# Artificial Intelligence and Machine Learning

## Information Based Learning

# Big Idea

- Information based machine learning algorithms try to build predictive models using only the most informative features.
- In this context an informative feature is a descriptive feature whose values split the instances in the dataset into homogeneous sets with respect to the target feature value.
- Model Representation:
  - Expert systems
  - Decision trees

# Applications

- Diagnosis
- Making decisions
- Customer propensity prediction

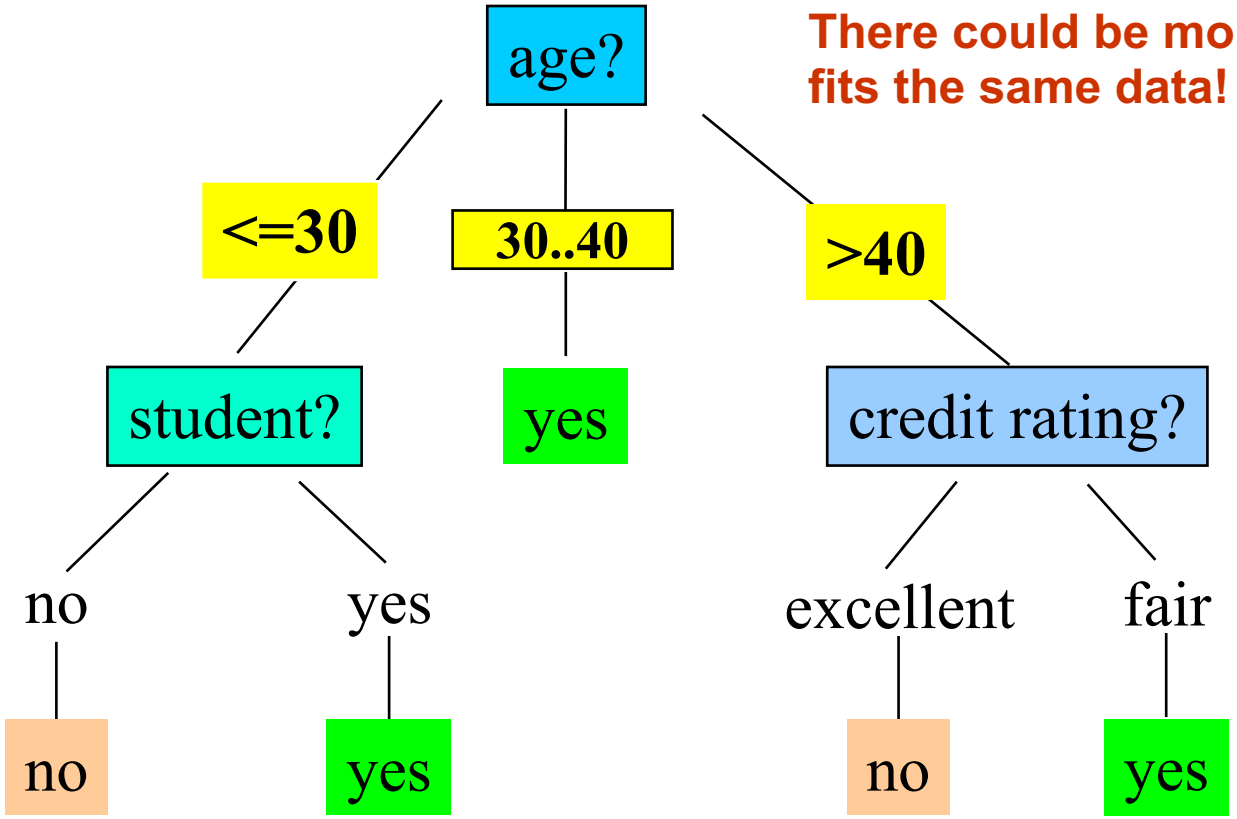
# Decision Tree

- A decision tree consists of:
  - a root node (or starting node),
  - interior nodes
  - and leaf nodes (or terminating nodes).
- Each of the non-leaf nodes (root and interior) in the tree specifies a test to be carried out on one of the query's descriptive features.
- Each of the leaf nodes specifies a predicted classification for the query.

# An Example Training Dataset

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

# Sample Decision Tree




**There could be more than one tree that fits the same data!**

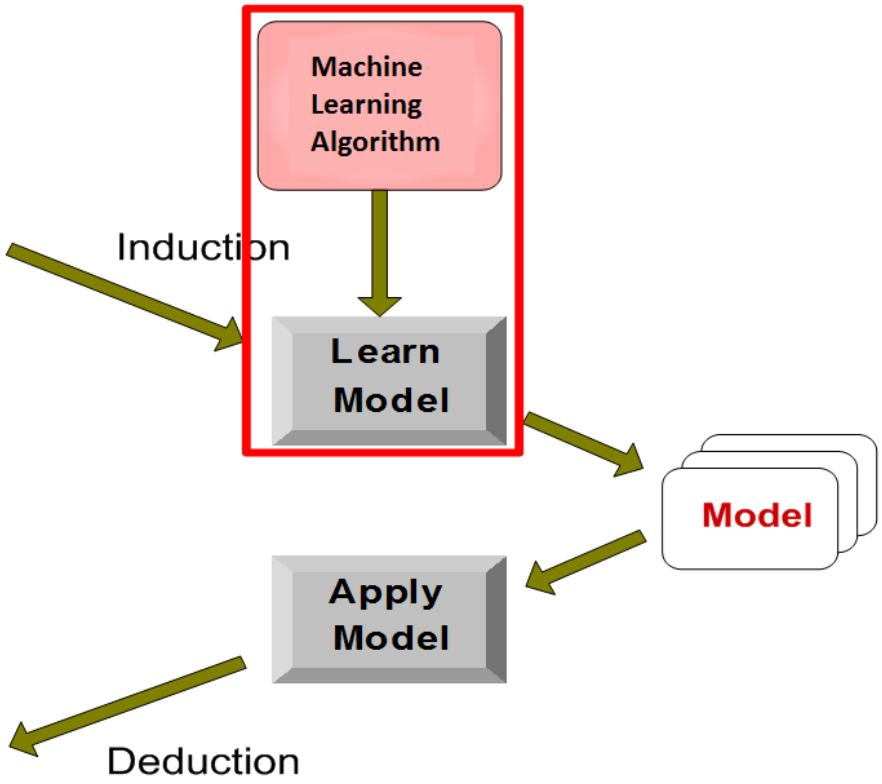
# Decision Tree

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|---|---|---|---|---|
| 1 | Yes | Large | 125K | No |
| 2 | No | Medium | 100K | No |
| 3 | No | Small | 70K | No |
| 4 | Yes | Medium | 120K | No |
| 5 | No | Large | 95K | Yes |
| 6 | No | Medium | 60K | No |
| 7 | Yes | Large | 220K | No |
| 8 | No | Small | 85K | Yes |
| 9 | No | Medium | 75K | No |
| 10 | No | Small | 90K | Yes |

Training Set

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|---|---|---|---|---|
| 11 | No | Small | 55K | ? |
| 12 | Yes | Medium | 80K | ? |
| 13 | Yes | Large | 110K | ? |
| 14 | No | Small | 95K | ? |
| 15 | No | Large | 67K | ? |

Test Set



# Advantages and Limitations

- Simple to understand and interpret
- Uses a white box model
- Performs well with large datasets

- Prone to overfitting
- Not suitable for some concepts, such as XOR
- The problem of learning an optimal decision tree is known to be NP-complete.

# How do we solve the NP-complete problem?

- Use greedy algorithms
- Apply to building decision tree:
  - In each step, choose the attribute that seems to be the "best"
  - "best" -- the attribute that most likely splits the dataset into pure sets with respect to the target feature
  - Result: shallower trees
- Computational metric of the purity of a set
  - Entropy
  - Gini Index
  - Misclassification

# Entropy

- Claude Shannon's entropy model defines a computational measure of the impurity of the elements of a set.
- An easy way to understand the entropy of a set is to think in terms of the uncertainty associated with guessing the result if you were to make a random selection from the set.
- Entropy is related to the probability of an outcome:
  - High probability → Low entropy
  - Low probability → High entropy

# Entropy (II)

- Shannon's model of entropy is a weighted sum of the logs of the probabilities of each of the possible outcomes when we make a random selection from a set.
- Entropy at a given node t:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Measures homogeneity of a node.
    - Maximum (log $n_c$) when records are equally distributed among all classes implying least information
    - Minimum (0.0) when all records belong to one class, implying most information

# Examples for computing Entropy

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

| C1 | **0** |
|---|---|
| C2 | **6** |

**P(C1) = 0/6 = 0  P(C2) = 6/6 = 1**

**Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0**

| C1 | **1** |
|---|---|
| C2 | **5** |

**P(C1) = 1/6  P(C2) = 5/6**

**Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65**

| C1 | **2** |
|---|---|
| C2 | **4** |

**P(C1) = 2/6  P(C2) = 4/6**

**Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92**

# Greedy algorithm that uses entropy

- Our intuition is that the ideal discriminatory feature will partition the data into pure subsets where all the instances in each subset have the same classification.
- One way to implement this idea is to use a metric called information gain.
- The information gain of a descriptive feature can be understood as a measure of the reduction in the overall entropy of a prediction task by testing on that feature.

# Information Gain

- Information Gain:

$$GAIN_{split} = Entropy(p) - \left( \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node, p is split into k partitions;

$n_i$ is number of records in partition i

  - Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
  - Used in ID3
  - Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

# ID3 Algorithm

- Iterative Dichotomizer 3
- Attempts to create the shallowest tree that is consistent with the data that it is given
- The ID3 algorithm builds the tree in a recursive, depth-first manner, beginning at the root node and working down to the leaf nodes.

# ID3 (II)

- The algorithm begins by choosing the best descriptive feature to test (i.e., the best question to ask first) using information gain.
- A root node is then added to the tree and labelled with the selected test feature.
- The training dataset is then partitioned using the test.
- For each partition a branch is grown from the node.
- The process is then repeated for each of these branches using the relevant partition of the training set in place of the full training set and with the selected test feature excluded from further testing.

# Stop condition of ID3

- The algorithm defines three situations where the recursion stops and a leaf node is constructed:
  - All of the instances in the dataset have the same classification (target feature value) then return a leaf node tree with that classification as its label.
  - The set of features left to test is empty then return a leaf node tree with the majority class of the dataset as its classification.
  - The dataset is empty return a leaf node tree with the majority class of the dataset at the parent node that made the recursive call.

# Information Gain Ratio

- Information Gain tends to prefer splits that result in large number of partitions, each being small but pure.
- Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions

$n_i$ is the number of records in partition i

- Adjusts Information Gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5
- Designed to overcome the disadvantage of Information Gain

# Alternative Feature Selection Metrics – GINI index

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

| C1 | **0** |
|---|---|
| C2 | **6** |
| **Gini=0.000** | |

| C1 | **1** |
|---|---|
| C2 | **5** |
| **Gini=0.278** | |

| C1 | **2** |
|---|---|
| C2 | **4** |
| **Gini=0.444** | |

| C1 | **3** |
|---|---|
| C2 | **3** |
| **Gini=0.500** | |

# Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

| | |
|---|---|
| C1 | **0** |
| C2 | **6** |

**P(C1) = 0/6 = 0  P(C2) = 6/6 = 1**

**Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0**

| | |
|---|---|
| C1 | **1** |
| C2 | **5** |

**P(C1) = 1/6  P(C2) = 5/6**

**Gini = 1 – (1/6)² – (5/6)² = 0.278**

| | |
|---|---|
| C1 | **2** |
| C2 | **4** |

**P(C1) = 2/6  P(C2) = 4/6**

**Gini = 1 – (2/6)² – (4/6)² = 0.444**

# Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, $n_i$ = number of records at child i,

n = number of records at node p.

- Information gain can be calculated using Gini index by replacing the entropy measure with the Gini index.

# Alternative Feature Selection Metrics – Misclassification

- Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Measures misclassification error made by a node.
  - Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

# Examples of Misclassification

$$Error(t) = 1 - \max_i P(i \mid t)$$

| C1 | **0** |
|---|---|
| C2 | **6** |

**P(C1) = 0/6 = 0  P(C2) = 6/6 = 1**

**Error = 1 – max (0, 1) = 1 – 1 = 0**

| C1 | **1** |
|---|---|
| C2 | **5** |

**P(C1) = 1/6  P(C2) = 5/6**

**Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6**

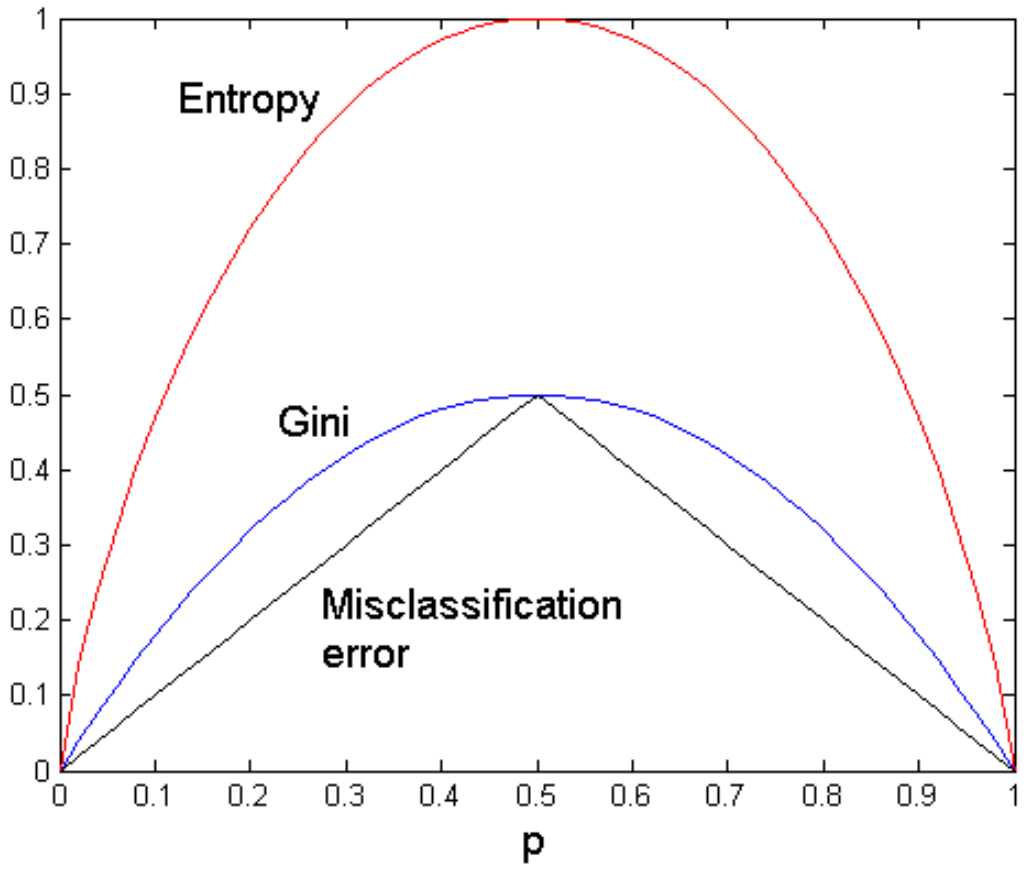
| C1 | **2** |
|---|---|
| C2 | **4** |

**P(C1) = 2/6  P(C2) = 4/6**

**Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3**

# Comparison among Splitting Criteria

**For a 2-class problem:**



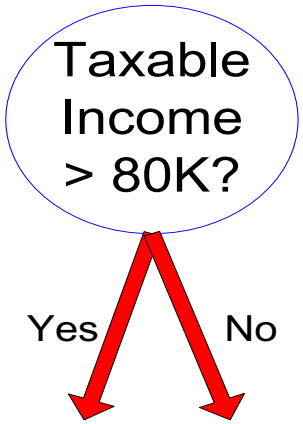
# How to Specify Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

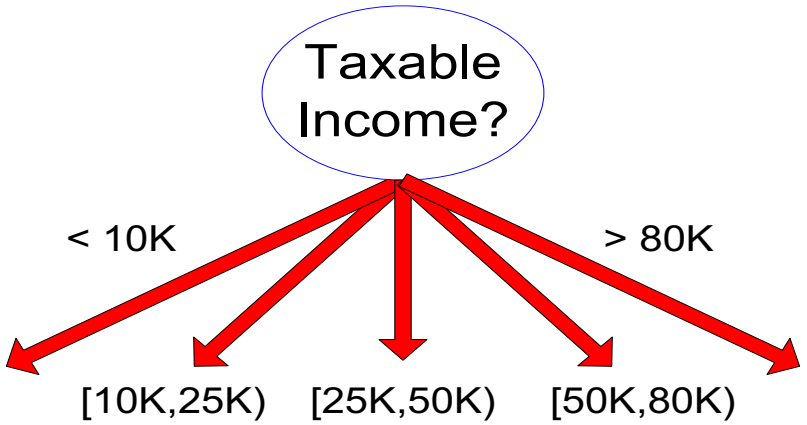
# Handling Continuous Descriptive Features

- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

# Continuous Features Example




(i) Binary split

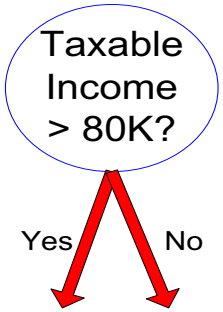



(ii) Multi-way split

# Handling Continuous Descriptive Features Using Binary Decision

- Use Binary Decisions based on a threshold value
- How to choose the threshold value?
  - Number of possible threshold values = Number of distinct values
  - Using Gini index
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A < v and A ≥ v
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |



# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| Cheat | No | No | No | Yes | Yes | Yes | No | No | No | No |
|---|---|---|---|---|---|---|---|---|---|---|
| Taxable Income | | | | | | | | | | |
| Sorted Values → | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |
| Split Positions → | 55 | 65 | 72 | 80 | 87 | 92 | 97 | 110 | 122 | 172 | 230 |

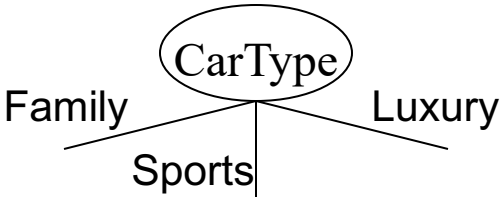
| | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > | <= | > |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Yes | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 1 | 2 | 2 | 1 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| No | 0 | 7 | 1 | 6 | 2 | 5 | 3 | 4 | 3 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | 5 | 2 | 6 | 1 | 7 | 0 |
| Gini | 0.420 | | 0.400 | | 0.375 | | 0.343 | | 0.417 | | 0.400 | | ***0.300*** | | 0.343 | | 0.375 | | 0.400 | | 0.420 | |

# Handling Continuous Descriptive Features

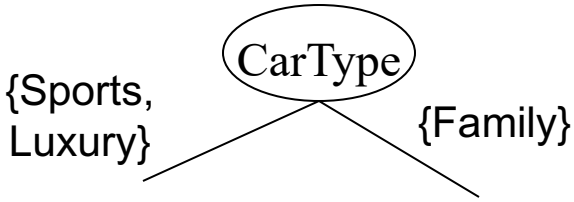
- Once a threshold has been set, the dynamically created new Boolean feature can compete with the other categorical features for selection as the splitting feature at that node.
- This process can be repeated at each node as the tree grows.

# Handling Nominal Features

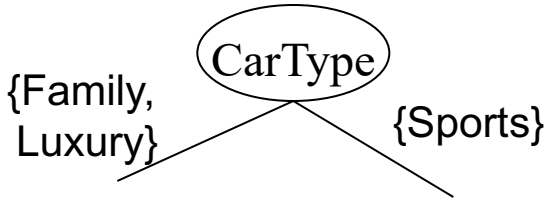
- Multi-way split: Use as many partitions as distinct values.




- Binary split: Divides values into two subsets. Need to find optimal partitioning.




OR



# Predicting Continuous Targets

- If a decision tree is used to predict continuous target, then it is called a regression tree.
- Regression trees are constructed so as to reduce the variance in the set of training examples at each of the leaf nodes.
- The ID3 algorithm can be adapted to use a measure of variance rather than a measure of classification impurity (entropy) when selecting the bet attribute.

# Measure of Variance

- The impurity (variance) at a node can be calculated using the following equation:

$$var(t, D) = \frac{\sum_{i=1}^{n} (t_i - \bar{t})^2}{n - 1}$$

- The best feature to split on at a node is selected as the one that minimizes the weighted variance across the resulting partitions:

$$d[best] = argmin \left( \sum_{l \in levels(d)} \frac{|D_{d=l}|}{|D|} \times var(t, D_{d=l}) \right)$$

# Regression Tree Example

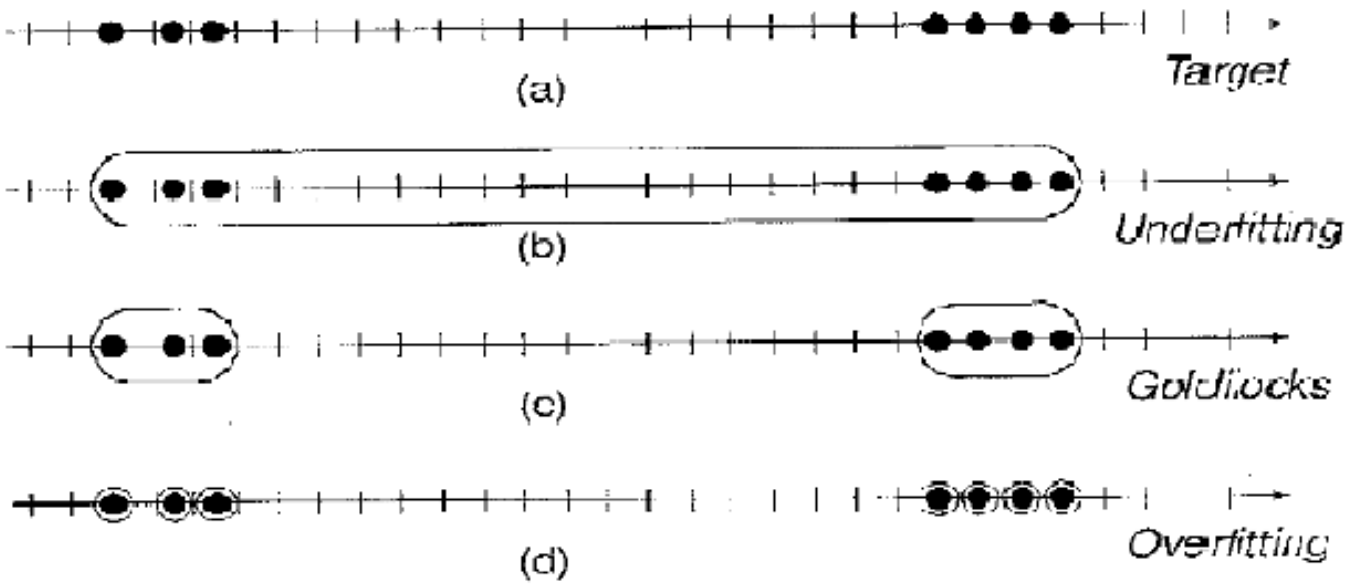



**Figure:** (a) A set of instances on a continuous number line; (b), (c), and (d) depict some of the potential groupings that could be applied to these instances.

# Practical Issues with Decision Trees

- Overfitting --- splitting the data on an irrelevant feature
  - To address the overfitting
    - Pre-pruning
    - Post-pruning
- Under-fitting
- Missing value in training data
- Missing value in query

# Pre-Pruning

- Early Stopping Rule
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using $\chi^2$ test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

# Post-Pruning

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined by majority voting

# Example of Post-Pruning

| | |
|---|---|
| Class = Yes | 20 |
| Class = No | 10 |
| Error = 10/30 | |

**Training Error (Before splitting) = 10/30**

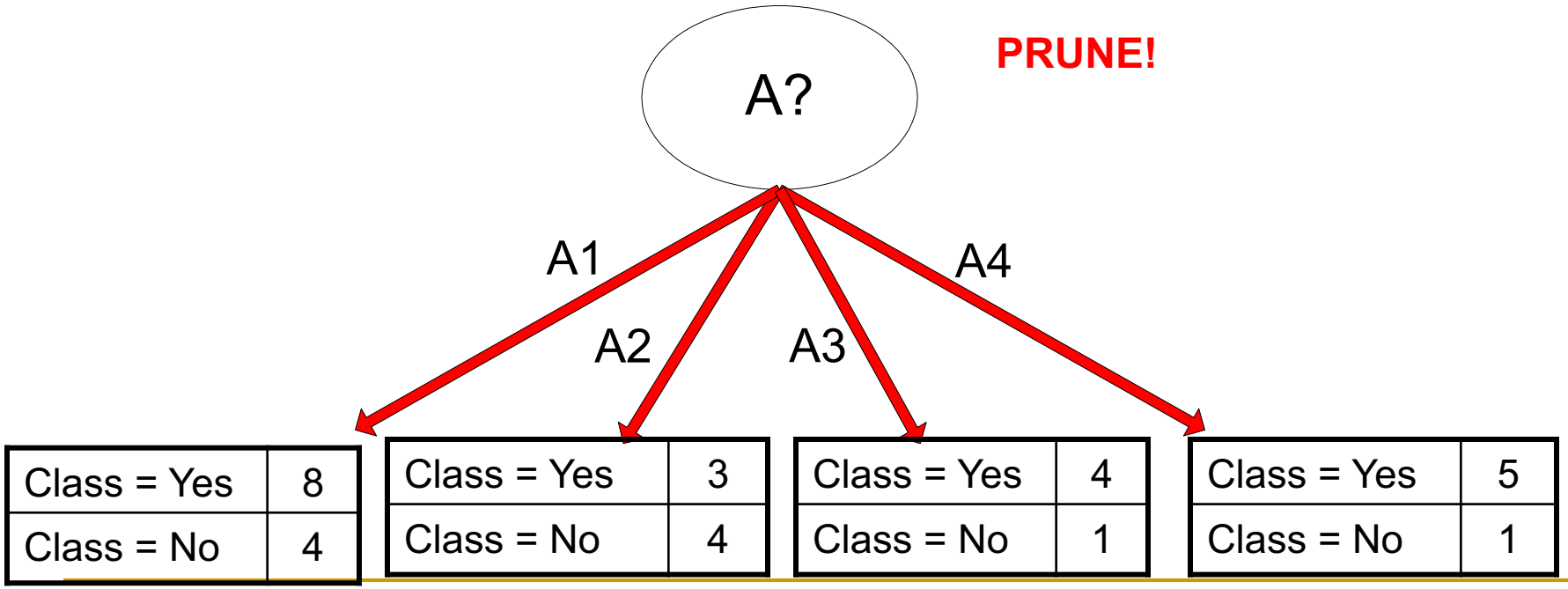
**Pessimistic error = (10 + 0.5)/30 = 10.5/30**

**Training Error (After splitting) = 9/30**

**Pessimistic error (After splitting)**

**= (9 + 4 × 0.5)/30 = 11/30**

**PRUNE!**



# Another Common Post-Pruning Approach

- Using the validation set evaluate the prediction accuracy achieved by both the fully grown tree and the pruned copy of the tree. If the pruned copy of the tree performs no worse than the fully grown tree, then the node is a candidate for pruning.

# Advantages of Pruning

- Smaller trees are easier to interpret
- Increased generalization accuracy when there is noise in the training data (noise dampening)

# Model Ensembles

- Rather than creating a single model, we can generate a set of models and then make predictions by aggregating the outputs of these models.
- A prediction model that is composed of a set of models is called a model ensemble.
- In order for this approach to work the models that are in the ensemble must be different from each other.
- There are two standard approaches to creating ensembles:
  - Boosting
  - bagging

# Boosting

- Boosting works by iteratively creating models and adding them to the ensembles.
- The iteration stops when a predefined number of models have been added.
- Each new model is biased to pay more attention to instances that previous models miss-classified.
- This is done by using a weighted dataset and incrementally adapting the dataset used to train the models.

# Weighted Dataset

- Each instance has an associated weight (>=0)
- Initially set to 1/n where n is the number of instances in the dataset.
- After each model is added to the ensemble, it is tested on the training data and the weights of the instances the model gets correct are decreased and the weights of the instances the model gets incorrect are increased.
- These weights are used as a distribution over which the dataset is sampled to create a replicated training set, where the replication of an instance is proportional to its weight.

# Boosting Ensemble

- The total error of a model, E, is calculated as the sum of the weights of the training instances for which the model predicted wrong.
- The model's confidence factor, A, increases as E decreases, and can be calculated as:

$$A = \frac{1}{2} \times \log_e\left(\frac{1-E}{E}\right)$$

- Once the set of models have been created, the ensemble makes predictions using a weighted aggregate of the predictions made by the individual models.
- The weights used in this aggregation are simply the confidence factors associated with each model.

# Bagging

- Each model in the ensemble using bagging is trained on a random sample of the dataset known as bootstrap samples.
- Each random sample is the same size as the dataset and sampling with replacement is used.
- Consequently, every bootstrap sample will be missing some of the instances from the dataset, so each bootstrap sample will be different . And this means that each model will also be different.
- Further more, each bootstrap sample can use subspace sampling to randomly select a subset of the descriptive features in the dataset.
- The combination of bagging, subspace sampling, and decision trees is know as a random forest model.