SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

Classification and Prediction

UNIT-3

- Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends.

- *Classification*- predicts categorical (discrete, unordered) labels.
- *prediction* - models continuous valued functions.
- Eg: classification model to categorize bank loan applications as either safe or risky.

What Is Classification? What Is Prediction?

- A bank loans officer needs analysis of her data in order to learn which loan applicants are “safe” and which are “risky” for the bank.
- A marketing manager at *AllElectronics* needs data analysis to help guess whether a customer with a given profile will buy a new computer.
- The data analysis task is **classification**, where a model or classifier is constructed to predict categorical labels, such as **“safe” or “risky”** for the loan application data; **“yes” or “no”** for the marketing data.

- These categories can be represented by discrete values, where the ordering among values has no meaning.

- The marketing manager would like to predict how much a given customer will **spend** during a sale at *AllElectronics*.

- This data analysis task is an **example** of **numeric prediction**, where the model constructed predicts a ***continuous-valued function*, or *ordered value***, as opposed to a categorical label.

- This model is a **predictor**. **Regression analysis** is a statistical methodology that is most often used for numeric prediction.

How does classification work?

Data classification is a **two-step** process

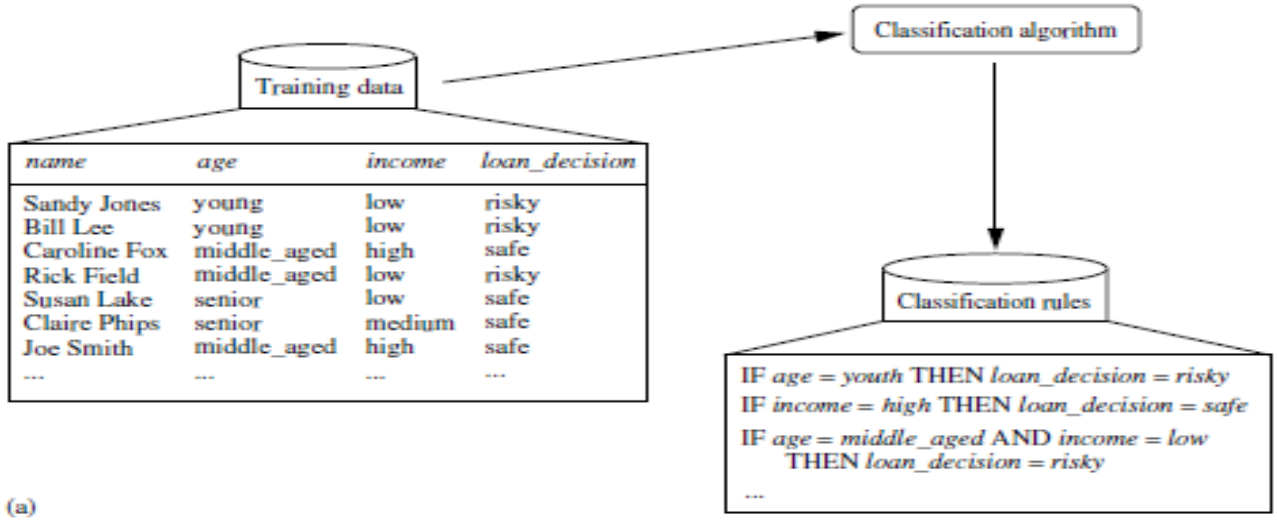
- In the first step, a classifier is built describing a predetermined set of data classes or concepts.
- This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels.

Classification—A Two-Step Process

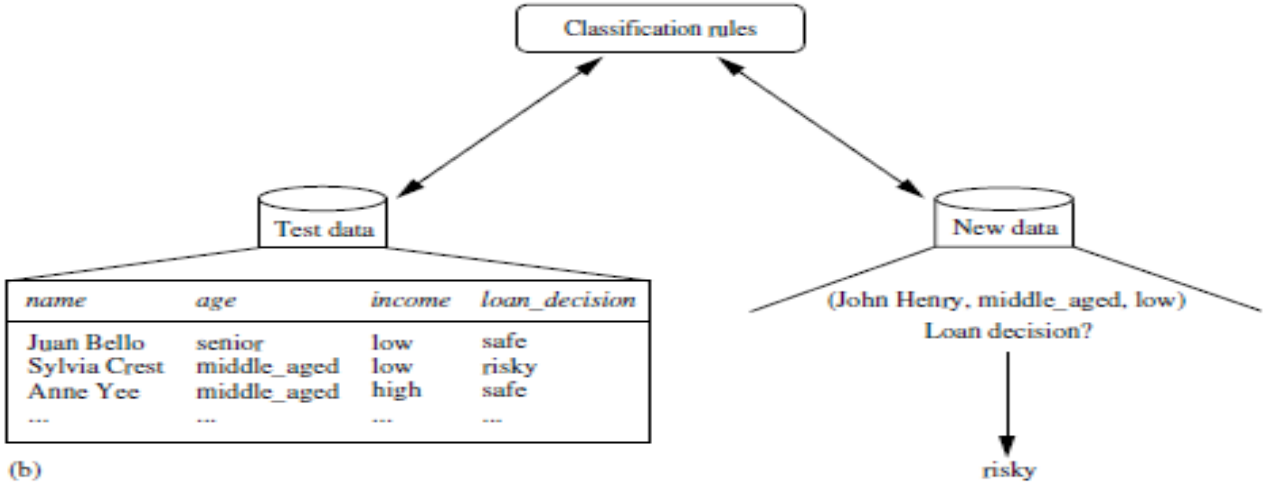
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision 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
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation (test) set

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. with the aim of establishing the existence of classes or clusters in the data


Classification algorithm
Training data
name age income loan_decision
Sandy Jones young low risky
Bill Lee young low risky
Caroline Fox middle_aged high safe
Rick Field middle_aged low risky
Susan Lake senior low safe
Claire Phips senior medium safe
Joe Smith middle_aged high safe
... ...
Classification rules
IF age = youth THEN loan_decision = risky
IF income = high THEN loan_decision = safe
IF age = middle_aged AND income = low
THEN loan_decision = risky
...

(a)


Classification rules
Test data
New data
name age income loan_decision
Juan Bello senior low safe
Sylvia Crest middle_aged low risky
Anne Yee middle_aged high safe
... ...
(John Henry, middle_aged, low)
Loan decision?
risky

(b)

Issues Regarding Classification and Prediction

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

Preparing the Data for Classification and Prediction

- preprocessing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process.
- **Data cleaning:** remove or reduce noise (by applying smoothing techniques).
- The treatment of missing values (e.g., by replacing a missing value with the most commonly occurring value for that attribute, or with the most probable value based on statistics).
- This step can help reduce confusion during learning.

- **Relevance analysis:** Many of the attributes in the data may be *redundant*.

- **Correlation analysis** can be used to identify whether any two given attributes are statistically related.

- A database may also contain ***irrelevant* attributes**. **Attribute subset selection** can be used in these cases to find a reduced set of attributes.
- These two are used to detect attributes that do not contribute to the classification or prediction task.
- *Time spent on relevance analysis + time spent on learning from the resulting "reduced" attribute subset < time spent on learning from the original set of attributes*

- **Data transformation and reduction:** The data may be transformed by **normalization.**
- Normalization involves scaling all values for a given attribute so that they fall within a small specified range, such as -1:0 to 1:0, or 0:0 to 1:0.
- The data can also be transformed by ***generalizing*** it to higher-level concepts.
- **Concept hierarchies** may be used for this purpose.
- **Eg:** street, can be generalized to higher-level concepts, like city.

Comparing Classification and Prediction Methods

- **Accuracy:** The **accuracy of a classifier** refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information).

- The **accuracy of a predictor** refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data.

- **Speed:** This refers to the computational costs involved in generating and using the given classifier or predictor.

- **Robustness:** This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.

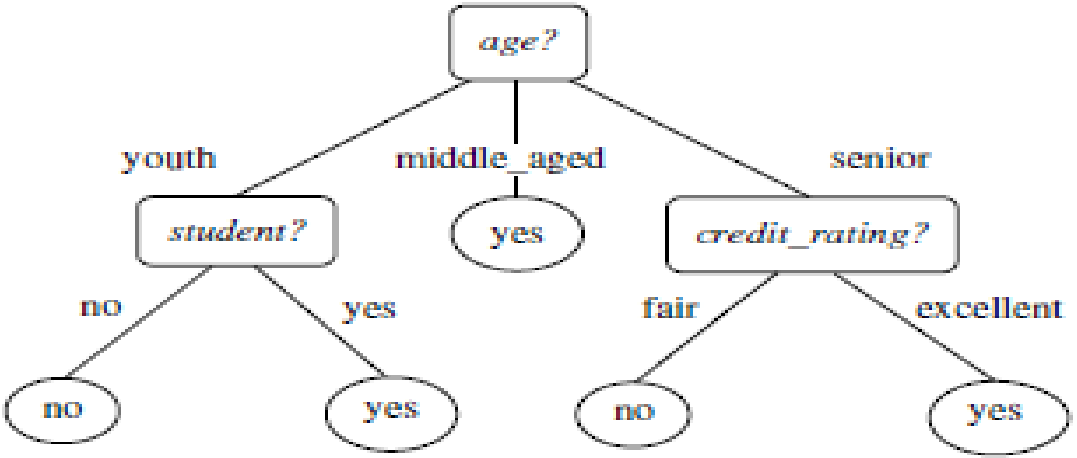
- **Scalability:** This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

- **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier or predictor.
- Interpretability is **subjective** and therefore more difficult to assess.

Classification by Decision Tree Induction

A **decision tree** is a flowchart-like tree structure.

- Internal node (non leaf node) - test on an attribute.
- Branch - outcome of the test.
- Leaf node (or *terminal node*) - holds a class label.
- The topmost node in a tree is the **root** node.



Why are decision tree classifiers so popular?

- The construction of decision tree classifiers does not require any domain knowledge or parameter setting.
- Decision trees can handle high dimensional data.
- The learning and classification steps of decision tree induction are simple and fast.
- Decision tree classifiers have good accuracy.

Decision Tree Induction

- During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as **ID3** (Iterative Dichotomiser).

- P. T. Stone. Quinlan later presented **C4.5** (a successor of ID3).

- In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book ***Classification and Regression Trees*** **(CART)**, which described the generation of binary decision trees.

- ID3, C4.5, and CART adopt a **greedy** (i.e., non backtracking) approach in which decision trees are constructed in a **top-down** recursive divide-and-conquer manner.

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C then
(3)       return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)       return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
          multiway splits allowed then // not restricted to binary trees
(9)       attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)      let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)      if Dj is empty then
(13)           attach a leaf labeled with the majority class in D to node N;
(14)      else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Basic algorithm for inducing a decision tree from training tuples.

- The algorithm is called with three parameters: ***D***, ***attribute list***, and ***Attribute selection method***.
- *D* - data partition.
- *Attribute list* - list of attributes describing the tuples.
- *Attribute selection method* – procedure for selecting the attribute.

1. The tree starts as a single node, *N* representing the training tuples in *D.*
2. If the tuples in *D* are all of the same class, then node *N* becomes a leaf and is labelled with that class.
3. Otherwise, the algorithm calls *Attribute selection method* to determine the splitting criterion.
4. The splitting criterion tells us which attribute to test at node *N* by determining
5. the "best" way to separate or partition the tuples in *D* into individual classes.
6. A partition is pure if all of the tuples in it belong to the same class.
7. The node *N* is labeled with the splitting criterion, which serves as a test at the node

***A is discrete-valued*:**

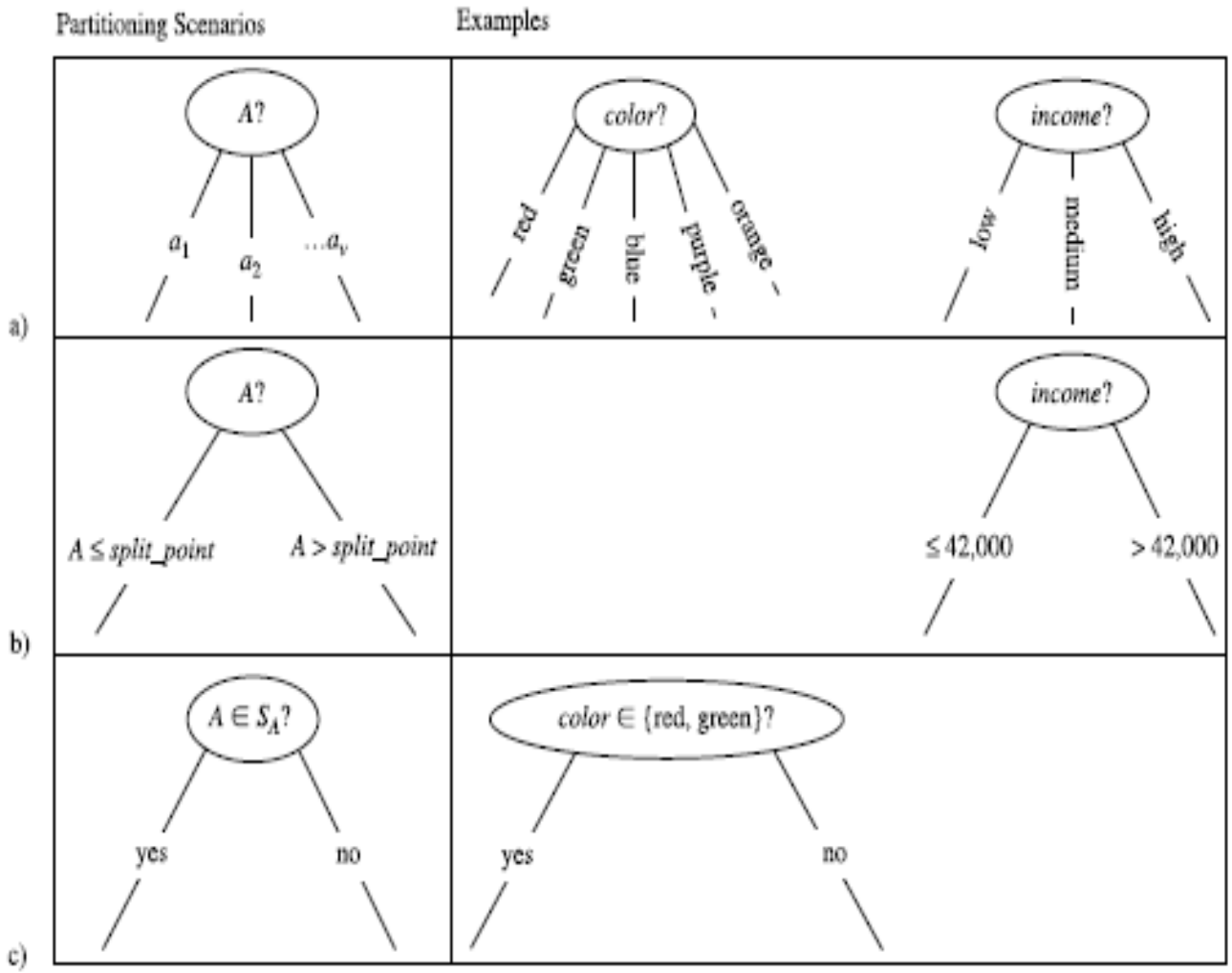
- The outcomes of the test at node *N* correspond directly to the known values of *A*.
- A branch is created for each known value, *aj*, of *A* and labeled with that value.
- Partition *Dj* is the subset of class-labeled tuples in *D* having value *aj* of *A*.
- Because all of the tuples in a given partition have the same value for *A*, then *A* need not be considered in any future partitioning of the tuples.
- Therefore, it is removed from *attribute list*

***A is continuous-valued*:**

- The test at node *N* has two possible outcomes, corresponding to the conditions *A <= split point* and *A > split point*, respectively.
- Where *split point* is the split-point returned by *Attribute selection method* as part of the splitting criterion.
- The tuples are partitioned such that*D*1 holds the subset of class-labeled tuples in *D* for which *A <=split point*, while *D*2 holds the rest.

***A is discrete-valued* and a *binary tree* must be produced**

- The test at node *N* is of the form "$A \in SA$?"
- *SA* is the splitting subset for *A*, returned by *Attribute selection method* as part of the splitting criterion.
- The left branch out of *N* is labeled *yes* so that *D*1 corresponds to the subset of class-labeled tuples in *D* that satisfy the test.
- The right branch out of *N* is labeled *no* so that *D*2 corresponds to the subset of class-labeled tuples from *D* that do not satisfy the test.


Partitioning Scenarios
Examples
a)
A?
a_1
a_2
$\ldots a_v$
color?
red
green
blue
purple
orange
income?
low
medium
high
b)
A?
$A \leq split\_point$
$A > split\_point$
income?
≤ 42,000
> 42,000
c)
$A \in S_A$?
yes
no
color ∈ {red, green}?
yes
no

The recursive partitioning stops only when any one of the following terminating conditions is true:

1. All of the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3), or
2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node N into a leaf and labeling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.
3. There are no tuples for a given branch, that is, a partition D_j is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).

The resulting decision tree is returned (step 15).

Attribute Selection Measures

- Three popular attribute selection **measures—*information gain, gain ratio*, and *gini index*.**

Information gain

- **ID3** uses information gain as its attribute selection measure.
- The attribute with the **highest information gain** is chosen as the splitting attribute for node *N*.
- Where pi is the probability that an arbitrary tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|.
- A log function to the base 2 is used, because the information is encoded in bits.
- Info(D) is just the **average amount of information** needed to identify the class label of a tuple in D.
- *Info(D)* is also known as the **entropy of *D*.**

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

How much more information would we still need (after the partitioning) in order to arrive at an exact classification? This amount is measured by

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

- The term |*Dj* |/|*D*| acts as the weight of the *j*th partition.
- *InfoA*(*D*) is the expected information required to classify a tuple from *D* based on the partitioning by *A*.
- The smaller the expected information (still) required, the greater the purity of the partitions.
- **Information gain** is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on *A*).

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

Induction of a decision tree using information gain

Class-labeled training tuples from the *AllElectronics* customer database.

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

- The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}).
- There are two distinct classes (that is, $m = 2$).
- Let class ***C1*** correspond to ***yes*** and class ***C2*** correspond to ***no***.
- There are **nine** tuples of class ***yes*** and **five** tuples of class ***no***.
- A (root) node N is created for the tuples in D.

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

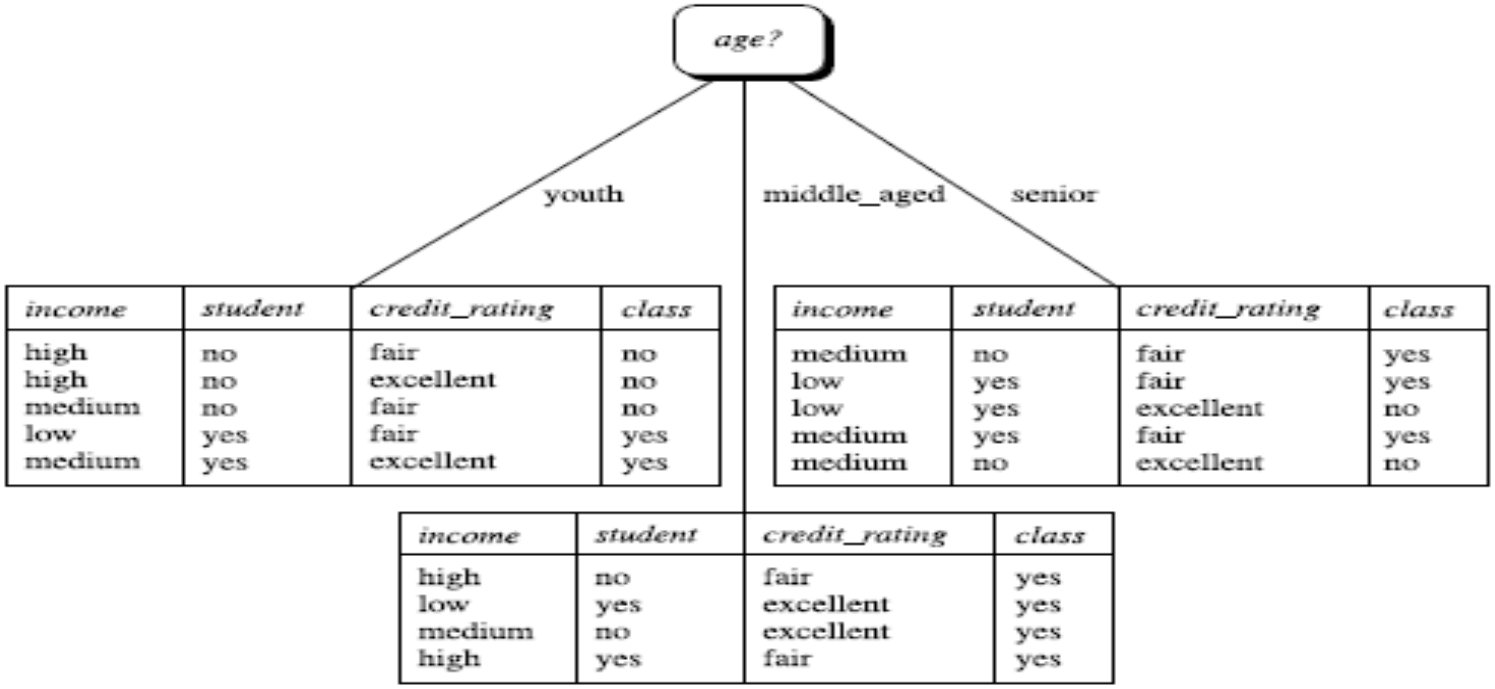
- Compute the expected information requirement for each attribute.
- *Age* category *youth – 2 yes &3 no, Middle aged – 4 yes & 0 no, Senior – 3 yes & 2 no.*

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ &+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) \\ &+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

- Compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits.
- Age has the highest information gain among the attributes, it is selected as the splitting attribute.

- Tuples falling into the partition for *age = middle aged* all belong to the same class.
- Because they all belong to class *"yes,"* a leaf should therefore be created at the end of this branch and labeled with *"yes."*



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Information gain of attribute – continuous valued

- Split-point for A, where the split-point is a threshold on A.
- We first sort the values of A in increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point.

$$\frac{a_i + a_{i+1}}{2}.$$

Predict if John will play tennis

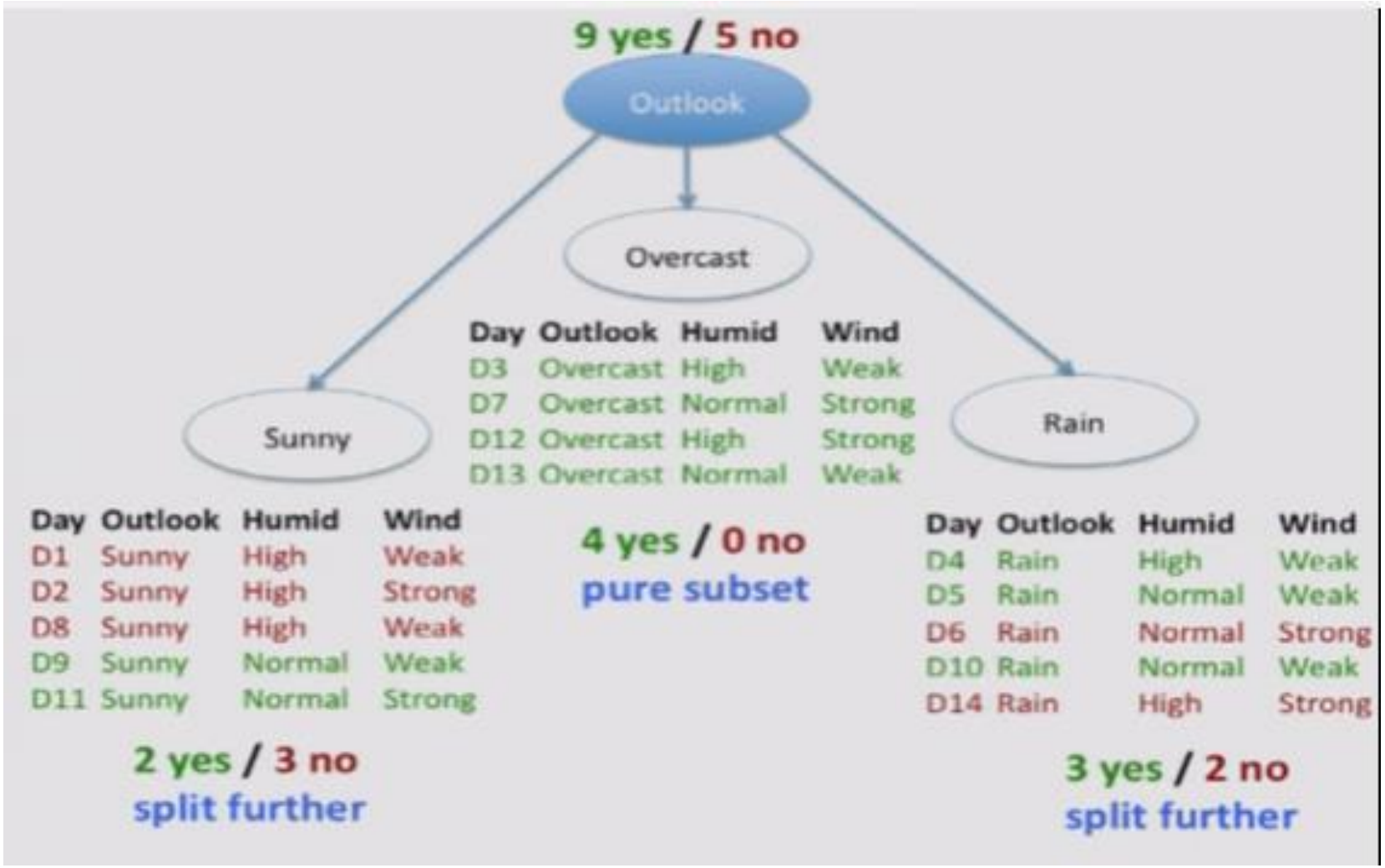
- Hard to guess
- Try to *understand* when John plays
- Divide & conquer:
  - split into subsets
  - are they pure? (all yes or all no)
  - if yes: stop
  - if not: repeat
- See which subset new data falls into

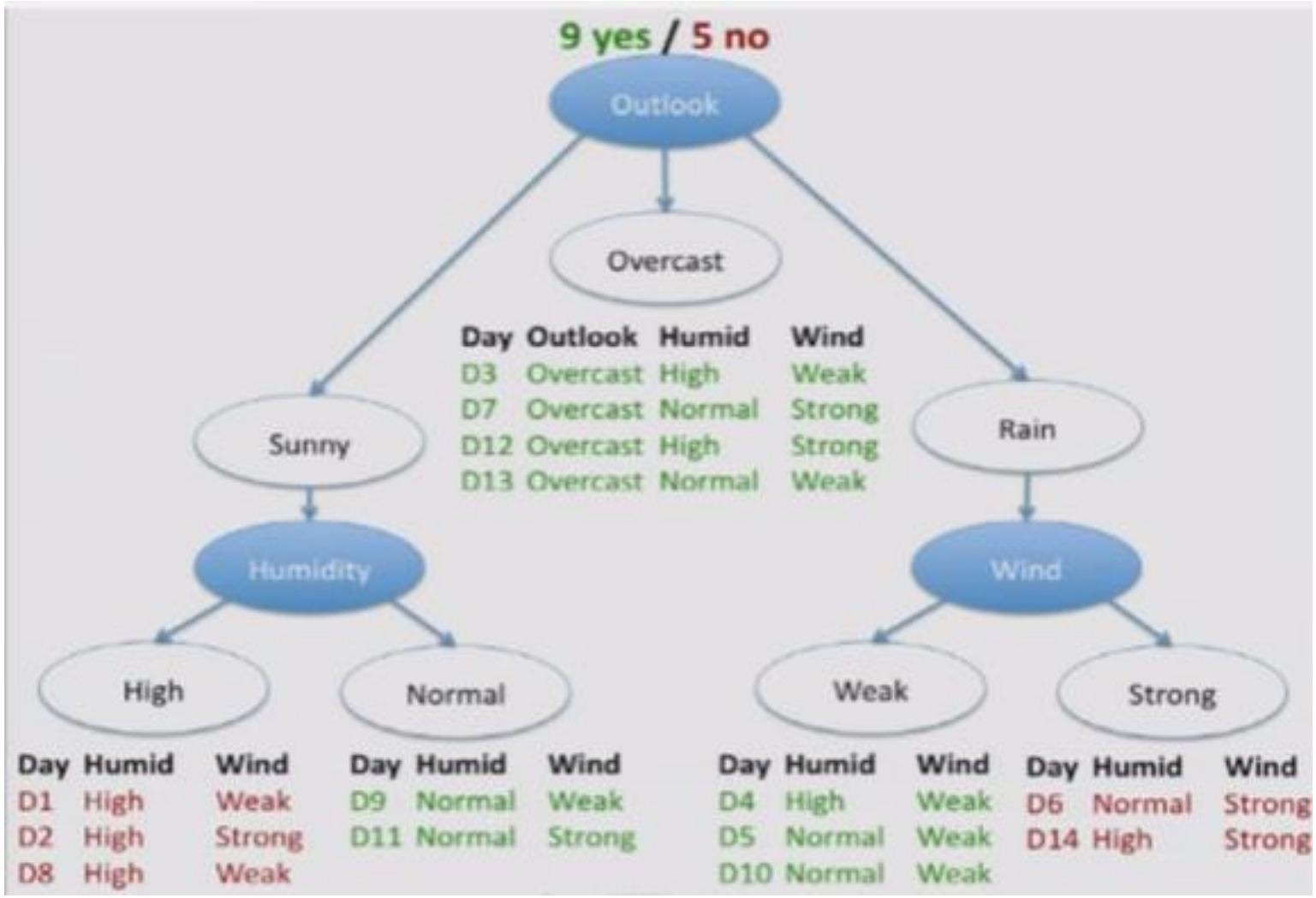
Training examples: **9 yes / 5 no**

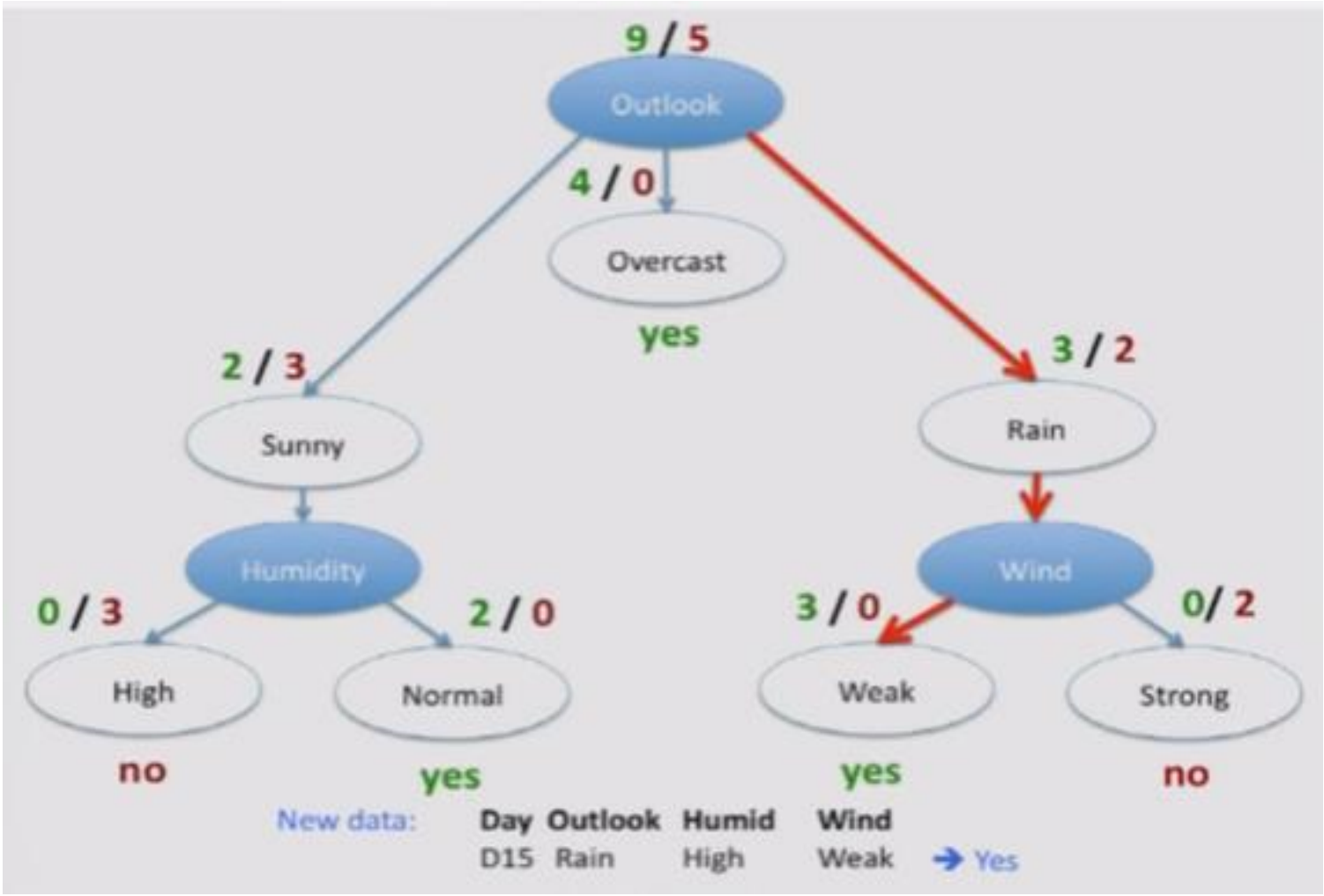
Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?


9 yes / 5 no
Outlook
Overcast
Sunny
Rain
Day Outlook Humid Wind
D3 Overcast High Weak
D7 Overcast Normal Strong
D12 Overcast High Strong
D13 Overcast Normal Weak
4 yes / 0 no
pure subset
Day Outlook Humid Wind
D1 Sunny High Weak
D2 Sunny High Strong
D8 Sunny High Weak
D9 Sunny Normal Weak
D11 Sunny Normal Strong
2 yes / 3 no
split further
Day Outlook Humid Wind
D4 Rain High Weak
D5 Rain Normal Weak
D6 Rain Normal Strong
D10 Rain Normal Weak
D14 Rain High Strong
3 yes / 2 no
split further


9 yes / 5 no
Outlook
Overcast
Sunny
Rain
Humidity
Wind
High
Normal
Weak
Strong
Day Outlook Humid Wind
D3 Overcast High Weak
D7 Overcast Normal Strong
D12 Overcast High Strong
D13 Overcast Normal Weak
Day Humid Wind
D1 High Weak
D2 High Strong
D8 High Weak
Day Humid Wind
D9 Normal Weak
D11 Normal Strong
Day Humid Wind
D4 High Weak
D5 Normal Weak
D10 Normal Weak
Day Humid Wind
D6 Normal Strong
D14 High Strong


9 / 5
Outlook
4 / 0
Overcast
yes
2 / 3
Sunny
3 / 2
Rain
Humidity
Wind
0 / 3
2 / 0
3 / 0
0/ 2
High
Normal
Weak
Strong
no
yes
yes
no
New data:
Day Outlook Humid Wind
D15 Rain High Weak
Yes

Gain ratio

- It prefers to select attributes having a large number of values.
- C4.5, a successor of ID3, uses an extension to information gain known as *gain ratio.*
- It applies a kind of normalization to information gain using a "split information" value defined analogously with *Info*(*D*) as

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

- It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning.

- The gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}.$$

Computation of gain ratio for the attribute *income*. A test on *income* splits the data of Table 6.1 into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*, we first use Equation (6.5) to obtain

$$SplitInfo_A(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right).$$

$$= 0.926.$$

$$Gain(income) = 0.029. \qquad GainRatio(income) = 0.029/0.926 = 0.031.$$

Gini index

- The Gini index is used in CART.

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

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_{i,D}|/|D|$. The sum is computed over m classes.

- The Gini index considers a binary split for each attribute.
- where A is a discrete-valued attribute having v distinct values, {$a1$, $a2$, : : : , av}, occurring in D.
- Determine the best binary split on A, we examine all of the possible subsets that can be formed using known values of A.
- If A has v possible values, then there are $2v$ possible subsets.

Eg: *income* has three possible values, namely {*low,medium, high*}, then the possible subsets are {*low, medium, high}*, {*low, medium*}, {*low,high*}, {*medium, high}*, {*low*}, {*medium*}, {*high}*, and {}.

- Exclude the power set,{*low, medium, high*}, and the empty - they do not represent a split.
- 2^v - 2 possible ways to form two partitions of the data, *D*, based on a binary split on *A*.

$$Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2).$$

- For a discrete-valued attribute, the subset that gives the minimum gini index for that attribute is selected as its splitting subset.

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute *A* is

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

- The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini index) is selected as the splitting attribute.

Induction of a decision tree using gini index:

Let *D* be the training data where there are nine tuples belonging to the class *buys computer = yes* and the remaining five tuples belong to the class *buys computer = no*. A (root) node *N* is created for the tuples in *D*. We first use Equation for Gini index to compute the impurity of *D*

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

Let's start with the attribute *income* and consider each of the possible splitting subsets. Consider the subset {*low, medium*}. This would result in 10 tuples in partition *D*1 satisfying the condition "*income* ∈ {*low, medium*}." The remaining four tuples of *D* would be assigned to partition *D*2.

$$\begin{aligned} &Gini_{income \in \{low,medium\}}(D) \\ &= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\ &= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\ &= 0.450 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

The Gini index values for splits on the remaining subsets are:

- ({*low, high*} and {*medium*}) = 0.315
- ({*medium, high*} and {*low*}) = 0.300

- Therefore, the best binary split for attribute *income* is on {*medium, high*} (or {*low*})because it minimizes the gini index.

- {*youth, senior}* (or {*middle aged*}) best split for *age* with a Gini index of 0.375.

- {*student*} = 0.367
- {*credit rating*} = 0.429

- The attribute *income* and splitting subset {*medium, high*} therefore give the minimum gini index overall, with a reduction in impurity of 0.459-0.300 = 0.159.

- The binary split "*income* ∈ {*medium, high*}" results in the maximum reduction in impurity of the tuples in *D* and is returned as the splitting criterion.

Bayesian Classification

- Bayesian classification is based on Bayes' theorem.

- Studies comparing classification algorithms have found a simple Bayesian classifier known as the *naïve Bayesian classifier.*

- Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called *class conditional independence*.

Bayes' Theorem

- Let $\boldsymbol{X}$ be a data tuple.
- In Bayesian terms, $\boldsymbol{X}$ is considered "evidence."
- Let H be some hypothesis, such as that the data tuple $\boldsymbol{X}$ belongs to a specified class C.
- For classification problems, we want to
- determine $P(H|\boldsymbol{X})$, the probability that the hypothesis H holds given the "evidence" or observed data tuple $\boldsymbol{X}$.
- P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.
- The attributes age and income, respectively, and that X is a 35-year-old customer with an income of $40,000.
- Suppose that H is the hypothesis that our customer will buy a computer. Then P(H|X) reflects the probability that customer X will buy a computer given that we know the customer's age and income.

- $P(H)$ is the prior probability, or *a priori probability,* of H.
- $P(\boldsymbol{X}|H)$ is the posterior probability of $\boldsymbol{X}$ conditioned on H.
- it is the probability that a customer, $\boldsymbol{X}$, is 35 years old and earns $40,000, given that we know the customer will buy a computer.
- $P(\boldsymbol{X})$ is the prior probability of $\boldsymbol{X}$. Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000.
- calculating the posterior probability, P(H|X).
- **Bayesian classifiers** have the **minimum error rate** in comparison to all other classifiers.

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

Naïve Bayesian Classification

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $\mathbf{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, $\mathbf{X}$, the classifier will predict that $\mathbf{X}$ belongs to the class having the highest posterior probability, conditioned on $\mathbf{X}$. That is, the naïve Bayesian classifier predicts that tuple $\mathbf{X}$ belongs to the class C_i if and only if

$$P(C_i|\mathbf{X}) > P(C_j|\mathbf{X}) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize $P(C_i|\mathbf{X})$. The class C_i for which $P(C_i|\mathbf{X})$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Equation (6.10)),

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}. \tag{6.11}$$

3. As $P(\mathbf{X})$ is constant for all classes, only $P(\mathbf{X}|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are

equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(\boldsymbol{X}|C_i)$. Otherwise, we maximize $P(\boldsymbol{X}|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of training tuples of class C_i in D.

4. In order to reduce computation in evaluating $P(\boldsymbol{X}|Ci)$, the naïve assumption of class conditional independence is made.

$$
\begin{aligned}
P(\boldsymbol{X}|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\
&= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).
\end{aligned}
$$

(a) If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{6.13}$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \tag{6.14}$$

5. In order to predict the class label of $\boldsymbol{X}$, $P(\boldsymbol{X}|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple $\boldsymbol{X}$ is the class C_i if and only if

$$P(\boldsymbol{X}|C_i)P(C_i) > P(\boldsymbol{X}|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \tag{6.15}$$

In other words, the predicted class label is the class C_i for which $P(\boldsymbol{X}|C_i)P(C_i)$ is the maximum.

Predicting a class label using naïve Bayesian classification

- The data tuples are described by the attributes *age*, *income*, *student*, and *credit rating*.
- The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}).
- Let $C1$ correspond to the class *buys computer = yes* and $C2$ correspond to *buys computer = no.*
- The tuple we wish to classify is
- $\boldsymbol{X}$ = (*age = youth, income = medium, student = yes, credit rating = fair*)
- We need to maximize $P(\boldsymbol{X}|Ci)P(Ci)$, for $i = 1, 2$. $P(Ci)$, the prior probability of each class, can be computed based on the training tuples

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

To compute $PX|C_i)$, for $i = 1, 2$, we compute the following conditional probabilities:

$$P(age = youth \mid buys\_computer = yes) = 2/9 = 0.222$$
$$P(age = youth \mid buys\_computer = no) = 3/5 = 0.600$$
$$P(income = medium \mid buys\_computer = yes) = 4/9 = 0.444$$
$$P(income = medium \mid buys\_computer = no) = 2/5 = 0.400$$
$$P(student = yes \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(student = yes \mid buys\_computer = no) = 1/5 = 0.200$$
$$P(credit\_rating = fair \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(credit\_rating = fair \mid buys\_computer = no) = 2/5 = 0.400$$

Using the above probabilities, we obtain

$$P(\mathbf{X}|buys\_computer = yes) = P(age = youth \mid buys\_computer = yes) \times$$
$$P(income = medium \mid buys\_computer = yes) \times$$
$$P(student = yes \mid buys\_computer = yes) \times$$
$$P(credit\_rating = fair \mid buys\_computer = yes)$$
$$= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044.$$

Similarly,

$$P(\mathbf{X}|buys\_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019.$$

To find the class, C_i, that maximizes $P(\mathbf{X}|C_i)P(C_i)$, we compute

$$P(\mathbf{X}|buys\_computer = yes)P(buys\_computer = yes) = 0.044 \times 0.643 = 0.028$$
$$P(\mathbf{X}|buys\_computer = no)P(buys\_computer = no) = 0.019 \times 0.357 = 0.007$$

Therefore, the naïve Bayesian classifier predicts $buys\_computer = yes$ for tuple $\mathbf{X}$.

Using the Laplacian correction to avoid computing probability values of zero:

- Suppose that for the class ***buys computer = yes*** in some training database, *D*, containing **1,000 tuples**, we have **0 tuples** with ***income = low*****, 990 tuples with *income = medium*, and 10 tuples with *income = high*.**

- The probabilities of these events, **without** the **Laplacian correction**, are 0, 0.990 (from 999/1000), and 0.010 (from 10/1,000), respectively.

- Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair.

$$\frac{1}{1,003} = 0.001, \frac{991}{1,003} = 0.988, \text{ and } \frac{11}{1,003} = 0.011,$$

The "corrected" probability estimates are close to their "uncorrected" counterparts, yet the zero probability value is avoided.

Rule-Based Classification

Using IF-THEN Rules for Classification

- A rule-based classifier uses a set of IF-THEN rules for classification.
- An IF-THEN rule is an expression of the form
- IF *condition* THEN *conclusion*.

An example is rule *R*1,

- R1: IF *age* = *youth* AND *student* = *yes* THEN *buys computer* = *yes*.
- The “IF”-part (or left-hand side)of a rule - rule antecedent or precondition.
- The “THEN”-part (or right-hand side) of a rule - rule consequent.

- The rule antecedent, the condition consists of one or more *attribute tests* that are logically ANDed.
- The rule's consequent contains a class prediction.

R1: (*age* = *youth*) ^ (*student* = *yes*))(*buys computer* = *yes*).

- If the condition in a rule antecedent holds true for a given tuple, we say that the rule antecedent is satisfied and that the rule covers the tuple.
- A rule *R* can be assessed by its coverage and accuracy
- let $\boldsymbol{n_{covers}}$ be the number of tuples covered by *R.*
- $\boldsymbol{n_{correct}}$ be the number of tuples correctly classified by *R.*
- $\boldsymbol{|D|}$ be the number of tuples in *D.*
- **Rule's Coverage** - percentage of tuples that are covered by the rule.
- **Rule's Accuracy**- percentage of them the rule can correctly classify.

Rule accuracy and coverage

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

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

- The class-labeled tuples from the *AllElectronics* customer database is taken. Our task is to predict whether a customer will buy a computer.
- Consider rule $R1$ above, which covers 2 of the 14 tuples.
- It can correctly classify both tuples.
- Therefore,
- $coverage(R1)$ = 2=14 = 14:28%
- *Accuracy* $(R1)$ = 2=2 = 100%.

- If a rule is satisfied by ***X***, the rule is said to be triggered.
- If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to ***X***.
- ***size ordering, rule ordering***
- **size ordering -** assigns the **highest priority** to the triggering rule that has the "**toughest**" requirements, where toughness is measured by the rule antecedent *size*.
- **Rule ordering** - prioritizes the rules beforehand. The ordering may be class based or rule-based.
- **class-based ordering** - classes are sorted in order of decreasing "importance," such as by decreasing order of prevalence.

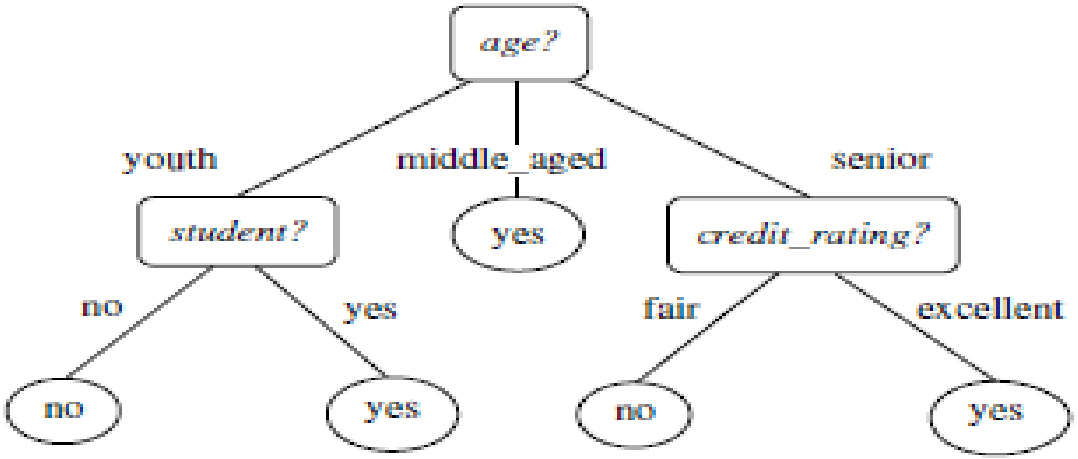
.

- **Rule-based ordering** - the rules are organized into one long priority list, according to some measure of rule quality such as accuracy, coverage, or size.
- When rule ordering is used, the rule set is known as a **decision list**.
- When there is **no rule satisfied** by X.
- A **fallback or default rule** can be set up to specify a default class, based on a training set.
- The **default rule is evaluated at the end**, if and only if no other rule covers X. The condition in the default rule is empty.

Rule Extraction from a Decision Tree

- Decision tree classifiers are a popular method of classification.
- They are known for their **accuracy**.
- Decision trees can become large and difficult to interpret.
- In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand.
- To extract rules from a decision tree, **one rule** is created for **each path** from the root to a leaf node.
- Each **splitting criterion** along a given path is **logically ANDed** to form the rule antecedent.
- The **leaf node** holds the **class prediction**, forming the rule Consequent.

Extracting classification rules from a decision tree



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

- A **disjunction (logical OR)** is implied between each of the extracted rules.
- Because the rules are extracted directly from the tree, they are **mutually exclusive and exhaustive**.
- ***Mutually exclusive*** - cannot have rule conflicts here because no two rules will be triggered for the same tuple.
- ***Exhaustive*** - there is one rule for each possible attribute-value combination, so that this set of rules does not require a default rule.
- We may end up in **repetition** and **replication** of some rules.
- we may need to do some more work by **pruning** the resulting rule set.

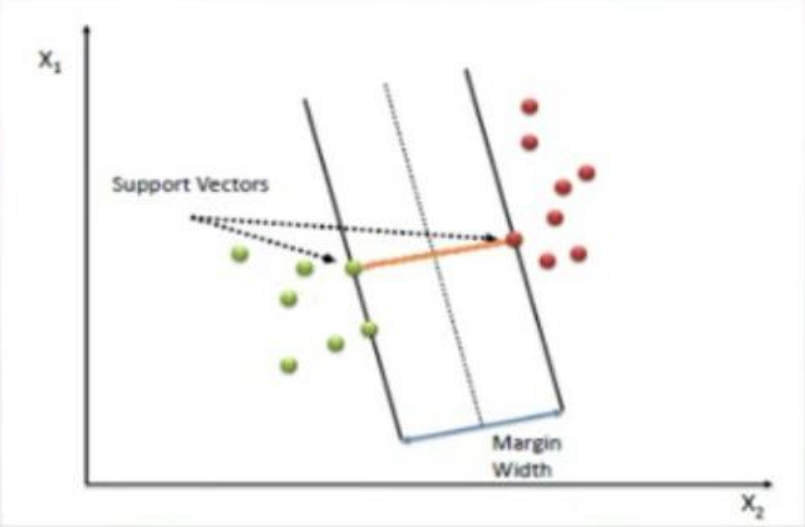
SVM – Support Vector Machine

Overview of SVM

- A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the margin between the two classes

Notice: linearly separable and binary

- *It draws the widest channel, or street, between the two classes*
- The two class labels are +1 (positive examples) and -1 (negative examples)



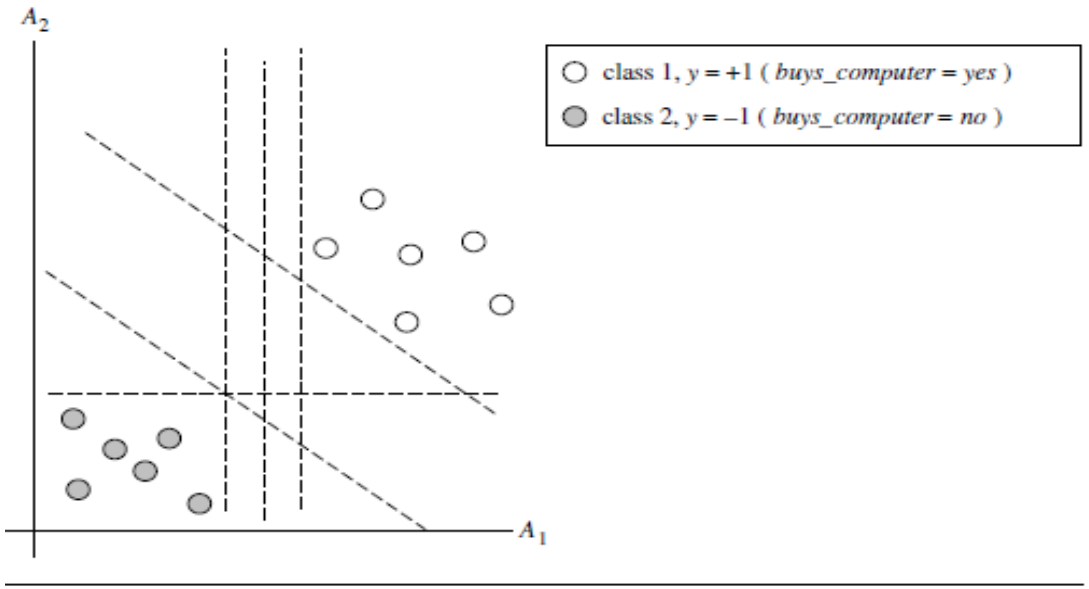
The Case When the Data Are Linearly Separable

- Let the data set D be given as $(\boldsymbol{X}1, y1)$, $(\boldsymbol{X}2, y2)$, … , $(\boldsymbol{X}|D|, y|D|)$,where $\boldsymbol{X}i$ is the set of training tuples with associated class labels, yi.

- Each yi can take one of two values, either+1 or -1 (i.e., $yi \in \{+1, -1\}$), corresponding to the classes *buys computer = yes* and *buys computer = no*, respectively.

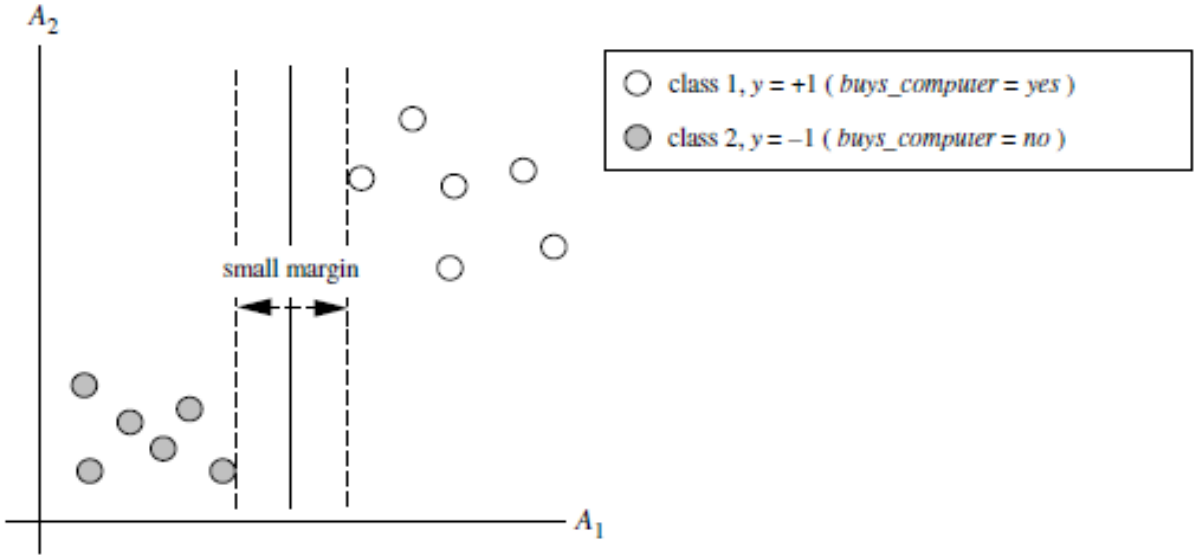
- The term "hyperplane" to refer to the decision boundary

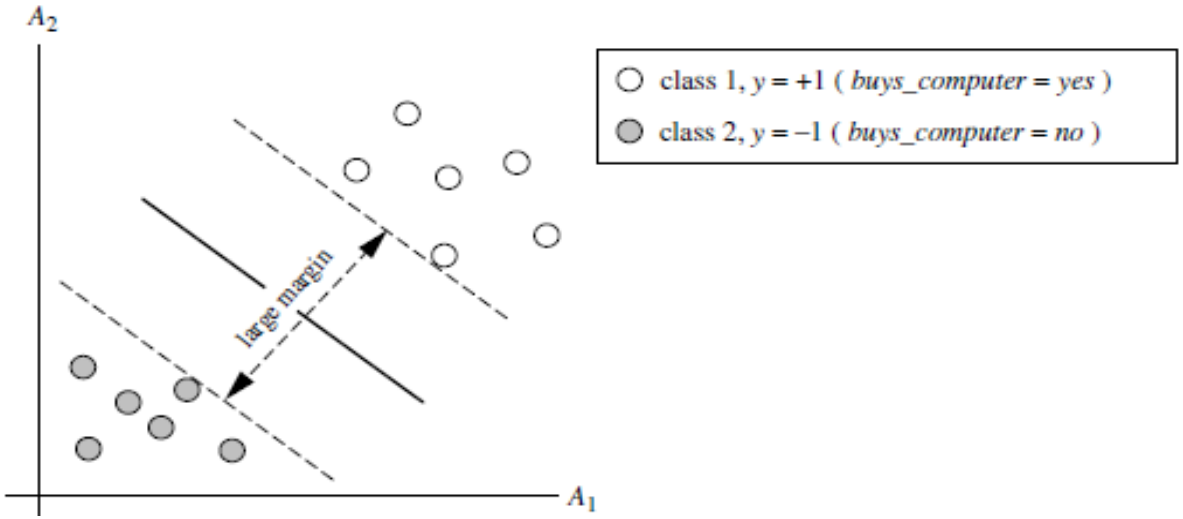
- SVM uses maximum marginal hyperplane.
- The hyperplane with the larger margin to be more accurate at classifying future data tuples than the hyperplane with the smaller margin.
- the SVM searches for the hyperplane with the largest margin, that is, the maximum marginal hyperplane (MMH).
- A separating hyperplane can be written as
- W.X+b = 0;



- Where $\boldsymbol{W}$ is a weight vector $\boldsymbol{W} = \{w1, w2, : : : , wn\}$
- n is the number of attributes;
- b is a scalar
- let's consider two input attributes, $A1$ and $A2$, Training tuples are 2-D, e.g., $\boldsymbol{X} = (x1, x2)$,
- where $x1$ and $x2$ are the values of attributes $A1$ and $A2$, respectively, for $\boldsymbol{X}$.

$$w_0 + w_1 x_1 + w_2 x_2 = 0.$$


A_2
small margin
A_1
class 1, y = +1 (buys_computer = yes)
class 2, y = –1 (buys_computer = no)

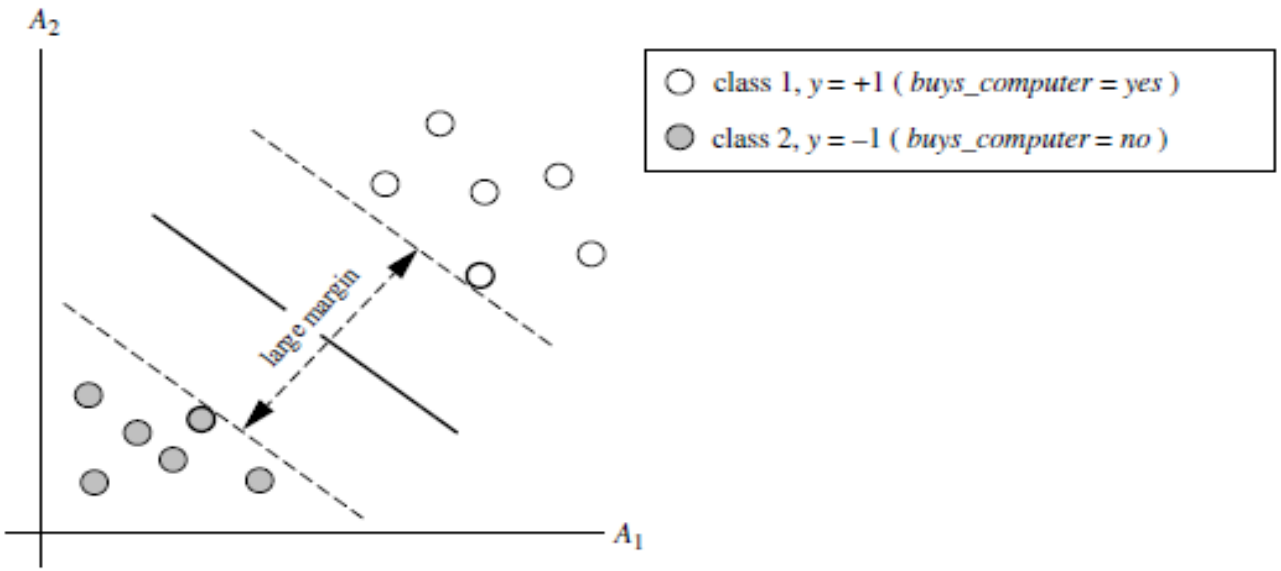

A_2
large margin
A_1
class 1, y = +1 (buys_computer = yes)
class 2, y = –1 (buys_computer = no)

Thus, any point that lies above the separating hyperplane satisfies

$$w_0 + w_1 x_1 + w_2 x_2 > 0.$$

Similarly, any point that lies below the separating hyperplane satisfies

$$w_0 + w_1 x_1 + w_2 x_2 < 0.$$



Support vectors. The SVM finds the maximum separating hyperplane, that is, the one with maximum distance between the nearest training tuples. The support vectors are shown with a thicker border.

The weights can be adjusted so that the hyperplanes defining the "sides" of the margin can be written as

$$H_1 : w_0 + w_1 x_1 + w_2 x_2 \geq 1 \text{ for } y_i = +1, \text{ and}$$
$$H_2 : w_0 + w_1 x_1 + w_2 x_2 \leq -1 \text{ for } y_i = -1.$$

That is, any tuple that falls on or above H_1 belongs to class $+1$, and any tuple that falls on or below H_2 belongs to class -1.

$$y_i(w_0 + w_1 x_1 + w_2 x_2) \geq 1, \ \forall i.$$

Any training tuples that fall on hyperplanes $H1$ or $H2$ (i.e., the "sides" defining the margin) satisfy Equation are called support vectors.

Formulae for the size of the maximal margin

The distance from the separating hyperplane to any point on $H1$ is $\frac{1}{||W||}$

where $||W||$ is the Euclidean norm of $\boldsymbol{W}$, that is $\sqrt{w.w}$.

- Based on the **Lagrangian formulation** the MMH can be rewritten as the decision boundary

$$d(\boldsymbol{X}^T) = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{X}_i \boldsymbol{X}^T + b_0,$$

- where y_i is the class label of support vector x_i
- x^T is a test tuple
- α_i and b_0 are numeric parameters that were determined automatically by the optimization or SVM algorithm.
- l is the number of support vectors.

Prediction

- Regression analysis can be used to model the relationship between one or more *independent* or predictor variables and a *dependent* or response variable.

Linear Regression

- Straight-line regression analysis involves a response variable, *y*, and a single predictor variable, *x*. It is the simplest form of regression, and models *y* as a linear function of *x*.

$$y = b + wx,$$

- where the variance of y is assumed to be constant, and b and w are regression coefficients specifying the Y-intercept and slope of the line, respectively.

$$y = w_0 + w_1 x.$$

- These coefficients can be solved for by the method of least squares, which estimates the best-fitting straight line as the one that minimizes the error between the actual data and the estimate of the line.
- D - training set
- x - values of predictor variable.
- y - associated values for response variable.
- The training set contains |D| data points of the form(x1, y1), (x2, y2), : : : , (x|D|, y|D|).

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

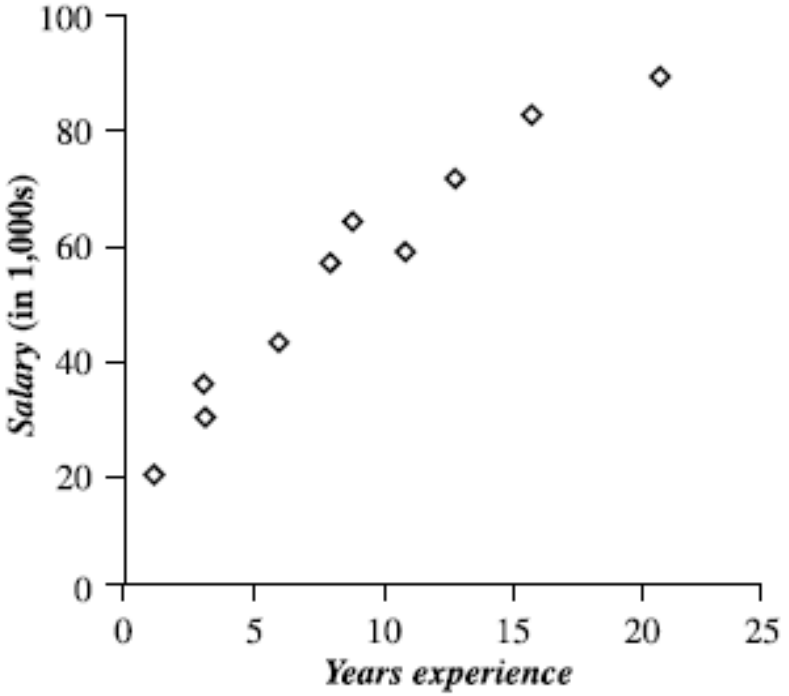
$$w_0 = \bar{y} - w_1 \bar{x}$$

- where x is the mean value of $x1, x2, : : : , x|D|$, and y is the mean value of $y1, y2, : : : , y|D|$.

Straight-line regression using the method of least squares

Salary data.

x years experience	*y salary (in $1000s)*
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83



$$\bar{x} = 9.1 \text{ and } \bar{y} = 55.4$$

$$w_1 = \frac{(3-9.1)(30-55.4)+(8-9.1)(57-55.4)+\cdots+(16-9.1)(83-55.4)}{(3-9.1)^2+(8-9.1)^2+\cdots+(16-9.1)^2} = 3.5$$

$$w_0 = 55.4-(3.5)(9.1) = 23.6$$

$$y = w_0 + w_1 x.$$

$$y = 23.6 + 3.5x.$$

Using this equation, we can predict that the salary of a college graduate with, say, 10 years of experience is $58,600.

Multiple linear regression

- Extension of straight-line regression so as to involve more than one predictor variable.
- multiple linear regression model based on two predictor attributes or variables, $A1$ and $A2$, is

$$y = w_0 + w_1 x_1 + w_2 x_2$$

- where x1 and x2 are the values of attributes A1 and A2, respectively, in X.

Nonlinear Regression

- Polynomial regression is often of interest when there is just one predictor variable.
- It can be modeled by adding polynomial terms to the basic linear model.
- By applying transformations to the variables, we can convert the nonlinear model into a linear one that can then be solved by the method of least squares.

Transformation of a polynomial regression model to a linear regression model

- Consider a cubic polynomial relationship given by

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

To convert this equation to linear form, we define new variables:

$$x_1 = x \qquad x_2 = x^2 \qquad x_3 = x^3$$

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Other Regression-Based Methods

Can it also be used to predict categorical labels?

- **Generalized linear models** - the variance of the response variable, *y*, is a function of the mean value of *y*, unlike in linear regression, where the variance of *y* is constant

- **Logistic regression** - the probability of some event occurring as a linear function of a set of predictor variables.

- Count data frequently exhibit a Poisson distribution and are commonly modeled using **Poisson regression.**

- **Log-linear models** - approximate *discrete* multidimensional probability distributions.
- They may be used to estimate the probability value associated with data cube cells.

- Decision tree induction can be adapted so as to predict continuous (ordered) values, rather than class labels.

- There are two main types of trees for prediction—
  regression trees
  model trees.

- **Regression trees -** were proposed as a component of the CART learning system.
- Each regression tree leaf stores a **continuous-valued prediction**, which is actually the average value of the predicted attribute for the training tuples that reach the leaf.
- **Model trees** - each leaf holds a regression model—a multivariate linear equation for the predicted attribute.

- Regression and model trees tend to be more accurate than linear regression.