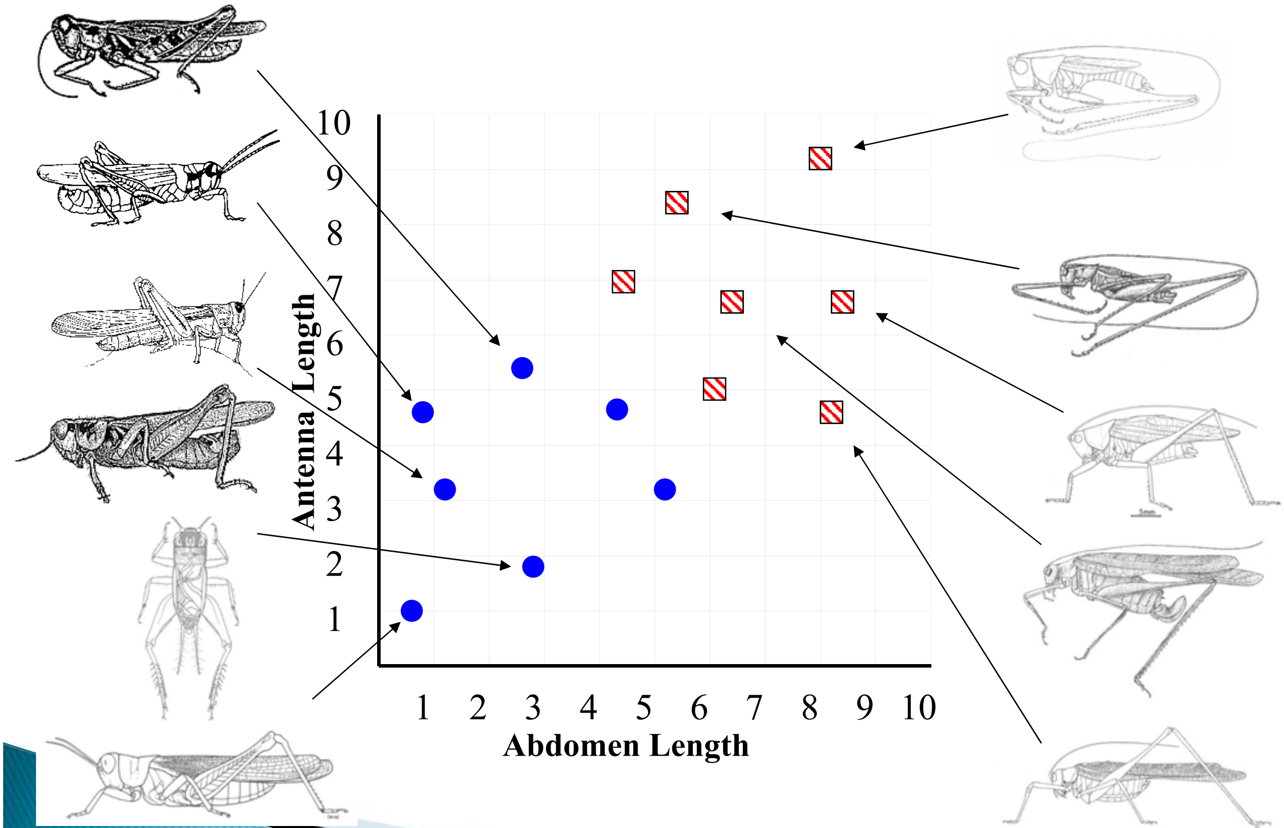
Data Mining
-- Classification

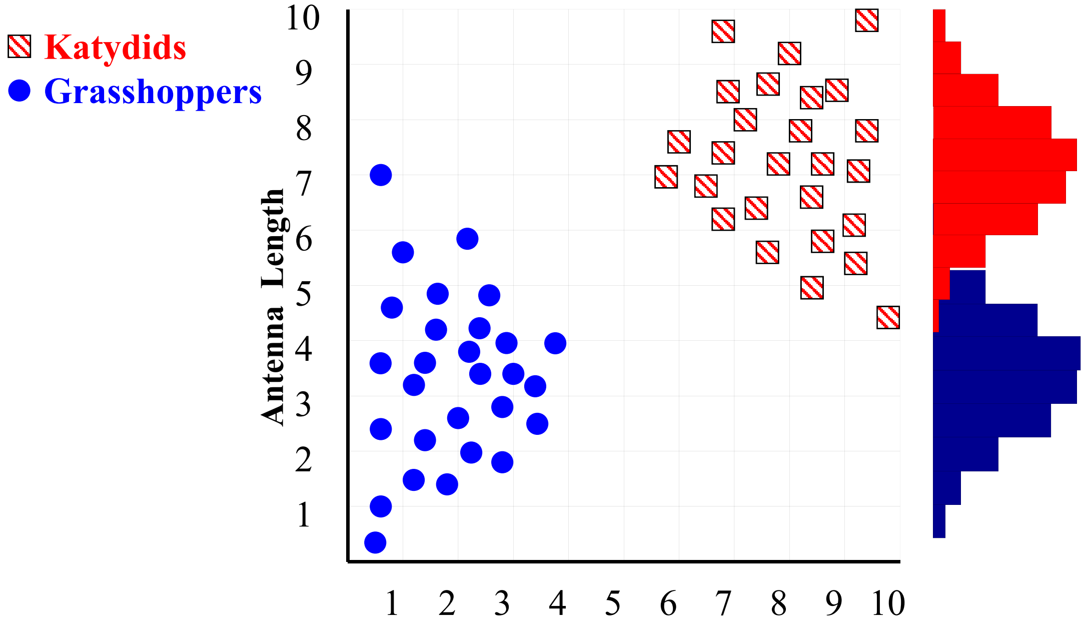
Instructor: Jen-Wei Huang

Office: 92528 in the EE building
jwhuang@mail.ncku

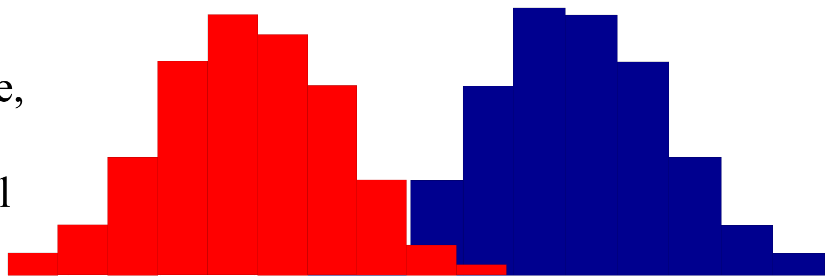
Grasshoppers **Katydids**



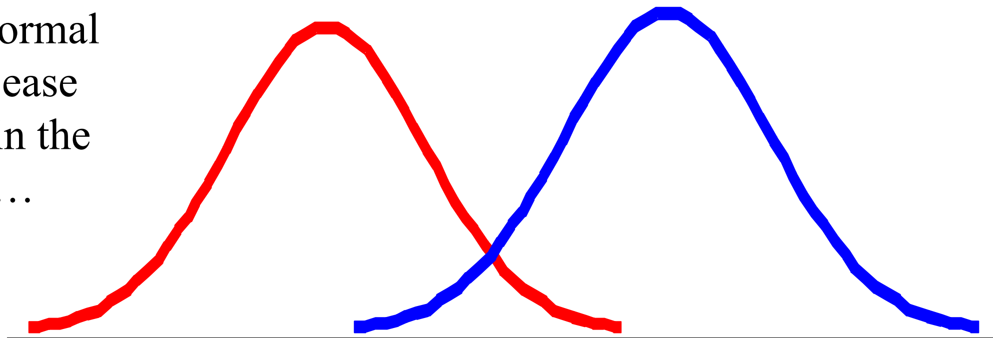
With a lot of data, we can build a histogram. Let us just build one for "Antenna Length" for now…



We can leave the histograms as they are, or we can summarize them with two normal distributions.



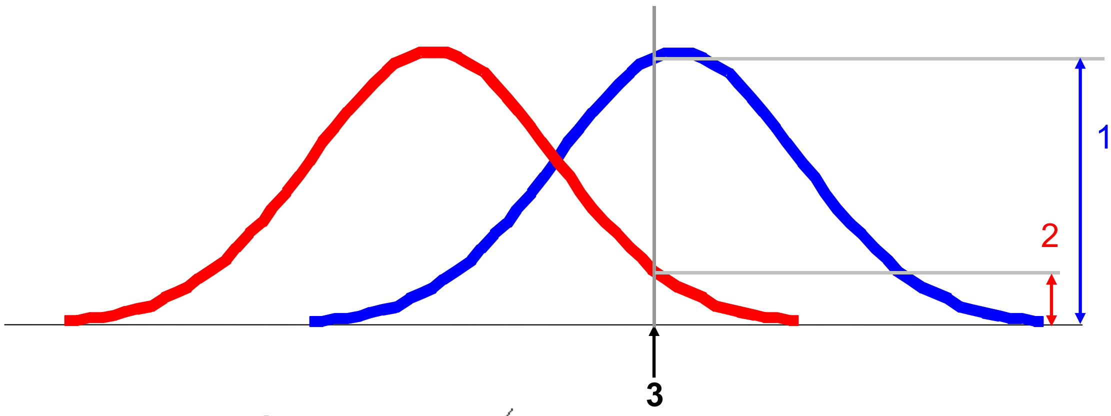
Let us use two normal distributions for ease of visualization in the following slides…

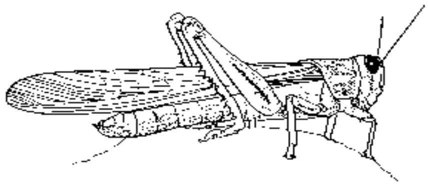


$p(c_j \mid d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **3**) = 10 / (10 + 2) = 0.833

P(**Katydid** | **3**) = 2 / (10 + 2) = 0.166



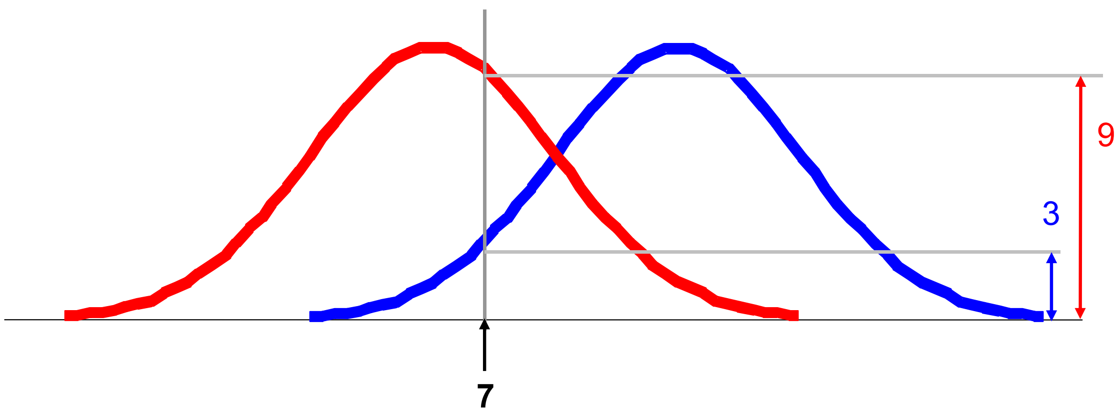


Antennae length is **3**

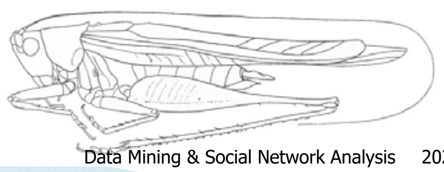
$p(c_j \mid d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **7**) = 3 / (3 + 9) = 0.250

P(**Katydid** | **7**) = 9 / (3 + 9) = 0.750



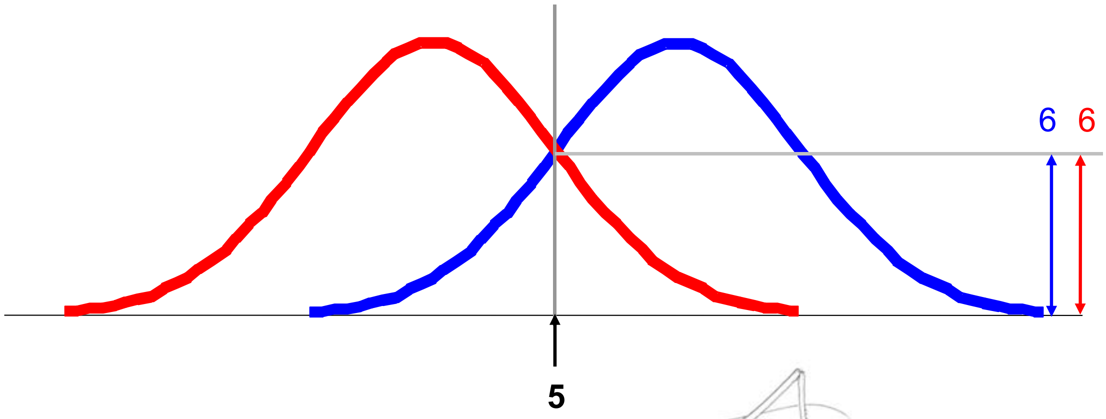
Antennae length is **7**

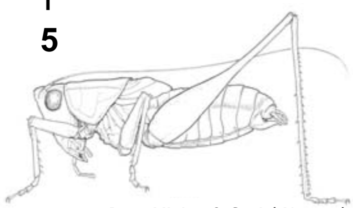


$p(c_j \mid d)$ = probability of class c_j, given that we have observed d

P(**Grasshopper** | **5**) = 6 / (6 + 6) = 0.500

P(**Katydid** | **5**) = 6 / (6 + 6) = 0.500





Antennae length is **5**

Bayesian Classification

- A statistical classifier: performs *probabilistic prediction* based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Basics Bayesian Theorem

- Let **X** be a data sample ("*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 observed data sample **X**

Basics Bayesian Theorem

- P(H) (*prior probability*), the initial probability
  - E.g., **X** will buy computer, regardless of age, income, …
- P(**X**): probability that sample data is observed
- P(**X**|H) (*posteriori probability*), the probability of observing the sample **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Basics 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

- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Naïve Bayesian Classifier

- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- Followed by Bayesian Theorem,

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

since P(X) is the same for all classes, we need only to maximize $P(\mathbf{X}|C_i)P(C_i)$

Derivation of Naïve Bayes Classifier

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

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

- 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 $|C_{i,D}|$ (# 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

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

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

Example

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

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

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

- $P(C_i)$: 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")

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
    - Dependencies among these cannot be modeled by Naïve Bayesian Classifier
  - How to deal with these dependencies?
    - Bayesian Belief Networks

K-Nearest Neighbor Classifier

- All instances correspond to points in the n-D space.
- The nearest neighbors are defined in terms of Euclidean distance.
- The class label is voted by k nearest neighbors to the testing instance.
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes.
  - To overcome it, elimination of the least relevant attributes

Association-Based Classifier

- Several methods for association-based classification
  - ARCS: Quantitative association mining and clustering of association rules (Lent et al'97)
    - It beats C4.5 in (mainly) scalability and also accuracy
  - Associative classification: (Liu et al'98)
    - It mines high support and high confidence rules in the form of "cond_set => y", where y is a class label
  - CAEP (Classification by aggregating emerging patterns) (Dong et al'99)
    - Emerging patterns (EPs): the itemsets whose support increases significantly from one class to another
    - Mine Eps based on minimum support and growth rate

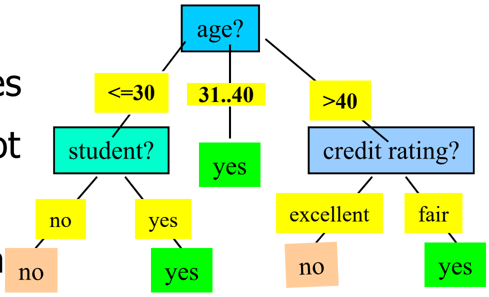
Rule-Based Classifier

- Represent the knowledge in the form of IF-THEN rules
  - R: IF *age* = youth AND *student* = yes
    THEN *buys_computer* = yes
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R
  - D: training data set
  - coverage(R) = n_{covers} /|D|
  - accuracy(R) = $n_{correct}$ / n_{covers}

Conflict Rules

- If more than one rule is triggered, need **conflict resolution**
  - Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute test*)
  - Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
  - Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

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, e.g., FOIL, AQ, CN2, RIPPER
- 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
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithm

- Star 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 $$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg})$$

    It favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

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

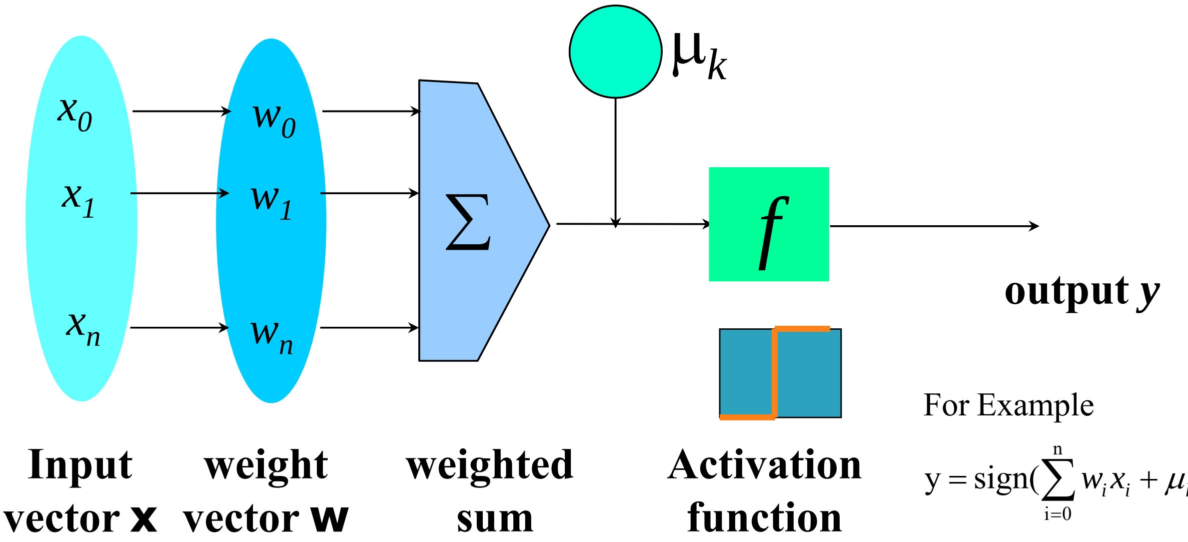
Pos/neg are # of positive/negative tuples covered by R.

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

Neural Network

- Analogy to Biological Systems (Indeed a great example of a good learning system)
- Massive Parallelism allowing for computational efficiency
- The first learning algorithm came in 1959 (Rosenblatt) who suggested that if a target output value is provided for a single neuron with fixed inputs, one can incrementally change weights to learn to produce these outputs using the perceptron learning rule

A Neuron (= a perceptron)

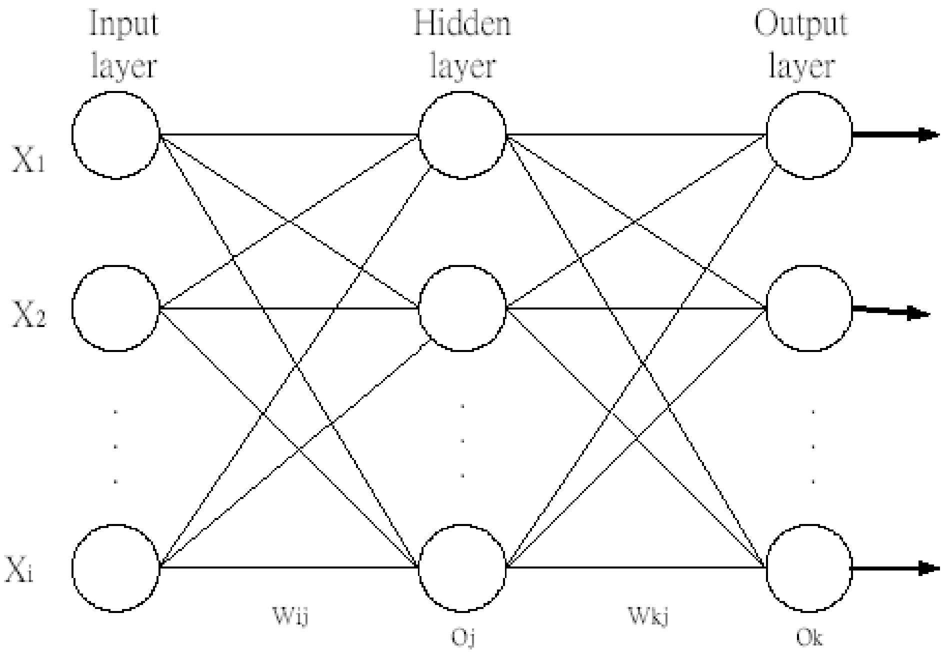


For Example

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

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

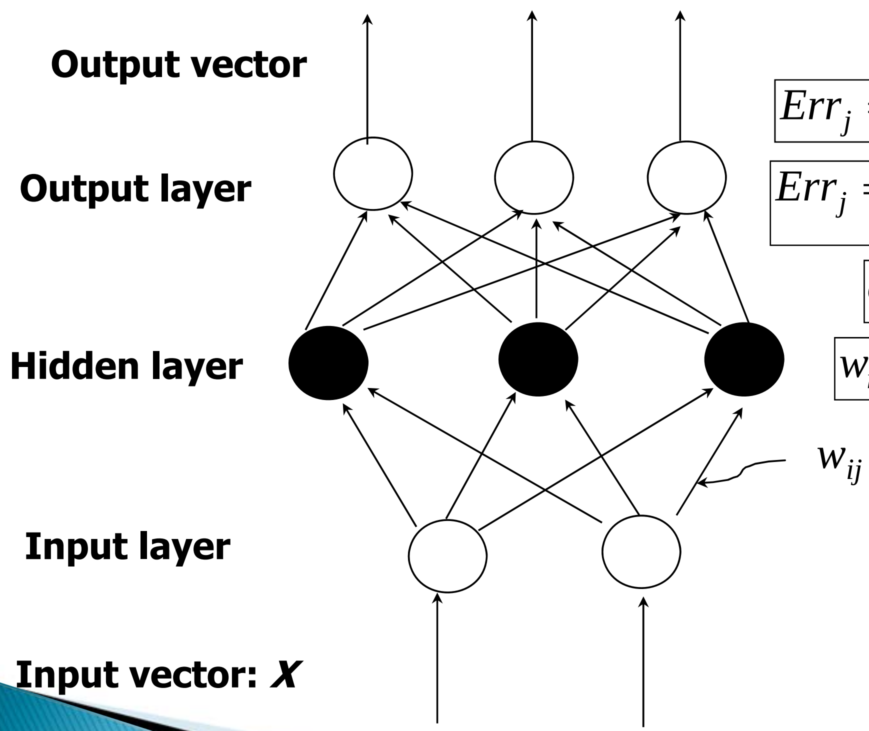
Neural Networks



Learning Phase

- 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 Neural Network



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

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

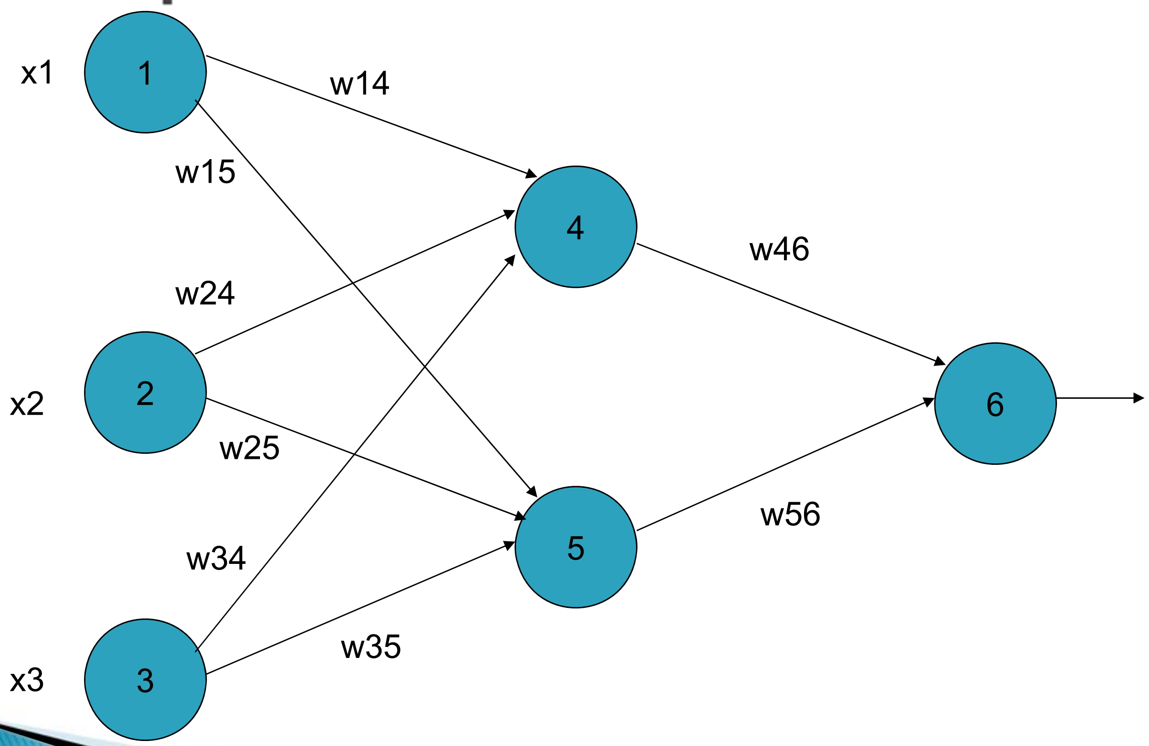
$$\theta_j = \theta_j + (l)Err_j$$

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

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

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

Example of Neural Networks



Example

Initial input, weight, and bias values														Class
x1	x2	x3	w14	w15	w24	w25	w34	w35	w46	w56	$\Theta 4$	$\Theta 5$	$\Theta 6$	C
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	1

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	$1/(1 + e^{0.105}) = 0.474$

Example

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

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

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

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

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

Calculation for weight and bias updating	
Weight or bias	New value
w46	-0.3+(0.9)(0.1311)(0.332)=-0.261
Θ4	-0.4+(0.9)(-0.0087)=-0.408

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

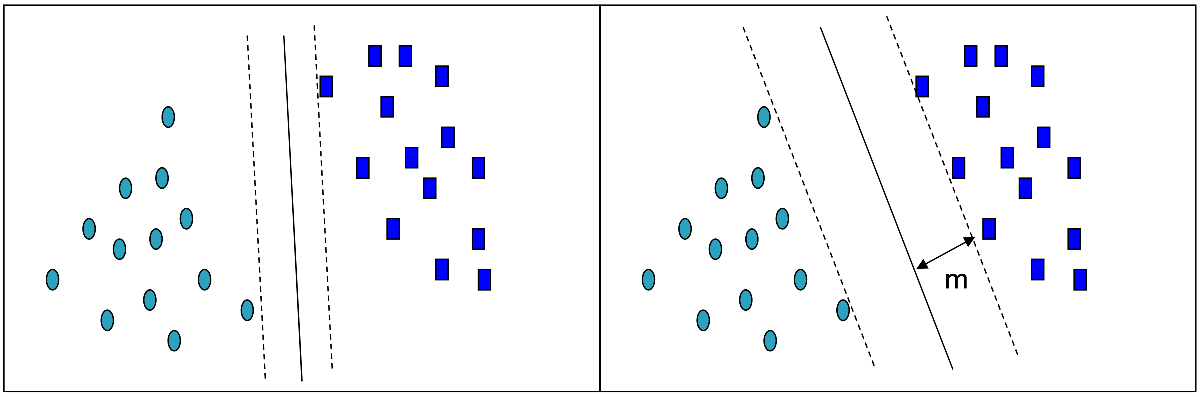
Strength

- 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

Support Vector Machines

- A new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., “decision boundary”)

SVM



Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), where $\mathbf{X}_i$ is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

SVM

- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)

Finding Hyperplane

- A separating hyperplane can be written as

  $$W \bullet X + b = 0$$

  where **W**$=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

Comments

- Deterministic algorithm (v.s. Neural Network)
- Nice Generalization properties
- Hard to learn – learned in batch mode using quadratic programming techniques
- Using kernels can learn very complex functions

SVM Resources

- http://www.kernel-machines.org/
- LIBSVM: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
- SVM-light: simpler but performance is not better than LIBSVM, support only binary classification and only C language
- SVM-torch: another recent implementation also written in C.

Other Classifiers

- Genetic Algorithms (GA)
- Rough Set Approaches
- Fuzzy Set Approaches
- Regression Models

References

- C. M. Bishop, **Neural Networks for Pattern Recognition.** Oxford University Press, 1995.
- C. J. C. Burges. **A Tutorial on Support Vector Machines for Pattern Recognition.** *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998.
- W. Cohen. **Fast effective rule induction**. ICML'95.
- G. Cong, K.-L. Tan, A. K. H. Tung, and X. Xu. **Mining top-k covering rule groups for gene expression data.** SIGMOD'05.
- Y. Freund and R. E. Schapire. **A decision-theoretic generalization of on-line learning and an application to boosting.** J. Computer and System Sciences, 1997.

References

- D. Heckerman, D. Geiger, and D. M. Chickering. **Learning Bayesian networks: The combination of knowledge and statistical data**. Machine Learning, 1995.
- B. Liu, W. Hsu, and Y. Ma. **Integrating Classification and Association Rule**. KDD'98.
- W. Li, J. Han, and J. Pei, **CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules**, ICDM'01.
- T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. **A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms.** Machine Learning, 2000.

References

- I. H. Witten and E. Frank. **Data Mining: Practical Machine Learning Tools and Techniques**, 2ed. Morgan Kaufmann, 2005.
- X. Yin and J. Han. **CPAR: Classification based on predictive association rules**. SDM'03
- H. Yu, J. Yang, and J. Han. **Classifying large data sets using SVM with hierarchical clusters**. KDD'03.

References

- Slides from Prof. J.-W. Han, UIUC
- Slides from Prof. M.-S. Chen, NTU
- Slides from Prof. W.-Z. Peng, NCTU