Unit-III
Classification and Prediction

Dr. K. Raghava Rao
Professor of CSE
Dept. of MCA
KL University

Bayesian Classification

Thomas Bayes (1702-1761)

Bayesian Classification: Why?

A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities, such as the probability that a given a tuple belong to a particular class.

- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers.
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data.
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured.

Bayesian Classification: Why?

- Naïve Baysian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes.

  This assumption is called class conditional independence.

Bayesian Theorem: Basics

- Let **X** be a data tuple ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C .
- Classification is to determine P(H|**X**), the probability that the hypothesis holds given the "evidence" or observed data tuple **X.**
- P(H) (*prior probability*), the initial probability
  - E.g., **X** will buy computer, regardless of age, income, ...
- P(**X**): probability that tuple data is observed
- P(**X**|H) (*posteriori probability*), the probability of observing the **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayesian Theorem: Basics

- P(X) is the prior probability of X. An example is the probability that a person from our set of customers is 35 years old and earns $40,000.

Bayesian Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_2 iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Towards Naïve Bayesian Classifier

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem $P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$
- Since P(X) is constant for all classes, only $P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$

  needs to be maximized

  Bayesian classifier predicts that tuple X belongs to the class Ci
  If and only if P(Ci/X) > P(Cj/X) for i<= j<=m, i not equal to j.

Derivation of Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution

- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i, D}|$, the no. of tuples of C_i in D

- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

defined as $$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

so that $$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Towards Naïve Bayesian Classifier

- In order to predict the class label of X, P(X/Ci) is evaluated for each class Ci.
- The classifier predicts that the class label of tuple X is the class Ci if and only if
- P(X|Ci) P(Ci)>P(X/Cj) P(Cj) for i<=j<=m, j not equal to i.

Naïve Bayesian Classifier: Training Dataset

age	income	student	credit_rating	_comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

Naïve Bayesian Classifier: An Example

- We need to maximize P(X/C_i)P(Ci): The prior probability of each class
- P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

- Compute P(X|C_i) for each class

  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

P(X|C_i) : P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

P(X|C_i)*P(C_i) : P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Avoiding the 0-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be non-zero. Otherwise, the predicted prob. will be zero

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0) tuples, income= medium (990) tuples, and income = high (10) tuples,
- Use Laplacian correction (or Laplacian estimator)
  - Adding 1 to each case

    Prob(income = low) = 1/1003=0.001

    Prob(income = medium) = 991/1003=0.988

    Prob(income = high) = 11/1003=0.011
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

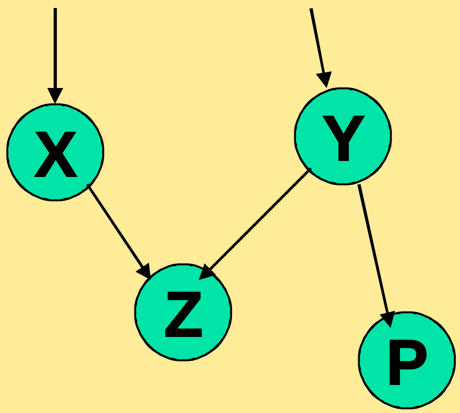
Naïve Bayesian Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.

    Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayesian Classifier
- How to deal with these dependencies?
  - Bayesian Belief Networks

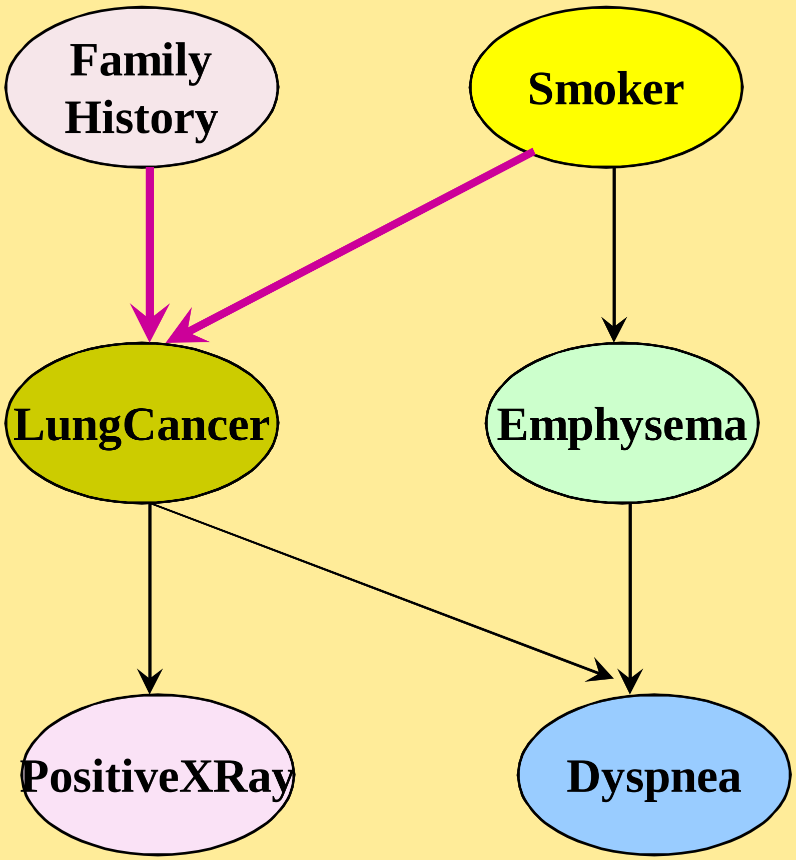
Bayesian Belief Networks

- Bayesian belief network allows a *subset* of the variables conditionally independent
- A graphical model of causal relationships
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution



- ❑ Nodes: random variables
- ❑ Links: dependency
- ❑ X and Y are the parents of Z, and Y is the parent of P
- ❑ No dependency between Z and P
- ❑ Has no loops or cycles

Bayesian Belief Network: An Example



Bayesian Belief Networks

The **conditional probability table** (**CPT**) for variable LungCancer:

	(FH, S)	(FH, ~S)	(~FH, S)	(~FH, ~S)
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

CPT shows the conditional probability for each possible combination of its parents

Derivation of the probability of a particular combination of values of **X**, from CPT:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(Y_i))$$

Training Bayesian Belief Networks

- Several scenarios:
  - Given both the network structure and all variables observable: *learn only the CPTs*
  - Network structure known, some hidden variables: *gradient descent* (greedy hill-climbing) method, analogous to neural network learning
  - Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*
  - Unknown structure, all hidden variables: No good algorithms known for this purpose
- Ref. D. Heckerman: Bayesian networks for data mining

Training Bayesian Belief Networks

- for data mining

Rule Based Classification

Using IF-THEN Rules for Classification

- Represent the knowledge in the form of IF-THEN rules

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

  - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

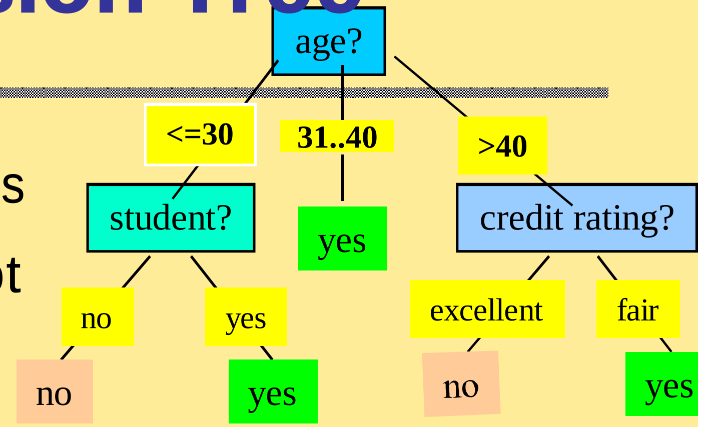
  coverage(R) = n_{covers} /|D| /* D: training data set */

  accuracy(R) = $n_{correct}$ / n_{covers}
- If more than one rule is triggered, need **conflict resolution strategy are 2**
  - 1)Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute test*)
  - 2)Rule ordering : this scheme prioritizes the rules before hand. Ordering may be based class based or rule based. This is known as **Decision list**

  - **2.1 rule based**: rules are organized into one long priority list, according to some measure of rule quality or by experts. Ordering may be class based or rule based.
  - **2.2 Class-based** ordering: classes are sorted in order of decreasing "importance".

Rule Based Classification

Rule Extraction from a Decision Tree



- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive
- Example: Rule extraction from our *buys_computer* decision-tree

IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*

IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*

IF *age* = mid-age THEN *buys_computer* = *yes*

IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*

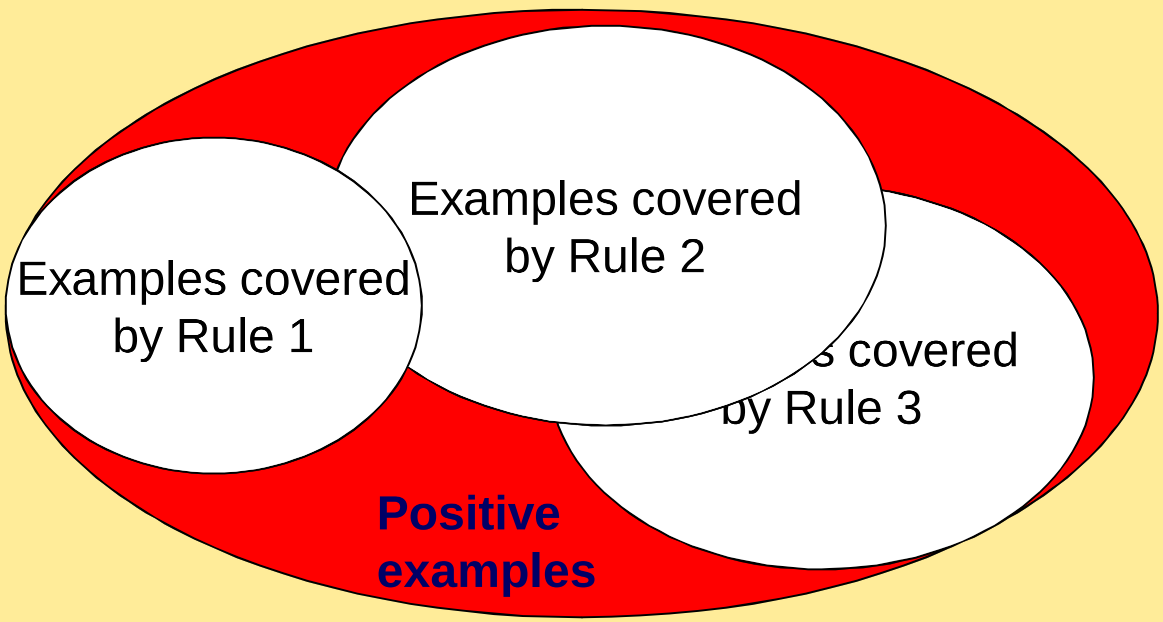
IF *age* = young AND *credit_rating* = *fair* THEN *buys_computer* = *no*

Rule Extraction from the Training Data

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - The process repeats on the remaining tuples unless *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithm

while (enough target tuples left)
    generate a rule
    remove positive target tuples satisfying this rule



Rule Generation

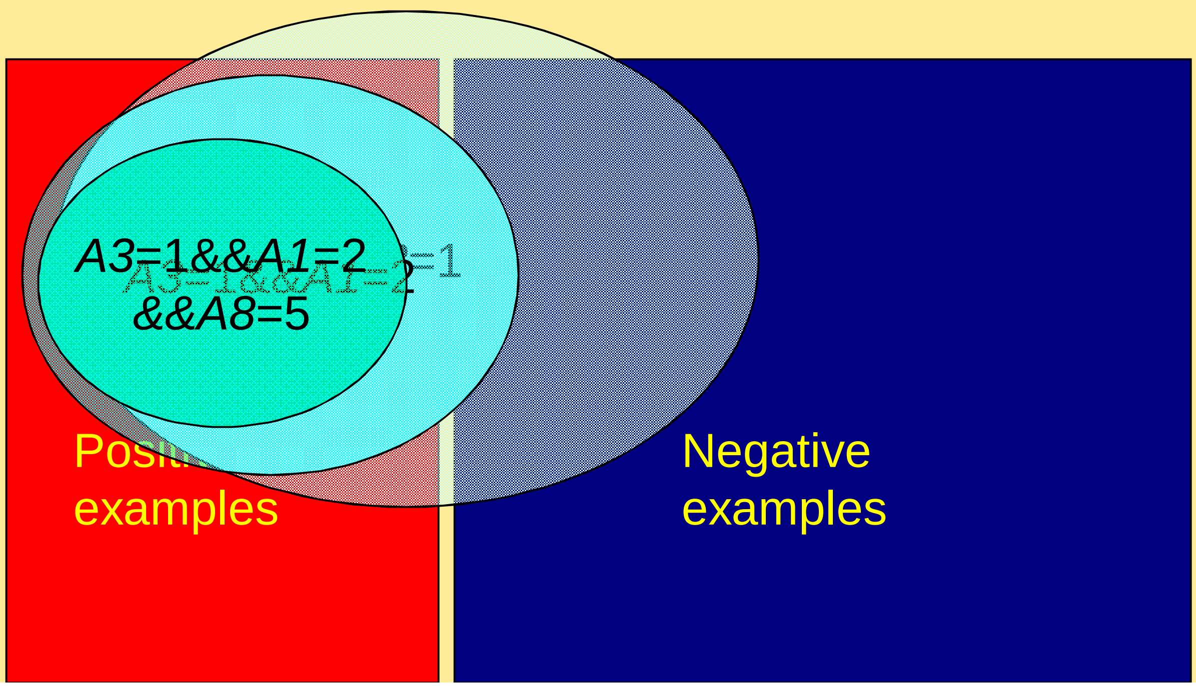
- To generate a rule

  while(true)

  find the best predicate *p*

  if foil-gain(*p*) > threshold **then** add *p* to current rule

  else break



Rule Quality Measures

How to Learn-One-Rule?

- Start with the *most general rule* possible: condition = empty
- *Adding new attributes* by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Rule-Quality measures: consider both coverage and accuracy
  - Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition $$FOIL\_Gain = pos' \times \left(\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg}\right)$$
    - favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

Pos/neg are # of positive/negative tuples covered by R.

If *FOIL_Prune* is higher for the pruned version of R, prune R

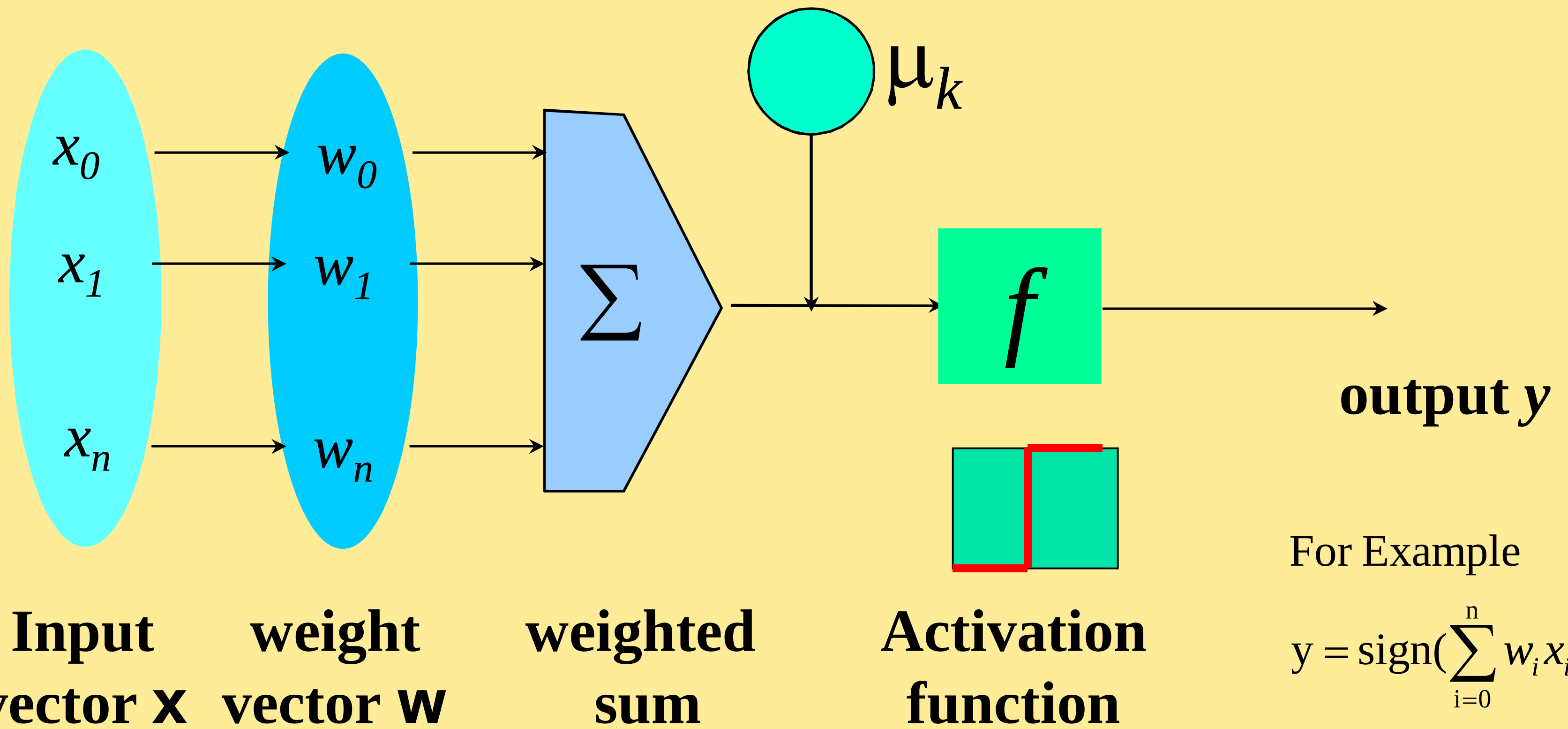
Classification by Backpropagation

- **Backpropagation**: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- **A neural network**: A set of connected input/output units where 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
- Also referred to as **connectionist learning** due to the connections between units

Neural Network as a Classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

A Neuron (= a perceptron)

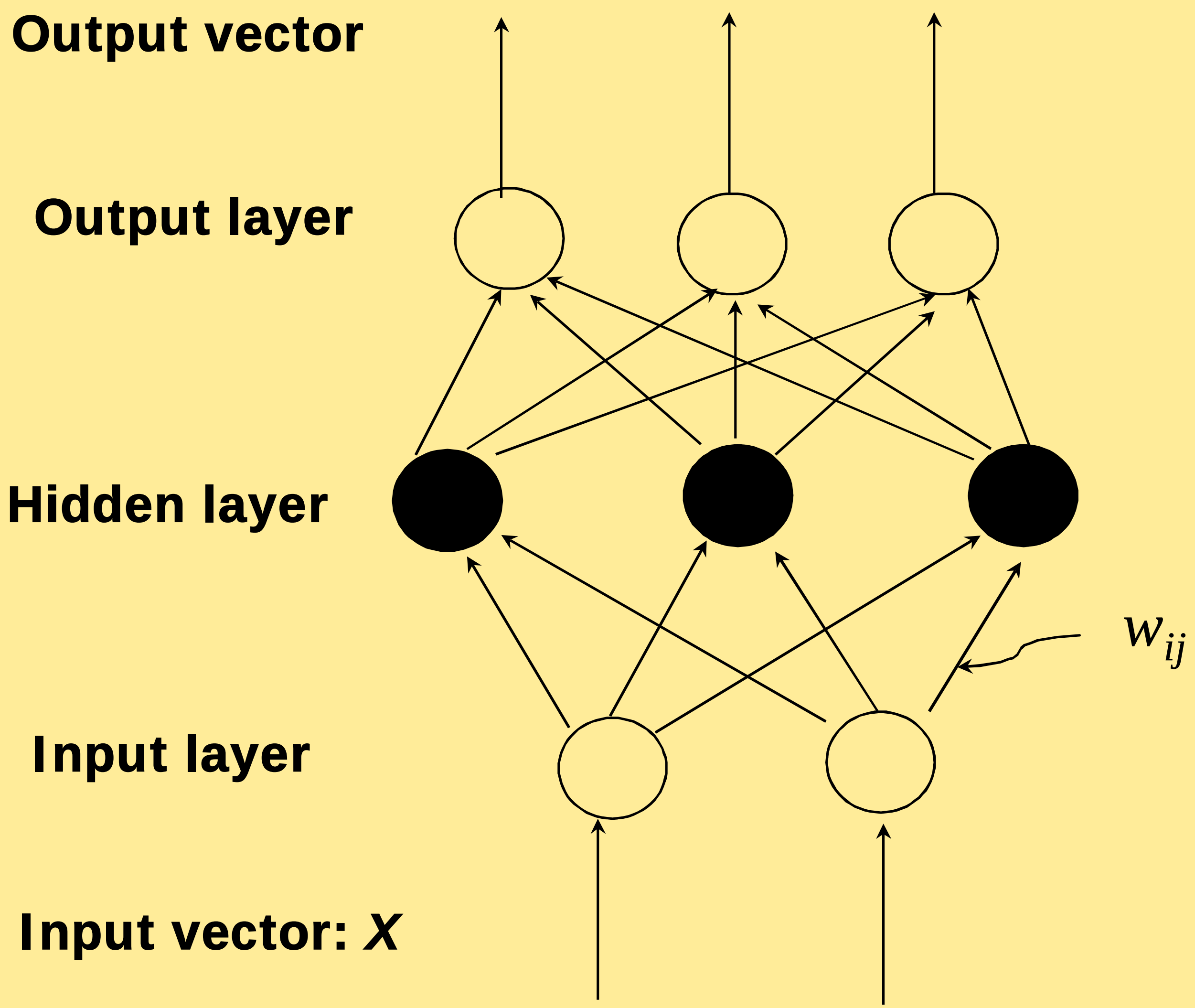


For Example

$$y = \text{sign}\left(\sum_{i=0}^{n} w_i x_i - \mu_k\right)$$

- The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

A Multi-Layer Feed-Forward Neural Network



How A Multi-Layer Neural Network Works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Defining a Network Topology

- First decide the **network topology:** # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalizing the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- Neural networks can be used for both classification and prediction
- For classification one output unit may be used to represent two classes
- for classification and more than two classes, one output unit per class is used

Contd.. Defining a Network Topology

- There are no clear rules for the ‘best’ number of hidden layer units
- NN design is a trial and error process
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

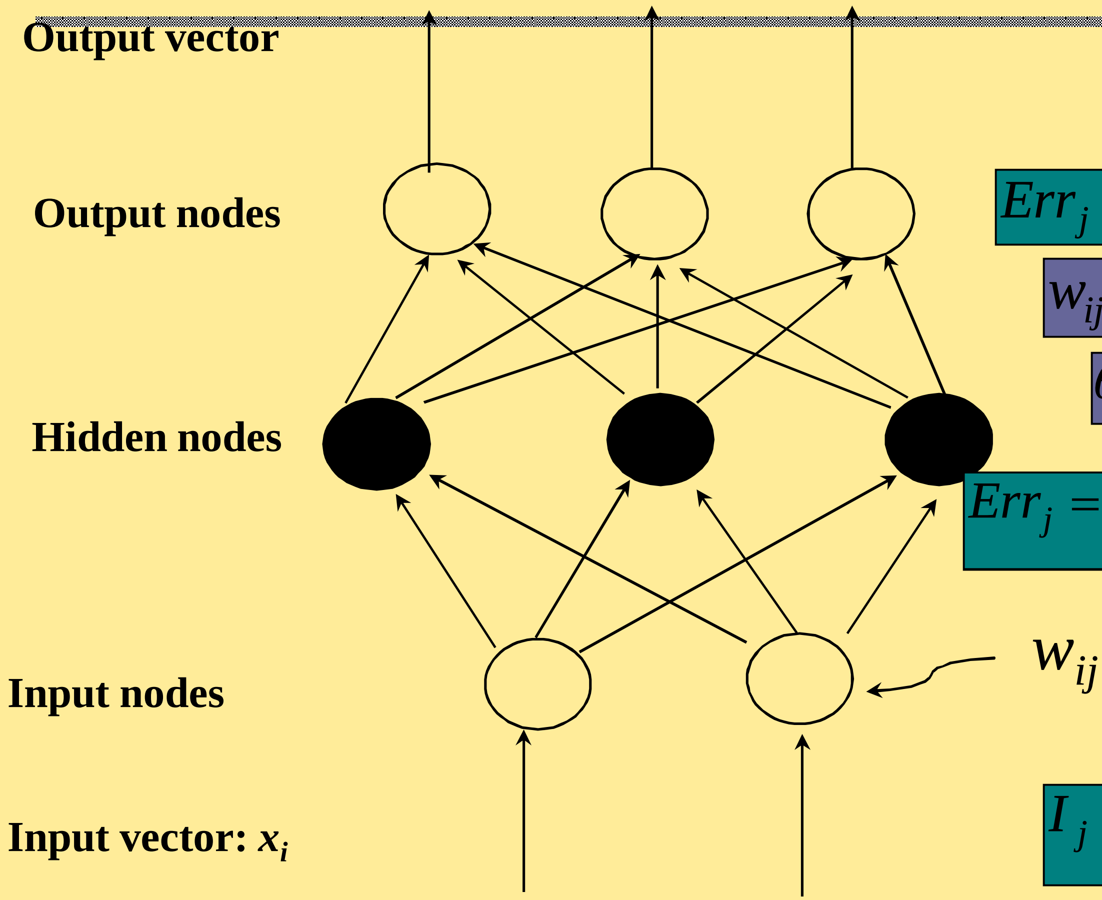
Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Network Training

- The ultimate objective of training
  - obtain a set of weights that makes almost all the tuples in the training data classified correctly
- Steps
  - Initialize weights with random values
  - Feed the input tuples into the network one by one
  - For each unit
    - Compute the net input to the unit as a linear combination of all the inputs to the unit
    - Compute the output value using the activation function
    - Compute the error
    - Update the weights and the bias

Multi-Layer Perceptron



Oj(1-Oj): derivative of the sig. function

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

Error of output layer

$$w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\theta_j = \theta_j + (l) Err_j$$

l: learning rate [0..1]

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

Error of hidden layer

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

Oj: the output of unit j
sigmoid function: map Ij to [0..1]

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

Ij: the net input to the unit j

l: too small → learning pace is too slow
too large → oscillation between wrong solutions
Heuristic: l=1/t (t: # iterations through training set so far)

Backpropagation algorithm

Input :

- D, a data set consisting of the training tuples and their associated target values
- L, the learning rate
- Network, a multilayer feed-forward network

Output : A trained neural network

Method :

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. Oj = Ij; // output of an input unit is its actual input value
7. for each hidden or output layer unit j {
8. //compute the net input of unit j w.r.t the previous layer, i $I_j = \sum_i w_{ij} O_i + \theta_j$

9. $O_j = \frac{1}{1+e^{-I_j}}$ } // compute the output of each unit j

Contd.. Backpropagation algorithm

10. // Backpropagation 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 w.r.t the next higher layer, k

15. for each weight wij in network {

16. Δwij = (l)Errkwjk ; // weight increment

17. *wij* = *wij* + Δwij ; } // weight update

18. for each bias θj in network {

19. Δθj = (l) Errj; // bias increment

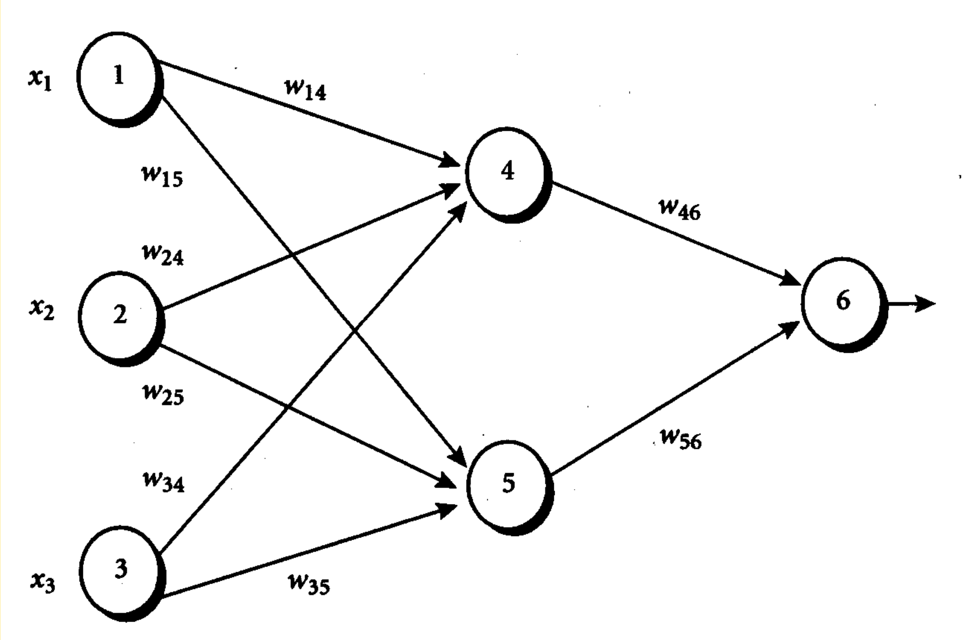
20. θj = θj + Δθj ; // bias update

21. } }

Multi-Layer Perceptron

- Case updating vs. epoch updating
  - Weights and biases are updated after presentation of each sample
  - Deltas are accumulated into variables throughout the whole training examples and then update
  - Case updating is more common (more accurate)
- Termination condition
  - Delta is too small (converge)
  - Accuracy of the current epoch is high enough
  - Pre-specified number of epochs
  - In practice, hundreds of thousands of epochs

Example



Initial input, weight, and bias values.

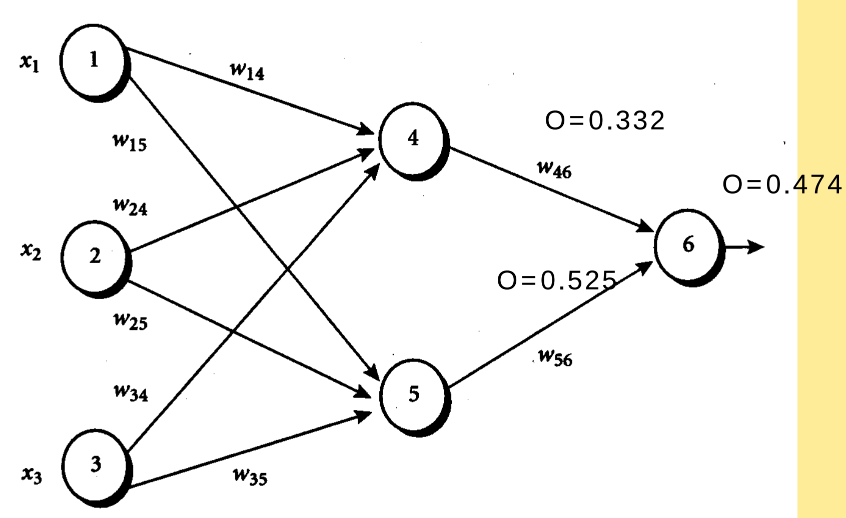
x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Class label = 1

Example

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



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

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

The 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	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

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

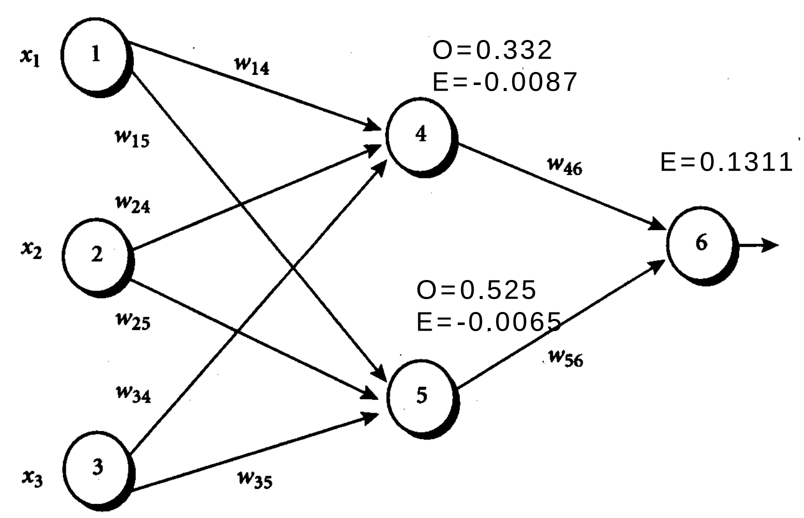
$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

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	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Example

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



$$w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\theta_j = \theta_j + (l) Err_j$$

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	$-0.4 + (0.9)(-0.0087) = -0.408$

Backpropagation and Interpretability

- Efficiency of backpropagation: Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in the worst case.

Network Pruning and Rule Extraction

- Network pruning
  - Fully connected network will be hard to articulate
  - N input nodes, h hidden nodes and m output nodes lead to h(m+N) weights
  - Pruning: Remove some of the links without affecting classification accuracy of the network
- Extracting rules from a trained network
  - Discretize activation values; replace individual activation value by the cluster average maintaining the network accuracy
  - Enumerate the output from the discretized activation values to find rules between activation value and output
  - Find the relationship between the input and activation value
  - Combine the above two to have rules relating the output to input

Network Pruning and Rule Extraction

- Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

What Is Prediction?

- (Numerical) prediction is similar to classification
  - construct a model
  - use model to predict continuous or ordered value for a given input
- Prediction is different from classification
  - Classification refers to predict categorical class label
  - Prediction models continuous-valued functions
- Major method for prediction: regression
  - model the relationship between one or more *independent* or **predictor** variables and a *dependent* or **response** variable
- Regression analysis
  - Linear and multiple regression
  - Non-linear regression
  - Other regression methods: generalized linear model, Poisson regression, log-linear models, regression trees

Linear Regression

- <u>Linear regression</u>: involves a response variable y and a single predictor variable x

  $y = w_0 + w_1 x$

  where w_0 (y-intercept) and w_1 (slope) are regression coefficients
- <u>Method of least squares</u>: estimates the best-fitting straight line

$$w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2} \qquad w_0 = \bar{y} - w_1 \bar{x}$$

- <u>Multiple linear regression</u>: involves more than one predictor variable
  - Training data is of the form ($\mathbf{X_1}$, y_1), ($\mathbf{X_2}$, y_2),..., ($\mathbf{X_{|D|}}$, $y_{|D|}$)
  - Ex. For 2-D data, we may have: $y = w_0 + w_1 x_1 + w_2 x_2$
  - Solvable by extension of least square method or using SAS, S-Plus
  - Many nonlinear functions can be transformed into the above

Nonlinear Regression

- Some nonlinear models can be modeled by a polynomial function
- A polynomial regression model can be transformed into linear regression model. For example,

  $$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

  convertible to linear with new variables: $x_2 = x^2$, $x_3 = x^3$

  $$y = w_0 + w_1 x + w_2 x_2 + w_3 x_3$$

- Other functions, such as power function, can also be transformed to linear model
- Some models are intractable nonlinear (e.g., sum of exponential terms)
  - possible to obtain least square estimates through extensive calculation on more complex formulae

Other Regression-Based Models

- Generalized linear model:
  - Foundation on which linear regression can be applied to modeling categorical response variables
  - Variance of y is a function of the mean value of y, not a constant
  - Logistic regression: models the prob. of some event occurring as a linear function of a set of predictor variables
  - Poisson regression: models the data that exhibit a Poisson distribution
- Log-linear models: (for categorical data)
  - Approximate discrete multidimensional prob. distributions
  - Also useful for data compression and smoothing
- Regression trees and model trees
  - Trees to predict continuous values rather than class labels

Accuracy and Error Measures

Classifier Accuracy Measures

	C_1	C_2
C_1	True positive	False negative
C_2	False positive	True negative

classes	buy_computer = yes	buy_computer = no	total	recognition(%)
buy_computer = yes	6954	46	7000	99.34
buy_computer = no	412	2588	3000	86.27
total	7366	2634	10000	95.52

- Accuracy of a classifier M, acc(M): percentage of test set tuples that are correctly classified by the model M
  - Error rate (misclassification rate) of M = 1 – acc(M)
  - Given *m* classes, $CM_{i,j}$, an entry in a **confusion matrix**, indicates # of tuples in class *i* that are labeled by the classifier as class *j*
- Alternative accuracy measures (e.g., for cancer diagnosis)

  sensitivity = t-pos/pos /* true positive recognition rate */

  specificity = t-neg/neg /* true negative recognition rate */

  precision = t-pos/(t-pos + f-pos)

  accuracy = sensitivity * pos/(pos + neg) + specificity * neg/(pos + neg)

  - This model can also be used for cost-benefit analysis

Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
- **Loss function**: measures the error between y_i and the predicted value y_i'
  - Absolute error: $| y_i - y_i'|$
  - Squared error: $(y_i - y_i')^2$
- Test error (generalization error): the average loss over the test set
  - Mean absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{d}$ Mean squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{d}$
  - Relative absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{\sum_{i=1}^{d} | y_i - \bar{y} |}$ Relative squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2}$

The mean squared-error exaggerates the presence of outliers

Popularly use (square) root mean-square error, similarly, root relative squared error

End of Unit-3