Chapters 8-9. Classification

- **Classification: Basic Concepts**
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: training data are labeled indicating classes
  - New instances are classified based on training set
- Unsupervised learning (clustering)
  - class labels are unknown
  - Given a set of objects, establish the existence of classes or clusters in the data

Prediction: Classification vs. Numeric Prediction

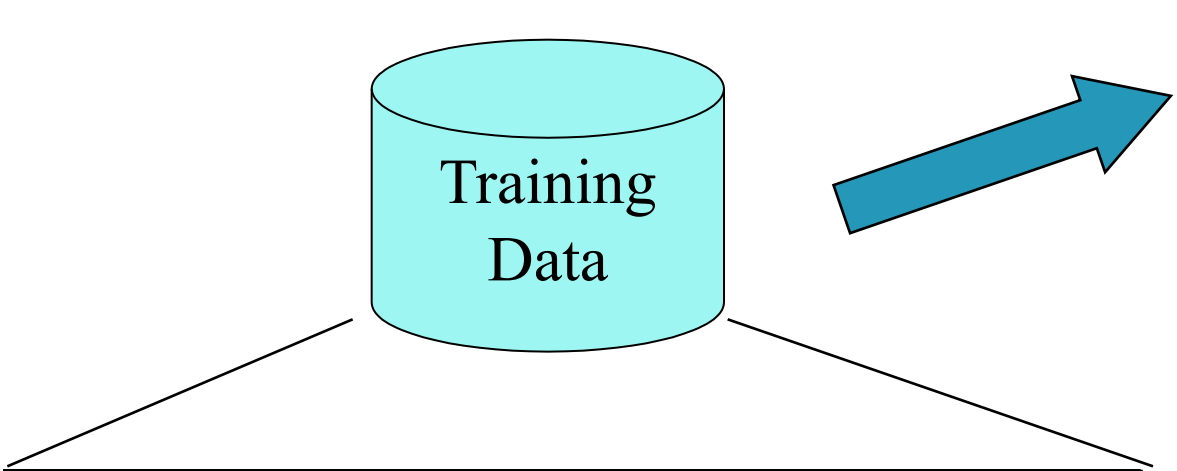
- Classification
  - predicts categorical class labels
- Numeric prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values

- Typical applications
  - Credit/loan approval
  - Medical diagnosis
  - Fraud detection
  - Web page categorization

Classification: A Two-Step Process

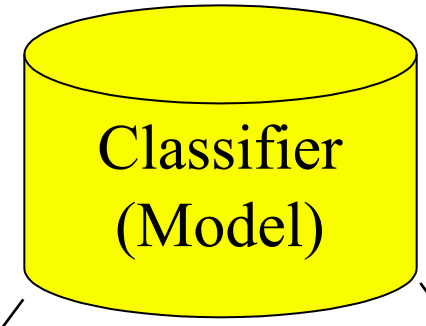
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as indicated by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown instances
  - Estimate accuracy of the model
    - Use an independent (of training set) testing set, compare predicted class labels with true class labels
    - Compute accuracy (percentage of correctly classified instances)
  - If the accuracy is acceptable, use the model to classify new data

Process 1: Model Construction



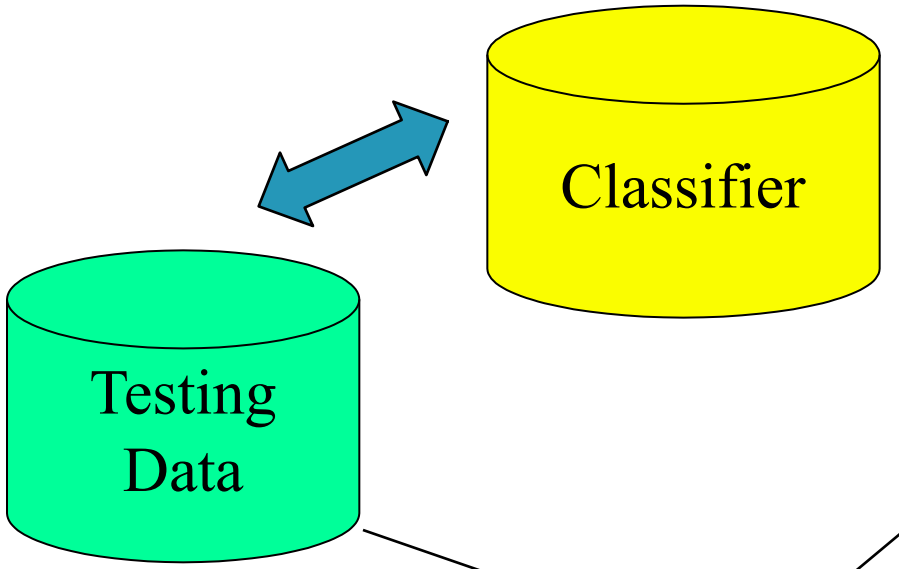
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Classification Algorithms



IF rank = 'professor'
OR years > 6
THEN tenured = 'yes'

Process 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

(Jeff, Professor, 4)

Tenured?

Yes

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

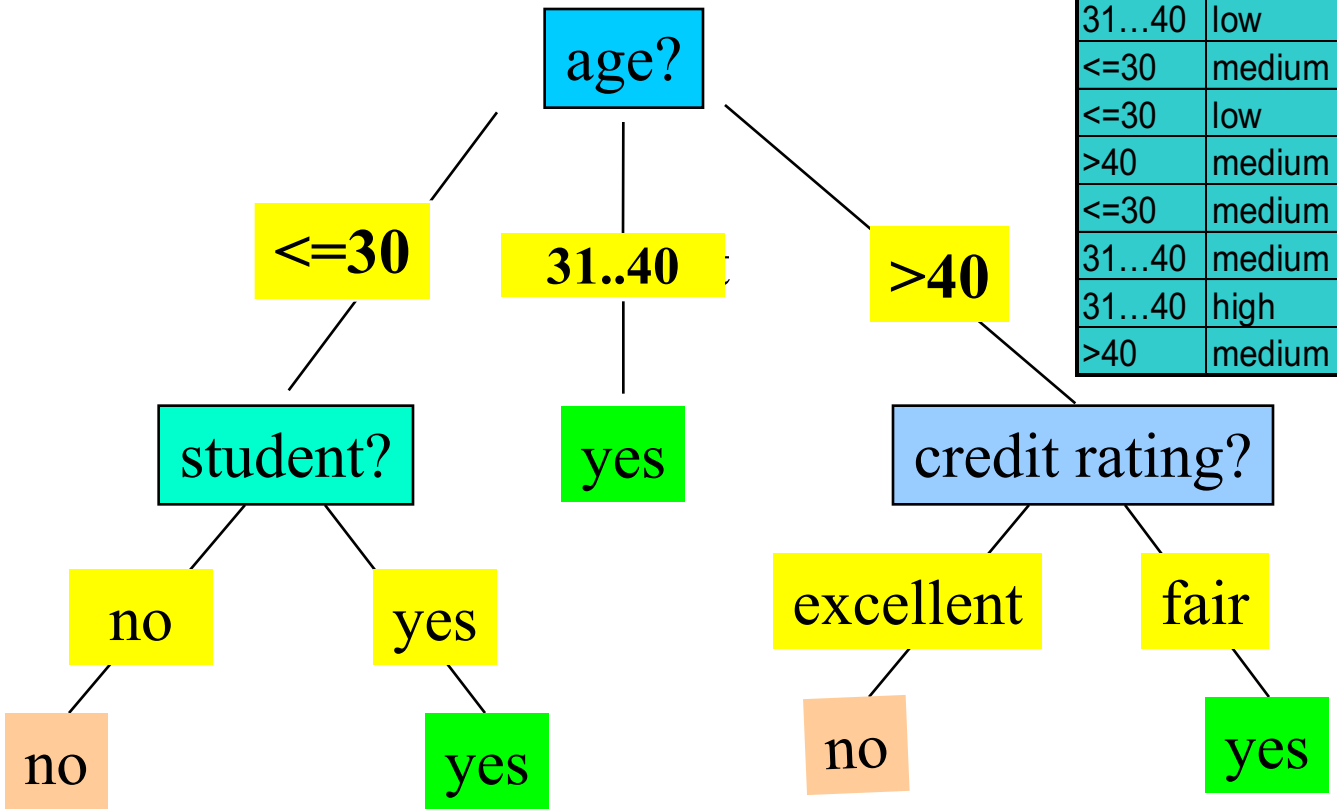
Chapters 8-9. Classification

- Classification: Basic Concepts
- **Decision Tree Induction**
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Decision Tree Induction: An Example

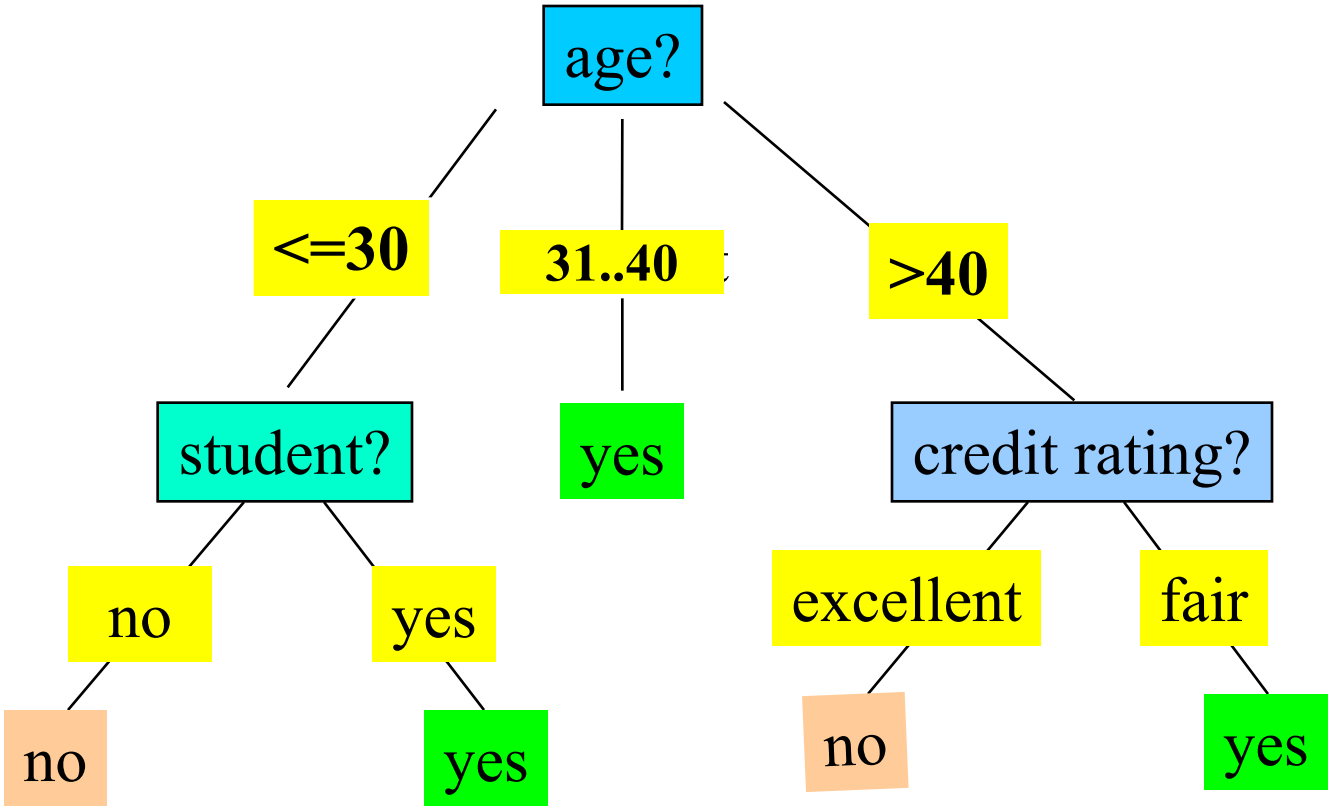
- Training data set: Buys_computer
- Resulting tree:

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



Decision Tree Properties

- Exhaustive (completely covers whole instance space)
- Mutually exclusive (no conflicting predictions)
- Interpretable (axis-parallel, as in rule learning)



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down (from general to specific) recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, discretization in advance)
  - Examples are partitioned recursively based on selected attributes
  - Attributes are selected based on heuristic or statistical measure (e.g., information gain)

- When to stop
  - All example for a given node belong to the same class (pure), or
  - No remaining attributes to select from, or
    - majority voting to determine class label for the node
  - No examples left

Random Tree Induction

Let *a* be the number of attributes. Let *v* be the maximum number of values any attribute can take

- Upper bound on the number of trees?
- Lower bound on the number of trees?

- Random tree induction
  - Randomly choose an attribute for split
  - Same stopping criteria

- The design of decision trees has been largely influenced by the preference for simplicity.

Occam's Razor

1288 - 1348

- Occam's Razor: rule of parsimony, principle of economy
  - plurality should not be assumed without necessity
  - meaning, one should not increase, beyond what is necessary, the number of entities required to explain anything
- Argument: the simplicity of nature and rarity of simple theories can be used to justify Occam's Razer.
  - First, nature exhibits regularity and natural phenomena are more often simple than complex. At least, the phenomena humans choose to study tend to have simple explanations.
  - Second, there are far fewer simple hypotheses than complex ones, so that there is only a small chance that any simple hypothesis that is wildly incorrect will be consistent with all observations.
- Occam's two razors: The sharp and the blunt (KDD'98)
  - Pedro Domingos

Attribute Selection Measure: Information Gain (ID3/C4.5)

- How to obtain smallest (shortest) tree?
- Careful design on selection of attribute
- Quinlan pioneered using entropy in his ID3 algorithm
- Entropy: in information theory, also called expected information, is a measure of uncertainly
- Intuition: chaos, molecular disorder, temperature, thermodynamic system, universe
  - High entropy = high disorder

Attribute Selection Measure: Information Gain (ID3/C4.5)

- 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:

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

  - entropy: measure of uncertainty. larger entropy -> larger uncertainty
- Information needed to classify D (aggregated entropy after using A to split D into v partitions) :

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

- Information gained (entropy dropped) by branching on attribute A

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

Attribute Selection: Information Gain

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

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

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

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

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

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

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

Similarly,

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

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

Gain Ratio for Attribute Selection (C4.5)

- Information gain 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)

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

gain_ratio(income) = 0.029/1.557 = 0.019

- The attribute with the largest gain ratio will be selected

Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class *j* 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*)

Computation of 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

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- 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

Comparing Attribute Selection Measures

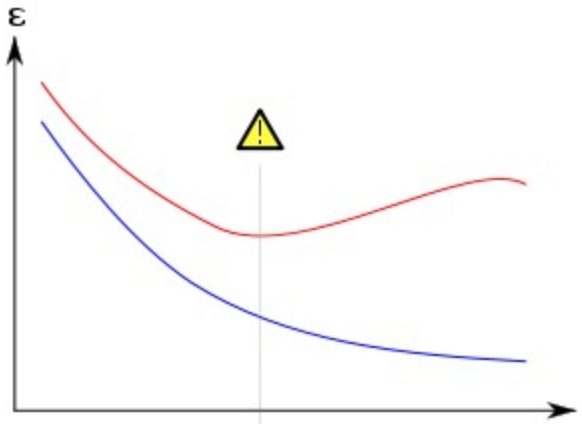
- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes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

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

Overfitting and Tree Pruning

- Overfitting: 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
  - Blue: training error, red: generalization error



- 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 (**validation set**) different from the training data to decide which is the "best pruned tree"

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

Classification in Large Databases

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why is decision tree induction popular?
  - relatively faster learning speed (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use SQL queries for accessing databases
  - comparable classification accuracy with other methods
- RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
  - Builds an AVC-list (attribute, value, class label)

Scalability Framework for RainForest

- Separates the scalability aspects from the criteria that determine the quality of the tree
- Builds an AVC-list**: AVC (Attribute, Value, Class_label)**
- **AVC-set** (of an attribute *X*)
  - Projection of training dataset onto the attribute *X* and class label where counts of individual class label are aggregated
- **AVC-group** (of a node *n*)
  - Set of AVC-sets of all predictor attributes at the node *n*

Rainforest: Training Set and Its AVC Sets

Training Examples

age	income	student	credit_rating	com
<=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

AVC-set on *Age*

Age	Buy_Computer	
	yes	no
<=30	2	3
31..40	4	0
>40	3	2

AVC-set on *income*

income	Buy_Computer	
	yes	no
high	2	2
medium	4	2
low	3	1

AVC-set on *Student*

student	Buy_Computer	
	yes	no
yes	6	1
no	3	4

AVC-set on *credit_rating*

Credit rating	Buy_Computer	
	yes	no
fair	6	2
excellent	3	3

BOAT (Bootstrapped Optimistic Algorithm for Tree Construction)

- Use a statistical technique called *bootstrapping* to create several smaller samples (subsets), each fits in memory
- Each subset is used to create a tree, resulting in several trees
- These trees are examined and used to construct a new tree *T'*
  - It turns out that *T'* is very close to the tree that would be generated using the whole data set together
- Adv: requires only two scans of DB, an incremental alg.

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- **Model Evaluation/Learning Algorithm Evaluation**
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Model Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Model Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate (recall for +)
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate (recall for -)
    - **Specificity = TN/N**

Model Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier (model) labeled as positive are actually positive

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

- **Recall:** completeness – what % of positive tuples did the classifier (model) label as positive?
- Perfect score is 1.0

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

- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Model Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Evaluating Learning Algorithm: Holdout & Cross-Validation Methods

- Holdout method
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Testing set (e.g., 1/3) for accuracy (or another metric) 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 common)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as testing 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

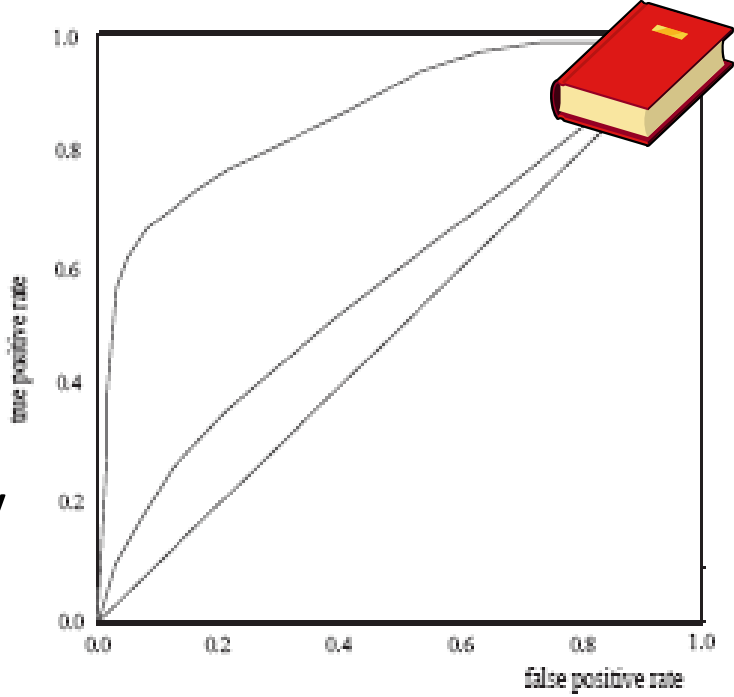
Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - 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 re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

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

Model Selection: ROC Curves

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows 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
- 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
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model



- Vertical axis represents the true positive rate
- Horizontal axis rep. the false positive rate
- The plot also shows a diagonal line
- A model with perfect accuracy will have an area of 1.0

Model Selection Issues

- **Accuracy**
  - classifier accuracy: predicting class label
- **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
  - Model (e.g., decision tree) size or compactness

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- **Rule-Based Classification**
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Using IF-THEN Rules for Classification

- Represent knowledge in the form of IF-THEN rules

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

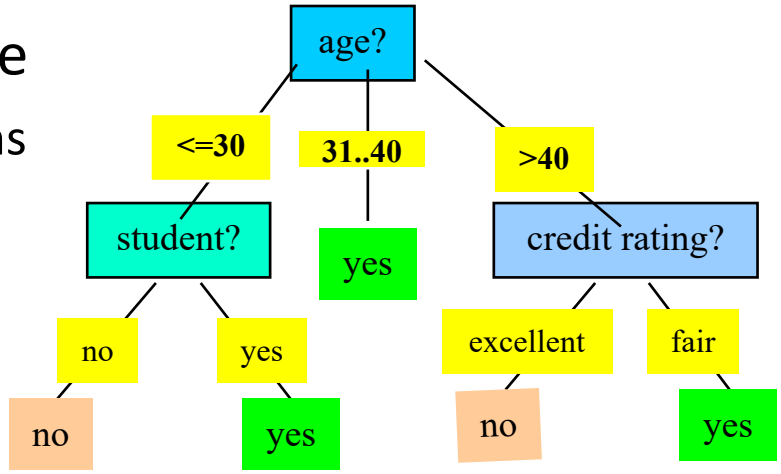
  - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

  coverage(R) = n_{covers} /|D|

  accuracy(R) = $n_{correct}$ / n_{covers}
- If more than one rule are triggered, need **conflict resolution**
  - Size ordering: assign the highest priority to the triggering rules that have the "toughest" requirement (i.e., with the *most attribute tests*)
  - 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 Decision Tree

- A root-to-leaf path corresponds to a rule
  - Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are exhaustive and mutually exclusive



- 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* = *no*

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

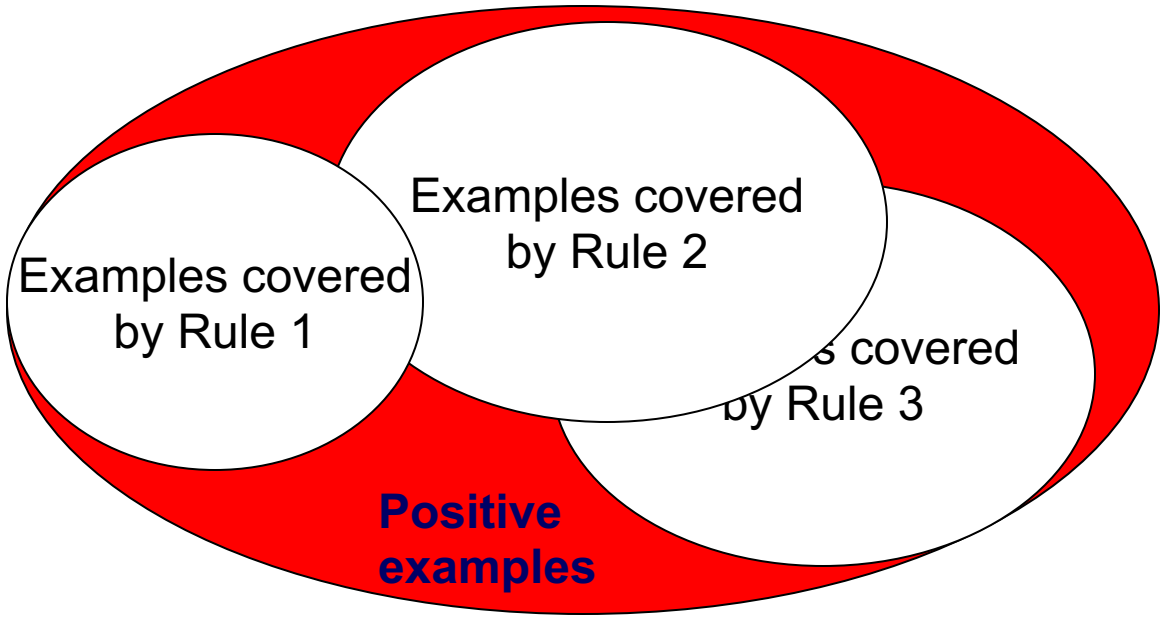
Rule Induction: Sequential Covering Method

- Sequential covering: 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 covered positive tuples are removed
  - Repeat until *termination condition* is met. e.g., no more training examples or the quality of a rule generated is below a user-specified threshold

- Unlike decision-trees that learn a set of rules *simultaneously*

Sequential Covering Algorithm

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



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

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

    - 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

Learn one rule

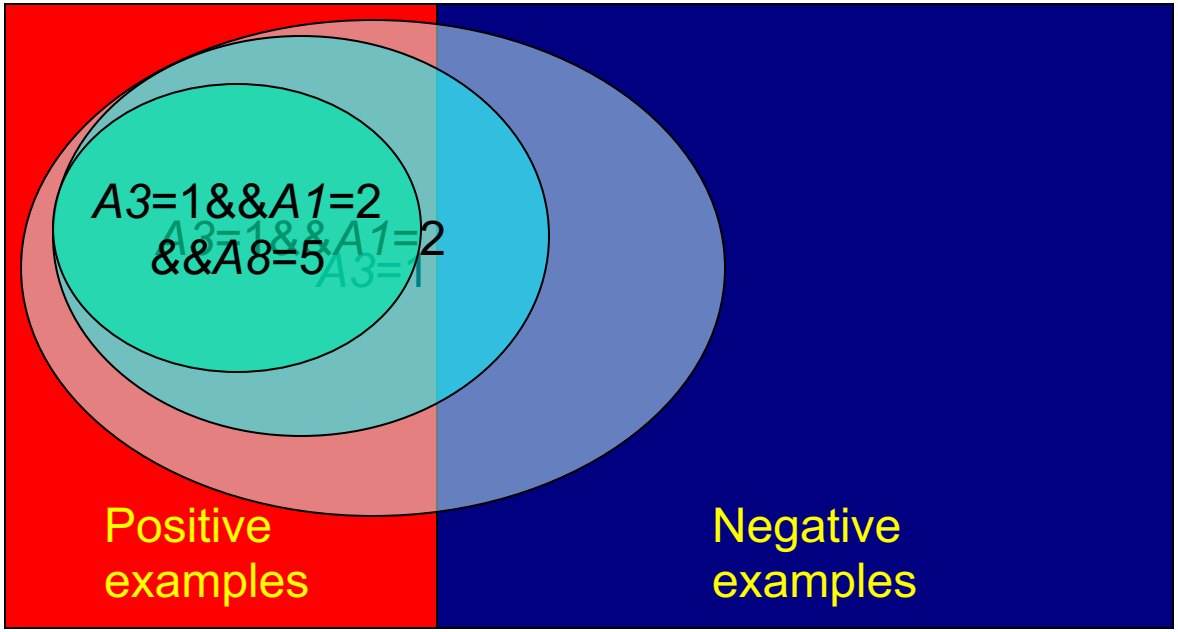
- To generate a rule

  while(true)

    find the best predicate *p*

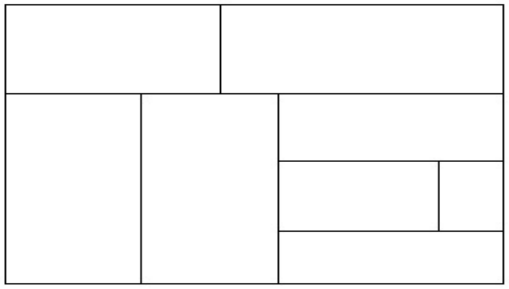
    if foil-gain(*p*) > threshold **then** add *p* to current rule

    else break

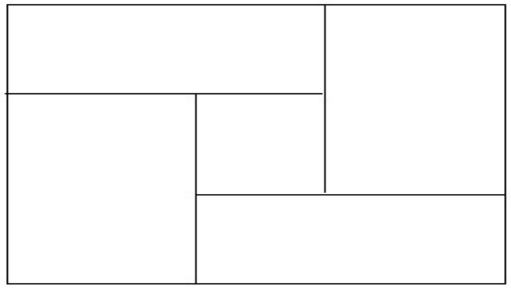


Trees and rules

- Most tree learners: divide and conquer
- Most rule learners: separate and conquer, i.e., sequential covering, (AQ, CN2, RIPPER …)
  - Some conquering-without-separating (RISE, from Domingos, biased towards complex models), rules are learned simultaneously, instance-based
- Decision space, decision boundary



(a) allowed by decision trees

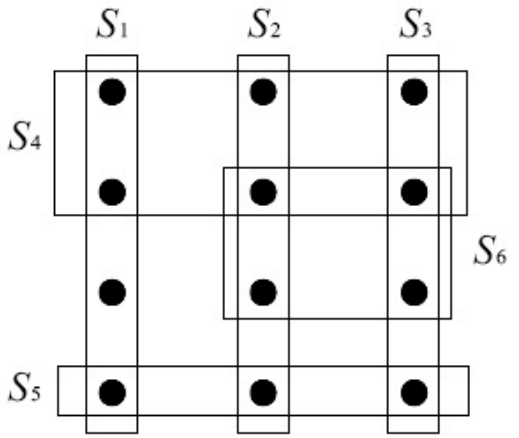


(b) not allowed by decision trees

- Both are interpretable classifiers
- Other usage of rule learning: rule extraction, e.g., from ANN

Separate and conquer vs. set cover

- Set covering problem (minimum set cover): one of the most studied combinatorial optimization problems
  - Given a finite ground set X and $S_1, S_2, \ldots S_m$ as subsets of X, find $I \subseteq \{1, \ldots m\}$ with $\mathsf{U}_{i \in I} S_i = X$ such that $|I|$ is minimized.
  - select as few as possible subsets from a given family such that each element in any subset of the family is covered
  - NP-hard
- Greedy algorithm: iteratively pick the subset that covers the maximum number of uncovered elements
  - Achieves 1 + ln n approximation ratio, optimal
- Greedy set cover vs. sequential covering
  - Select one subset (learn one rule) at a time
  - Consider uncovered elements (remove covered examples)
  - Iterate until all elements (examples) are covered



- Other related problems: graph coloring, minimum clique partition

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- **Bayes Classification Methods**
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- 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

Probability Model for Classifiers

- Let **X** = (x_1, x_2, …, x_n) be a data sample ("*evidence*"): class label is unknown
- The probability model for a classifier is to determine P(C|**X**), the probability that **X** belongs to class *C* given the observed data sample **X**
  - predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes

Bayes' Theorem

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

- P(C | X) : posterior
- P(C): prior, the initial probability
  - E.g., one will buy computer, regardless of age, income, …
- P(**X**): probability that the sample **X** is observed
- P(**X**|C): likelihood, 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
- Informally, this can be written as

  posterior = prior x likelihood / evidence

Maximizing joint probability

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

- In practice we are only interested in the numerator of that fraction, since the denominator does not depend on *H* and the same value is shared by all classes.
- The numerator is the joint probability

$$P(C\ )P(\mathbf{X}|C\ ) = P(C,\mathbf{X}) = P(C,X1,X2,\ldots Xn)$$

Maximizing joint probability

$$P(C\ )P(\mathbf{X}|C\ )=P(C,\mathbf{X})=P(C,X1,X2,...Xn)$$

repeatedly apply conditional probability, $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$.

$$=P(C)P(X1,X2,...Xn|C)$$
$$=P(C)P(X1|C)P(X2,...Xn|C,X1)$$
$$=P(C)P(X1|C)P(X2|C,X1)P(X3,...Xn|C,X1,X2)$$
$$=P(C)P(X1|C)P(X2|C,X1)...P(Xn|C,X1,X2,...Xn-1)$$

Naïve Bayes Classifier: Assuming Conditional Independence

Simplifying assumption: features are conditionally independent of each other, then,

$$P(Xi|C,Xj)=P(Xi|C) \qquad P(B \mid A) = P(B)\,.$$

$$\begin{aligned}&P(C,X1,X2,...Xn)\\&=P(C)P(X1|C)P(X2|C,X1)...P(Xn|C,X1,X2,...Xn-1)\\&=P(C)P(X1|C)P(X2|C)...P(Xn|C)\end{aligned}$$

- This greatly reduces the computation cost: Only counts the class distribution
- Only requires a small number of training data to estimate the parameters

Naïve Bayes Classifier

- 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

and $P(x_k|C_i)$ is

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

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

Naïve Bayes Classifier: Training Dataset

Class:
C1:buys_computer = ‘yes’
C2:buys_computer = ‘no’

Data to be classified:
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	com
<=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 Bayes Classifier: Example

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**
- **P(C):** P(buys_computer = "yes") = 9/14 = 0.643
  P(buys_computer = "no") = 5/14= 0.357

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

- Compute P(X|C) for each class
  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4
- **P(X|C) :** 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(C, X) = P(X|C)*P(C)
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 Zero-Probability Problem

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

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

- Suppose training set has 1000 tuples for class buys_computer= yes. 0 for income=low, 990 for income=medium, and 10 for income=high
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low | buys_computer = "yes") = 1/1003

    Prob(income = medium | buys_computer = "yes") = 991/1003

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

Naïve Bayes Classifier: Comments

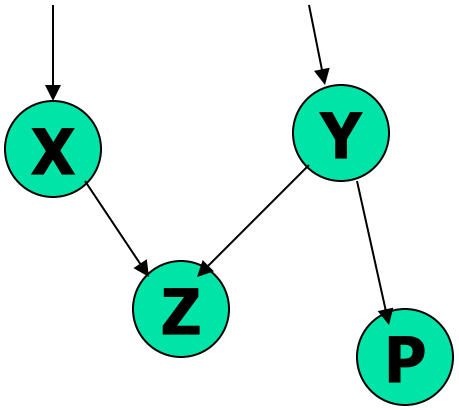
- Advantages
  - Easy to implement
  - Good results obtained in most of the cases. **Optimal if assumption holds true.**
- Disadvantages
  - Assumption: class conditional independence, loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.
      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier
- How to deal with these dependencies? Bayesian Belief Networks (Chapter 9)

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- **Bayesian Belief Networks (ch9)**
- Techniques to Improve Classification
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

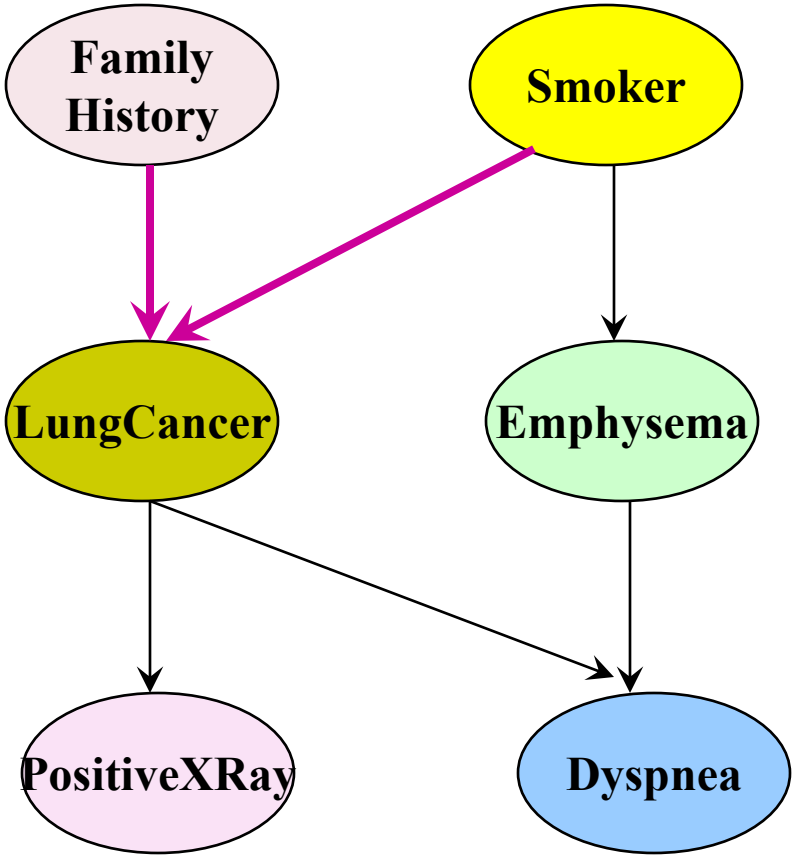
Bayesian Belief Networks

- Bayesian belief network relieves the conditional independence assumption in naïve bayes
- 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:

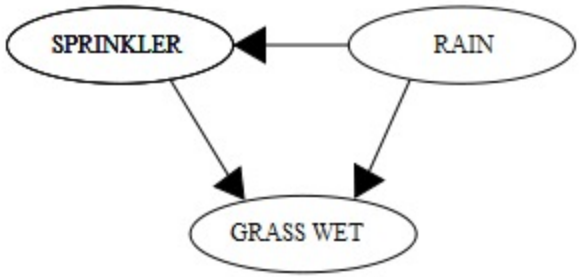
$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(xi))$$

Training Bayesian 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

Example

RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99



RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

- Two events could cause grass to be wet: either the sprinkler is on or it's raining
- The rain has a direct effect on the use of the sprinkler
  - when it rains, the sprinkler is usually not turned on

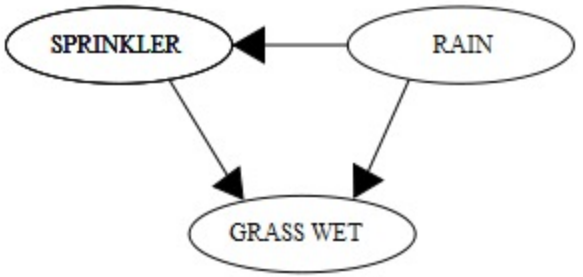
Then the situation can be modeled with a Bayesian network.
All three variables have two possible values, T and F.
The joint probability function is:

$$P(G,S,R) = P(G \mid S,R)P(S \mid R)P(R)$$

where *G = Grass wet*, *S = Sprinkler*, and *R = Rain*

Example

RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99



RAIN T	RAIN F
0.2	0.8

The joint probability function is:

$$P(G,S,R) = P(G \mid S,R)P(S \mid R)P(R)$$

where *G = Grass wet*, *S = Sprinkler*, and *R = Rain*

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

- The model can answer questions like "What is the probability that it is raining, given the grass is wet?"

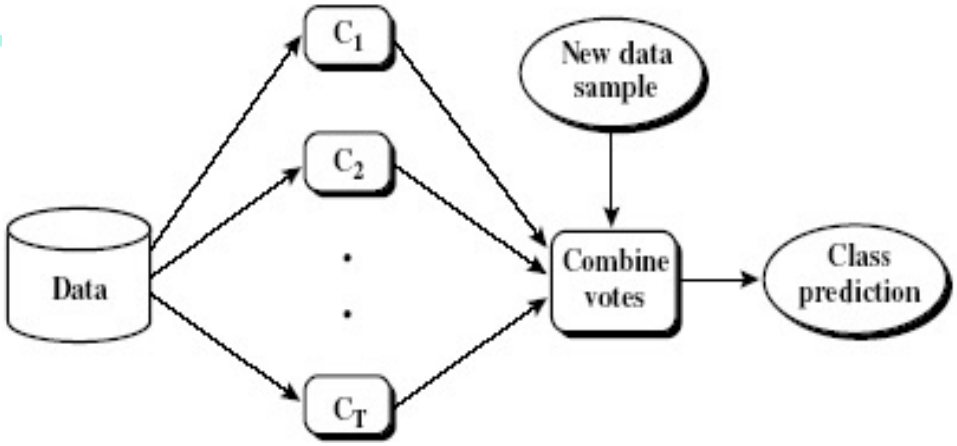
$$P(R = T \mid G = T) = \frac{P(G = T, R = T)}{P(G = T)} = \frac{\sum_{S \in \{T,F\}} P(G = T, S, R = T)}{\sum_{S,R \in \{T,F\}} P(G = T, S, R)}$$

$$= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%.$$

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- **Techniques to Improve Classification**
- Lazy Learners (ch9)
- Other known methods: SVM, ANN (ch9)

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - 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*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - 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: classify 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
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

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 algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost (Freund and Schapire, 1997)

- 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 misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Random Forest (Breiman 2001)

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods for imbalance data in 2-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: moves the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - Ensemble techniques: Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- **Lazy Learners (ch9)**
- Other known methods: SVM, ANN (ch9)

Lazy vs. Eager Learning

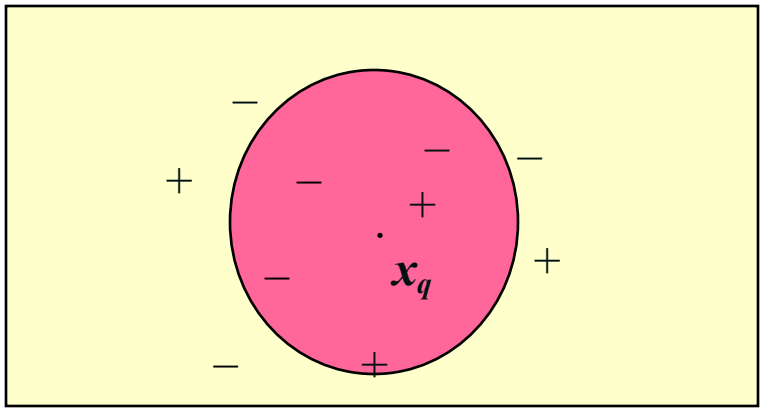
- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

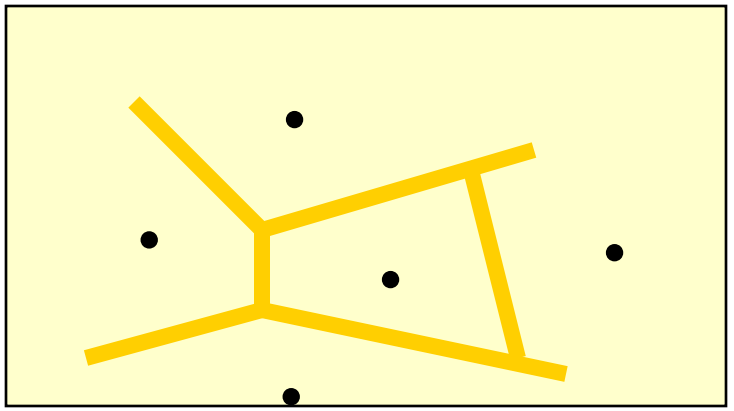
Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

The *k*-Nearest Neighbor Algorithm

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X_1}$, $\mathbf{X_2}$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





Discussion on the *k*-NN Algorithm

- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by averaging *k*-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

Chapters 8-9. Classification

- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation/Learning Algorithm Evaluation
- Rule-Based Classification
- Bayes Classification Methods
- Bayesian Belief Networks (ch9)
- Techniques to Improve Classification
- Lazy Learners (ch9)
- **Other known methods: SVM, ANN (ch9)**

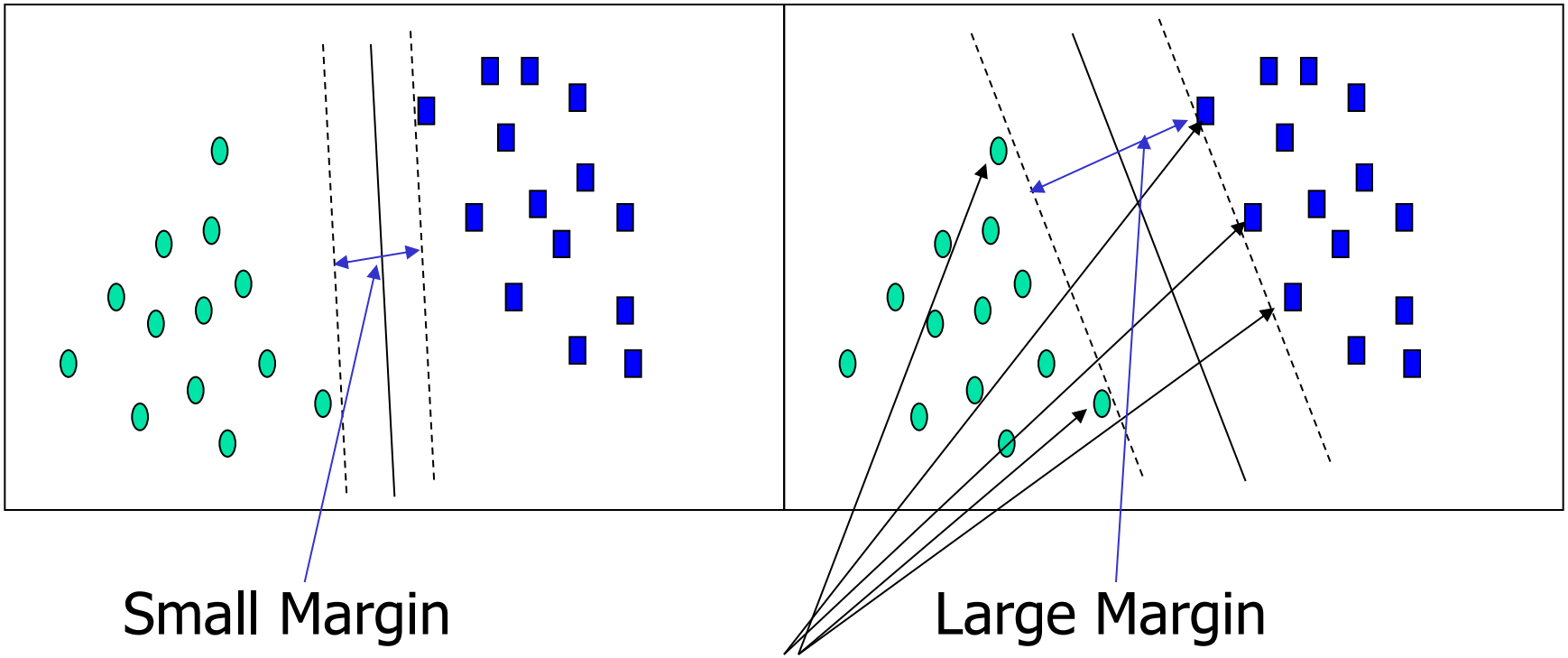
SVM—Support Vector Machines

- 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”)
- 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)

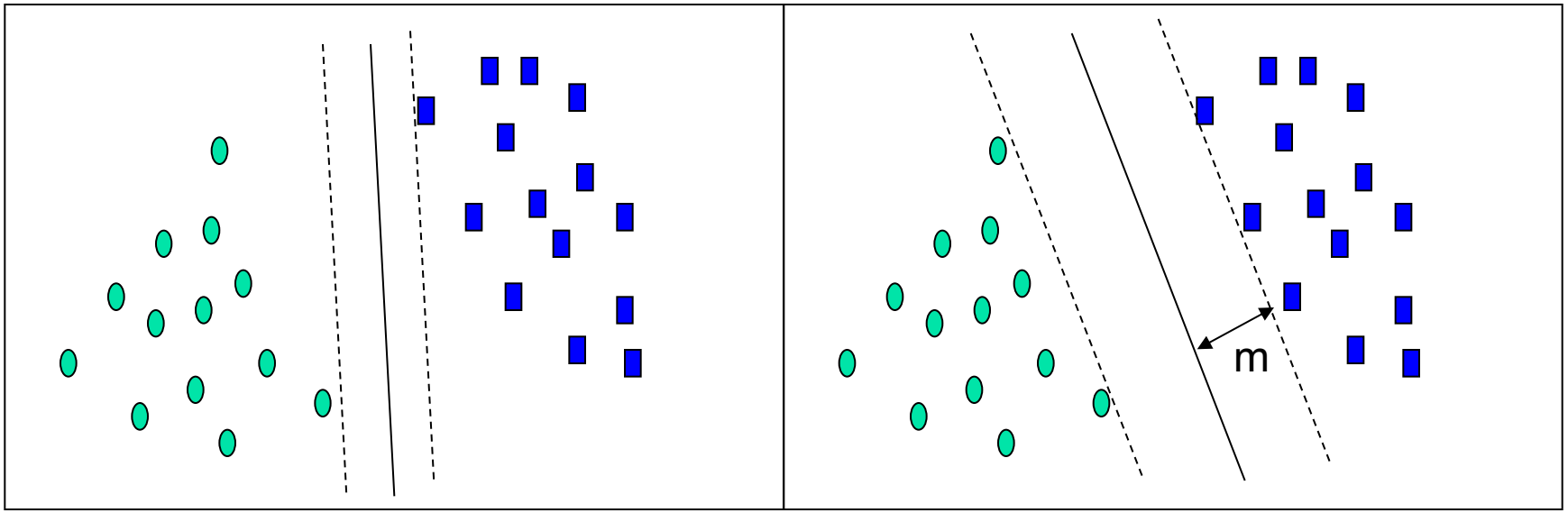
History and Applications

- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

General Philosophy



When Data Is Linearly Separable



Let data D be $(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_{|D|}, y_{|D|})$, 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)

Kernel functions

- Instead of computing the dot product on the transformed data tuples, it is mathematically equivalent to instead applying a kernel function $K(\mathbf{X_i}, \mathbf{X_j})$ to the original data, i.e., $K(\mathbf{X_i}, \mathbf{X_j}) = \Phi(\mathbf{X_i})\ \Phi(\mathbf{X_j})$
- Typical Kernel Functions

$$\text{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

$$\text{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\text{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional user parameters)

Why Is SVM Effective on High Dimensional Data?

- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

SVM—Introduction Literature

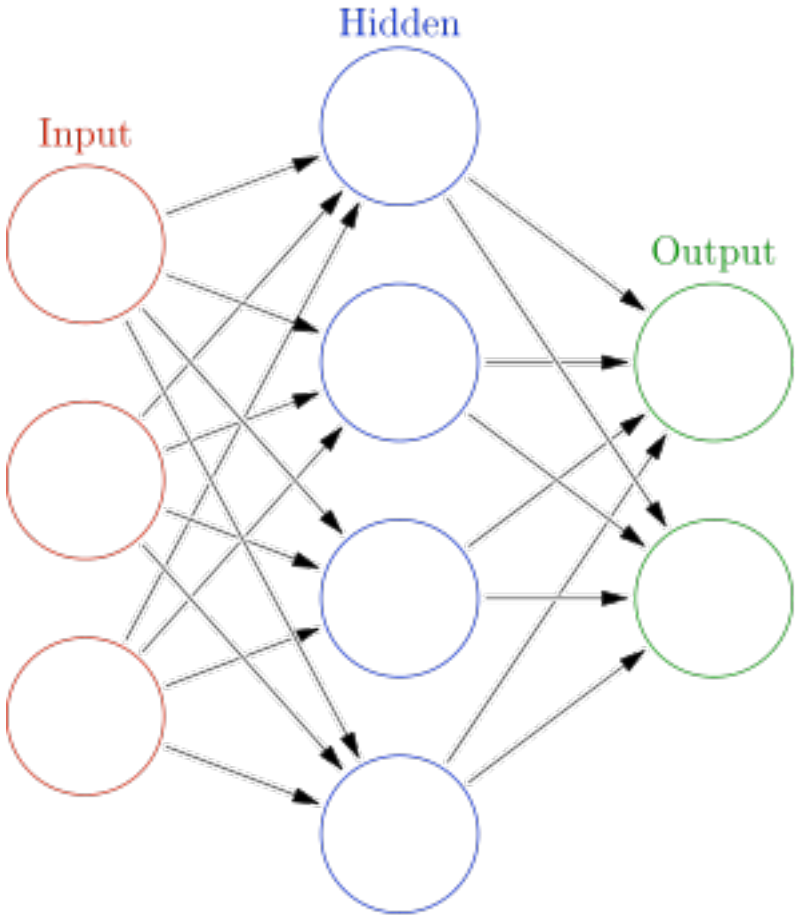
- "Statistical Learning Theory" by Vapnik: extremely hard to understand, containing many errors too.
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
  - Better than the Vapnik's book, but still written too hard for introduction, and the examples are so not-intuitive
- The book "An Introduction to Support Vector Machines" by N. Cristianini and J. Shawe-Taylor
  - Also written hard for introduction, but the explanation about the mercer's theorem is better than above literatures
- The neural network book by Haykins
  - Contains one nice chapter of SVM introduction

SVM Related Links

- SVM Website
  - http://www.kernel-machines.org/
- Representative implementations
  - 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.

ANN—Artificial Neural Network (Classification by Backpropagation)

- An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.
- Deep learning: deep neural networks



SVM vs. ANN

- SVM
  - Relatively new concept
  - Deterministic
  - Nice Generalization properties
  - Hard to learn – learned in batch mode using quadratic programming techniques
  - Using kernels can learn very complex functions
- ANN
  - Relatively old (but …)
  - Nondeterministic
  - Generalizes well but doesn't have strong mathematical foundation
  - Can easily be learned in incremental fashion
  - To learn complex functions—use multilayer perceptron (not that trivial)