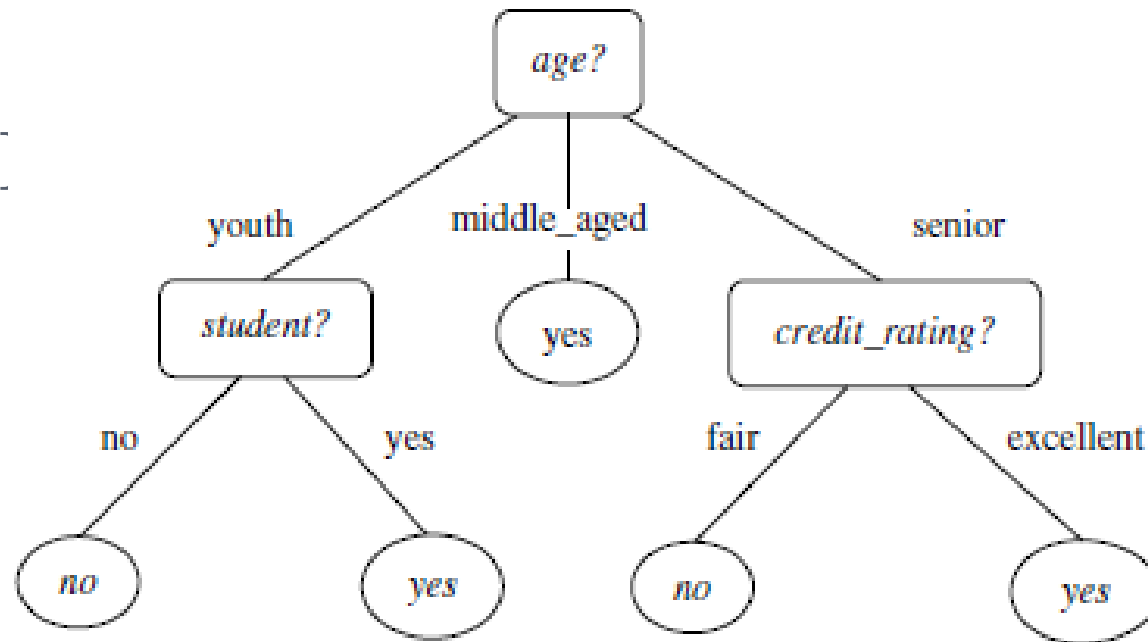
- What happens if there is no rule satisfied by X .
- How, then, can we determine the class label of X ?
- In this case, a fallback or **default rule** can be set up to specify a default class, based on a training set.
- This may be the class in majority or the majority class of the tuples that were not covered by any rule.
- The default rule is evaluated at the end, if and only if no other rule covers X .
- The condition in the default rule is empty.
- In this way, the rule fires when no other rule is satisfied.



RULE EXTRACTION FROM A DECISION TREE

- To extract rules from a decision tree, one rule is created for each path from the root to a leaf node.
- Each splitting criterion along a given path is logically ANDed to form the rule antecedent (“IF” part).
- The leaf node holds the class prediction, forming the rule consequent (“THEN” part).





R1: IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>no</i>	THEN <i>buys_computer</i> = <i>no</i>
R2: IF <i>age</i> = <i>youth</i>	AND <i>student</i> = <i>yes</i>	THEN <i>buys_computer</i> = <i>yes</i>
R3: IF <i>age</i> = <i>middle_aged</i>		THEN <i>buys_computer</i> = <i>yes</i>
R4: IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>excellent</i>	THEN <i>buys_computer</i> = <i>yes</i>
R5: IF <i>age</i> = <i>senior</i>	AND <i>credit_rating</i> = <i>fair</i>	THEN <i>buys_computer</i> = <i>no</i>



- A disjunction (logical OR) is implied between each of the extracted rules.
- They are **mutually exclusive** and **exhaustive**.
- *Mutually exclusive* - no rule conflicts here because no two rules will be triggered for the same tuple.
- *Exhaustive* - one rule for each possible attribute–value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter—they are *unordered*.



HOW CAN WE PRUNE THE RULE SET?

- For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned.
- The training tuples and their associated class labels are used to estimate rule accuracy.
- Any rule that does not contribute to the overall accuracy of the entire rule set can also be pruned.
- Other problems arise during rule pruning, however, as the rules *will no longer be mutually exclusive and exhaustive*.
- C4.5 orders the class rule sets so as to minimize the number of *false-positive errors* .
- The class rule set with the least number of false positives is examined first.



RULE INDUCTION USING A SEQUENTIAL COVERING ALGORITHM

- IF-THEN rules can be extracted directly from the training data - without having to generate a decision tree first.
- The name comes from the notion that the rules are learned *sequentially* (one at a time), where each rule for a given class will ideally *cover* many of the class's tuples.
- Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules.
- There are many sequential covering algorithms. Popular variations include AQ, CN2, and the more recent RIPPER.



- The general strategy is as follows.
- Rules are learned one at a time.
- Each time a rule is learned, the tuples covered by the rule are removed, and the process repeats on the remaining tuples



ALGORITHM

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- D , a data set of class-labeled tuples;
- Att_vals , the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
(1)  $Rule\_set = \{\}$ ; // initial set of rules learned is empty
(2) for each class  $c$  do
(3)     repeat
(4)          $Rule = \text{Learn\_One\_Rule}(D, Att\_vals, c)$ ;
(5)         remove tuples covered by  $Rule$  from  $D$ ;
(6)          $Rule\_set = Rule\_set + Rule$ ; // add new rule to rule set
(7)     until terminating condition;
(8) endfor
(9) return  $Rule\_Set$ ;
```

Basic sequential covering algorithm.



- The rules are grown in a *general-to-specific* manner
- We can think of this as a beam search, where we start off with an empty rule and then gradually keep appending attribute tests to it.
- To learn a rule for the class “accept,” we start off with the most general rule possible, that is, the condition of the rule antecedent is empty.
- The rule is

IF THEN loan decision = accept.

- We then consider each possible attribute test that may be added to the rule. These can be derived from the parameter *Att_vals*, which contains a list of attributes with their associated values.
- For example, for an attribute–value pair (*att*, *val*), we can consider attribute tests such as *att = val*, *att ≤ val*, *att > val*, and so on.



- *Learn_One_Rule* adopts a greedy depth-first strategy.
- Each time it is faced with adding a new attribute test (conjunct) to the current rule, it picks the one that most improves the rule quality, based on the training samples.
- Suppose *Learn_One_Rule* finds that the attribute test *income=high* **best improves the accuracy of our current** (empty) rule.
- We append it to the condition, so that the current rule becomes

IF income=high THEN loan decision=accept.

- The process repeats, where at each step we continue to greedily grow rules until the resulting rule meets an acceptable quality level.
- Greedy search does not allow for backtracking.




CLASSIFICATION BY BACK PROPAGATION

- Back propagation is a neural network learning algorithm.
- **Neural network** is a set of connected input/output units in which each connection has a weight associated with it.
- During the learning phase, the network learns by adjusting the **weights** so as to be able to predict the correct class label of the input tuples.

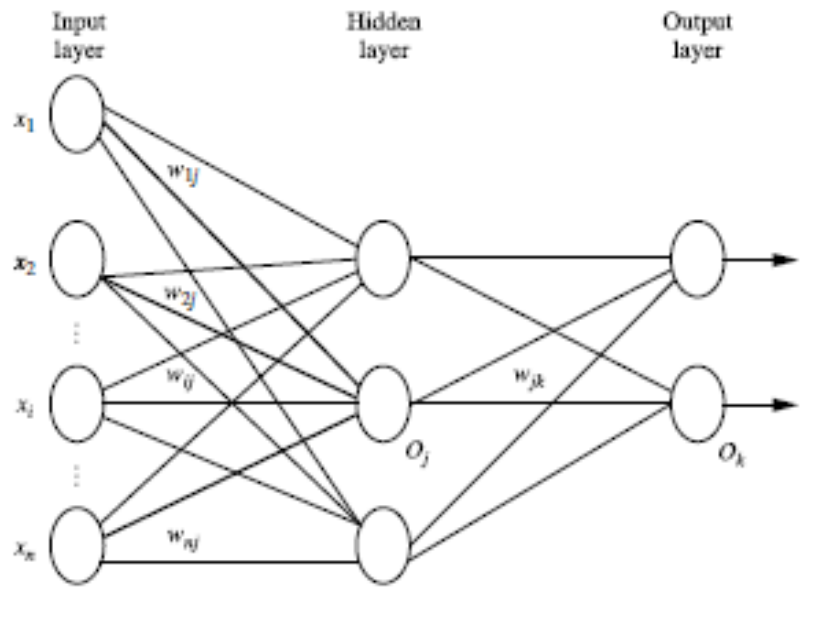


ADVANTAGES OF NEURAL NETWORKS

1. High tolerance of noisy data
 2. Ability to classify patterns on which they have not been trained.
 3. They can be used when you may have little knowledge of the relationships between attributes and classes.
 4. They are well suited for continuous-valued inputs *and* outputs, unlike most decision tree algorithms.
 5. They have been successful on a wide array of real-world data- including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text.
 6. Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process.
 7. In addition, several techniques have been recently developed for rule extraction from trained neural networks
- 

A MULTILAYER FEED-FORWARD NEURAL NETWORK

- The back propagation algorithm performs learning on a *multilayer feed-forward* neural network.
- It iteratively learns a set of weights for prediction of the class label of tuples. A **multi-layer feed-forward** neural network consists of an *input layer*, one or more *hidden layers*, and an *output layer*



Multilayer feed-forward neural network.



- Each layer is made up of units.
- The inputs to the network correspond to the attributes measured for each training tuple.
- The inputs are fed simultaneously into the units making up the **input layer**.
- These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuron like” units, known as a **hidden layer**.
- The outputs of the hidden layer units can be input to another hidden layer, and so on.
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network’s prediction for given tuples.



- Classification by Back propagation



BACK PROPAGATION

- Back propagation learns by iteratively processing a data set of training tuples, comparing the network's prediction for each tuple with the actual known *target* value.
- The target value may be the known class label of the training tuple (for classification problems) or a continuous value (for numeric prediction).
- For each training tuple, the weights are modified so as to minimize the mean-squared error between the network's prediction and the actual target value.
- These modifications are made in the “backwards” direction (i.e., from the output layer) through each hidden layer down to the first hidden layer (hence the name *back propagation*).



ALGORITHM –MAJOR STEPS

1. Initialize the weights:

- The **weights** in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5).
- Each unit has a **bias** associated with it, as explained later.
- The biases are similarly initialized to small random numbers.

Each training tuple, X , is processed by the following steps.

2. Propagate the inputs forward:

- First, the training tuple is fed to the network's input layer.
- The inputs pass through the input units, unchanged.
- For an input unit, j , O_j - output and I_j - input value
- Next, the **net input and output** of each unit in the hidden and output layers are computed.
- The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs.



- To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed.
- Given a unit, j in a hidden or output layer, the net input, I_j , to unit j is

$$I_j = \sum_i w_{ij} O_i + \theta_j,$$

- w_{ij} - weight of the connection from unit i in the previous layer to unit j ;
- O_i - output of unit i from the previous layer;
- θ_j is the **bias** of the unit - threshold that serves to vary the activity of the unit.
- Each unit in the hidden and output layers takes its net input and then applies an **activation** function to it- **logistic** or **sigmoid**.
- Given the net input I_j to unit j , then O_j , the output of unit j , is computed as

$$O_j = \frac{1}{1 + e^{-I_j}}.$$



3. Back propagate the error:

- The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction.
- For a unit j in the output layer, the error Err_j is computed by

$$Err_j = O_j(1 - O_j)(T_j - O_j),$$

- where O_j is the actual **output** of unit j , and T_j is the known **target value** of the given training tuple.
- To compute the **error of a hidden layer unit j** , the weighted sum of the errors of the units connected to unit j in the next layer are considered.
- The error of a hidden layer unit j is

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk},$$

- where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and Err_k is the error of unit k .



- The weights and biases are updated to reflect the propagated errors.
- Weights are updated by the following equations, where Δw_{ij} is the change in weight w_{ij} .

$$\Delta w_{ij} = (l) Err_j O_i.$$

$$w_{ij} = w_{ij} + \Delta w_{ij}.$$

- The variable l is the **learning rate**, a constant typically having a value between 0.0 and 1.0
- Biases are updated by the following equations, where $\Delta \theta_j$ is the change in bias θ_j

$$\Delta \theta_j = (l) Err_j.$$

$$\theta_j = \theta_j + \Delta \theta_j.$$



- Note that here we are updating the weights and biases after the presentation of each tuple. This is referred to as **case updating**.
- Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all the tuples in the training set have been presented. This latter strategy is called **epoch updating**, where one iteration through the training set is an **epoch**.

Terminating condition: Training stops when

- All Δw_{ij} in the previous epoch are so small as to be below some specified threshold, or
- The percentage of tuples misclassified in the previous epoch is below some threshold,
- or
- A pre specified number of epochs has expired.



HOW CAN WE CLASSIFY AN UNKNOWN TUPLE

- To classify an unknown tuple, \mathbf{X} , the tuple is input to the trained network, and the net input and output of each unit are computed.
- There is no need for computation and/or back propagation of the error.
- If there is **one output node per class**, then the output node with the highest value determines the predicted class label for \mathbf{X} .
- If there is **only one output node**, then output values greater than or equal to 0.5 may be considered as belonging to the positive class, while values less than 0.5 may be considered negative.



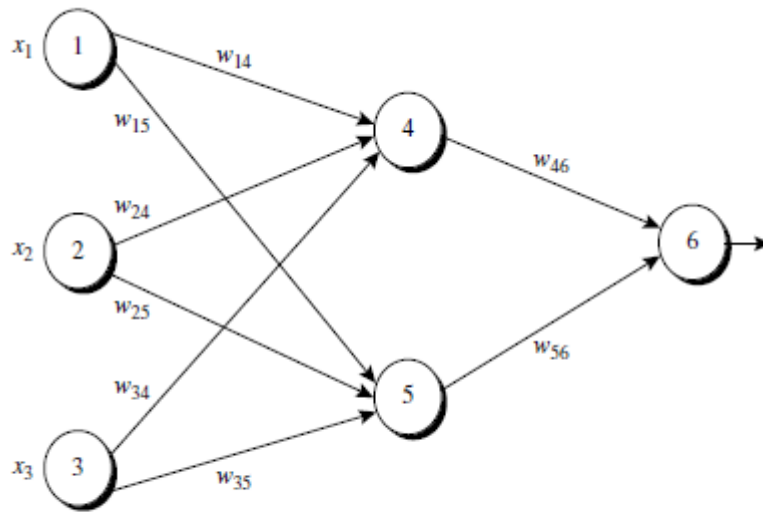
ALGORITHM

```
(1) Initialize all weights and biases in network;  
(2) while terminating condition is not satisfied {  
(3)   for each training tuple X in D {  
(4)     // Propagate the inputs forward:  
(5)     for each input layer unit j {  
(6)        $O_j = I_j$ ; // output of an input unit is its actual input value  
(7)     for each hidden or output layer unit j {  
(8)        $I_j = \sum_i w_{ij} O_i + \theta_j$ ; // compute the net input of unit j with respect to  
        the previous layer, i  
(9)        $O_j = \frac{1}{1 + e^{-I_j}}$ ; } // compute the output of each unit j  
(10)    // Backpropagate the errors:  
(11)    for each unit j in the output layer  
(12)       $Err_j = O_j(1 - O_j)(T_j - O_j)$ ; // compute the error  
(13)    for each unit j in the hidden layers, from the last to the first hidden layer  
(14)       $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$ ; // compute the error with respect to  
        the next higher layer, k  
(15)    for each weight  $w_{ij}$  in network {  
(16)       $\Delta w_{ij} = (l) Err_j O_i$ ; // weight increment  
(17)       $w_{ij} = w_{ij} + \Delta w_{ij}$ ; } // weight update  
(18)    for each bias  $\theta_j$  in network {  
(19)       $\Delta \theta_j = (l) Err_j$ ; // bias increment  
(20)       $\theta_j = \theta_j + \Delta \theta_j$ ; } // bias update  
(21)  }
```



BACK PROPAGATION- EXAMPLE

- Let the learning rate be 0.9. The initial weight and bias values of the network are given in Table along with the first training tuple, \mathbf{X} (1, 0, 1), with a class label of 1.

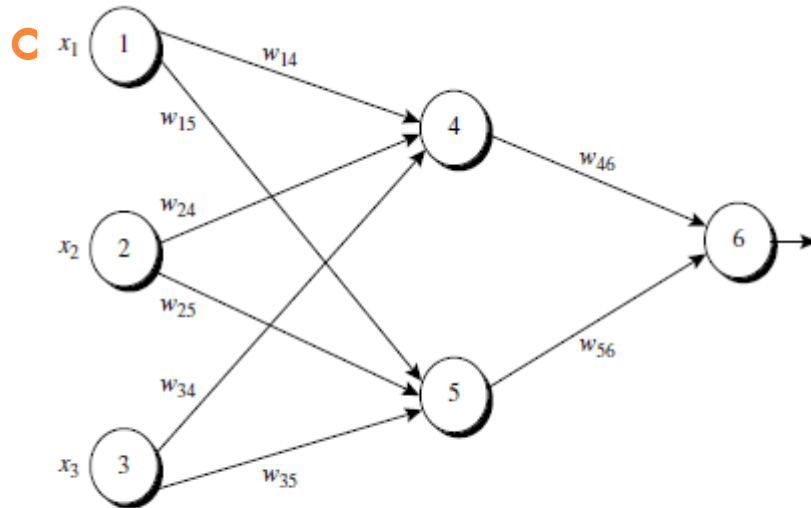


Example of a multilayer feed-forward neural network.

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

SOLUTION



Example of a multilayer feed-forward neural network.

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

$$I_j = \sum_i w_{ij} O_i + \theta_j, \quad O_j = \frac{1}{1 + e^{-I_j}}$$

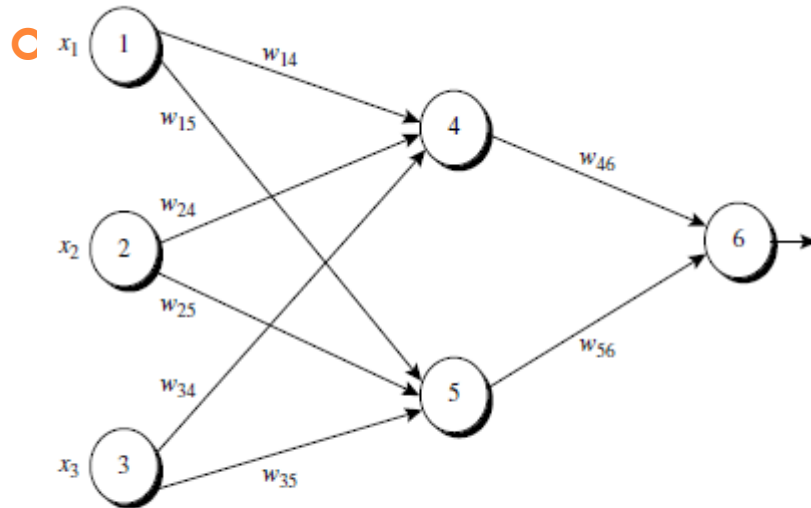
Net Input and Output Calculations

Unit, j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

$$2 \times 1 + 4 \times 0 + (-0.5) \times 1 + (-0.4) = -0.7$$



SOLUTION



Example of a multilayer feed-forward neural network.

$$I_j = \sum_i w_{ij} O_i + \theta_j, \quad O_j = \frac{1}{1 + e^{-I_j}}.$$

Net Input and Output Calculations

Unit, j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

$$Err_j = O_j(1 - O_j)(T_j - O_j),$$

Initial Input, Weight, and Bias Values

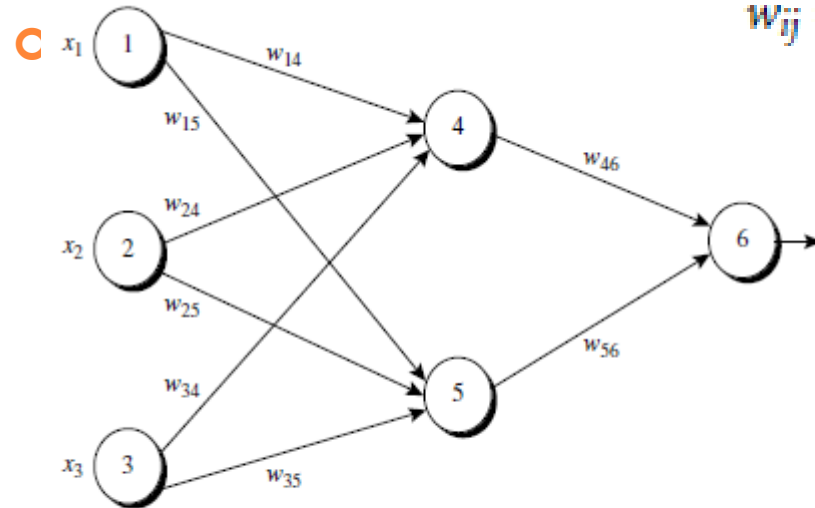
x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Calculation of the Error at Each Node

Unit, j	Err_j
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk},$$

SOLUTION



Example of a multilayer feed-forward neural network.

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Calculation of the Error at Each Node

Unit, j	Err_j
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

$$\Delta w_{ij} = (l) Err_j O_i.$$

$$w_{ij} = w_{ij} + \Delta w_{ij}.$$

$$\Delta \theta_j = (l) Err_j.$$

$$\theta_j = \theta_j + \Delta \theta_j.$$

Calculations for Weight and Bias Updating

Weight or Bias	New Value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

- Lazy Leaner (K-Nearest Neighbour)



EAGER LEARNERS

- **Eager learners**, when given a set of training tuples, will construct a classification model before receiving new (e.g., test) tuples to classify.
- We can think of the learned model as being ready and eager to classify previously unseen tuples.
- Eg. Decision tree induction, Bayesian classification, rule-based classification, classification by back propagation, support vector machines, and classification based on association rule mining.



LAZY LEARNER

- Waits until the last minute before doing any model construction to classify a given test tuple.
- When given a training tuple, a **lazy learner** simply stores it (or does only a little minor processing) and waits until it is given a test tuple.
- Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples.
- Lazy learners do less work when a training tuple is presented and more work when making a classification or numeric prediction.
- Because lazy learners store the training tuples or “instances,” they are also referred to as **instance-based learners**, even though all learning is essentially based on instances.



- When making a classification or numeric prediction, lazy learners can be computationally expensive.
- They require efficient storage techniques and are well suited to implementation on parallel hardware.
- They offer little explanation or insight into the data's structure.
- Lazy learners, however, naturally support incremental learning.



K-NEAREST-NEIGHBOR CLASSIFIERS

- Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it.
- The training tuples are described by n attributes.
- Each tuple represents a point in an n -dimensional space.
- In this way, all the training tuples are stored in an n -dimensional pattern space.
- When given an unknown tuple, a **k -nearest-neighbor classifier** searches the pattern space for the k training tuples that are closest to the unknown tuple.
- These k training tuples are the k “nearest neighbors” of the unknown tuple.



- “Closeness” is defined in terms of a distance metric, such as Euclidean distance.
- The Euclidean distance between two points or tuples, say, $X_1=(x_{11}, x_{12}, \dots, x_{1n})$ and $X_2=(x_{21}, x_{22}, \dots, x_{2n})$, is

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}.$$

- Typically, we normalize the values of each attribute before using above equation.
- This helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes).
- For example ,Min-max normalization,



- For k -nearest-neighbor classification, the unknown tuple is assigned the most common class among its k -nearest neighbors.
- When $k = 1$, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space.
- Nearest-neighbor classifiers can also be used for **numeric prediction**, that is, to return a real-valued prediction for a given unknown tuple.
- In this case, the classifier **returns the average value of** the real-valued labels associated with the k -nearest neighbors of the unknown tuple.



HOW TO HANDLE NOMINAL DATA

- For nominal attributes, a simple method is to compare the corresponding value of the attribute in tuple $X1$ with that in tuple $X2$.
- If the two are identical (e.g., tuples $X1$ and $X2$ both have the color blue), then the difference between the two is taken as 0.
- If the two are different (e.g., tuple $X1$ is blue but tuple $X2$ is red), then the difference is considered to be 1.
- Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a larger difference score is assigned, say, for blue and white than for blue and black).



WHAT ABOUT MISSING VALUES?"

- For **nominal attributes**, we take the difference value to be 1 if either one or both of the corresponding values of A are missing.
- If A is **numeric** and missing from both tuples $X1$ and $X2$, then the difference is also taken to be 1.
- If only one value is missing and the other (which we will call v') is present and normalized, then we can take the difference to be either $|1-v'|$ or $|0-v'|$ (i.e., $1-v'$ or v'), whichever is greater.



HOW CAN I DETERMINE FOR K

- This can be determined experimentally.
- Starting with $k = 1$, we use a test set to estimate the error rate of the classifier.
- This process can be repeated each time by incrementing k to allow for one more neighbor.
- The k value that gives the minimum error rate may be selected.
- In general, the larger the number of training tuples, the larger the value of k will be



- Model Evaluation and Selection



METRICS FOR EVALUATING CLASSIFIER PERFORMANCE

- It measures for assessing how good or how “accurate” your classifier is at predicting the class label of tuples.
- **positive tuples** - tuples of the main class of interest
- **negative tuples** - all other tuples
- There are four additional terms we need to know that are the “building blocks” used in computing many evaluation measures.



- **True positives - TP :** These refer to the positive tuples that were correctly labeled by the classifier. Let TP be the number of true positives.
- **True negatives - TN :** These are the negative tuples that were correctly labeled by the classifier. Let TN be the number of true negatives.
- **False positives- FP :** These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class *buys computer = no* for which the classifier predicted *buys computer = yes*). Let FP be the number of false positives.
- **False negatives- FN :** These are the positive tuples that were mislabeled as negative (e.g., tuples of class *buys computer = yes* for which the classifier predicted *buys computer = no*). Let FN be the number of false negatives.



CONFUSION MATRIX

- These terms are summarized in the **confusion matrix**
- The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes.
- TP and TN tell us when the classifier is getting things right, while FP and FN tell us when the classifier is getting things wrong.

		Predicted class		
		yes	no	Total
Actual class	yes	TP	FN	P
	no	FP	TN	N
Total		P'	N'	$P + N$

! Confusion matrix, shown with totals for positive and negative tuples.

Classes	$buys_computer = yes$	$buys_computer = no$	Total	Recognition (%)
$buys_computer = yes$	6954	46	7000	99.34
$buys_computer = no$	412	2588	3000	86.27
Total	7366	2634	10,000	95.42

i Confusion matrix for the classes $buys_computer = yes$ and $buys_computer = no$, where an entry in row i and column j shows the number of tuples of class i that were labeled by the classifier as class j . Ideally, the nondiagonal entries should be zero or close to zero.



EVALUATION MEASURES

<i>Measure</i>	<i>Formula</i>
accuracy, recognition rate	$\frac{TP + TN}{P + N}$
error rate, misclassification rate	$\frac{FP + FN}{P + N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP + FP}$
F , F_1 , F -score, harmonic mean of precision and recall	$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
F_β , where β is a non-negative real number	$\frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}}$



- In addition to accuracy-based measures, classifiers can also be compared with respect to the following additional aspects:
- **Speed:** This refers to the computational costs involved in generating and using the given classifier.
- **Robustness:** This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.
- **Scalability:** This refers to the ability to construct the classifier efficiently given large amounts of data. Scalability is typically assessed with a series of data sets of increasing size.
- **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor.



RELIABLE CLASSIFIER ACCURACY ESTIMATES - METHODS

- **Holdout Method and Random Subsampling**
- **Cross – validation**
- **Bootstrap**



HOLDOUT METHOD AND RANDOM SUBSAMPLING

- In holdout method, the given data are randomly partitioned into two independent sets, a *training set* and a *test set*.
- Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set.
- The training set is used to derive the model.
- **Random subsampling** is a variation of the holdout method in which the holdout method is repeated k times.
- The overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.



CROSS-VALIDATION

- In ***k*-fold cross-validation**, the initial data are randomly partitioned into k mutually exclusive subsets or “folds,” D_1, D_2, \dots, D_k , each of approximately equal size.
- Training and testing is performed k times.
- In iteration i , partition D_i is reserved as the test set, and the remaining partitions are collectively used to train the model.
- That is, in the first iteration, subsets D_2, \dots, D_k collectively serve as the training set to obtain a first model, which is tested on D_1 ; the second iteration is trained on subsets D_1, D_3, \dots, D_k and tested on D_2 ; and so on.
- Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and once for testing.
- For classification, the accuracy estimate is the overall number of correct classifications from the k iterations, divided by the total number of tuples in the initial data.



BOOTSTRAP

- The **bootstrap method** samples the given training tuples uniformly *with replacement*.
- That is, each time a tuple is selected, it is equally likely to be selected again and re-added to the training set.
- For instance, imagine a machine that randomly selects tuples for our training set.
- In *sampling with replacement*, the machine is allowed to select the same tuple more than once.



PREDICTION

- Predictor that predicts a continuous-valued-function or ordered value.
- **Regression analysis** is a statistical methodology that is most often used for numeric prediction.
- Linear regression is the simplest form of regression.
- In linear regression, data are modeled using a straight line.
- It models a random variable Y(called response variable) as a linear function of another variable X (called predictor variable)
- It is represented by the equation

$Y=a+bX$ a – intercept, b- slope of regression line

Also called regression coefficients



- Those coefficients can be solved by the method of least square.
- The best-fit line is achieved using the Least-Squared method.
- This method minimizes the sum of the squares of the deviations from each of the data points to the regression line.
- The negative and positive distances do not get cancelled out here as all the deviations are squared.
- Given S samples or data points of the form $(x_1, y_1), (x_2, y_2), \dots, (x_s, y_s)$, then regression coefficient can be estimated using the method



EXAMPLE

- Given the experience of an employee as years and their salary. Find the prediction equation

X <i>years experience</i>	Y <i>salary (in \$1000)</i>
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83



- The prediction equation is given by $Y=a+bX$ where a and b is given by

$$b = \frac{\sum_{i=1}^S (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^S (x_i - \bar{x})^2}$$
$$a = \bar{y} - b\bar{x}$$

- This can be calculated as

$$b = \frac{(3-9.1)(30-55.4) + (8-9.1)(57-55.4) + \dots + (16-9.1)(83-55.4)}{(3-9.1)^2 + (8-9.1)^2 + \dots + (16-9.1)^2}$$
$$= \underline{\underline{3.7}}$$



- Also a is given by

$$\begin{aligned}\therefore a &= \bar{y} - b\bar{x} \\ &= 55.4 - (3.7)(9.1) = \underline{\underline{21.7}}\end{aligned}$$

- Therefore prediction eqn. is

$$Y = a + bX$$

$$\underline{\underline{Y = 21.7 + 3.1X}}$$



MULTIPLE REGRESSION

- When X is made up of more than one variables (or features) it is termed as multiple linear regression.
- It uses two or more independent variables to predict an outcome and a single continuous dependent variable.
- Ie $Y = a + b_1X_1 + b_2X_2 + \dots + b_kX_k$



NON LINEAR REGRESSION

- Given response variable and predictor variable has a relationship that may be modeled by a polynomial function

$$Y = a + b \cdot X^2$$

- In this particular regression, the line of best fit is not a straight line like in Linear Regression.
- However, it is a curve that is fitted to all the data points.

