ECLT5810/SEEM5750
Practical Considerations for Classification Learning

Feature Selection

- Can improve generalization (performance)
  - Eliminates noise features
  - Avoids overfitting
- Reduces training time
  - Training time for some methods is quadratic or worse in the number of features

Feature selection

- Some ideas:
  - Hypothesis testing statistics:
    - Are we confident that the value of one categorical variable is associated with the value of another?
    - Chi-square test (χ^2)
  - Information theory:
    - How much information does the value of one categorical variable give you about the value of another?
    - Mutual information
  - Wrapper method

χ^2 Statistic (CHI) for Categorical Features

- $\chi 2$ is interested in $(f_o - f_e)^2/f_e$ summed over all table entries: is the observed number what you'd expect given the marginals?

$$\chi^2(Feature) = \sum (O-E)^2/E = (2-.25)^2/.25 + (3-4.75)^2/4.75$$
$$+ (500-502)^2/502 + (9500-9498)^2/9498 = 12.9 \ (p < .001)$$

- The null hypothesis is rejected with confidence .999, since 12.9 > 10.83 (the value for .999 confidence).
- Higher $\chi 2$ values imply higher dependency among the feature and the class

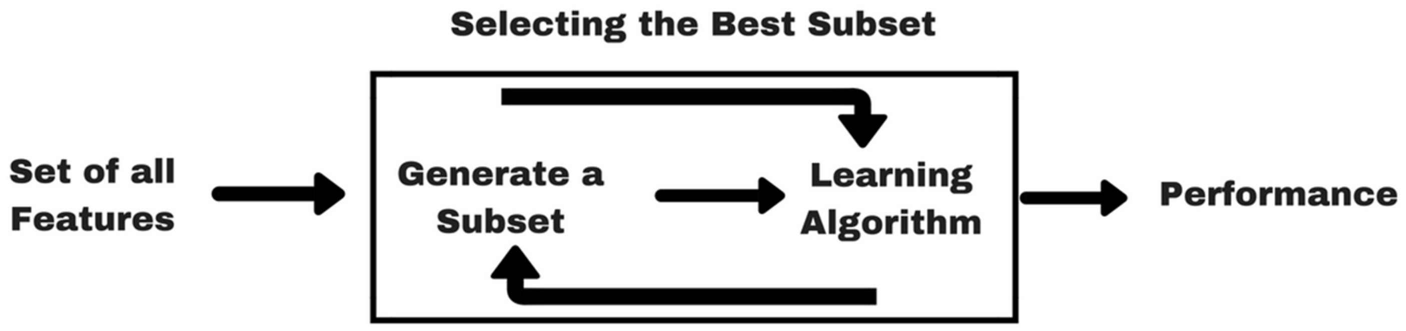
(5/10005)*(502/10005)*10005 = 0.25

	Feature=1	*Feature=0*	
Class = auto	2 *(0.25)*	500 *(502)*	502
Class ≠ auto	3 *(4.75)*	9500 *(9498)*	9503
	5	10000	

expected: f_e

observed: f_o

Wrapper Method



- Use a subset of features and train a model
- Based on the classification performance, we add or remove features from the subset.
- Forward Selection
  - Start with no feature. In each iteration, add the feature which best improves the model
- Backward Elimination
  - Start with all features. In each iteration, remove the feature which best improves the model

The Basic Decision Tree Induction Method - Revisited

Algorithm: **Generate_decision_tree.** Generate a decision tree from the given training data.

Input: The training samples, *samples*, represented by discrete-valued attributes; the set of candidate ~~attribu~~ attributes *attribute-list*.

Output: A decision tree.

Method:

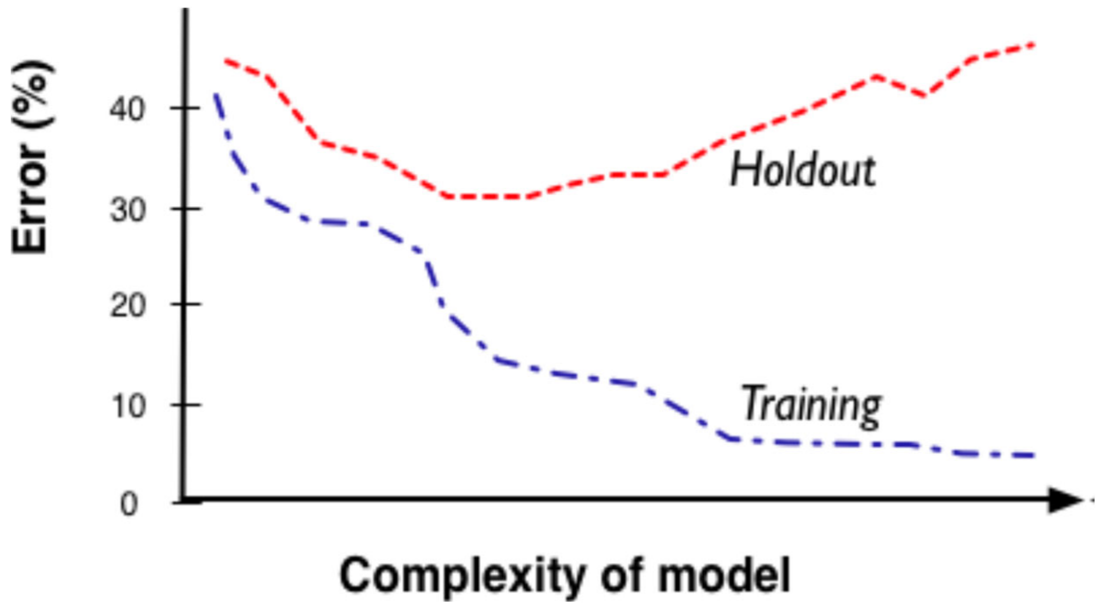
```
(1)   create a node $N$;
(2)   if samples are all of the same class, $C$ then
(3)       return $N$ as a leaf node labeled with the class $C$;
(4)   if attribute-list is empty then
(5)       return $N$ as a leaf node labeled with the most common class in samples; // majority voting
(6)   select test-attribute, the attribute among attribute-list with the highest information gain;
(7)   label node $N$ with test-attribute;
(8)   for each known value $a_i$ of test-attribute // partition the samples
(9)       grow a branch from node $N$ for the condition test-attribute $= a_i$;
(10)      let $s_i$ be the set of samples in samples for which test-attribute $= a_i$; // a partition
(11)      if $s_i$ is empty then
(12)          attach a leaf labeled with the most common class in samples;
(13)      else attach the node returned by Generate_decision_tree($s_i$, attribute-list–test-attribute);
```

Practical Considerations for Decision Tree Induction

- For a method to be useful in a wide range of real-world applications it should be robust in the presence of noise.
- The basic decision tree induction method needs to be refined.
- For example, it stops growing a branch when the majority of instances have the same label.
  - Change Line (2) in the basic algorithm to

    "if (the majority, denoted as A%, of the samples are of the same class label) then"
  - This is also called pre-pruning

Overfitting



- When the model is too simple, it is not very accurate.
- As the model gets complex, the accuracy improves.
- But when the model gets too complex, it looks very accurate on the training data, but in fact it is overfitting
- Overfitting – training accuracy diverges from the holdout (test) accuracy

Revisiting Tree Pruning

- When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers.
- Tree pruning methods address this problem of overfitting the data.
- Such methods typically use statistical measures to remove the least-reliable branches.
- An example of fully grown (unpruned) tree is shown below:

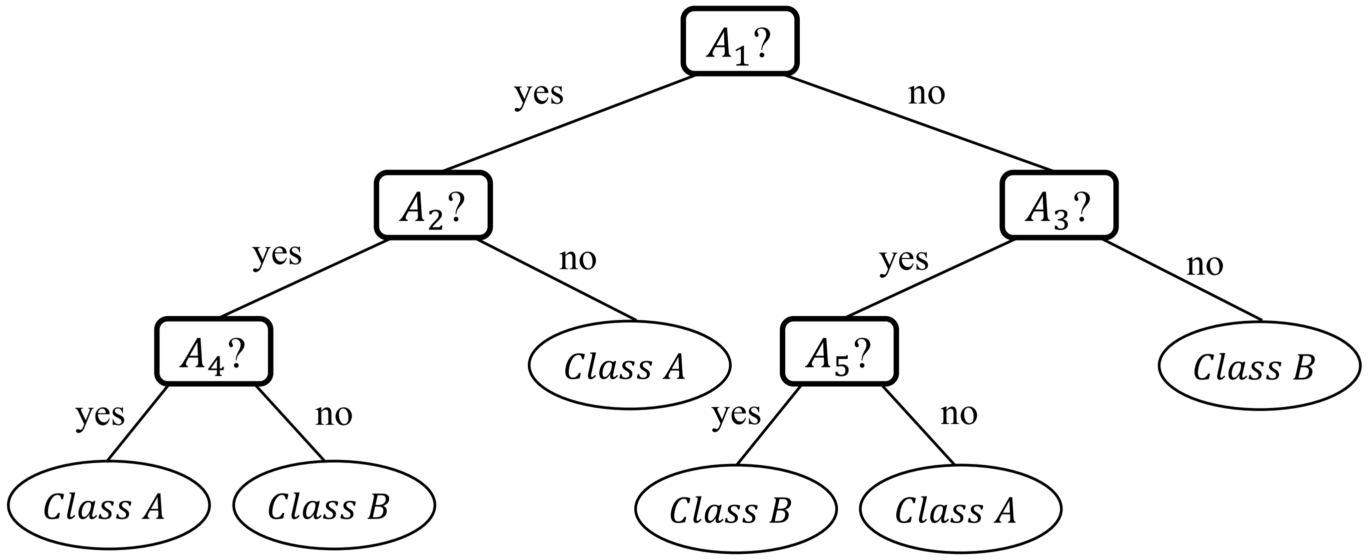


Figure 8.6 (a): An unpruned decision tree.

Revisiting Tree Pruning

- If a pruned tree is appropriately constructed in the learning process, it can avoid overfitting and achieve a better prediction performance.
- Pruned trees tend to be smaller and less complex and, thus, easier to comprehend.
- An example of a pruned version of the tree in the previous page is shown below:

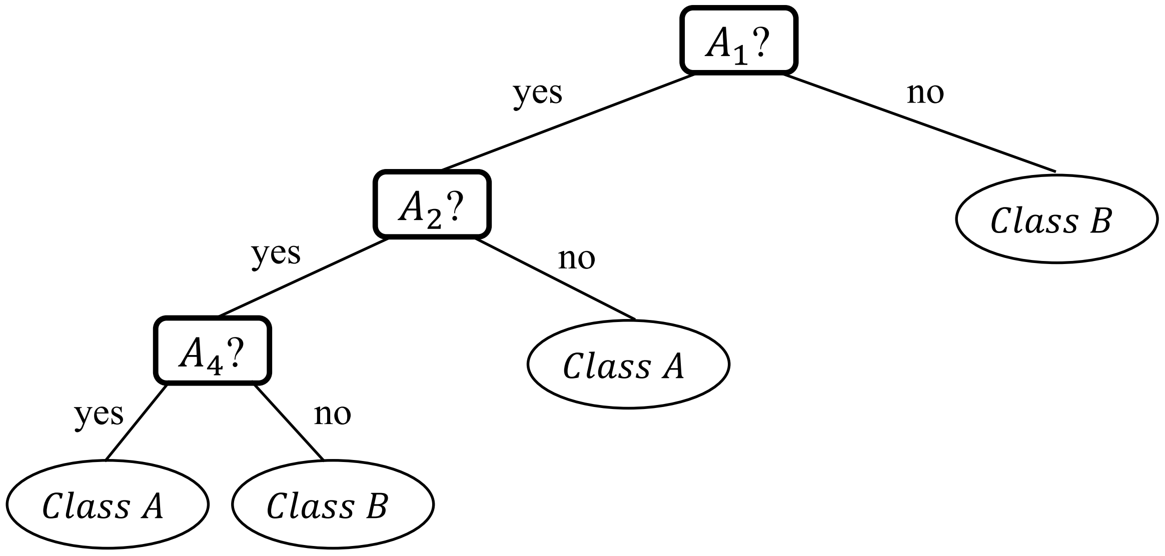


Figure 8.6 (b): The pruned version of Figure 8.6 (a).

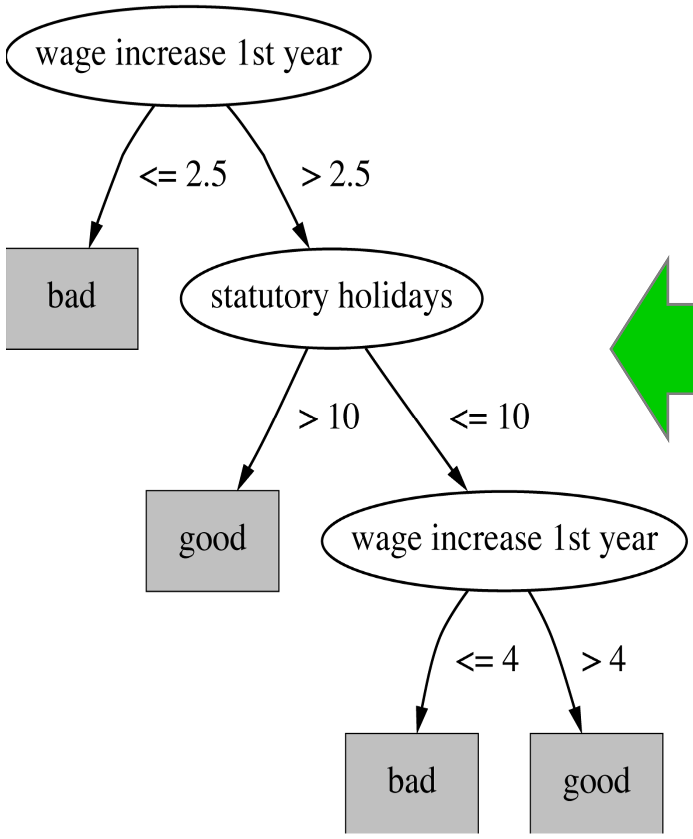
Tree Pruning - Prepruning

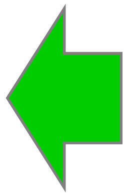
- In general, there are two common approaches to tree pruning
  - prepruning and postpruning
- In the **prepruning** approach, a tree is “pruned” by halting its construction early
  - For example, by deciding not to further split or partition the subset of training tuples at a given node).
  - Upon halting, the node becomes a leaf.
  - The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.
- When constructing a tree, measures such as statistical significance and information gain, can be used to assess the goodness of a split.
- If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted.
- There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

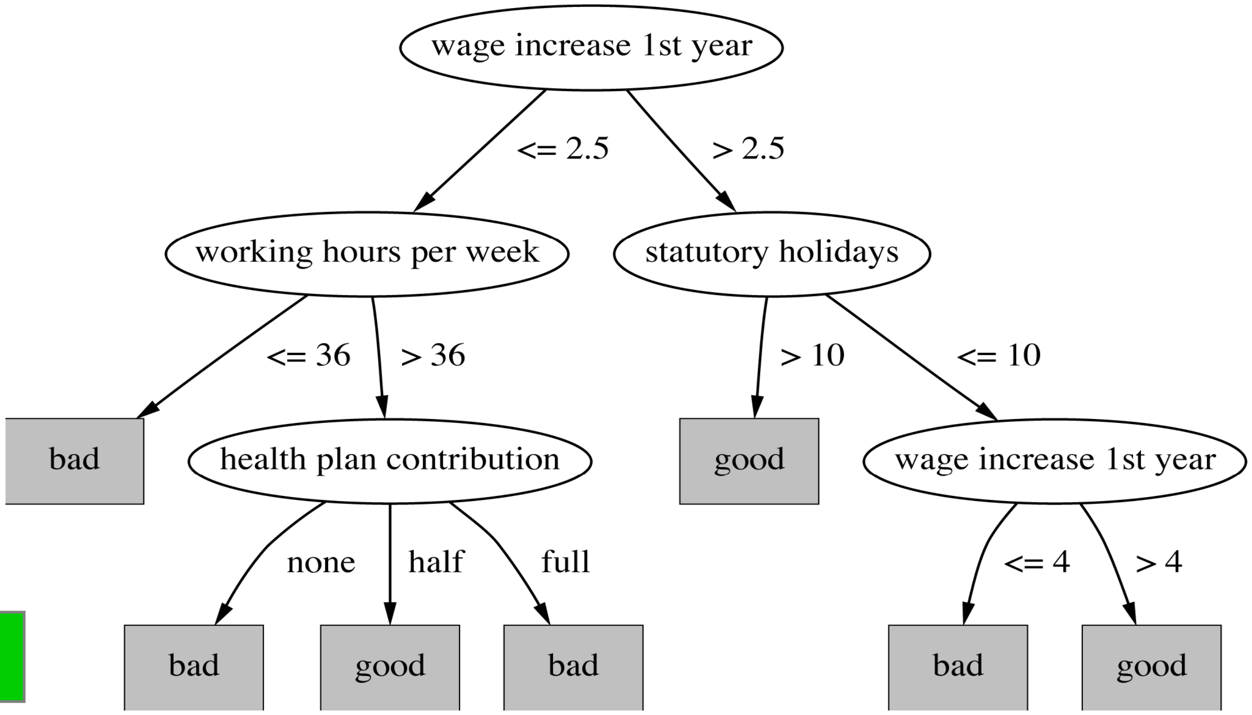
Tree Pruning - Postpruning

- The second and more common approach is **postpruning**, which removes subtrees from a "fully grown" tree.
  - A subtree at a given node is pruned by removing its branches and replacing it with a leaf.
  - The leaf is labeled with the most frequent class among the subtree being replaced.
  - In the previous example, notice the subtree at node "A_3?" in the unpruned tree of Figure 8.6 (a). Suppose that the most common class within this subtree is "$class\ B$." In the pruned version of the tree, the subtree in question is pruned by replacing it with the leaf "$class\ B$."

Another Example of Postpruning







fully-grown tree

pruned tree

Tree Pruning - Postpruning

- The **cost complexity** pruning algorithm used in CART is an example of the postpruning approach.
  - This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree.
    - where the **error rate** is the percentage of tuples misclassified by the tree.
  - It starts from the bottom of the tree. For each internal node, N, it computes the cost complexity of the subtree at N, and the cost complexity of the subtree at N if it were to be pruned (i.e., replaced by a leaf node).
  - The two values are compared. If pruning the subtree at node N would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept.
- A **pruning set** of class-labeled tuples is used to estimate cost complexity. This set is independent of the training set used to build the unpruned tree and of any test set used for accuracy estimation.
- The algorithm generates a set of progressively pruned trees. In general, the smallest decision tree that minimizes the cost complexity is preferred.

Tree Pruning - Postpruning

- C4.5 uses a method called **pessimistic pruning**, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning.
  - Pessimistic pruning uses the training set to estimate error rates.
    - It does not require the use of a pruning set.
  - Recall that an estimate of accuracy or error based on the training set is overly optimistic and, therefore, strongly biased.
  - The pessimistic pruning method therefore adjusts the error rates obtained from the training set by adding a penalty, so as to counter the bias incurred.

Revisiting Tree Pruning

- Alternatively, prepruning and postpruning may be interleaved for a combined approach.
- Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree.
- No single pruning method has been found to be superior over all others.
- Although some pruning methods do depend on the availability of additional data for pruning, this is usually not a concern when dealing with large databases.
- Although pruned trees tend to be more compact than their unpruned counterparts, they may still be rather large and complex.
- Decision trees can suffer from *repetition* and *replication* (Figure 8.7), making them overwhelming to interpret.

Revisiting Tree Pruning

- **Repetition** occurs when an attribute is repeatedly tested along a given branch of the tree (e.g., "$age < 60?$," followed by "$age < 45?$," and so on).

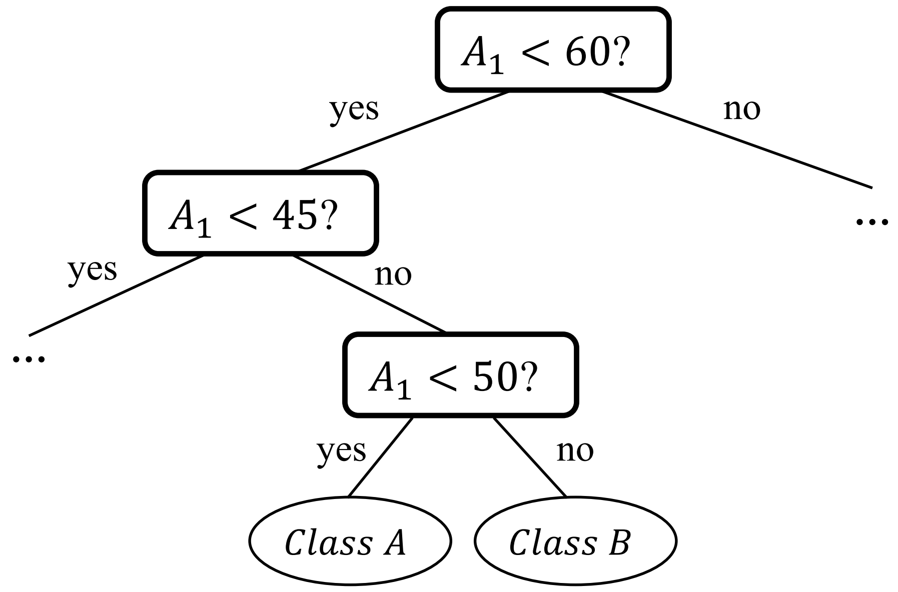


Figure 8.7 (a): An example of subtree **repetition**, where an attribute is repeatedly tested along a given branch of the tree (e.g., age).

Revisiting Tree Pruning

- In **replication**, duplicate subtrees exist within the tree. These situations can impede the accuracy and comprehensibility of a decision tree. The use of multivariate splits (splits based on a combination of attributes) can prevent these problems.

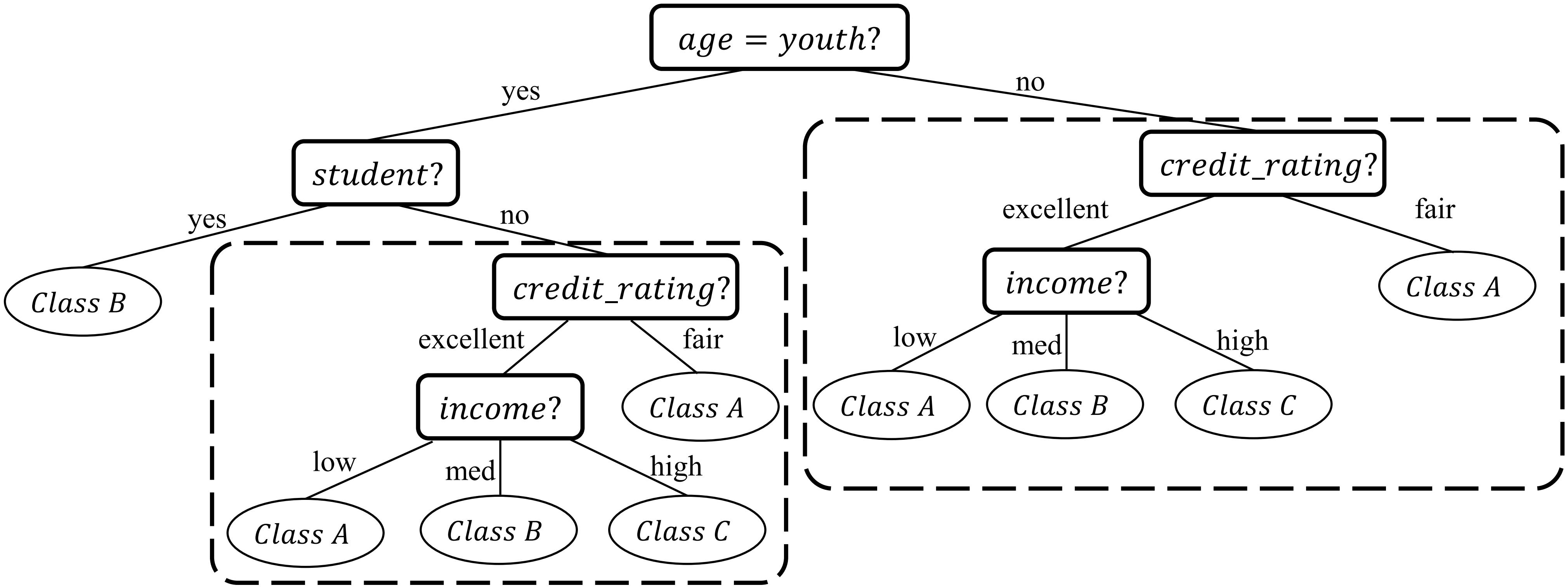


Figure 8.7 (b): An example of subtree **replication**, where duplicate subtrees exist within a tree (e.g., the subtree headed by the node "*credit_rating*?".

Validation data set

- It is important that the test data is not used in any way to create the classifier
- Sometimes, as part of the training process, we wish:
  - to choose or compare different model parameter values of the same data mining model (parameter tuning)
  - to choose or compare different variants of the same method
  - to choose or compare different methods
- The test data can’t be used for such purpose
- Proper procedure uses three sets: training data, validation data, and test data

Validation data set – for parameter tuning

- Validation data set can also be used for parameter tuning
- Different variants of the same method refer to same model but with different model parameters
- An example of model parameters is the criterion (i.e. A%) in the pre-pruning of the decision tree induction method.
- The parameter tuning process is regarded as part of the training process.

Validation data set – for parameter tuning

Training data set		Validation data set
Parameter value	Trained model	Error rate
0.7	$M_{0.7}$	0.13
0.8	$M_{0.8}$	0.11
0.9	$M_{0.9}$	0.08
0.95	$M_{0.95}$	0.13

The model $M_{0.9}$ will be the output of the training process

Validation data set – for comparing methods

- Validation data is used for performance comparison of different variants of the classifier during training
- Suppose there are two data mining methods, namely, decision tree and regression.
- Apply decision tree learning on the training data and obtain a model M_1
- Use validation data to evaluate the model M_1 and obtain an error rate r_1
- Apply regression learning on the training data and obtain a model M_2
- Use validation data to evaluate the model M_2 and obtain an error rate r_2
- Select as output the model that gives the smaller error
- Note that the above process is considered as part of the training process