Unit 1.5 - Classification

What Is Classification?

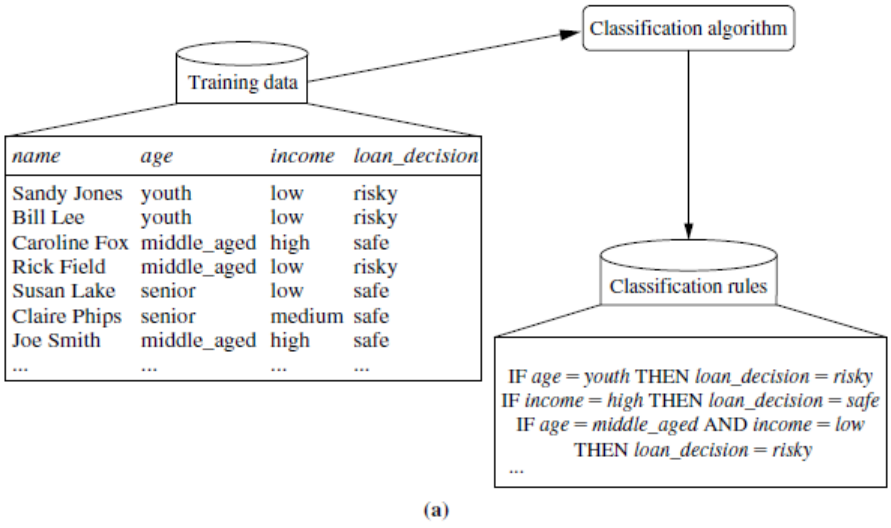
Classification is a form of data analysis that extracts models describing important data classes. Such models, called classifiers, predict categorical class labels.

Ex: A bank loans officer needs analysis of her data 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. In each of these examples, the data analysis task is classification, where a model or classifier is constructed to predict class (categorical) labels, such as “safe” or “risky” for the loan application data; “yes” or “no” for the marketing data;

General Approach to Classification

How does classification work?

Data classification is a two-step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data). The process is shown for the loan application data in below figure.



First step:

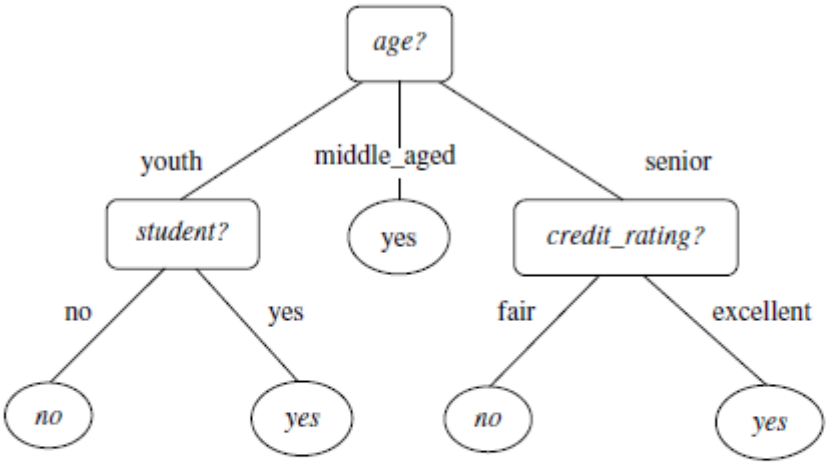
- A classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels.
- A tuple, X, is represented by an n-dimensional attribute vector, X= (x1, x2, … , xn), depicting n measurements made on the tuple from n database attributes A1, A2,…, An.
- Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute.
- The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples and are randomly sampled from the database under analysis.
- The classification process can also be viewed as the learning of a mapping or function, y=f(X), that can predict the associated class label y of a given tuple X

Second step:

- In the second step the model is used for classification.
- The predictive accuracy of the classifier is estimated using test set. The test set is made up of test tuples and their associated class labels.
- The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

Decision Tree Induction

- Decision tree induction is the learning of decision trees from class-labelled training tuples.
- A decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, each leaf node holds a class label. The topmost node in a tree is the root node.



A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

How are decision trees used for classification?

- Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree.
- A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

The algorithm strategy is as follows.

- The algorithm is called with three parameters: D, attribute list, and Attribute selection method.
- The D is a data partition. Initially, it is the complete set of training tuples and their associated class labels.
- The parameter attribute list is a list of attributes describing the tuples.
- Attribute selection method specifies a procedure for selecting the attribute that "best" discriminates the given tuples according to class.
- The tree starts as a single node, N, representing the training tuples in D (step 1).

Algorithm

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;
```

- If the tuples in D are all of the same class, then node N becomes a leaf and is labelled with that class (steps 2 and 3).
- Note that steps 4 and 5 are terminating conditions.
- The algorithm calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the "best" way to separate or partition the tuples in D into individual classes (step 6).
- The node N is labeled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node N for each of the outcomes of the splitting criterion.
- The tuples in D are partitioned accordingly (steps 10 to 11)
- If there are no tuples for a given branch, that is, a partition Dj is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).
- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, Dj , of D (step 14).
- The resulting decision tree is returned (step 15).

Bayes Classification Methods

What are Bayesian classifiers?

- Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.
- Bayesian classification is based on Bayes' theorem

Bayes' Theorem

Let H be some hypothesis such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine P(H|X), the probability that the hypothesis H holds given the "evidence" or observed data tuple X.

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

- P(H|X), is the posterior probability, or a posteriori probability, of H conditioned on X.
- P(X|H) is the posterior probability of X conditioned on H or likelihood probability
- P(H) is the priori probability of H.
- P(X) is the prior probability of X.

Naive Bayesian Classification

The naive 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, $X = (x_1, x_2, \ldots, x_n)$, 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, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

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

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

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

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ needs to 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 $P(X|C_i)$. Otherwise, we maximize $P(X|C_i)P(C_i)$. 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. Given data sets with many attributes, presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple

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

5. To predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple X is the class C_i if and only if

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

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

Example:

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 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 C_1 correspond to the class *buys_computer* = *yes* and C_2 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 $P(\boldsymbol{X}|C_i)P(C_i)$, for $i = 1, 2$. $P(C_i)$, 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 $P(\boldsymbol{X}|C_i)$, 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 these probabilities, we obtain

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

Similarly,

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

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

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

Therefore, the naïve Bayesian classifier predicts *buys_computer* = *yes* for tuple $\boldsymbol{X}$.

Rule-Based Classification

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

IF *condition* THEN *conclusion.*

An example is rule *R*1,

*R*1: IF *age* = *youth* AND *student* = *yes* THEN *buys_computer* = *yes.*

The "IF" part (or left side) of a rule is known as the **rule antecedent** or **precondition.** The "THEN" part (or right side) is the **rule consequent.** In the rule antecedent, the condition consists of one or more *attribute tests* (e.g., *age* = *youth* and *student* = *yes*) that are logically ANDed. The rule's consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer). *R*1 can also be written as

$$R1: (age = youth) \wedge (student = yes) \Rightarrow (buys\_computer = yes).$$

A rule R can be assessed by its coverage and accuracy. Given a tuple, $\boldsymbol{X}$, from a class-labeled data set, D, let n_{covers} be the number of tuples covered by R; $n_{correct}$ be the number of tuples correctly classified by R; and $|D|$ be the number of tuples in D. We can define the **coverage** and **accuracy** of R as

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

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

Rule Induction Using a Sequential Covering Algorithm

IF-THEN rules can be extracted directly from the training data using a sequential covering algorithm.

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
(1)  Rule_set = {}; // initial set of rules learned is empty
(2)  for each class c do
(3)      repeat
(4)          Rule = Learn_One_Rule(D, Att_vals, c);
(5)          remove tuples covered by Rule from D;
(6)          Rule_set = Rule_set + Rule; // add new rule to rule set
(7)      until terminating condition;
(8)  endfor
(9)  return Rule_Set;
```

Basic sequential covering algorithm.