Lecture Slides - 4

Classification & Prediction

Classification and Prediction

- What is classification? What is prediction?
- Issues regarding classification and prediction
- Classification techniques
- Prediction
- Accuracy and error measures
- Ensemble methods
- Model selection
- Summary

Classification vs. Prediction

- Classification
  - Predicts categorical class labels (discrete or nominal)
  - Classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction
  - Models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit approval
  - Target marketing
  - Medical diagnosis
  - Fraud detection

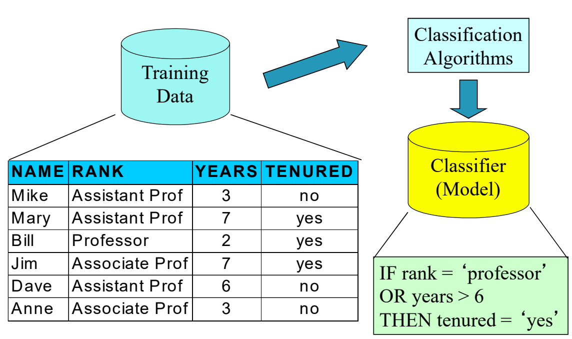
Classification: A Mathematical Mapping

- Classification:
  - predicts categorical class labels
- Mathematically
  - $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$
  - We want a function f: X → Y
- E.g., Personal homepage classification
  - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
    - x_1 : # of a word "homepage"
    - x_2 : # of a word "welcome"

Classification — A Two-Step Process

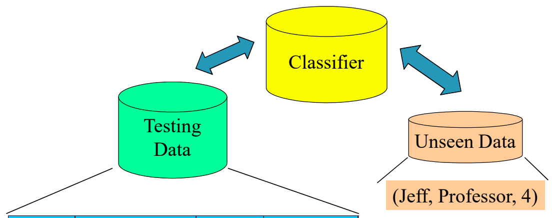
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

Step-1: Model Construction

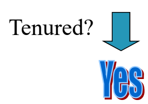


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Step-2: Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes



7

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. the aim is to establish the existence of classes or clusters in the data

8

Issues: Data Preparation

- Data cleaning
  - Preprocess data in order to reduce noise and handle missing values
- Relevance analysis (feature selection)
  - Remove the irrelevant or redundant attributes
- Data transformation
  - Generalize and/or normalize data

9

Issues: Evaluating Classification Methods

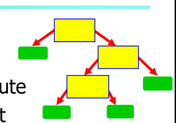
- Accuracy
  - Classifier accuracy: predicting class label
  - Predictor accuracy: guessing value of predicted attributes
- Speed
  - Time to construct the model (training time)
  - Time to use the model (classification/prediction time)
- Robustness:
  - Handling noise and missing values
- Scalability:
  - Efficiency in disk-resident databases
- Interpretability
  - Understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

10

Classification Techniques

- Well-known methods:
  - Decision Trees
  - Bayesian Classification
  - Bayesian Belief Networks
- Besides, we can also use
  - Instance-Based Methods
    - *k*-nearest neighbor (*k-NN*) approach
  - Neural Networks
  - Support Vector Machine

Classification by DT Induction



- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases:
  - Tree construction
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying unknown samples

Decision Tree Induction: Training Dataset

age	income	student	credit_rating	buys_computer
<=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

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Decision Tree Algorithms (ID3, C4.5, C5.0/See5)

- ID3 (Iterative Dichotomiser 3): A decision tree algorithm invented by Ross Quinlan (RQ)
- C4.5: An improvement of ID3 by RQ
  - Handles both continuous & discrete attributes
  - Handles training data with missing attributes
  - Prunes tree after creation
- C5.0/See5: Another improvement by RQ
- C5.0 - Unix/Linux; See5 - Windows
  - Speed
  - Memory usage
  - Smaller decision trees

Attribute Selection Measure: Information Gain

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times I(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection Measure: Information Gain

- Entropy H(D):
  - Measure of the amount of uncertainty in the dataset D

$$H(D) = -\sum_{c \in C} p(c) \log_2 p(c)$$

  - Where, D: The dataset for which entropy is calculated
  - C: The set of classes in D
  - p(c): The proportion of the number of elements in class c to the number of elements in D
- When H(D) =0, the dataset D is perfectly classified
  - i.e. all elements in D are of the same class

Attribute Selection Measure: Information Gain

- Information Gain IG(A):
  - Measure of the difference in entropy from before to after the dataset D is split on an attribute A.
  - i.e., how much uncertainty in D was reduced after splitting D on attribute A.

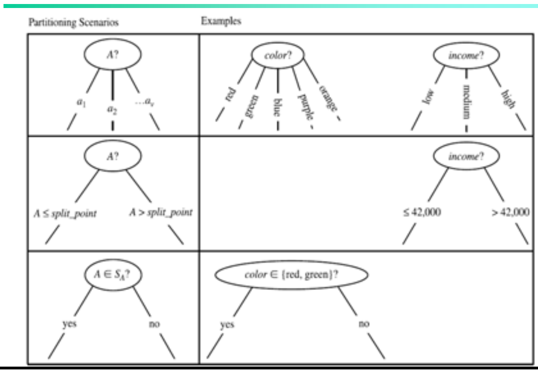
$$IG(A, D) = H(D) - \sum_{s \in S} p(s) H(s)$$

  - Where, H(D): Entropy of the dataset D
  - S: The subsets created from splitting D by attribute A
  - p(s): The proportion of the number of elements in s to the number of elements in D
  - H(s): Entropy of the subset s
- The attribute with the largest information gain is used to split the dataset D.

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Splitting Approaches in DT



Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

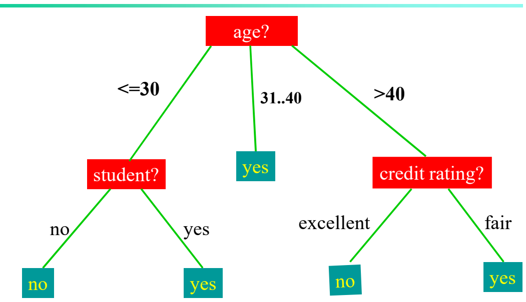
$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

age	income	student	credit_rating	buys_computer
<=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

Similarly,

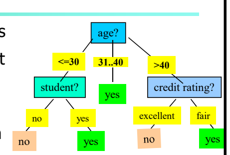
$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Output: A Decision Tree for "buys_computer"



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* <=30 AND *student* = *no* THEN *buys_computer* = *no*
IF *age* <= 30 AND *student* = *yes* THEN *buys_computer* = *yes*
IF *age* = 31…40 THEN *buys_computer* = *yes*
IF *age* > 40 AND *credit_rating* = *excellent* THEN *buys_computer* = *no*
IF *age* > 40 AND *credit_rating* = *fair* THEN *buys_computer* = *yes*

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v}\frac{|D_j|}{|D|}\times\log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex. $SplitInfo_A(D) = -\frac{4}{14}\times\log_2\left(\frac{4}{14}\right) - \frac{6}{14}\times\log_2\left(\frac{6}{14}\right) - \frac{4}{14}\times\log_2\left(\frac{4}{14}\right) = 0.926$
  - gain_ratio(income) = 0.029/0.926 = 0.031
- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini index (CART, IBM IntelligentMiner)

- A measure of impurity to decide how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset.
- If D a dataset containing examples from m classes, then

$$gini(D) = \sum_{i=1}^{m} p_i(1-p_i) = \sum_{i=1}^{m} p_i - \sum_{i=1}^{m} P_i^2 = 1 - \sum_{i=1}^{m} P_i^2$$

Where p_i is the relative frequency of class i in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index *gini*(D) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity: $\Delta gini(A) = gini(D) - gini_A(D)$
- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

25

Example: Gini index

- Ex. D has 9 tuples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2

$$gini_{income \in \{low, medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_1)$$
$$= \frac{10}{14}(1 - (\frac{6}{10})^2 - (\frac{4}{10})^2) + \frac{4}{14}(1 - (\frac{1}{4})^2 - (\frac{3}{4})^2)$$
$$= 0.450$$
$$= Gini_{income \in \{high\}}(D)$$

but $gini_{\{medium,high\}}$ is 0.30 and thus the best since it is the lowest

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

26

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - Information gain:
    - biased towards multivalued attributes
  - Gain ratio:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - Gini index:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

27

Overfitting and Tree Pruning

- Overfitting: Occurs when a model describes random error or noise instead of the underlying relationship.
- An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: Remove branches from a "fully grown" tree - get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

28

Pros & Cons of Using DT

- Pros:
  - Simple to understand and interpret
  - Requires little data preparation
  - Able to handle both numerical and categorical data
  - Robust
  - Perform well with large data in short time
- Cons:
  - Learning an optimal decision tree is NP-complete
  - Decision-tree learners create over-complex trees that do not generalize the data well.

29

Bayesian Classification

- A statistical classifier: predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: 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

Bayesian Theorem: Basics

- 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**
- P(H) - prior 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 H holds
  - e.g., Given that **X** will buy computer, the prob. that **X** is 31..40, medium income

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 × prior / evidence

- Predicts X belongs to C_i iff the probability $P(C_i\,|\,X)$ 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

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

Derivation of Naïve Bayes Classifier

- Assumption: attributes are conditionally independent

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

- 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 $|C_{i,D}|$ (# 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 $P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$

Naïve Bayesian Classifier: Training Dataset

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	s_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

Naïve Bayesian Classifier: An Example

- 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|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(buys_computer="yes"|X) = P(X|buys_computer = "yes") × P(buys_computer = "yes") = 0.028
P(X|buys_computer= "no"|X) = 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
- Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use Laplacian correction (or Laplacian estimator)
  - Adding 1 to each case

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

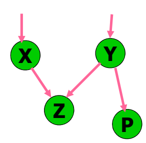
    Prob(income = high) = 11/1003

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., 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

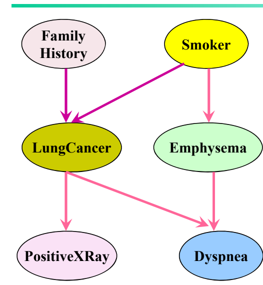
Bayesian Belief Networks

- 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:

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

Lazy vs. Eager Learning

- Lazy learning: Simply stores training data (or only minor processing) and waits until it is given a test tuple.
- Eager learning: Given a set of training set, constructs a classification model before receiving new (e.g., test) data to classify
- Pros & Cons:
  - Lazy requires less time in training but more time in predicting
  - Lazy methods are more accurate as they use many local linear functions to form its implicit global approximation to the target function. Whereas, eager methods must commit to a single hypothesis that covers the entire instance space.

41

Evaluation: What is Good Classification?

- Correct classification: The known label of test sample is identical with the class result from the classification model
- Accuracy ratio: the percentage of test set samples that are correctly classified by the model
- A distance measure between classes can be used
  - e.g., classifying "football" document as a "basketball" document is not as bad as classifying it as "crime".

Accuracy?

Accuracy is the percentage of test set samples that are correctly classified by the model

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*

Confusion Matrix

		System Classification	
		Class-1 (+ve)	Class-2 (-ve)
Actual Classification	Class-1 (+ve)	True Positive (TP)	False Negative (FN)
Actual Classification	Class-2 (-ve)	False Positive (FP)	True Negative (TN)

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Error rate (misclassification rate) = 1 – accuracy

Other Measures

- Alternative accuracy measures (e.g., for cancer diagnosis)

$$sensitivity = \frac{TP}{P} = \frac{TP}{TP + FN}$$ /* true positive recognition rate */

$$specificity = \frac{TN}{N} = \frac{TN}{TN + FP}$$ /* true negative recognition rate */

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$accuracy = sensitivity \times \frac{P}{P+N} + specificity \times \frac{N}{P+N}$$

/* P = total positives = TP + FN; N = total negatives = TN + FP */

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

45

Evaluating the Accuracy of a Classifier or Predictor

- Holdout method
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
- Random sampling (a variation of holdout)
    - Repeat holdout *k* times, accuracy = avg. of the accuracies obtained
- Cross-validation (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: k folds where k = # of tuples, for small sized data
  - Stratified cross-validation: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

46

Cont…

- Bootstrap
  - Samples the given training tuples uniformly with replacement
  - i.e., Each time a tuple is selected, it is equally likely to be selected again and readded to the training set.
  - 0.632 bootstrap: A commonly used bootstrap method
    - Given a dataset of d tuples, it is sampled d times, with replacement, resulting in a bootstrap sample or training set of d samples.

$$prob(selection) = \frac{1}{d}$$

$$prob(no\ selection) = 1 - \frac{1}{d}$$

If d is very large

$$In\ d\ repetitions,\ prob(no\ selection) = \left(1 - \frac{1}{d}\right)^d \approx e^{-1} = 0.368$$

36.8% of tuples will not selected for training set – used for test

Remaining 63.2% tuples will form the training set

47

Cont…

- The sampling procedure in bootstrap can be repeated k times
- The overall accuracy of model M is then calculated as

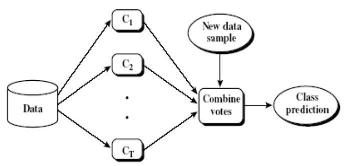
$$Acc(M) = \sum_{i=1}^{k} (0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

- Works well for small data sets

48

Ensemble Methods: Increasing the Accuracy

- Ensemble methods
  - Methods that use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*
  - Can works for both classification and prediction



Popular Ensemble Methods

- Bagging:
  - Using majority vote for classification
  - Averaging the predicted values for prediction
- Boosting:
  - Assigning weights to classifiers and using weighted vote for classification and prediction

Bagging

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training: learning models
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classifying an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses – weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - Weights are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to pay more attention to the training tuples that were misclassified by M_i
  - The final M* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting tends to achieve greater accuracy than bagging, but it also risks Overfitting the model to misclassified data

Adaboost (Freund and Schapire, 1997)

- A popular boosting algorithm
- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is correctly classified, its weight is decreased
  - If a tuple is incorrectly classified, its weight is increased

Adaboost (Cont…)

- A tuple's weight reflects how hard it is to classify
  - The higher the weight, the more often it has been misclassified.

Computing Error Rate of a Model

- Error rate of classifier M_i is the sum of the weights of the misclassified tuples in D_i, that is

$$error(M_i) = \sum_{j=1}^{d} w_j \times error(\mathbf{X}_j)$$

Where, error($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$ (if misclassification then 1, otherwise 0).

- If the performance of classifier M_i is so poor that its error exceeds 0.5, then we abandon it, and repeat the sampling to re-learn model M_i

Tuples Weight Updation

- If a tuple in round i was correctly classified, its weight is multiplied by $\frac{error(M_i)}{1 - error(M_i)}$
- The weights for all tuples (including the misclassified ones) are normalized so that their sum remains the same as it was before.
- To normalize a weight, multiply it by the sum of the old weights, and divided by the sum of the new weights.
- As a result, the weights of misclassified tuples are increased and the weights of correctly classified tuples are decreased

X	X/(1-X)
0.10	0.11
0.20	0.25
0.30	0.43
0.40	0.67
0.50	1.00
0.60	1.50
0.70	2.33
0.80	4.00
0.90	9.00

	Initial weight	prediction	Error Calculation	Weight updation	Weight Normalization
	0.10	T	0.00	0.07	0.03
	0.10	F	0.10	0.10	0.05
	0.10	F	0.10	0.10	0.05
	0.10	T	0.00	0.07	0.03
	0.10	T	0.00	0.07	0.03
	0.10	F	0.10	0.10	0.05
	0.10	T	0.00	0.07	0.03
	0.10	F	0.10	0.10	0.05
	0.10	T	0.00	0.07	0.03
	0.10	T	0.00	0.07	0.03
		Error(Mi)	0.40	0.80	0.40
Mutiplication factor			0.67		

Using Classifier Weights for Prediction

- The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be. The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$
- For each class, c, we sum the weights of each classifier that assigned class c to X. The class with the highest sum is the "winner" and is returned as the class prediction for tuple X.

How Adaboost learns models?

- Steps:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Equation 6.66)
(6) **if** $error(M_i) > 0.5$ **then**
(7) reinitialize the weights to $1/d$
(8) go back to step 3 and try again;
(9) **endif**
(10) **for** each tuple in D_i that was correctly classified **do**
(11) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(12) normalize the weight of each tuple;
(13) **endfor**

How Adaboost classifies a tuple X?

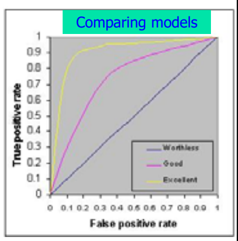
- Steps:
  - Initialize weight of each class to 0
  - For i=1 to k do // for each classifier
    - w_i = log ((1-error(M_i))/error(M_i)) // weight of the classifier's vote
    - c = M_i(X) // get class prediction for X from M_i
    - Add w_i to weight for class c
  - Endfor
  - Return the class with the largest weight

Model Selection: ROC Curves

- Receiver Operating Characteristics (ROC) curves are used for visual comparison of classification models
- Originated from signal detection theory developed during World War II for analysis of Radar images
- Presents the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model
- A model with perfect accuracy will have an area of 1.0

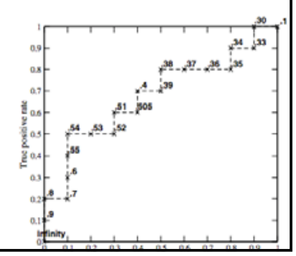
How to Draw ROC Curve?

- Rank the test tuples in decreasing order: the one that is most likely to belong to the positive class appears at the top of the list
- Vertical axis represents the true positive rate
- Horizontal axis represents the false positive rate
- The plot also shows a diagonal line



ROC Curve Plotting as a Step Function

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



References

- C. Apte and S. Weiss. **Data mining with decision trees and decision rules**. Future Generation Computer Systems, 13, 1997.
- C. M. Bishop, **Neural Networks for Pattern Recognition**. Oxford University Press, 1995.
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. **Classification and Regression Trees**. Wadsworth International Group, 1984.
- C. J. C. Burges. **A Tutorial on Support Vector Machines for Pattern Recognition**. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998.
- P. K. Chan and S. J. Stolfo. **Learning arbiter and combiner trees from partitioned data for scaling machine learning**. KDD'95.
- 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.
- A. J. Dobson. **An Introduction to Generalized Linear Models**. Chapman and Hall, 1990.
- G. Dong and J. Li. **Efficient mining of emerging patterns: Discovering trends and differences**. KDD'99.

References

- R. O. Duda, P. E. Hart, and D. G. Stork. **Pattern Classification**, 2ed. John Wiley and Sons, 2001
- U. M. Fayyad. **Branching on attribute values in decision tree generation**. AAAI'94.
- 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.
- J. Gehrke, R. Ramakrishnan, and V. Ganti. **Rainforest: A framework for fast decision tree construction of large datasets**. VLDB'98.
- J. Gehrke, V. Gant, R. Ramakrishnan, and W.-Y. Loh, **BOAT -- Optimistic Decision Tree Construction**. SIGMOD'99.
- T. Hastie, R. Tibshirani, and J. Friedman. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction.** Springer-Verlag, 2001.
- D. Heckerman, D. Geiger, and D. M. Chickering. **Learning Bayesian networks: The combination of knowledge and statistical data**. Machine Learning, 1995.
- M. Kamber, L. Winstone, W. Gong, S. Cheng, and J. Han. **Generalization and decision tree induction: Efficient classification in data mining**. RIDE'97.
- 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.

References

- 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.
- J. Magidson. **The Chaid approach to segmentation modeling: Chi-squared automatic interaction detection**. In R. P. Bagozzi, editor, Advanced Methods of Marketing Research, Blackwell Business, 1994.
- M. Mehta, R. Agrawal, and J. Rissanen. **SLIQ : A fast scalable classifier for data mining**. EDBT'96.
- T. M. Mitchell. **Machine Learning**. McGraw Hill, 1997.
- S. K. Murthy, **Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey**, Data Mining and Knowledge Discovery 2(4): 345-389, 1998
- J. R. Quinlan. **Induction of decision trees**. *Machine Learning*, 1:81-106, 1986.
- J. R. Quinlan and R. M. Cameron-Jones. **FOIL: A midterm report**. ECML' 93.
- J. R. Quinlan. **C4.5: Programs for Machine Learning**. Morgan Kaufmann, 1993.
- J. R. Quinlan. **Bagging, boosting, and c4.5**. AAAI'96.

References

- R. Rastogi and K. Shim. **Public: A decision tree classifier that integrates building and pruning**. VLDB' 98.
- J. Shafer, R. Agrawal, and M. Mehta. **SPRINT : A scalable parallel classifier for data mining**. VLDB' 96.
- J. W. Shavlik and T. G. Dietterich. **Readings in Machine Learning**. Morgan Kaufmann, 1990.
- P. Tan, M. Steinbach, and V. Kumar. **Introduction to Data Mining**. Addison Wesley, 2005.
- S. M. Weiss and C. A. Kulikowski. **Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems**. Morgan Kaufman, 1991.
- S. M. Weiss and N. Indurkhya. **Predictive Data Mining**. Morgan Kaufmann, 1997.
- 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.