U O W

Classification by Splitting Data

– Dive Into ML Model Training

CSCI316: Big Data Mining Techniques and Implementation

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Open the black box of model training…

- Recall the following fragment of the end-to-end project (see page 32 of the "End-to-End Big Data Lifecycle" lecture note:

> - Try Decision Tree
>
> ```
> > from sklearn.tree import DecisionTreeRegressor
> > tree_reg = DecisionTreeRegressor()
> > tree_reg.fit(housing_prepared, housing_labels)
> > housing_predictions = tree_reg.predict(housing_prepared)
> ```

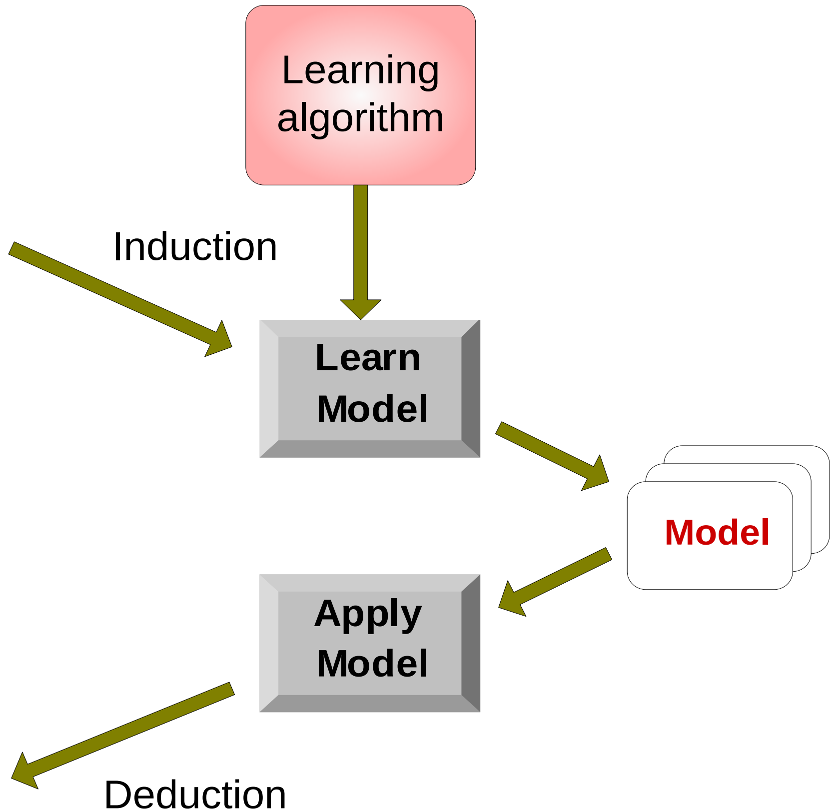
- *What is a DT? How does it work? What is the theory behind?*

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

The Classification Problem: An Example

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

What is a Decision Tree

- A decision tree is a ***flowchart-like tree structure***
  - Each *internal node* (non-leaf node) denotes a test on an attribute
  - Each *branch* (i.e., subtree) represents an outcome of the test
  - Each *leaf node* (or terminal node) holds a class label
- It simulates the process of human decision-marking.
  - Thus, one advantage of decision trees is *understandability*

Example of a Decision Tree

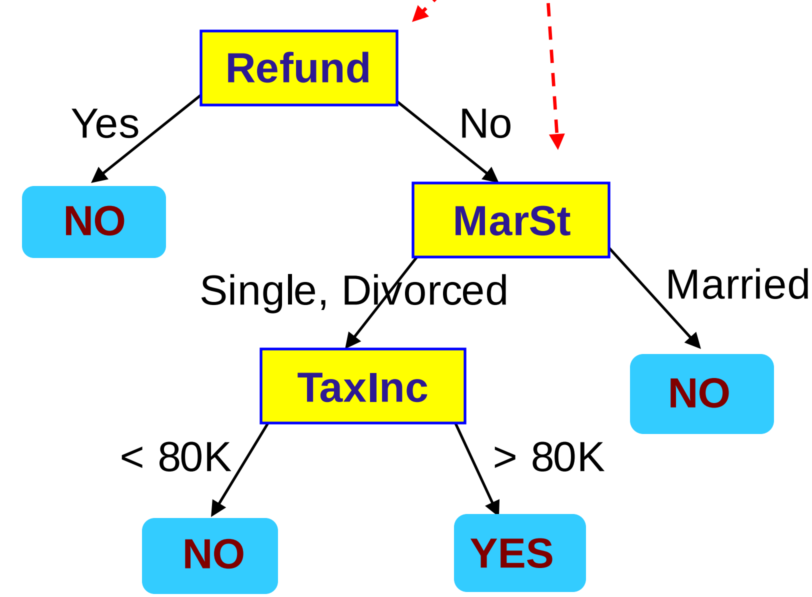
categorical categorical continuous class

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**

Training Data

Each node is associated with a (sub)set of records

Splitting Attributes

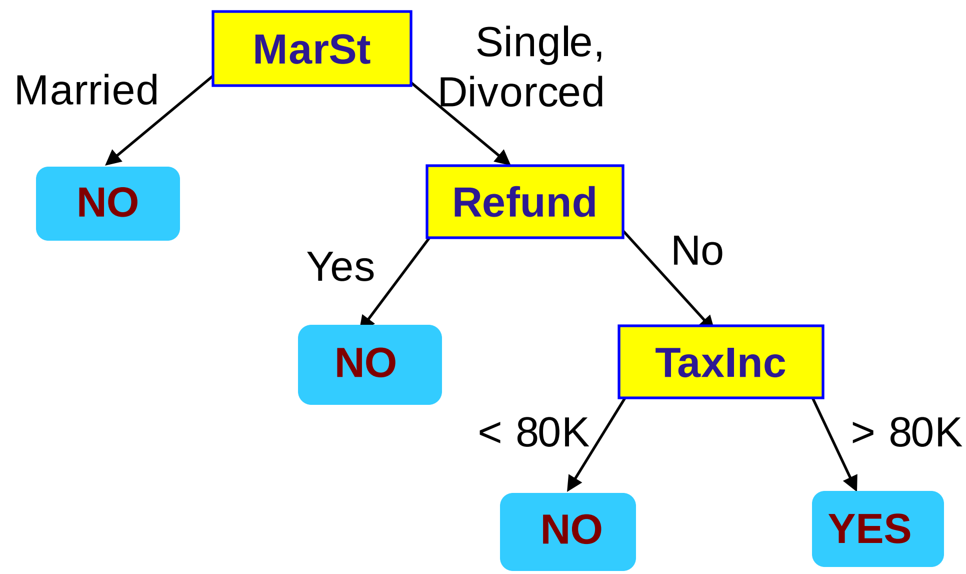


Model: Decision Tree

Another Example of Decision Tree

categorical categorical continuous class

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

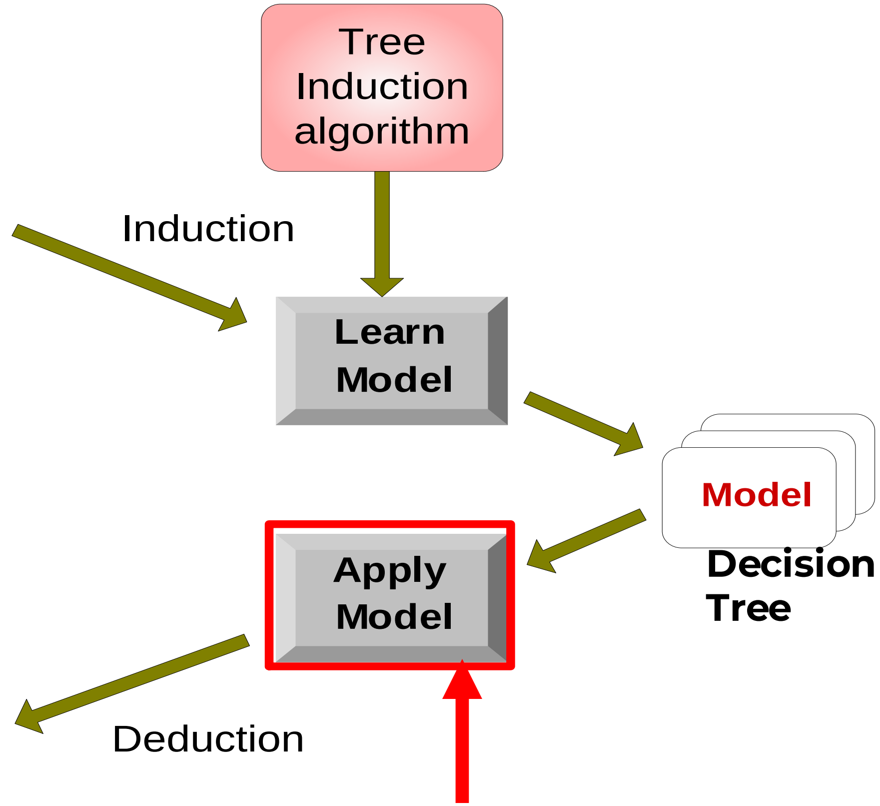


There could be multiple trees that fit the same data!

Decision Tree Classification Task

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

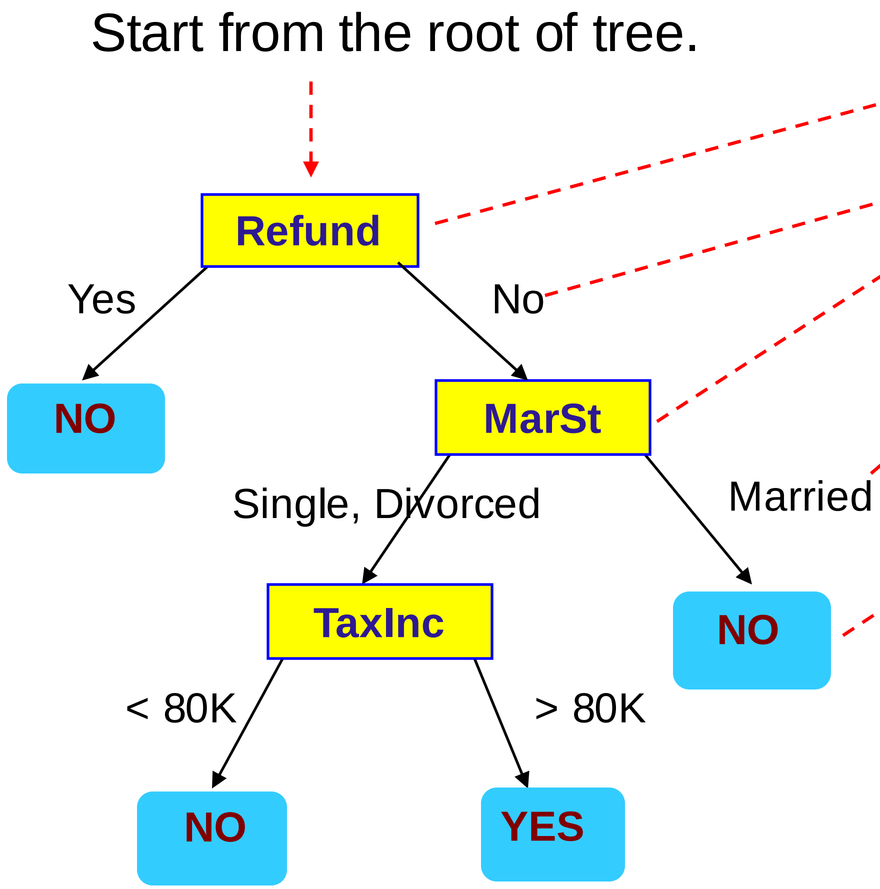
Training Set



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

Apply Model to Test Data



Test Data

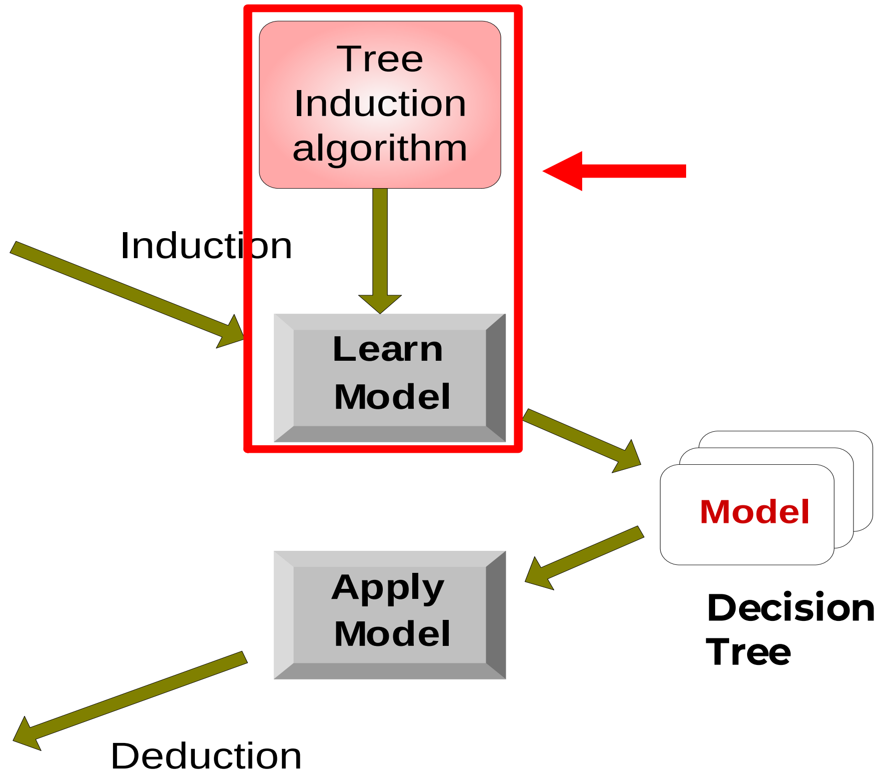
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	? **No**

UNIVERSITY OF WOLLONGONG AUSTRALIA

Decision Tree Classification Task

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



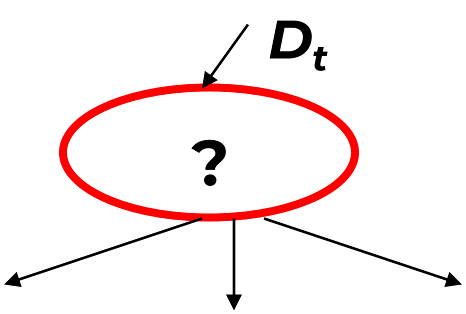
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

General Structure of Decision Tree Induction Algorithms

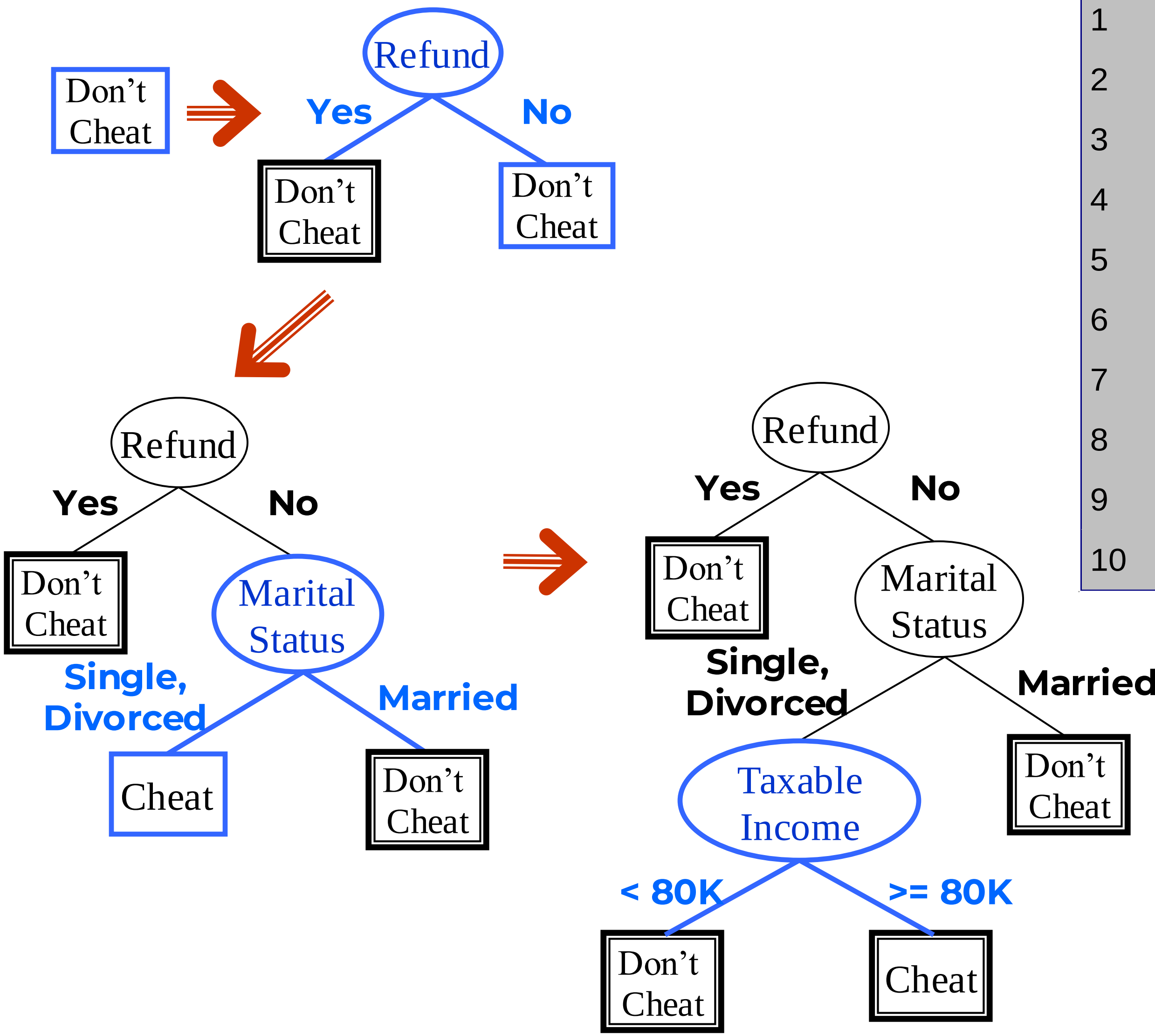
- Let D_t be the associated set of training records that reach a node t
- General Procedure:
  - If D_t contains records that belong the same class y_t, then t is a leaf node, labeled as y_t
  - If D_t is an empty set, then t is a leaf node, labeled as the same class as its parent node
  - If no more attributes to split D_t, then t is a leaf node, labeled as the *majority class*
  - Otherwise, ***split*** the dataset into smaller subsets, each of which is associated with a child node of the node t, and ***recursively*** apply the same procedure to child node

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



UNIVERSITY OF WOLLONGONG AUSTRALIA

Hunt's Algorithm



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

UNIVERSITY OF WOLLONGONG AUSTRALIA

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition? (focus)
    - How to determine the best split?
  - Determine when to stop splitting

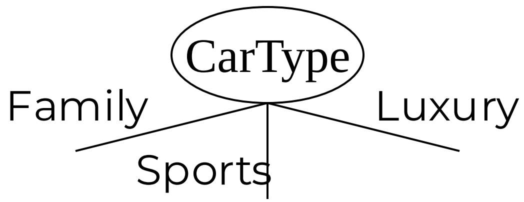
How to Specify Test Condition?

- Depends on the attribute types
  - Nominal/categorical
  - Ordinal
  - Continuous

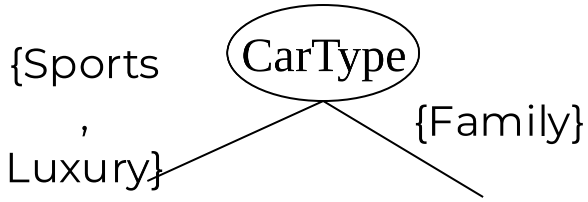
- Depends on the number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

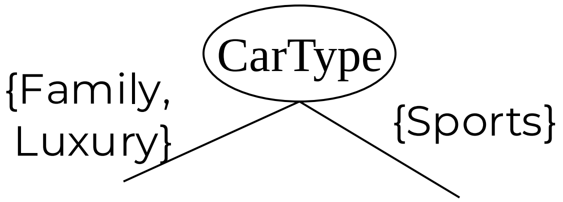
- Multi-way split: Use as many partitions as distinct values.



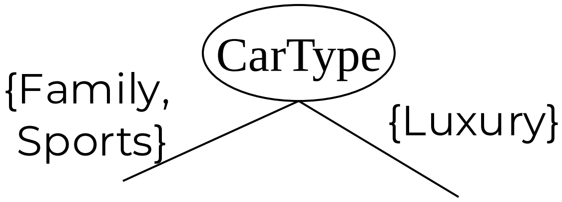
- Binary split: Divide values into two subsets. Need to find optimal partitioning.



OR



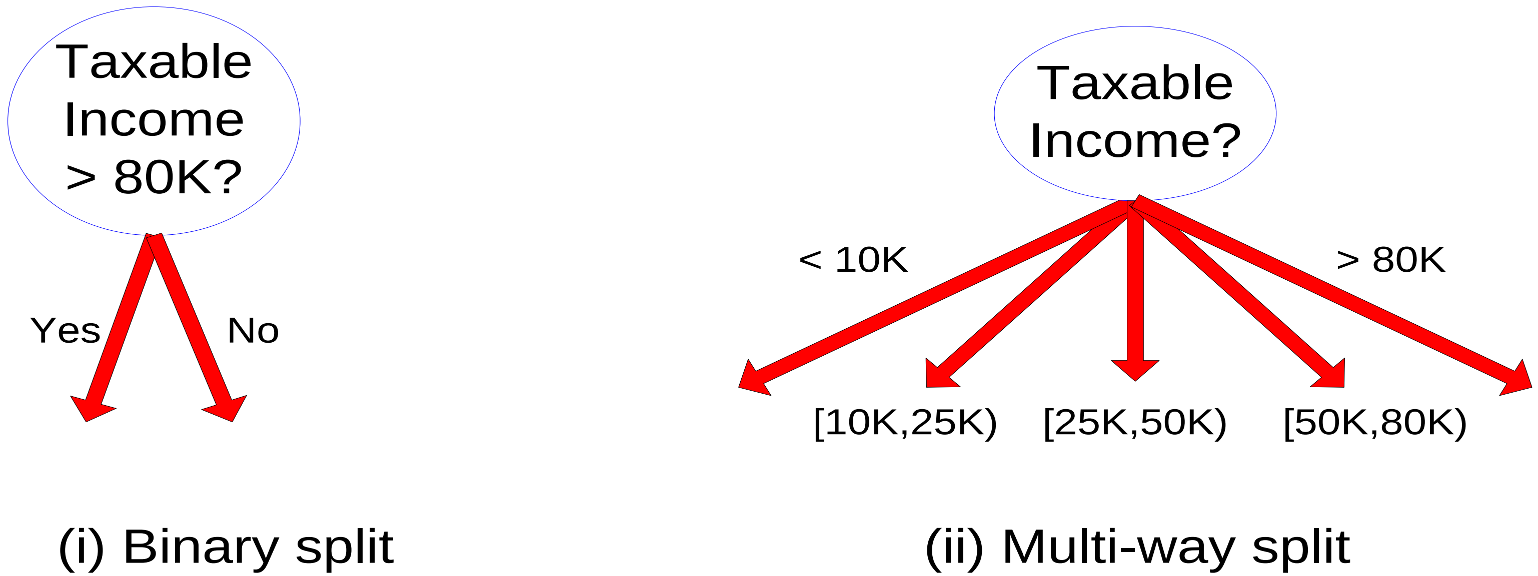
OR



Splitting Based on Ordinal/Continuous Attributes

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

Splitting Based on Ordinal/Continuous Attributes



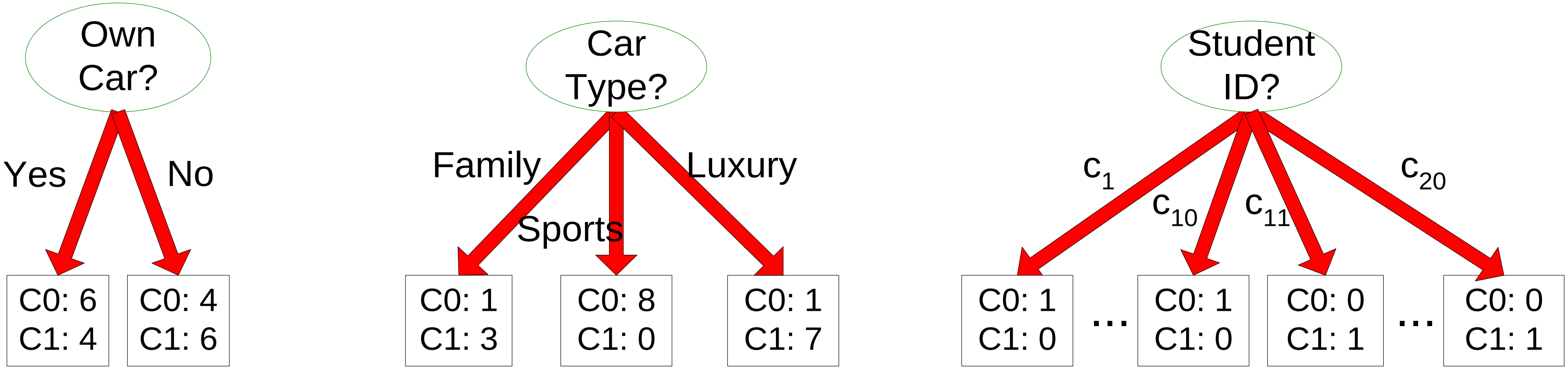
Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split? (focus)
  - Determine when to stop splitting

How to determine the Best Split

**Before Splitting: 10 records of class 0,
10 records of class 1**



Which test condition is the best?

How to determine the Best Split

- Greedy approach:
  - Nodes with homogeneous class distributions are preferred
- Need a measure of node **impurity** (or information **uncertainty**):

C0: 5	C0: 9
C1: 5	C1: 1

Non-homogeneous,

High degree of impurity

Homogeneous,

Low degree of impurity

Another way to look at Impurity and Uncertainty

- We flip two different coins: (0 is “head”, 1 is “tail”)
  - 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0…
  - 0 1 0 1 0 1 1 1 0 0 1 1 0 1 0 1 0 1…

- Question: *How to measure/quantify the information uncertainty with the two coins?*

UNIVERSITY OF WOLLONGONG AUSTRALIA

Different Measures of Impurity/Uncertainty

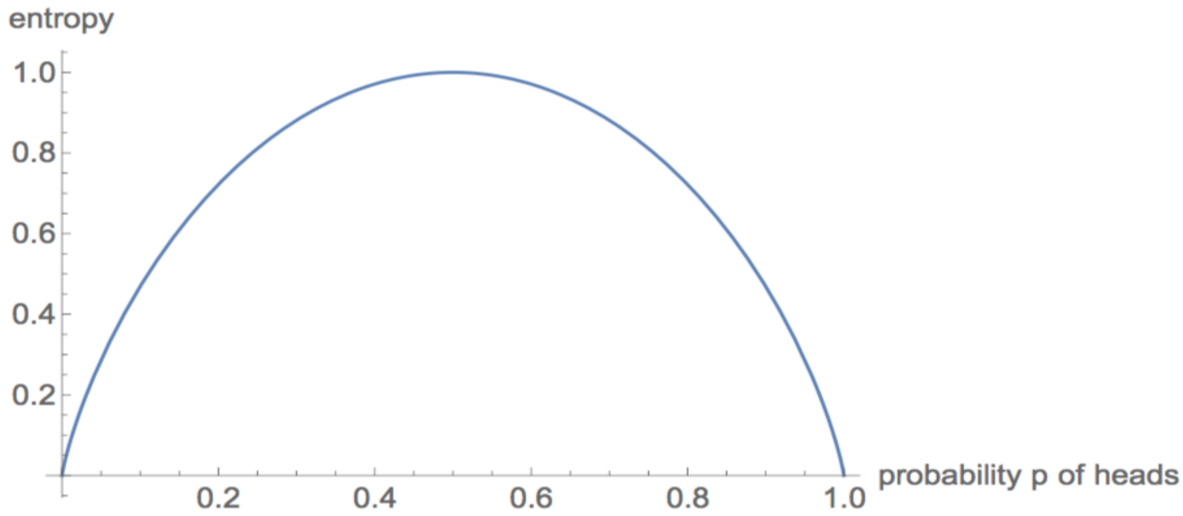
- Entropy (information gain)
- Gain ratio
- Gini Index
- Variance
- Others …

Shannon Entropy

- Logarithm: $y = \log_a x$
  - $2^3 = 8 \Leftrightarrow \log_2 8 = 3$
  - $2^{-1} = 0.5 \Leftrightarrow \log_2 0.5 = -1$
- Shannon Entropy:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Entropy of a coin:



Conditional Entropy

- Example: X = {Raining, Not raining}, Y= {Cloudy, not cloudy}

	Cloudy	Not cloudy	Total
Is Raining	24	1	25
Not Raining	25	50	75
Total	49	51	100

- What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

Note. $H(Y|X) \neq H(Y)$

$$
\begin{aligned}
H(Y|X) &= \sum_{x \in X} p(x) H(Y|X=x) \\
&= \frac{1}{4} H(\quad Y \quad |\text{is raining}) + \frac{3}{4} H(\quad Y \quad |\text{not raining}) \\
&\approx 0.75 \text{ bits}
\end{aligned}
$$

Information Gain

- If I don't know whether it is raining or not, the entropy of cloudiness is $H(Y) \approx 1.00$ bit (*verifying this as an exercise*)
- How much information about cloudiness do we gain by discovering whether it is raining?
- The Shannon entropy tells $\text{InfoGain}(Y|X) = H(Y) - H(Y|X) \approx 0.25$ bit
- How do we make use of this measure to construct our decision tree?
  - E.g., to determine the best split of the dataset.

Splitting Based on InfoGain

- Let D be the set of training records that reach a node
  - Compute the entropy $H(D)$ for D
- Let *Attribute_List* be a set of attributes associated with D
  - Each split with an attribute in *Attribute_List* produces a **partition** on $P = \{D_1, \ldots, D_v\}$ on D
  - Compute the conditional entropy for each split and then calculate the InfoGain:

$$H_P(D) = \sum_{i=1}^{v} \frac{|D_i|}{|D|} H(D_i)$$

$$\text{InfoGain}(P) = H(D) - H_P(D)$$

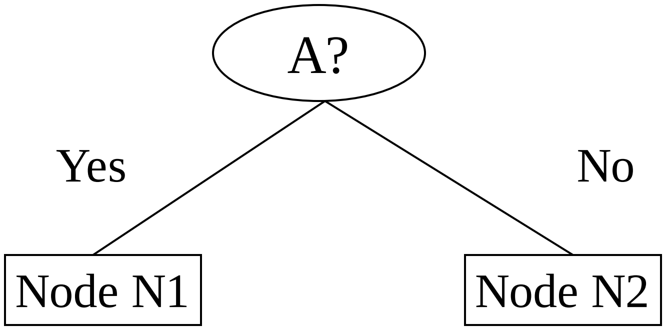
- Select an attribute that gives the best split (one with the *largest* InfoGain)

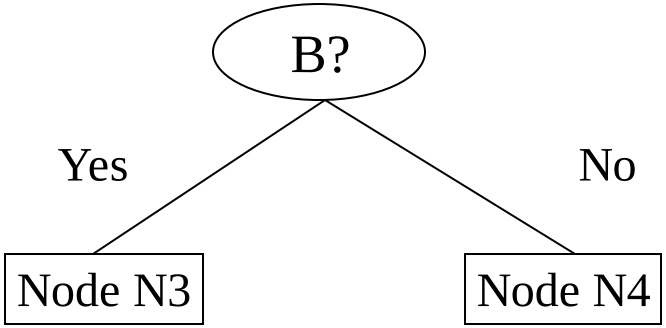
How to Find the Best Split

Before Splitting:

class	counts
C0	N_{00}
C1	N_{01}

→ **H0**





C0	N_{10}
C1	N_{11}

C0	N_{20}
C1	N_{21}

C0	N_{30}
C1	N_{31}

C0	N_{40}
C1	N_{41}

H1 * p1 **H2 * (1-p1)** **H3 * p3** **H4 * (1-p3)**

$$\underbrace{H1 * p1 \quad H2 * (1-p1)}_{H12} \qquad \underbrace{H3 * p3 \quad H4 * (1-p3)}_{H34}$$

Gain = H0 – H12 vs H0 – H34

p1 (resp. p3) is proportion of samples meeting "yes" when applying A (resp. B)

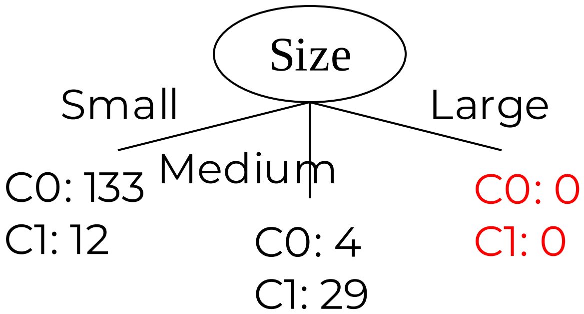
Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting (focus)

Stopping Criteria

1. No more attribute for splitting the dataset D_t
   - Majority vote: select the class label with most records to report
2. All tuples in D_t share the same class label
3. D_t is empty (no tuples)
4. Non-basic criteria
   - Tree pre-pruning (talked later), such as
     - set a threshold for the impurity measured
     - minimum dataset size
     - largest tree depth
     - etc.



UNIVERSITY OF WOLLONGONG AUSTRALIA

Tree Induction Algorithm

Assumption: the training tuples contain categorical values only; multi-split is used.

Procedure: **generate_decision_tree**(*D*, *Attribute_List*).

- ❖ Generate a decision tree from a set of training tuples of *D*.

Input:

- Dataset, *D*, which is a set of training tuples (each includes a tuple of feature values and one class label)
- *Attribute_List*, the set of candidate attributes for split

Output: A decision tree

Tree Induction Algorithm

Pseudo-code:

(1) create a node *N*;

(2) **if** tuples in *D* are all of the same class, i.e. 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 C_0

(6) find the *best_splitting_attribute* in *Attribute_List* to split *D*;

(7) *New_Attribute_List* ← *Attribute_List/{best_ splitting_attribute}*;

Tree Induction Algorithm

(8) **foreach** value *s* of *best_ splitting_attribute*;

(9) let D_s be a subset of *D* with *best_ splitting_attribute* being *s*;

(10) **if** D_s is empty **then**

(11) attach a (leaf) node labeled with the majority class in *D* to node *N*;

(12) **else** attach a new node, N_{child}, returned by applying **generate_decision_tree**(D_s, *New_Attribute_List*) to node *N*;

(13) **return** *N*;

UNIVERSITY OF WOLLONGONG AUSTRALIA

Classification with Decision Trees

- Given a testing tuple, the classification with a decision tree is just by traversing the tree until a leaf is reached.
- Procedure: **classify**(*N*, *d*)
- Input: testing tuple *d*.
- Output: a class label C
- Pseudo-code:

(1) **if** *N* is a leaf node **then**

(2) **return** the class label C with *N*;

(3) **else** traverse to the child node N_{child} of *N* where the value of the *best_splitting_feature* matches the value in *d*;

(4) let C = **classify**(N_{child}, *d*);

(5) **return** C;

Python Implementation

- Python **dictionaries** are a convenient data structure to represent a decision tree
  - Each splitting feature is a node
  - For a multi-split tree with categorical features (JSON style):

```
tree = {
        index_of_splitting_feature: {
          v_0: subtree_0 or leaf_0,
            ...
          v_l: subtree_l or leaf_l
         }
}
```

    where each *v* is a (unique) value of the splitting feature.
  - Access to the split feature and values:

```
split_feature = tree.keys()
subtree= tree[split_feature]
feature_values = subtree.keys()
```

Python Implementation

- A **leaf** can just be a class label, say, C_i.
- But more generally, a leaf can be represented by a NumPy array (i.e., vector) $ary = (q_1, \dots, q_m)$
  - such that $q_i = |D_{c_i}|$ is a class frequency where:
    - D is the set of training tuples associated with splitting_feature (as a node), and
    - $D_{c_i} \subseteq D$ contains all tuples in D that belong to class C_i
  - Note that a class label can be determined immediately from the vector ary.
    - E.g., just choose the class with the largest q_i

Python Implementation

- It is not hard to observe that both the tree induction and the classification involve a ***recursive function***.
- Recursive function example in Python:

```python
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

- factorial is called within itself.
- Running:

  4! = 4 * 3!

  3! = 3 * 2!

  2! = 2 * 1!

  1! = 1

Python Implementation

- To check whether a node in a tree (as a Python dictionary) is a leaf or grows a subtree:

```
# python3
isinstance(somenode, dict) == True #a subtree
# or
type(somenode).__name__=='dict' #a subtree
```


UNIVERSITY OF WOLLONGONG AUSTRALIA

Sample Python Code (Compute Shannon Entropy)

```
# calculate Shannon Entropy of a dataset
def calcShannonEnt(dataSet):
    numEntries = len(dataSet) # number of tuples
    labelCounts = {}
    for featVec in dataSet:
        # a class label is the last element in each tuples
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / numEntries
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt
```

Sample Python Code (Multi-Split , Categorical Features)

```
def chooseBestMultiSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1  # number of features
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures): # iterate over all features
        uniqueVals = set([tuple[i] for tuple in dataSet])
        newEntropy = 0.0
        for value in uniqueVals:
# "splitDataSet" function, implemented elsewhere, filters "dataset" such that the i-th feature equals to "value"
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet) / float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain; bestFeature = i
    return bestFeature  # returns a feature index
```

How to Implement a Decision Tree Classifier

- How to represent/encode your decision tree?
  - Consider a Python dictionary (see previous slides)
- How to implement your tree induction algorithm based on the calcShannonEnt and chooseBestMultiSplit functions?
  - Consider a recursive Python function that calls the two functions
  - Address all basic stopping criteria
- How to classify (new) records with your decision tree?
  - Also consider a recursive function

- *The implementation assumes categorical features, how about ordinal and continuous features*?
  - Use **binning** to generate a suitable number of bins (e.g., 5)

Gini Index

- Gini index (or Gini impurity) is a measure of how often a randomly chosen element from the set would be incorrectly labelled, if it was randomly labelled according to the distribution of labels in the subset.
  - Given D , a set of training tuples:

$$\text{Gini}(D) = \sum_{i=1}^{m} p_i \sum_{j \neq i} p_j = 1 - \sum_{i=1}^{m} p_i^2$$

where $p_i = |D_{C_i}|/|D|$, i.e. the probability that a tuple in D belongs to class C_i. (Here D_{C_i} refers to a subset of D such that the tuple belongs to class C_i.)

Gini Index

- For multi-way split on some feature $P = \{D_1, \dots, D_m\}$ on D, the Gini index of D given this partitioning is

$$\mathrm{Gini}_P(D) = \frac{|D_1|}{|D|}\mathrm{Gini}(D_1) + \cdots + \frac{|D_m|}{|D|}\mathrm{Gini}(D_m)$$

  - The reduction in impurity that would be incurred by the binary split is

$$\Delta\mathrm{Gini}_P = \mathrm{Gini}(D) - \mathrm{Gini}_P(D)$$

Variance

- Variance is the expectation of the squared deviation of a random variable from its mean.
  - is a simple error measure for binary classification (i.e., two class labels, often represented by 0 and 1)
  - Given D, a data partition or a set of training tuples:

$$\mathrm{Var}(D) = p(1-p)$$

where p is the probability that a tuple in D belongs to class C_0 and is estimated by $|D_{C_0}|/|D|$.

Gain Ratio*

- Disadvantage of InfoGain: Tends to prefer splits that result in large number of partitions, each being small but pure.
- Recall that each split on node results in a partition $P = \{D_1, \dots, D_v\}$ on D, the set of records associated with this node.
- $\text{SplitInfo}(P) = -\sum_{i=1}^{v} \frac{|D_i|}{|D|} \log\left(\frac{|D_i|}{|D|}\right)$
- $\text{GainRatio} = \text{InfoGain}(P)/\text{SplitInfo}(P)$

Comparison of Impurity Measures

- All impurity measures return good results in general, but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio:**
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index:**
    - biased to multivalued attributes
    - has difficulty when the number of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions
  - **Variance:**
    - suitable to binary classification, even though extension is possible

Advantages of Decision Tree Classifier

- Construction of the tree does not require any domain knowledge
- Can handle multidimensional data
- Representation of knowledge (as a decision tree) easy to assimilate by human
- The learning and classification steps are simple and fast
- Good accuracy in general.

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Pre-pruning: *Halt tree construction early*— do not split a node if this would result a measure falling below a threshold
    - Difficult to choose appropriate parameter thresholds
  - Post-pruning*: *Merge branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data *different* from the training data to decide which is the "best pruned tree"

Pre-pruning Sample Python Code (Multi-Split)

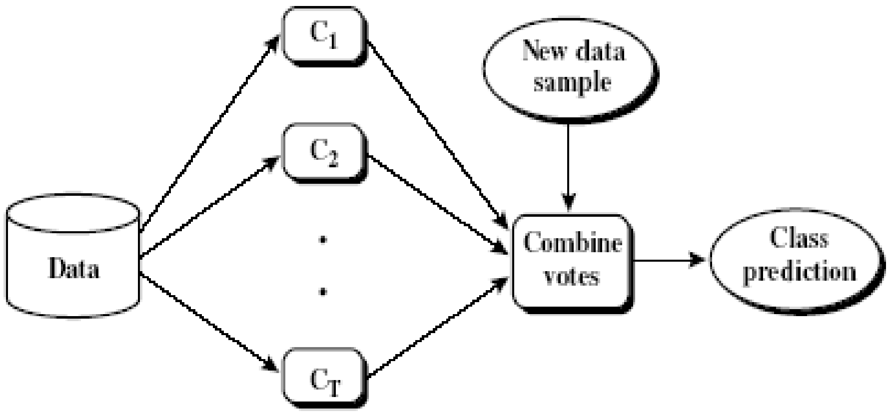
```
def chooseBestMultiSplit(dataSet, ops=(0.1,20)):
        tolG = ops[0]; tolN = ops[1]
    if (shape(dataSet)[0] < tolN):
        return None # exit
    numFeatures = len(dataSet[0]) - 1  # number of features
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures): # iterate over all features
        uniqueVals = set([tuple[i] for tuple in dataSet])
        newEntropy = 0.0
        for value in uniqueVals:
# "splitDataSet" function, implemented elsewhere, filters "dataset" such that the i-th
feature equals to "value"
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet) / float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain; bestFeature = I
         if bestInfoGain < tolG:
        return None #exit
    return bestFeature  # returns a feature index
```

ops is an optional argument. If the variance decrement is small than ops[0] or the size of the split dataset is small than ops[1], stop the split process. By default, ops=(0.5,4).

Random Forest:

Model Ensemble for Decision Trees

Ensemble Methods:

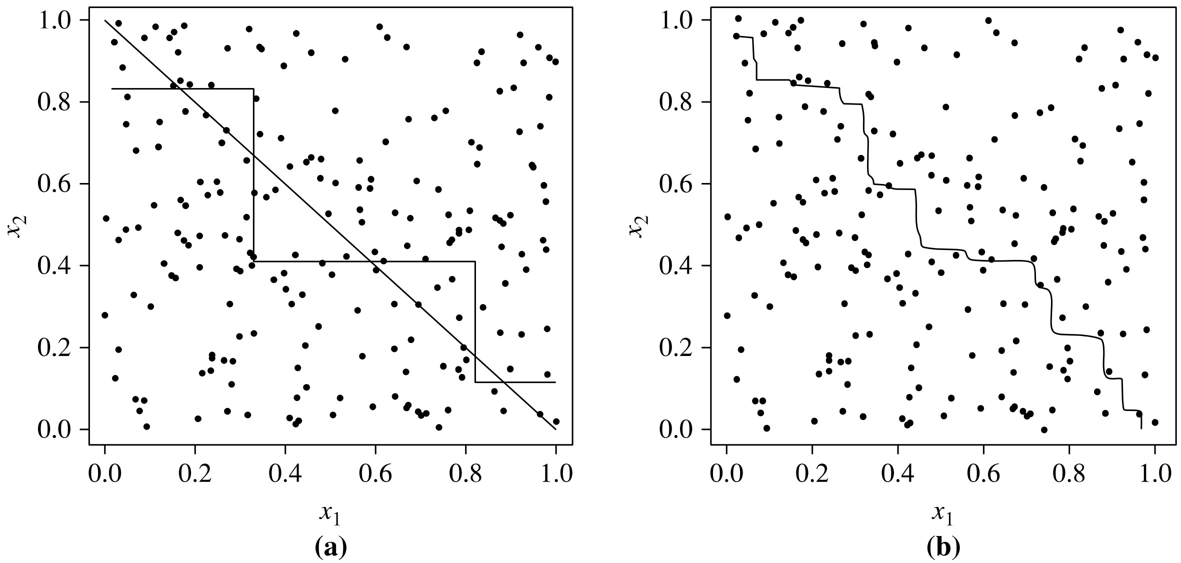


- Ensemble methods
  - Use a combination of models
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating a combined model M*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

Ensemble Methods:

- Advantages:
  - Increase accuracy: Miss classification occurs only when more than half of base classifiers predict incorrectly (even better if the base classifiers are less correlated.
  - Can deal with data in sheer volume (too many records or attributes)
  - Can run in parallel

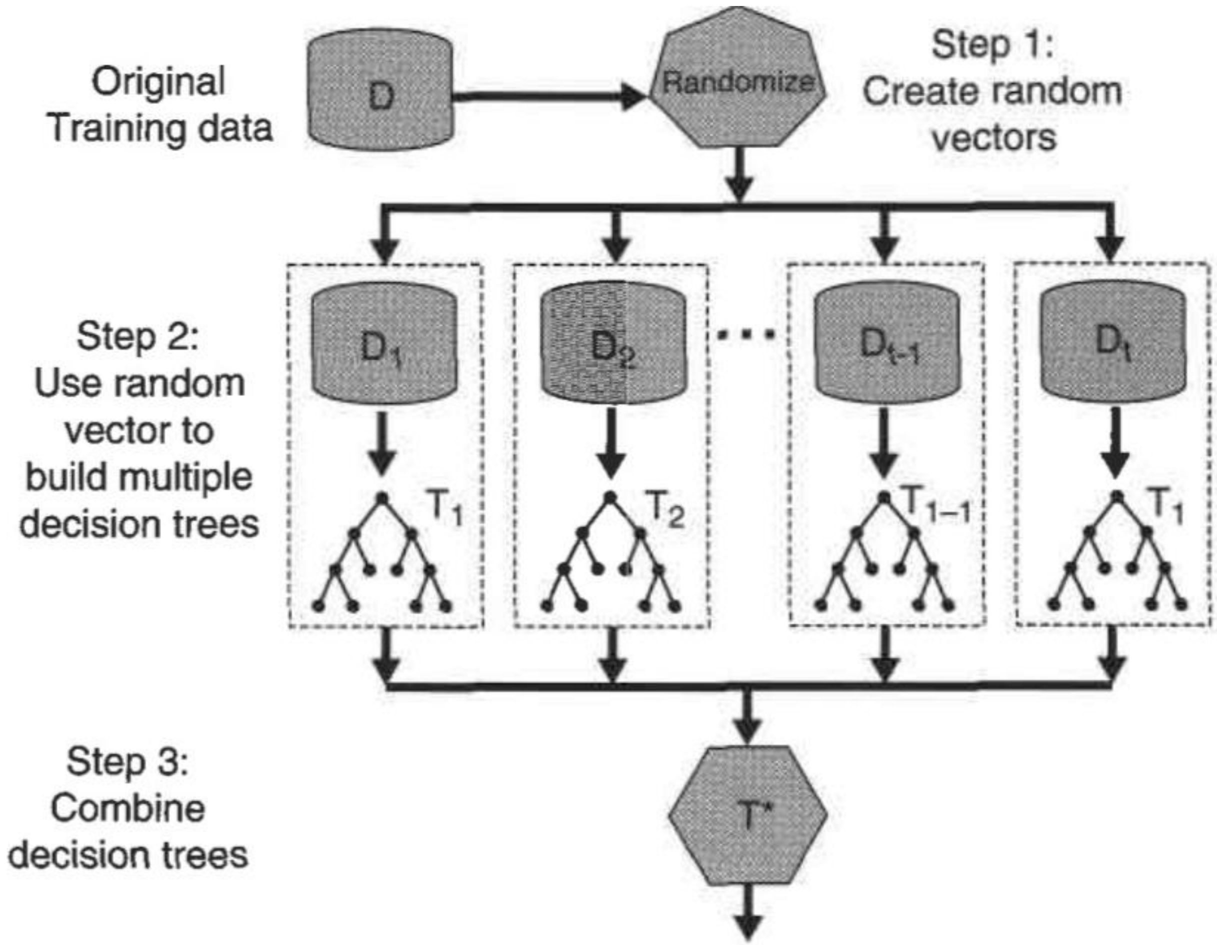
Decision boundary by (a) a single decision tree and (b) a random forest



Random Forest

- Random Forest is a class of ensemble methods specifically designed for decision tree classifiers.
  - It combines the predictions made by multiple decision trees.
  - Each tree is generated randomly based on the training tuples.
  - The final prediction output is produced by a voting function.
- A properly built random forest tends to be more accurate and less biased than individual decision tree classifiers.
  - The accuracy of RF depends on the *strength* of individual classifiers (trees) and a measure of *dependence* between them.
- But the computational cost grows as the number of trees in the forest increases.

Random Forest



Random Forest

- There are 3 common ways to associate randomization with decision trees.
- **(1) Bagging**: Given a set D of d tuples, bagging works as follows. For iteration i ($i = 1, 2, \ldots, k$), a training set D_i of d tuples is sampled *with replacement* from the original set D.
- Note that some of the original tuples of D may not be included in D_i, whereas others may occur more than once.
- A decision tree M_i is learned for each training set, D_i. To classify an unknown tuple $\boldsymbol{X}$, each classifier M_i returns its class prediction, which counts as one vote.
- The bagged classifier, say, M_* counts the votes and assigns the class with the most votes to $\boldsymbol{X}$.
- Random Forests can handle datasets that don't fit in memory

Random Forest

- **(2) Forest-RI** (*random input selection*)
  - When building the tree, randomly select *F* attributes (features) that are used to determine the split at each node, where *F* is much smaller than the number of available attributes.
  - Useful when the number of attributes is large

Random Forest

- **(3) Forest-RC** (*random linear combinations*)
  - creates new attributes (features) that are a linear combination of the existing attributes.
    - that is, randomly selected and added together with coefficients that are uniform random numbers on $[-1,1]$.
  - Useful when the number of attributes is small or large

Summary

- Decision Tree Classifier
  - Theory
  - Implementation
  - Tree Pruning
- Random Forest