Managing Big Data

Basic Classification: Concepts, Decision Trees and Evaluation

Manolis Tzagarakis
Assistant Professor
Department of Economics
University of Patras

tzagara@upatras.gr
2610 969845
google:tzagara
Facebook: tzagara
SkypeID: tzagara
QuakeLive: DeusEx
CoD: CoDFather

The classification problem

- Given a collection of records **(training set)**
  - Each record contains a set of ***attributes***, one of the attributes is (always) the ***class***.
- Find a ***model*** **for class attribute** as a function of the values of other attributes.
- Goal: **previously unseen** records should be **assigned a class** as accurately as possible.
  - A ***test set*** is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

The classification problem

- Example of classification

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

One record

Here, *class = cheat* attribute of records.

Goal: Try to guess value of cheat (the "class") based on the other values of that record

It's called classification because we try to put each record in one class/category ("yes", "no")

Goal: Find function f() such that:

f (Refund, Marital Status, Taxable Income) = Cheat

The classification problem

- Prerequisites for classification
  - **Must have** class attribute
  - **<u>Class attribute </u>MUST BE discrete**
    - I.e. set of values of class attribute countable, fixed and known beforehand.
  - **Other attributes** of records (except class attribute) **can be anything: discrete and/or continuous**.

The classification problem

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	Class (assume discrete)
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

One record

NOTE:
When *class is a continuous attribute*, then such tasks are *called regression (which you probably know but we won't deal with this)*.
Basic *<u>difference between classification and regression</u>*.

Again, here we're interested only in situations where class is a discrete attribute (=classification).

"Class" also mentioned as: categories, category labels, labels.

The classification problem

- Formal definition of the classification problem

  "Classification is the task of learning a target-function f, that maps each attribute set x to one of the predefined class labels y."

- **Target-function f** also known as **classification model** or simply **model**.

Usefulness?

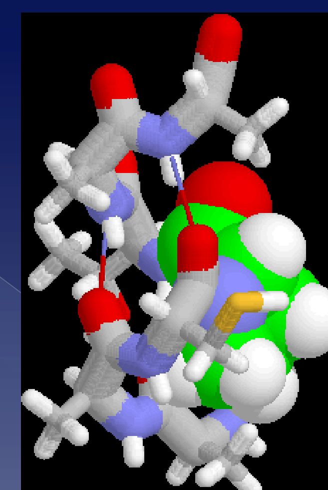
- **Descriptive modeling**
  - Used as an explanatory tool for distinguishing objects in different categories
    - E.g. for biologists to explain how a mammal, bird, fish is defined based on some characteristics
- **Predictive modeling**
  - Used as a tool to predict the label of a class of unknown objects
    - E.g. predict whether or not customers will be defaulting on loans or not

When does classification perform best?

- **Best**
  - When class attribute is binary (i.e. only has only two distinct values) or is nominal
- **Not so good**
  - When class attribute is ordinal (=values can be ordered e.g. Small, Medium, Large, XLarge)
    - Classification does not take into consideration the ordering that ordinals imply – be careful
  - When class attribute resembles hierarchy (category/subcategory)
- **Our focus: class attribute is binary or nominal**

Real application domains?

- Predicting tumor cells as **benign** or **malignant**
- Classifying credit card transactions as **legitimate** or **fraudulent**
- Classifying secondary structures of protein
  as **alpha-helix**, **beta-sheet**, or **random**
  coil
- Categorizing news stories as **finance**,
  weather, **entertainment**, **sports**, etc



Classification techniques

- **Decision Tree based Methods**
- k-nearest neighbors
- Rule-based Methods
- Memory based reasoning
- Neural Networks
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines (SVD)

General approach of existing classification techniques

In **training set**: class of each record already known and correct and this creates the model!

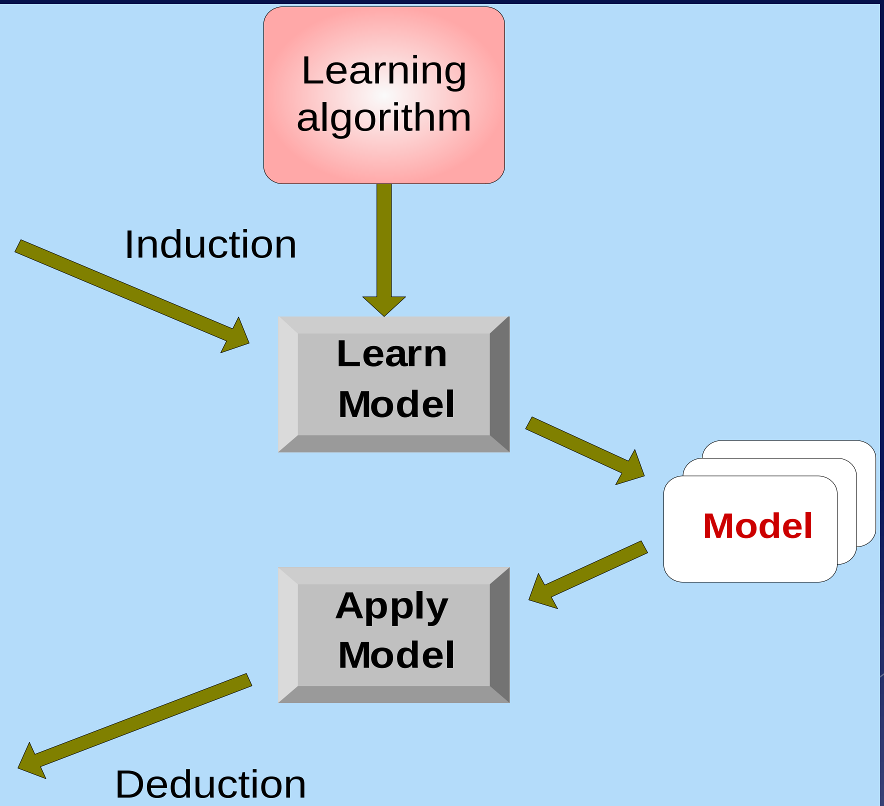
In **testing set**: class of each record unknown and the goal is to find it by applying the model.

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



Performance of model?

- The goal is to **find a model** that assigns to each record the correct class
- However, errors may occur
  - **What is an "Error"?** Model puts record in class **j** when it in reality, it belongs to class **k** .
- Confusion matrix tells us the performance

		Predicted class by model	
		Class 1	**Class 0**
True class it belongs to	**Class 1**	f_{11}	f_{10}
	Class 0	f_{01}	f_{00}

Confusion matrix:
f11 tells us the number of class 1 items that have been put by algorithm in class 1 .
In general: fij = number of items in class i predicted by model as class j.

Performance of model?

- Metrics based on confusion matrix
  - **Accuracy**
    - Pct of **correctly identified** classes

$$Accuracy = \frac{NC}{T} = \frac{f11 + f00}{f11 + f10 + f01 + f00}$$

  - **Error rate**
    - Pct of **incorrectly identified** classes

$$Error\ rate = \frac{NW}{T} = \frac{f10 + f01}{f11 + f10 + f01 + f00}$$

Decision Trees

Decision Tree

- What is a decision tree?
  - A **hierarchical structure**, with nodes and edges, **which recursively partitions the data (records) into classes**, by examining the values of its attributes.
  - Build classification of regression models in the **form of a tree** (hierarchical structure)
  - Decision trees are models
    - Built by the training set to determine the structure
    - Used by the testing set to assign records into a class
  - **The basic idea**: perform **a series of steps/comparisons** in order to reach a conclusion (=i.e. class). The decision tree tells you which comparisons to make.

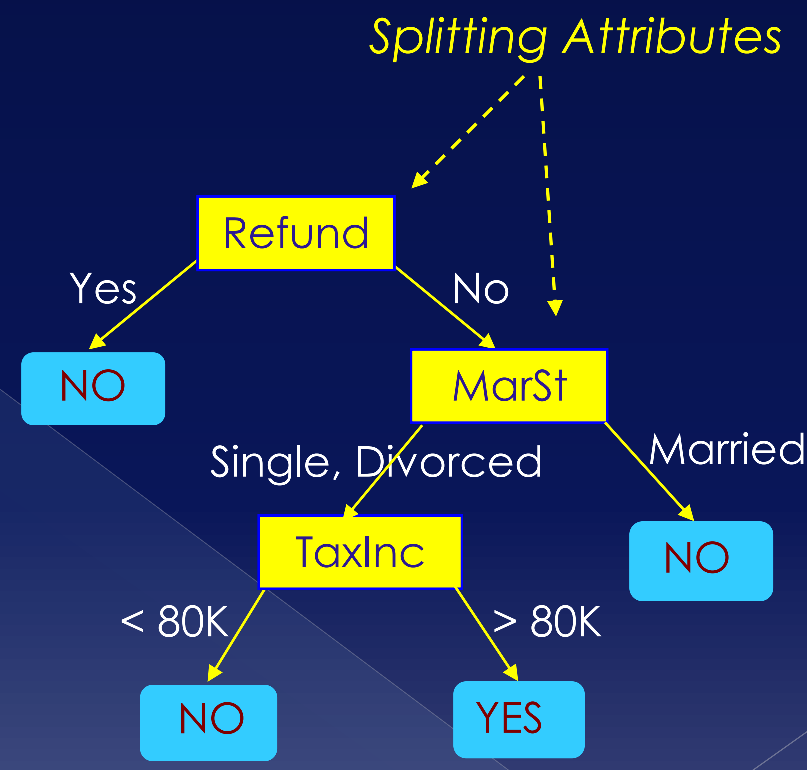
Decision Tree: example

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

Builds

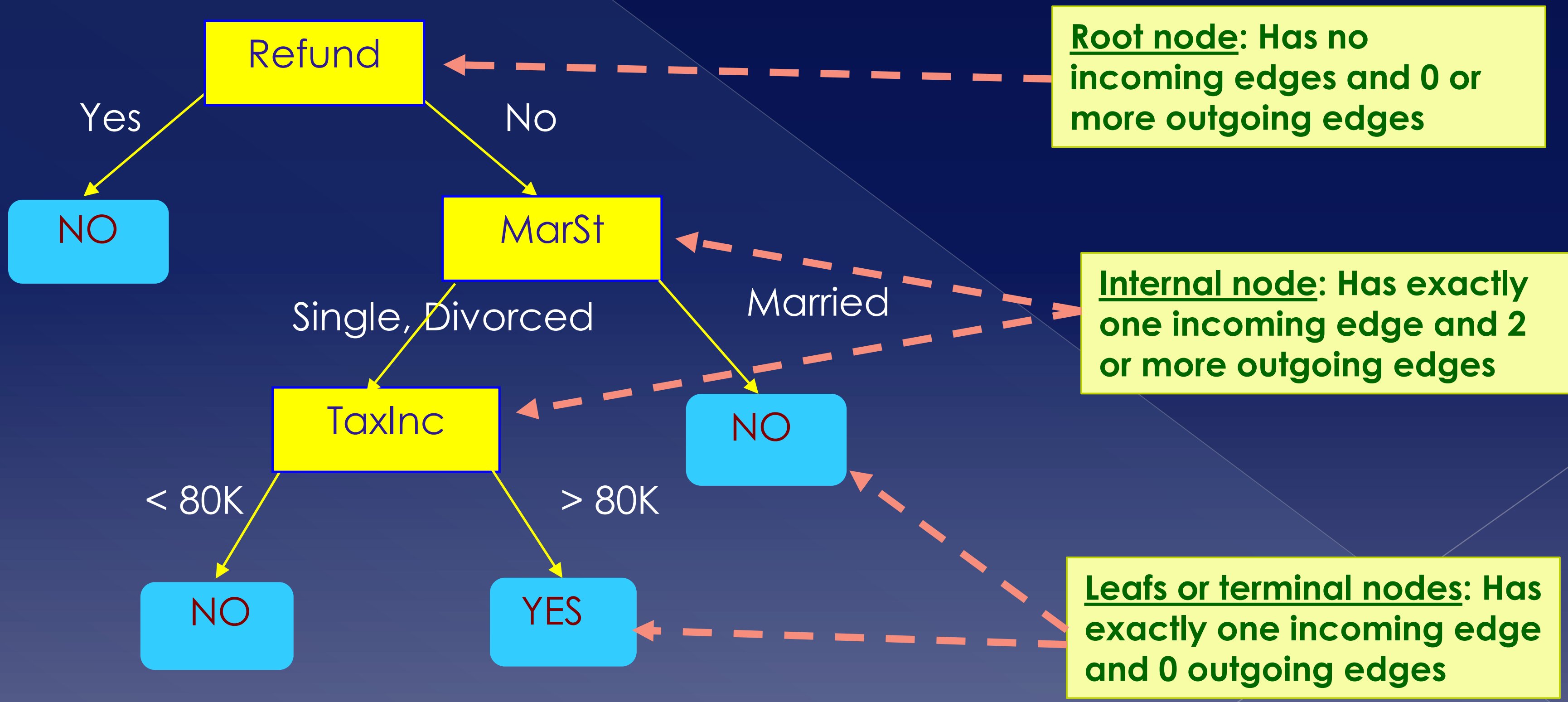


Model: Decision Tree

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

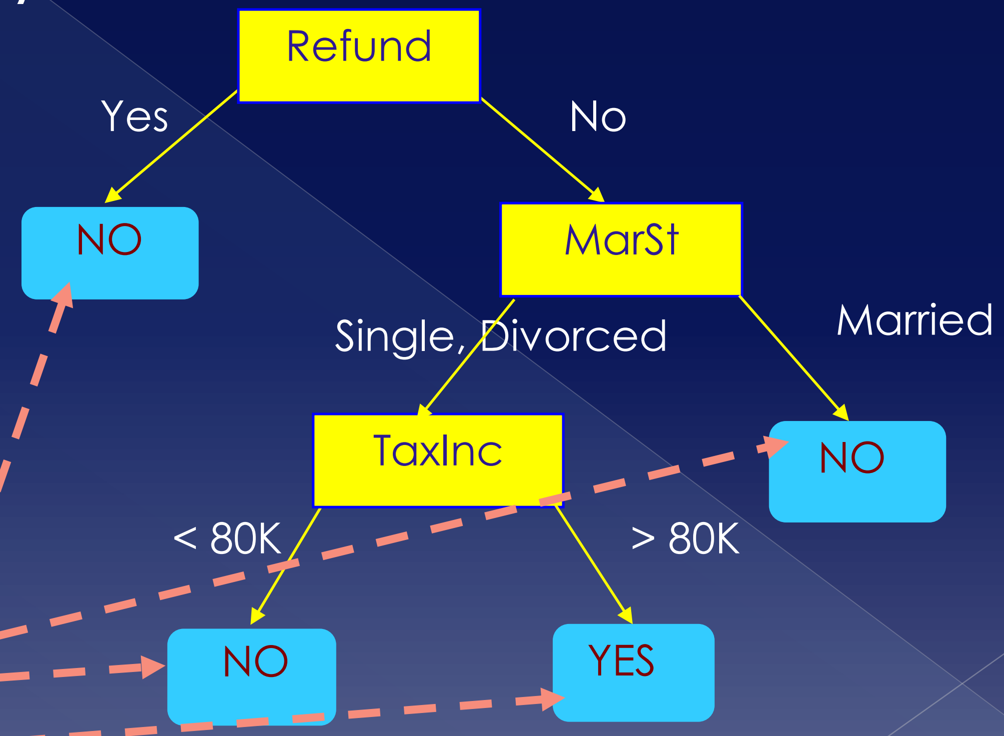
Decision Trees

- Types of nodes in decision trees



Decision Trees

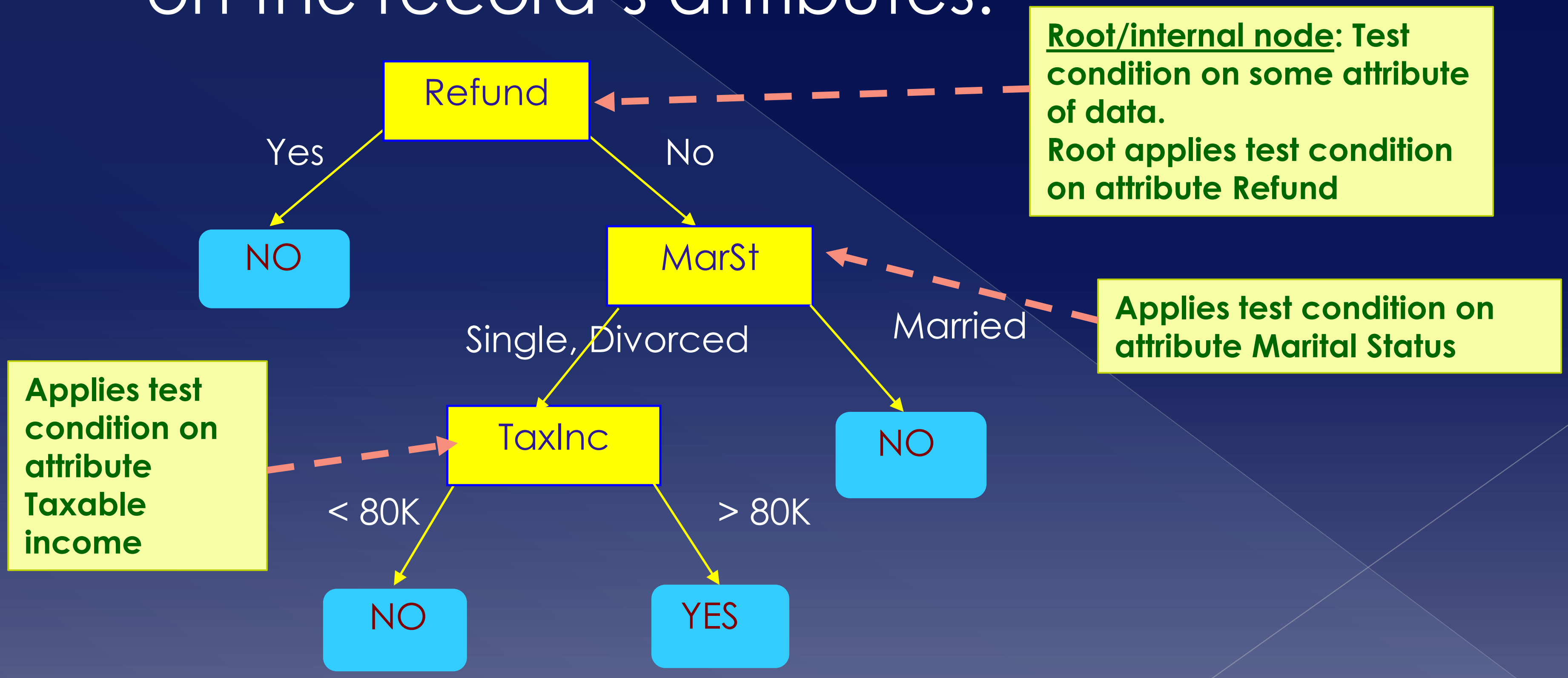
- In decision trees, **each leaf/terminal node** is assigned a class label (i.e. one value of the class attribute)



<u>Leaf or terminal node</u>: Assigned one value of class attribute (in each leaf one of the 2 values of cheat: {YES, NO})

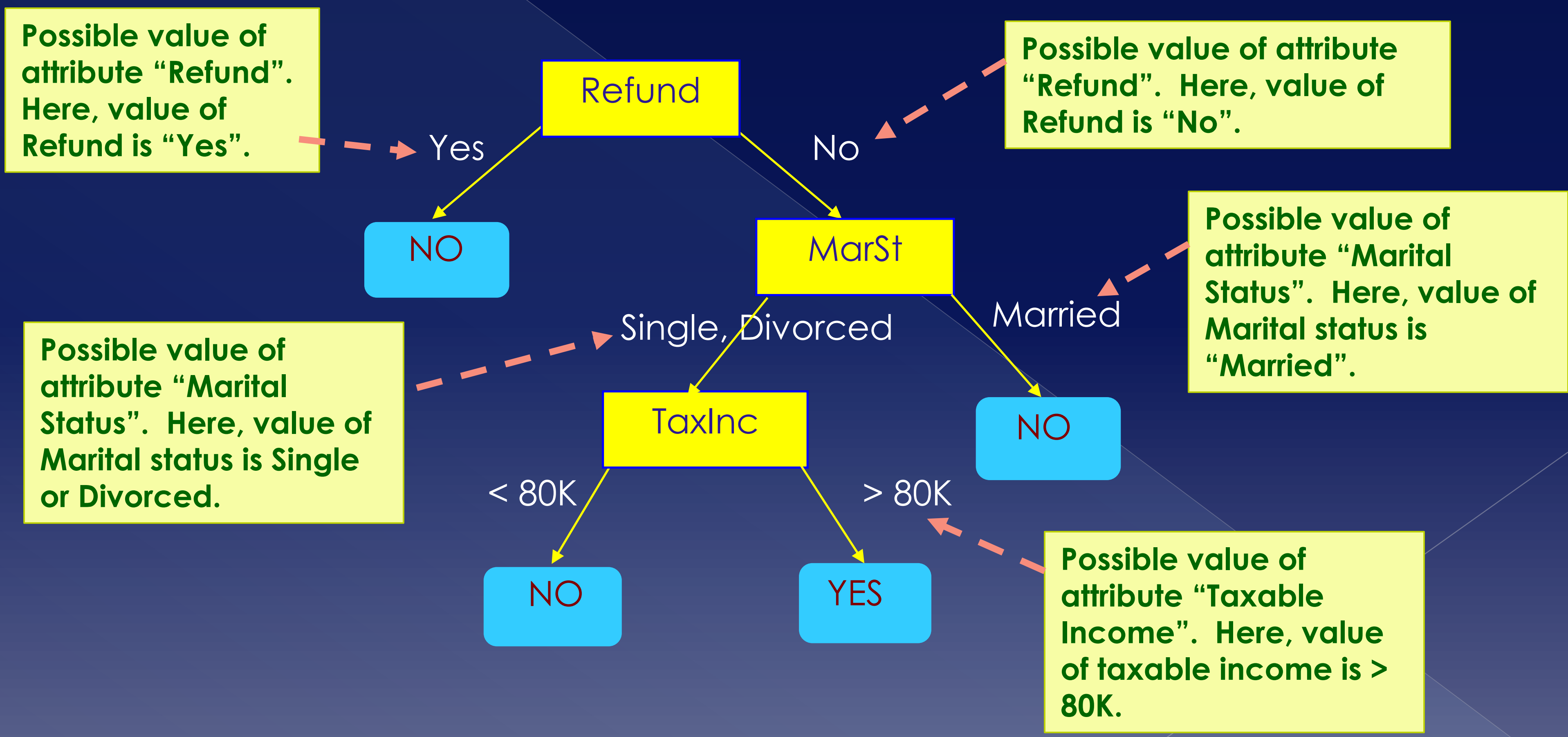
Decision Trees

- **Non-terminal nodes (i.e. root and internal nodes)** contain test conditions on the record's attributes.



Decision Trees

- **Edges have labels**, indicating the values of the test condition



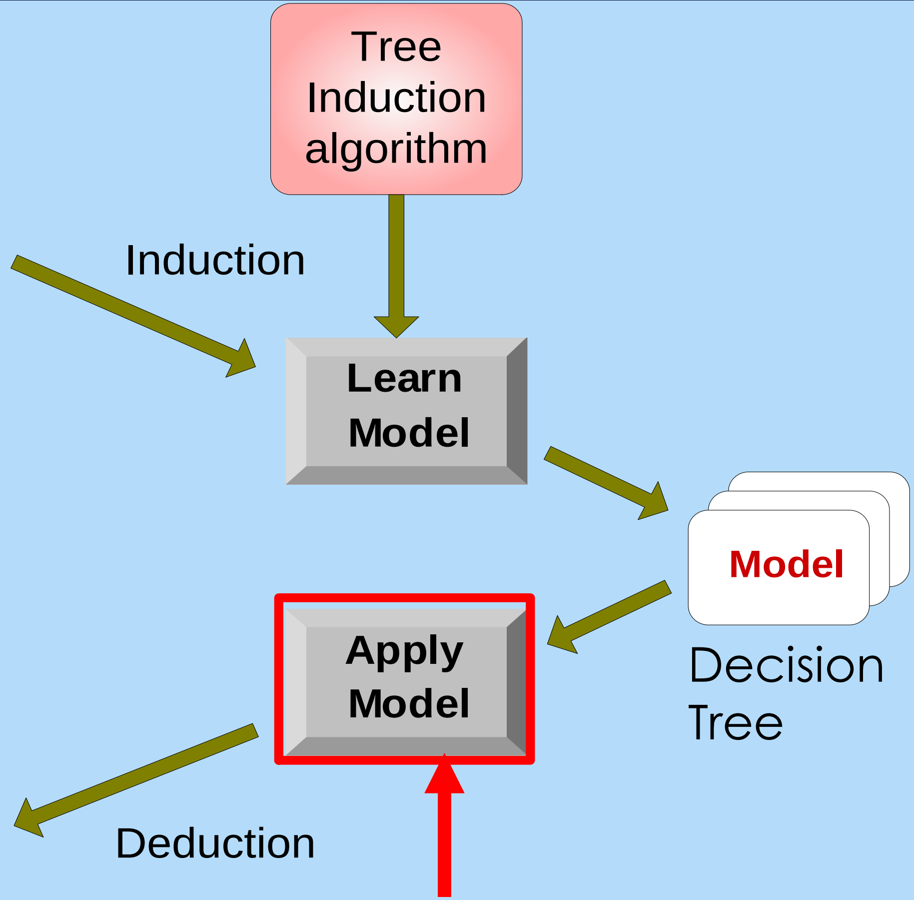
Using Decision Trees

- Decision trees (=the model) are **built by using the training data**
  - Where the class is already known and correct
- Decision trees (=the model) are **used by testing and unknown sets** to classify the data

Using Decision Trees

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

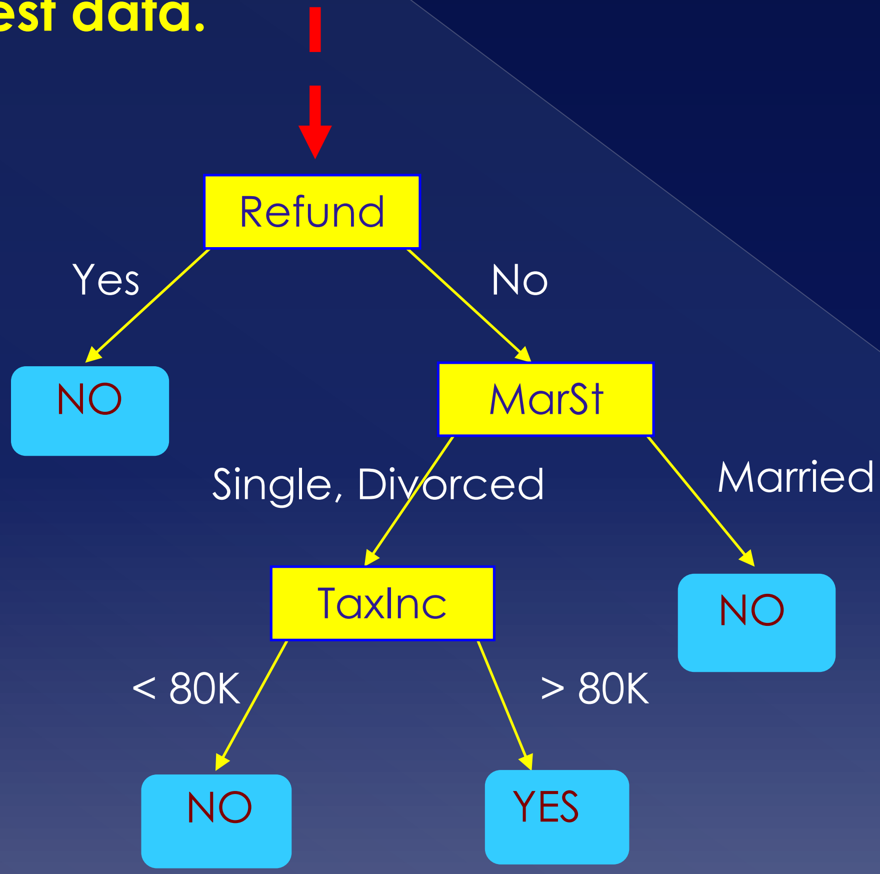


Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

How to apply Decision Trees to Test Data ?

Start from the root of tree and apply conditions to record of test data.



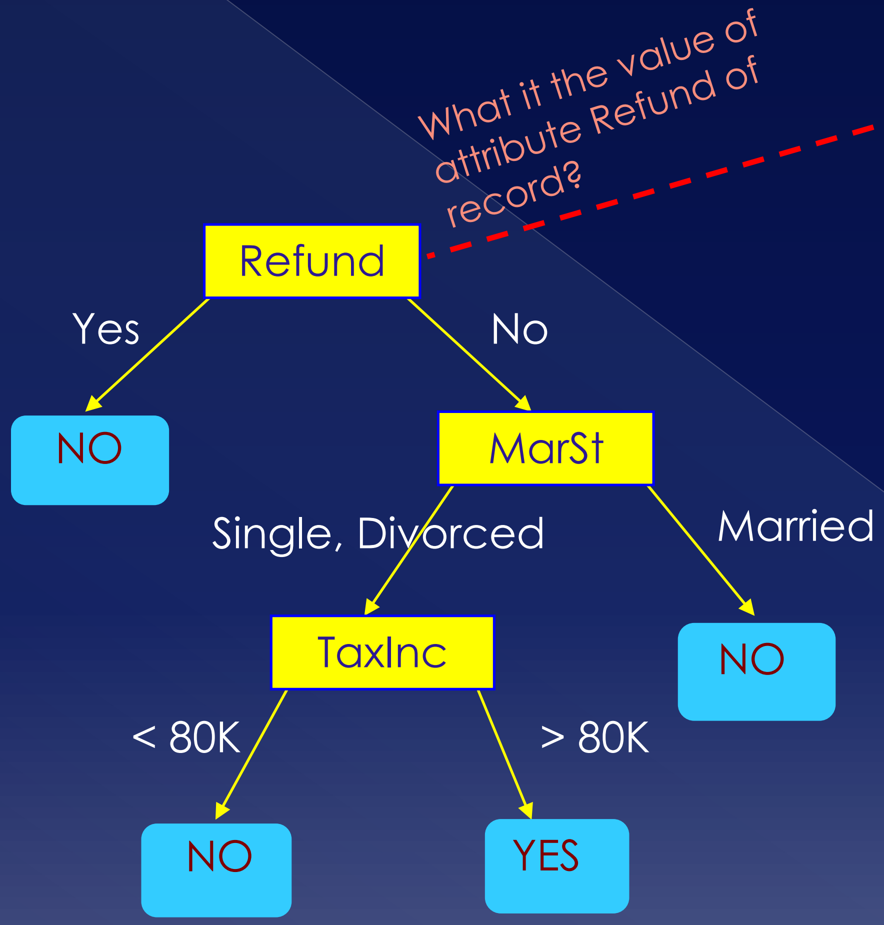
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

<u>Our goal</u>: Try to predict value of Cheat for this particular record.

We've built this decision tree (model) from the training data.

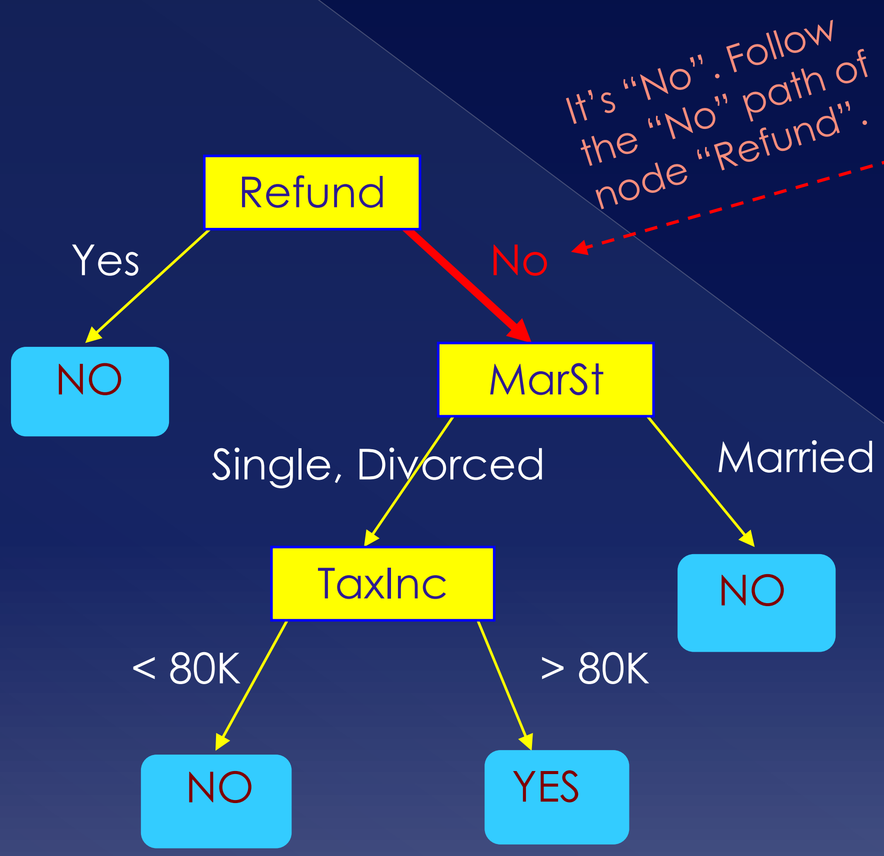
How to apply Decision Trees to Test Data ?



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

How to apply Decision Trees to Test Data ?



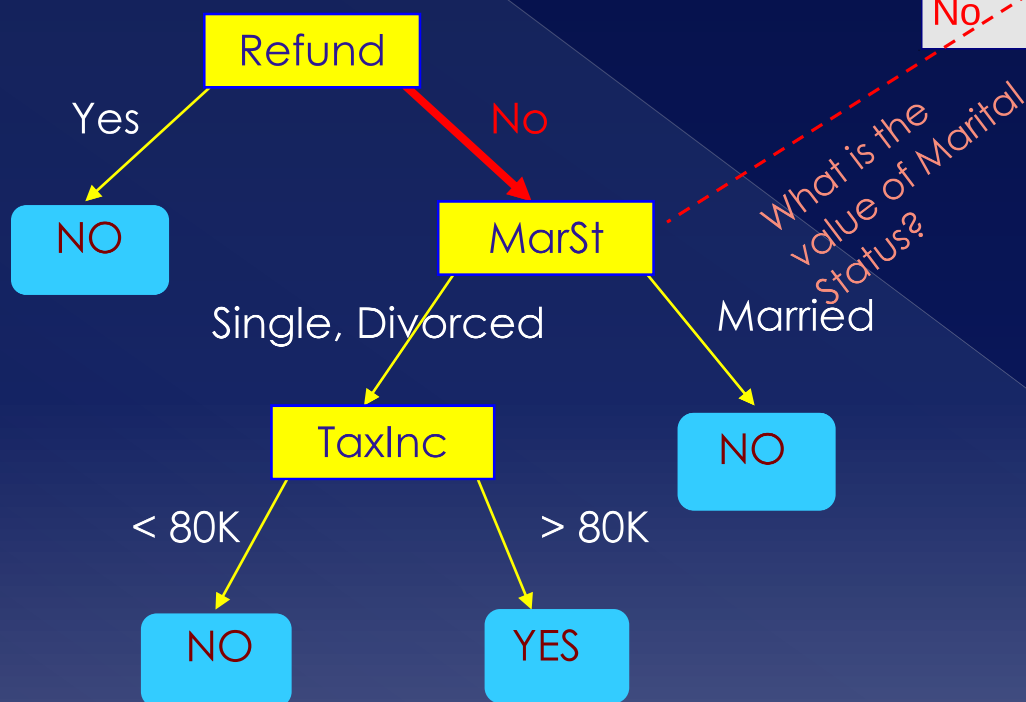
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

How to apply Decision Trees to Test Data ?

Test Data

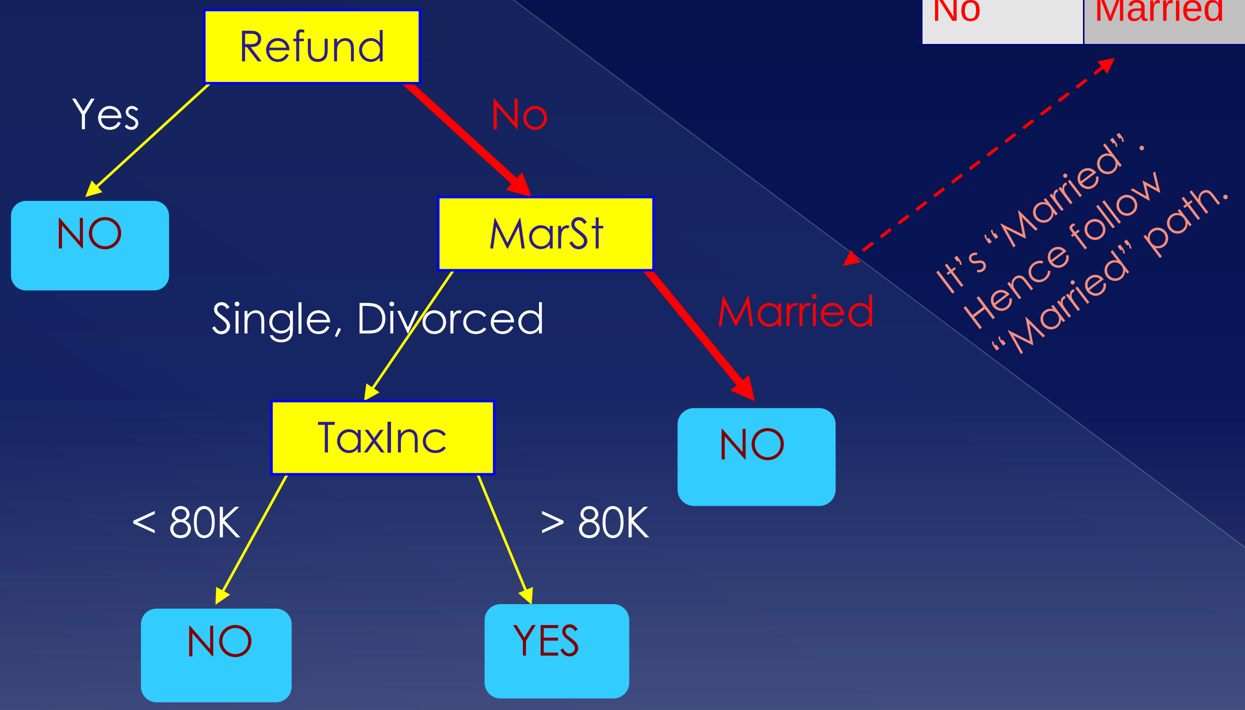
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



How to apply Decision Trees to Test Data ?

Test Data

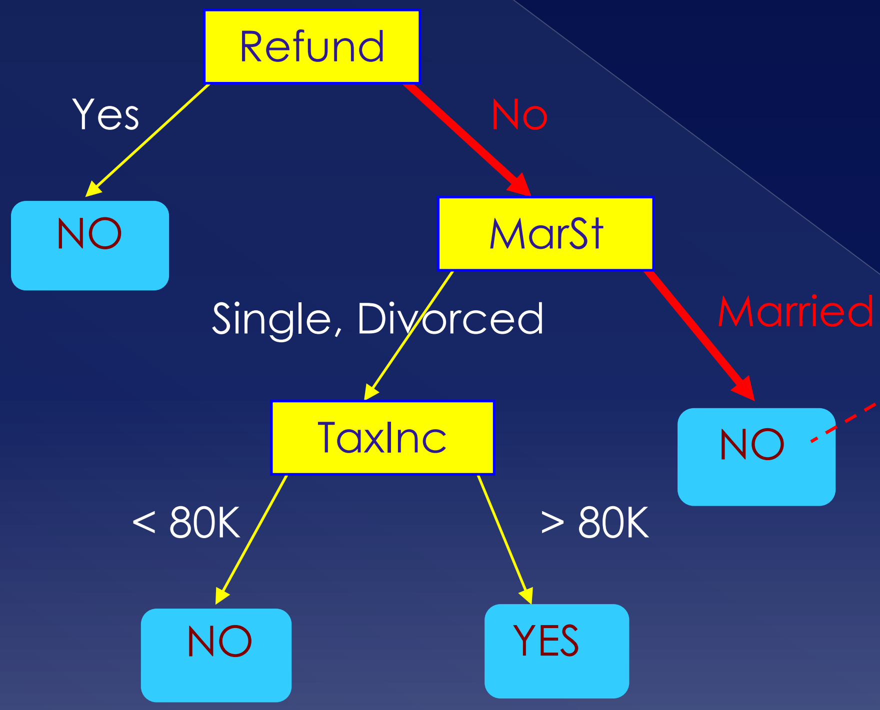
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



How to apply Decision Trees to Test Data ?

Test Data

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



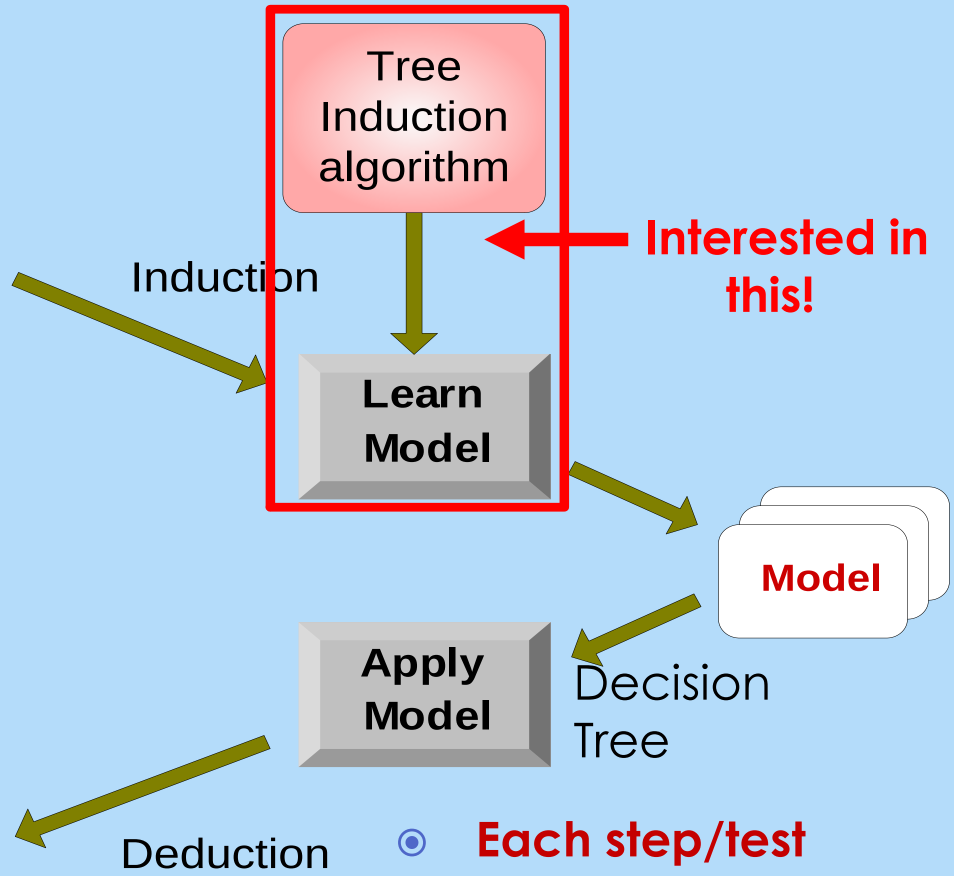
Reached leaf node which contains a class.

Assign value of leaf node to cheat i.e. assign cheat to "No" . Since record has been assigned to class ("No"), terminate.

How to build Decision Trees?

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

- **Each step/test condition is determined from training set**

How to build Decision Trees?

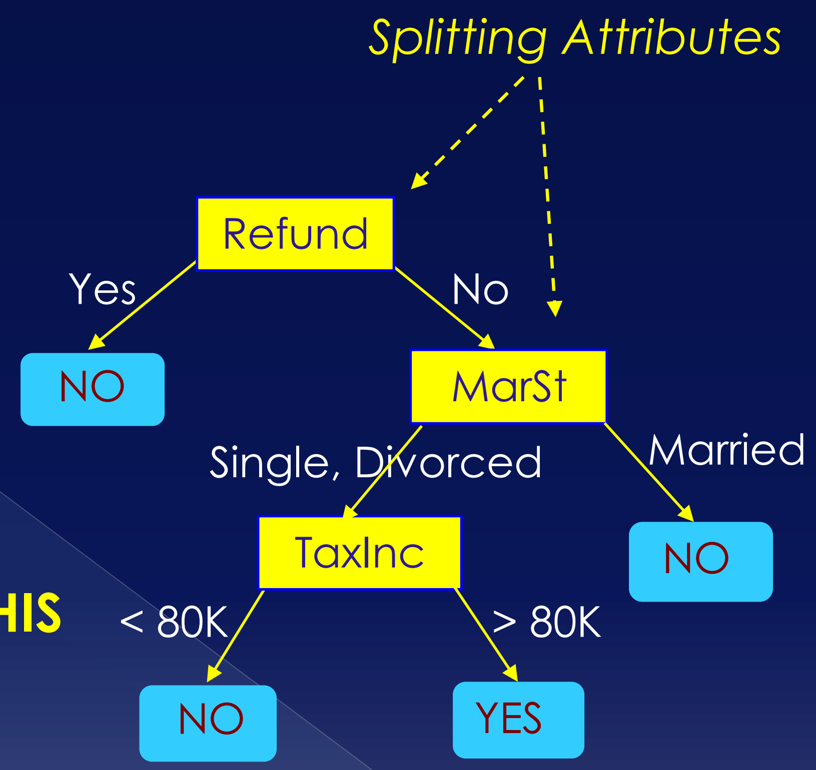
- The problem again!

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

Need to construct THIS



Model: Decision Tree

You start with this, AND NO DECISION TREE AT ALL

How to build Decision Trees?

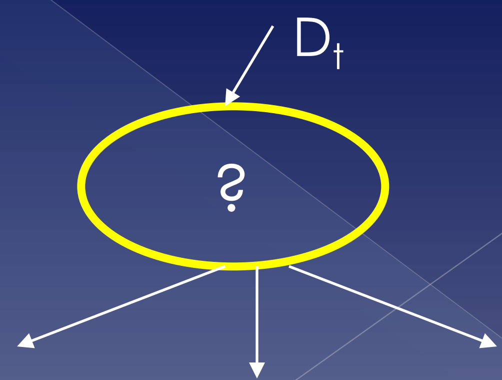
- Building Decision Tree based on training set using Tree Induction
- Many Algorithms :
  - **Hunt's algorithm (one of the earliest)**
  - **CART**
  - **ID3, C4.5**
  - **SLIQ,SPRINT**

Basic approach of Hunt's algorithm

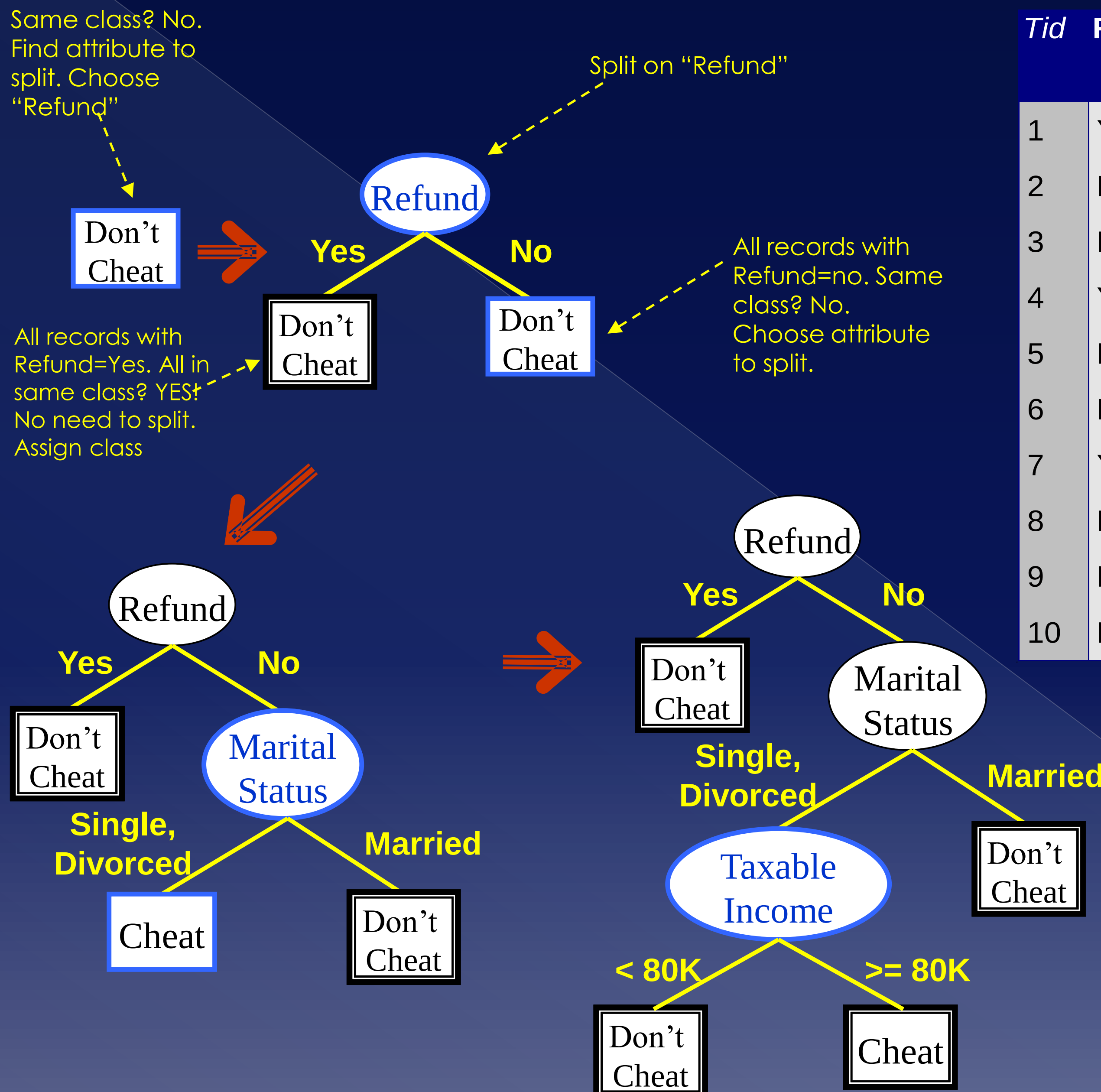
- Let D_t be the set of **training records that reach a node t**
- General Procedure:
  - If D_t contains **records that belong the same class y_t**, then t is a leaf node labeled as y_t
  - If **D_t is an empty set**, then t is a leaf node labeled by the **default class, y_d**
  - If **D_t contains records** that belong to **more than one class** (note: this is the training set and we know the class), use an **attribute test to split the data** into smaller subsets. **Recursively** apply the procedure to each subset.

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



Hunt's algorithm: example



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

Tree Induction

- **Greedy strategy** (greedy algorithms)
  - Split the records of the training **set based on an attribute that optimizes *now*** a certain criterion
- Some serious issues though
  - **How to split the records?**
    - How to specify attribute test condition for non-terminal nodes?
    - Which attribute to select, i.e. how to determine best split?
  - **When to stop splitting?**

How to specify test condition?

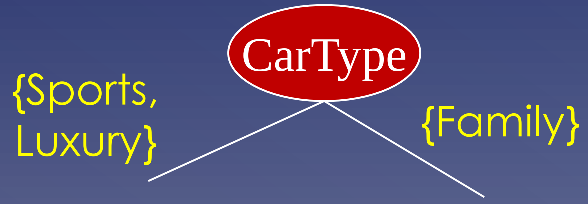
- Depends on attribute types of records. Can be:
  - Nominal
  - Ordinal
  - Continuous

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

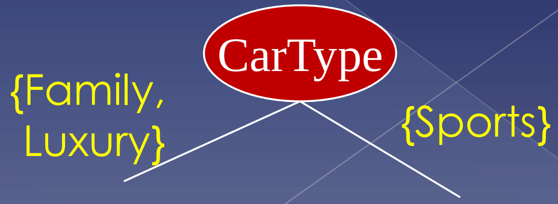
Splitting based on **Nominal attributes**

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

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

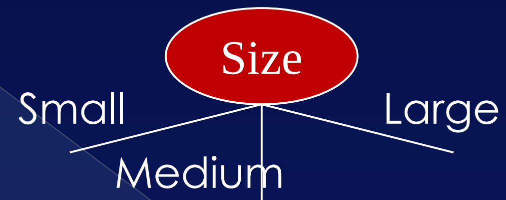


OR

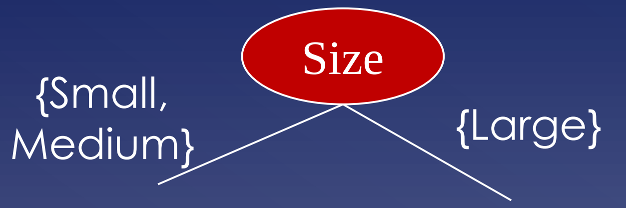


Splitting based on **Ordinal attributes**

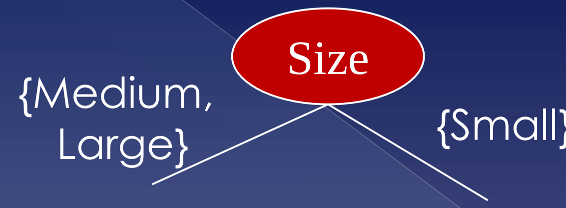
- **Multi-way split:** Use as many partitions as distinct values.



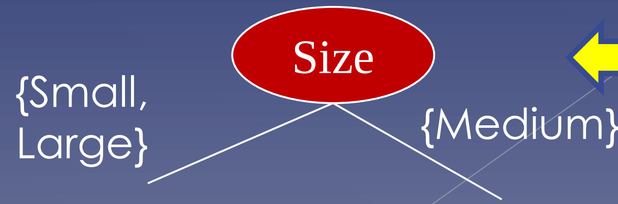
- **Binary split:** Divides values into two subsets. Need to find optimal partitioning.



OR



- What about this split?



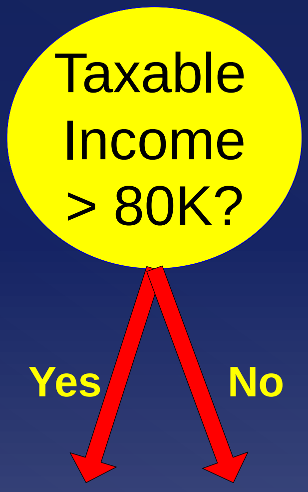
NOPE! Does not preserve order. REMEMBER: Ordinals are about order

Splitting based on **continuous attributes**

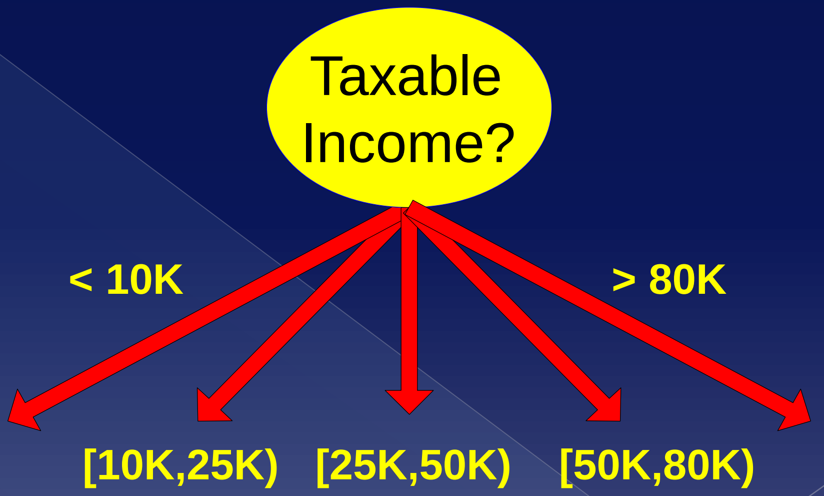
- Different ways of handling
  - **Discretization** to **form an <u>ordinal categorical attribute</u>**
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles) or clustering.
  - **Binary Decision: (A < v)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

Splitting based on **continuous attributes**

- Examples of continuous attributes



(i) Binary split



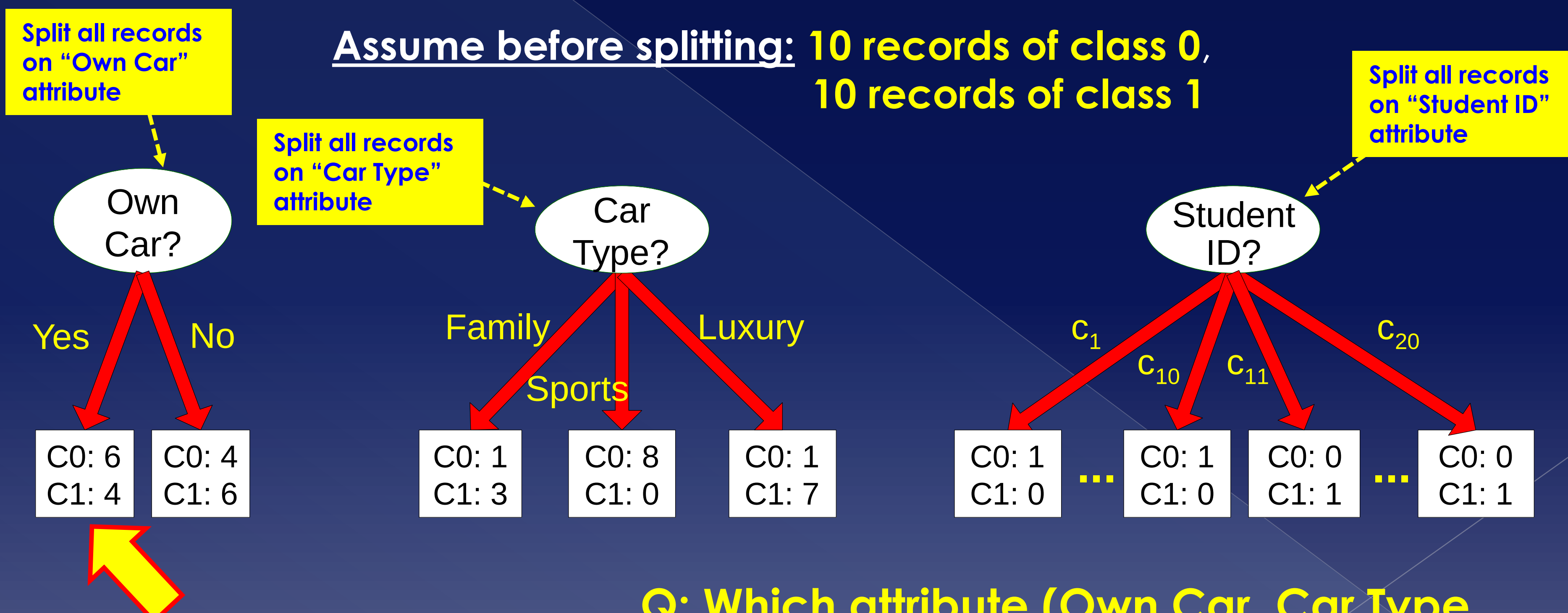
(ii) Multi-way split

Tree Induction

- How to determine which attribute to select for split i.e. determine best split?
  - Is it possible to somehow **measure the “goodness”/quality** of a split based on some attribute?
    - If yes, select **the split with the best quality**
    - **(Answer: YES, there are measures.)**

Determine best split

- Intuitive observations in order to come up with a measure



This means: If splitting based on attribute "Own car", then out of all records with value "Yes" in "Own Car", 6 belong in class 0 and 4 in class 1.

Q: Which attribute (Own Car, Car Type, Student ID) is better? i.e. Which split is better? ANSWER: Car Type.

Determine best split

- All algorithms have **greedy** and **divide and conquer** approach:
  - Nodes with **homogeneous** class distribution are preferred
    - **Homogeneous?** Low impurity, low intermixing of records belonging to different classes, records to belong mostly to one class
- Need a measure of **node impurity**:

Split records on attribute A

C0: 5
C1: 5

Non-homogeneous, High degree of impurity

Split (same) records on attribute B

C0: 9
C1: 1

This is better!

Homogeneous, **Low degree of impurity**

Measures of node impurity

- Three measures of node impurity
  - **Gini index**
  - **Entropy**
  - **Misclassification error**
- Different algorithms use different node impurity measures. E.g.
  - **ID3, C4.5 uses Entropy**
  - **Hunt, CART, SLIQ, SPRINT uses Gini index**

Gini Index

- Gini index **for a given node t**

$$Gini(t) = 1 - \sum_{j} [p(j|t)]^2$$

…where **p(j|t)** the **relative frequency** of class j at node t
(Note: each node may contain records from any class)

- Observations:
  - **Maximum = 1 – 1/n_c** when records of node t are distributed equally among classes. **Greatest impurity**
  - **Minimum = 0.0** when all records of node t belong to only one class. **Smallest (no) impurity**.
  - **Lower values are better/preferred!**
  - **Understanding Gini index? Measures how often (=probability) a randomly chosen record would be placed in the incorrect class.**

Gini index

- Examples: calculating the **Gini index for** various **nodes**

$$Gini(t) = 1 - \sum_{j} [p(j|t)]^2$$

A node with: 0 rec in class 0, 6 rec in class 1

Class 1 (C1)	**0**
Class 2 (C2)	**6**

$P(C1) = 0/6 = 0$ $P(C2) = 6/6 = 1$ (rel. fr/prob.)

Gini = $1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$

Class 1 (C1)	**1**
Class 2 (C2)	**5**

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

Gini = $1 - (1/6)^2 - (5/6)^2 = 0.278$

Class 1 (C1)	**2**
Class 2 (C2)	**4**

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

Gini = $1 - (2/6)^2 - (4/6)^2 = 0.444$

Gini Index

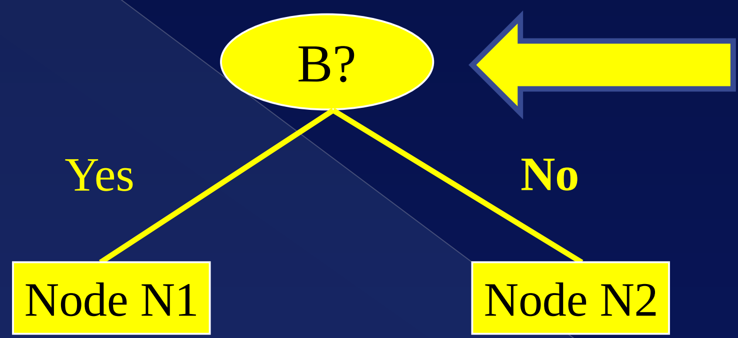
- Calculate **Gini Index for splits** (also referred to as "**Gini Index of attribute**")
- Assume a **node p** is split (based on attribute) into **k partitions** ("children") then Gini Index of split:

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

…where **n_i = number of records at node i** and **n = number of records at node p**

Gini Index

- Example: assume binary split. What is the **Gini index of this split?**



	B? (parent)
C1	6
C2	6

	N1
C1	5
C2	2

	N2
C1	1
C2	4

Gini index of each of the 2 nodes (N1, N2)

$\text{Gini}(N1) = 1 - (5/7)^2 - (2/7)^2 = 0.408$

$\text{Gini}(N2) = 1 - (1/5)^2 - (4/5)^2 = 0.320$

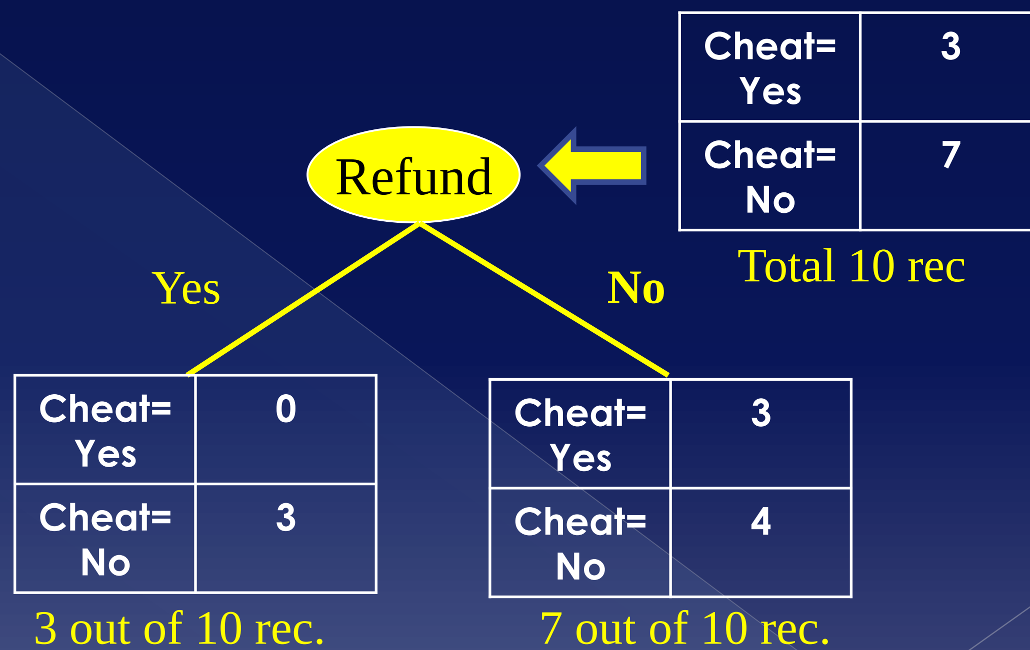
Gini(Split or B) = 7/12 * Gini(N1) + 5/12 * Gini(N2) = 7/12 * 0.408 + 5/12 * 0.320 = <u>0.413</u> QED

Gini index

- Examples

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

Compute Gini index for "Refund" attribute (i.e. compute Gini index of split based on "Refund"). **Cheat is class!**
If we split based on "Refund" we get:



Gini(Left node) = $1-(0/3)^2-(3/3)^2 = 0$
Gini(Right node) = $1 - (3/7)^2 - (4/7)^2 = 0.489$

Gini(Refund) = $(3/10) * 0 + (7/10) * 0.489 =$ 0.3423 QED

Gini Index – Gini gain

- **Gini gain**
  - The difference between **parent node's Gini index** and **Gini index of split:**

$$\text{Gini}_{gain} = \text{Gini}_{Parent} - \text{Gini}_{Split}$$

  - **Measures how Gini index improves**
  - **Goal: maximize gain, i.e. this difference,** which determines which attribute to select for splitting in this step.
  - **Important:** Used in algorithms to select attributes and build decision trees.

Gini Index – Gini gain: The main idea

Before Splitting:

C0	N00
C1	N01

→ M0

Split on A or B attribute???

A?

Yes → Node N1

C0	N10
C1	N11

→ M1

No → Node N2

C0	N20
C1	N21

→ M2

$\underbrace{M1 \qquad M2}_{M12}$

B?

Yes → Node N3

C0	N30
C1	N31

→ M3

No → Node N4

C0	N40
C1	N41

→ M4

$\underbrace{M3 \qquad M4}_{M34}$

Gain = M0 – M12 vs M0 – M34 => choose attribute with biggest gain!

Gini Index – Gini gain

- Example

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**

Compute Gini gain when splitting on attribute "Refund". **Cheat is class!**

Gini gain(Refund) = Gini of entire dataset – Gini(Refund)

We have already calculated Gini of attribute "Refund" **(=0.3423**).
Gini of our entire data set is also called "Gini of overall collection of training examples" or "Gini of system":
Gini(training set) = 1-$(3/10)^2$ – $(7/10)^2$ = 0.42

Gini gain(Refund) = 0.42 – 0.3423 = 0.0777 QED

Building Decision Trees Phase-1

- Using Gini and **Gini gain to build decision trees**

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

STEP 1: Compute Gini index for our entire dataset:

Gini = $1 – (2/6)^2 – (4/6)^2 = 0.444$

STEP 2: Compute Gini index for each attribute (Split):

Gini(M) = $(3/6)*0 + (3/6)*0.444 = 0.222$

Gini(N) = $(2/6)*0 + (2/6)*0.5+(2/6)*0.5=0.333$

Gini(Q) = $(2/6)*0.5 + (2/6)*0.5 + (2/6)*0 = 0.333$

STEP 3: Calculate Gini gains for each attribute:

Gini Gain(M) = 0.444 – 0.222 = 0.222 (biggest!)

Gini Gain(N) = 0.444 – 0.333 = 0.111

Gini Gain(Q) = 0.444 – 0.333 = 0.111

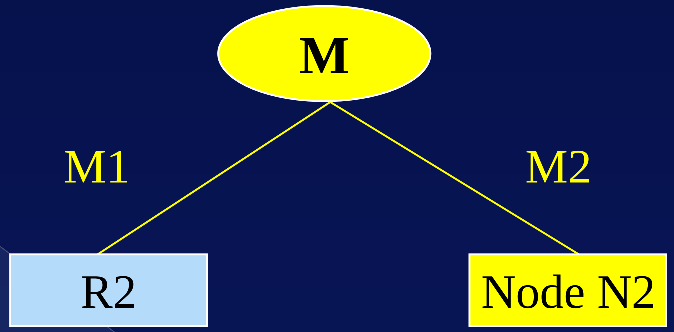
Hence, **first split based on attribute M**

Building Decision Trees Phase-2

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

Root node of tree will be test on attribute M



R1	**2**
R2	**1**

If M has value M1, then R is always R2. Homogeneous node! Hence no further splitting and make this a leaf.
We make a rule:
M1 → R2

Here, non-homogeneous node. Apply same method to split based on other attribute. Tree will grow at this branch.

Building Decision Trees Phase-3

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

Calculate again with the same steps.

STEP 1: Compute Gini index for our **new "entire" dataset** (leave out red-ish rows. These have already been classified):

Gini = $1 - (2/3)^2 - (1/3)^2 = 0.444$

STEP 2: Compute Gini index for each attribute:
Gini(N) = $(1/3)*0 + (1/3)*0+(1/3)*0 = 0$
Gini(Q) = $(1/3)*0 + (1/3)*0 + (1/3)*0 = 0$

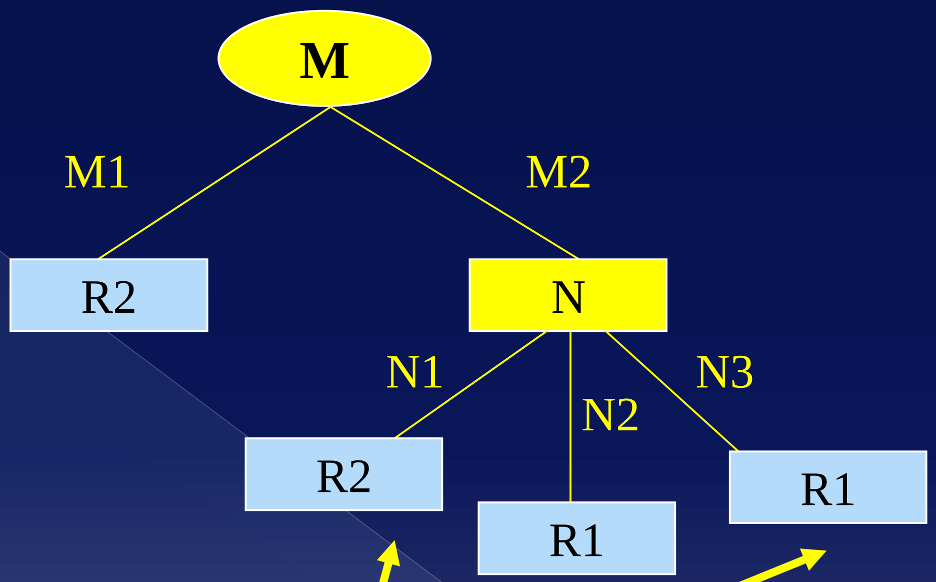
STEP 3: Calculate Gini gains:
Gini Gain(N) = 0.444 – 0.0 = **0.444**
Gini Gain(Q) = 0.444 – 0.0 = **0.444**

Equal Gini gains. Hence, both are equally good. **choose one of N, Q. Assume we choose N (can also choose Q).**

Building Decision Trees Phase-4

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2



Homogeneous nodes. Hence, no need to split further. Make them leaves with proper class values (from nodes) and devise rules (include incoming M2):

M2, N1 → R2
M2, N2 → R1
M2, N3 → R1

Dataset empty, hence done.
Decision tree built.

Computing Gini Index for **categorical** attributes

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	**Sports**	**Luxury**
C1	1	2	1
C2	4	1	1
Gini	**0.393**		

Gini(CarType)

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	**{Family}**
C1	3	1
C2	2	4
Gini	**0.400**	

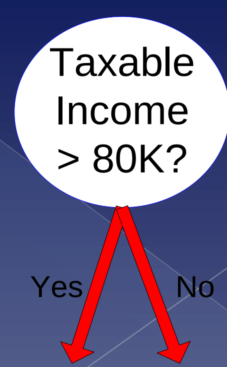
	CarType	
	{Sports}	**{Family, Luxury}**
C1	2	2
C2	1	5
Gini	**0.419**	

Note: for k categorical values you need to check $(2^k - 2)/2 = 2^{k-1} - 1$ partitions

Computing Gini Index for **continuous** attributes

- Use Binary Decisions **based on one value**
- Several Choices for the splitting value
  - Number of possible splitting values **= Number of distinct values**
- 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! Complexity: O(n^2) computing Gini.**
    - **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



Computing Gini Index for **continuous** attributes

- How to improve speed?
- For **efficient computation**: for each attribute,
  - Sort the attribute on values. **Complexity O(nlogn) =>better!**
  - **Linearly scan these values**, each time updating the count matrix and computing Gini index. Some optimization tricks:
    - Choose split points at **midpoint between values**
    - Identify **adjacent examples** that **differ in their target (class) labels** and attribute values => Set of candidate splits
  - Calculate Gini Index and choose the split position that has **the least gini index**

Sorted Values →

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income										
	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		<u>0.300</u>		0.343		0.375		0.400		0.420	

Alternative splitting criteria (besides Gini): **Entropy**

- **Entropy – measures information (INFO)**
- Calculate **Entropy for node t :**

$$Entropy(t) = -\sum_{j} p(j|t) \log p(j|t)$$

…where **p(j | t)** the relative frequency of class j at node t. (note **log** = base-2 logarithm i.e. **$\log_2$**)

- Measures **homogeneity of a node**
  - **Maximum = $\log n_c$** when all records of node equally distributed among classes
  - **Minimum = 0.0** when all records of node belong to one class

Entropy

- Entropy **measures information in a node**
  - Yes, you can measure amount of information!
  - **Intuitively:** when **all records of node belong to one class**, implies **most information**. Wen **records of node belong to different classes**, **least information**
  - Try to maximize **amount of information gain**
  - Entropy based computations **similar to Gini Index computations**.

Entropy

- Examples: calculating the **Entropy for** various **nodes**

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

A node with: 0 rec in class 0, 6 rec in class 1

Class 1 (C1)	**0**
Class 2 (C2)	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1 (rel. fr/prob.)

Entropy = – $0\log_2 0 - 1\log_2 1 = 0$ (note: 0log0 = 0)

Class 1 (C1)	**1**
Class 2 (C2)	**5**

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

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

Class 1 (C1)	**2**
Class 2 (C2)	**4**

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

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

Entropy

- Entropy of split
  - Assume **node p** is split into **k children**, then **Entropy of split (or Entropy of attribute):**

$$Entropy_{SPLIT} = \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$$

…where **n_i = number of records at node i, Entropy(i) = Entropy of node/child i, n = number of records at node p.**

Note: Compare to Gini Index of Split: Similar!

Entropy

- Information gain
  - Measuring the **gain of information** when **splitting on an attribute**. Assume parent node p split into k partitions (children):

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

…where **n_i** number of records at node **i**, **n** # of records at parent and **Entropy(i)** entropy of child i (note similar to Gini)

- **Measures reduction in entropy** because of split. Choose split that achieves most reduction (maximizes GAIN)
- **Disadvantage**: Prefers splits resulting in large number of "pure" nodes/partitions (pure=low intermixing)

Building Decision Trees Phase-1 – **Using Entropy**

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

STEP 1: Compute Entropy for our entire training data (Entropy of node):

Entropy Node = -(2/6)log2(2/6) – (4/6)log2(4/6) = 0.9182

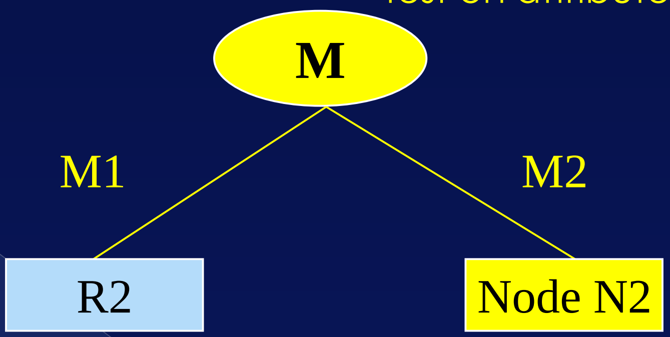
STEP 2: Compute Entropy for each attribute of node i.e. split node based on each attribute (split – use formula on slide 62):
Entropy(M) = (3/6)*0 + (3/6)*[-(2/3)*log2(2/3) – (1/3)*log2(1/3)] = 0.4591
Entropy(N) = (2/6)*0 + (2/6)*1+ (2/6)*1 = 0.6666
Entropy(Q) = (2/6)*1 + (2/6)*1 + (2/6)*0 = 0.6666

STEP 3: Calculate Entropy gains for each attribute:
Entropy Gain(M) = 0.9182 – 0.4591 = 0.4591 (biggest!)
Entropy Gain(N) = 0.9182 – 0.6666 = 0.2516
Entropy Gain(Q) = 0.9182 – 0.6666 = 0.2516

Hence, **first split based on attribute M (biggest gain)**

Building Decision Trees Phase-2 – **Using Entropy**

Root node of tree will be test on attribute M



R1	2
R2	1

If M has value M1, then R is always R2. Homogeneous node! Hence no further splitting and make this a leaf.
We make a rule:
M1 → R2

Here, non-homogeneous node. Apply same method to split based on other attribute. Tree will grow at this branch.

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

Building Decision Trees Phase-3 – **Using Entropy**

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

Calculate again following the same steps (leave out red rows).

STEP 1: Compute Entropy for our **new "entire" node/dataset** (leave out red-ish rows. These have already been classified):

Entropy node = -(2/3)log2(2/3) – (1/3)log2(1/3) = 0.9182

STEP 2: Compute Entropy for each attribute (use formula on slide 62):
Entropy(N) = (1/3)*0 + (1/3)*0+(1/3)*0 = 0
Entropy(Q) = (1/3)*0 + (1/3)*0 + (1/3)*0 = 0

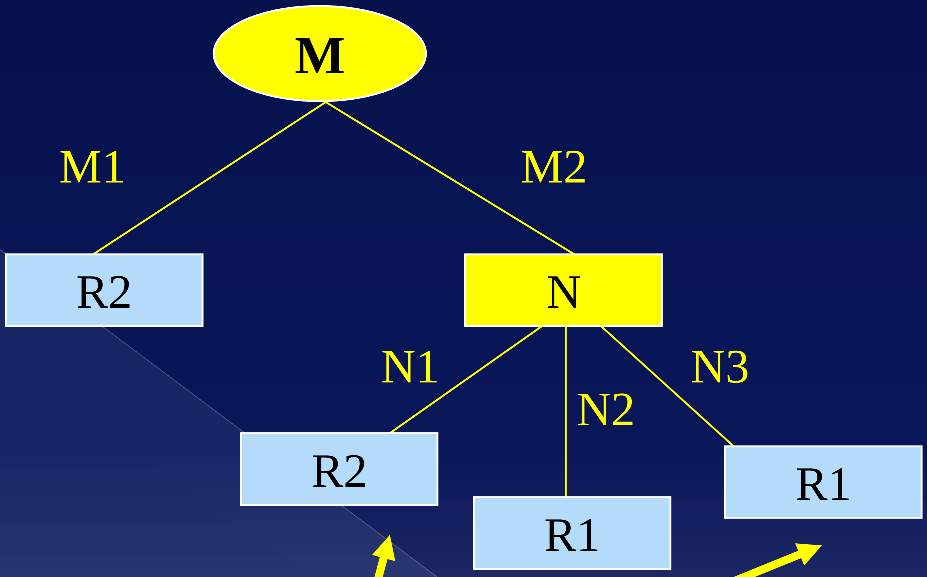
STEP 3: Calculate Entropy gains:
Gini Gain(N) = 0.9182 – 0.0 = **0.9182**
Gini Gain(Q) = 0.9182 – 0.0 = **0.9182**

Equal Entropy gains. Hence, both attributes are equally good. **Choose one of N, Q. Assume we choose N (can also choose Q).**

Building Decision Trees Phase-4 – **Using Entropy**

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2



Homogeneous nodes. Hence, no need to split further. Make them leaves with proper class values (from nodes) and devise rules (include incoming M2):

M2, N1 → R2
M2, N2 → R1
M2, N3 → R1

Dataset empty, hence done.
Decision tree built.

Entropy – Gain ratio

- **Gain ratio**
  - alternative way instead of information gain to solve information gain problems.
- Assume **node p** is split into **k partitions (children)**

$$GainRATIO = \frac{GAIN_{SPLIT}}{SplitINFO} \qquad SplitINFO = -\sum_{j=1}^{k} \frac{n_i}{n} log \frac{n_i}{n}$$

… where **n_i is number of records in partition i**

- **Adjust information gain** by the entropy of the partition. *Large number of small partitions is penalized (i.e. higher entropy partitions)*
- Overcomes disadvantages of Information gain
- Used in C4.5

Entropy

- Example – **Entropy split/ attribute**

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

Entire dataset

Compute Entropy for "Refund" attribute (i.e. compute Entropy of split based on "Refund"). **Cheat is class!**
If we split based on "Refund" we get:

Cheat= Yes	3
Cheat= No	7

Total 10 rec

Cheat= Yes	0
Cheat= No	3

3 out of 10 rec.

Cheat= Yes	3
Cheat= No	4

7 out of 10 rec.

Entropy(Refund="yes") = - (3/3)log(3/3) – (0/3)log(0/3) =0
Entropy(Refund="no") = - (3/7)log(3/7) – (4/7)log(4/7) = 0.9852

Entropy(Refund) = (3/10) * 0 + (7/10) * 0.9852 = 0.6894 QED

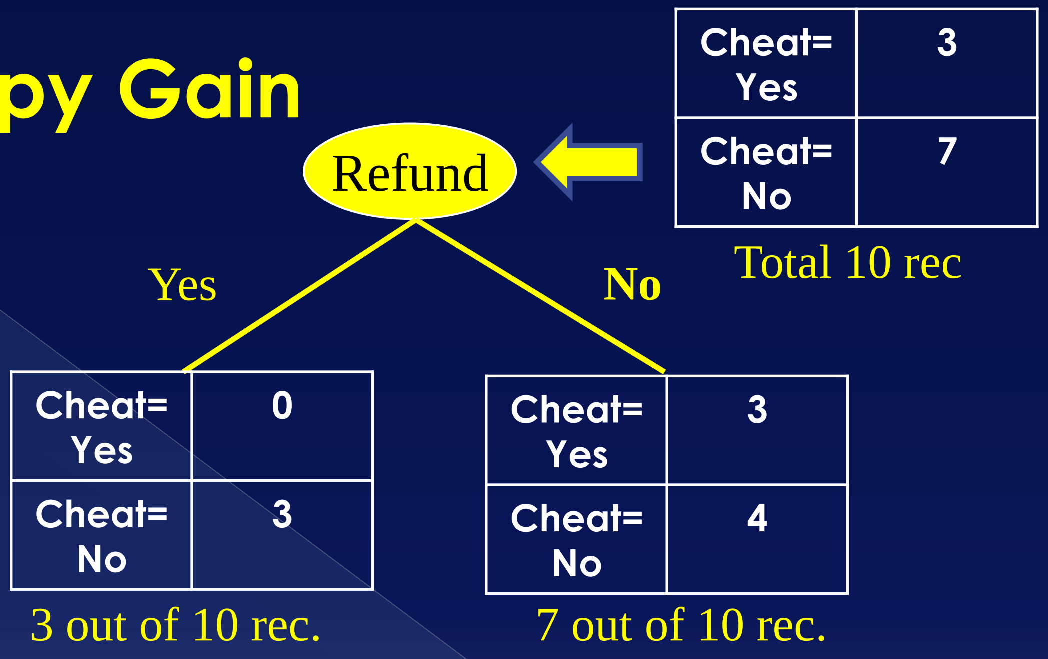
Entropy

Compute **Entropy gain** for split on "Refund" attribute. **Cheat is class!**

- Example – **Entropy Gain**

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

Entire dataset



Calculate Entropy for parent node (entire dataset):
Entropy(dataset) = -(3/10)log(3/10) – (7/10)log(7/10)=
= <u>0.8812</u>

Entropy(Refund) = <u>0.6894</u> (see previous slide)

Entropy Gain = 0.8812 – 0.6894 = <u>0.1918</u> QED

Alternative splitting criteria (besides Gini): Classification error

- Classification error at node i

$$\boldsymbol{Error(i) = 1 - max_i\big(p(i|t)\big)}$$

- Measures misclassification error made by a node.
  - **Maximum = 1-1/n_c** when records are equally distributed among all classes
  - **Minimum = 0.0**, when all records belong to one class

Classification error

- Examples: calculating the **Classification error for** various **nodes**

$$Error(i) = 1 - max_i(p(i|t))$$

A node with: 0 rec in class 0, 6 rec in class 1

Class 1 (C1)	**0**
Class 2 (C2)	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1 (rel. fr/prob.)

Error = 1 – max(0, 1) = 1-1 = 0

Class 1 (C1)	**1**
Class 2 (C2)	**5**

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

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

Class 1 (C1)	**2**
Class 2 (C2)	**4**

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

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

Building Decision Trees

- Using criteria **Entropy and Classification error** to build Decision Trees **in the same way the Gini index is used**

Training Data

ID	M	N	Q	R
1	M1	N3	Q2	R2
2	M2	N3	Q2	R1
3	M2	N2	Q1	R1
4	M1	N2	Q1	R2
5	M2	N1	Q3	R2
6	M1	N1	Q3	R2

STEP 1: Calculate Entropy/Classification error for system

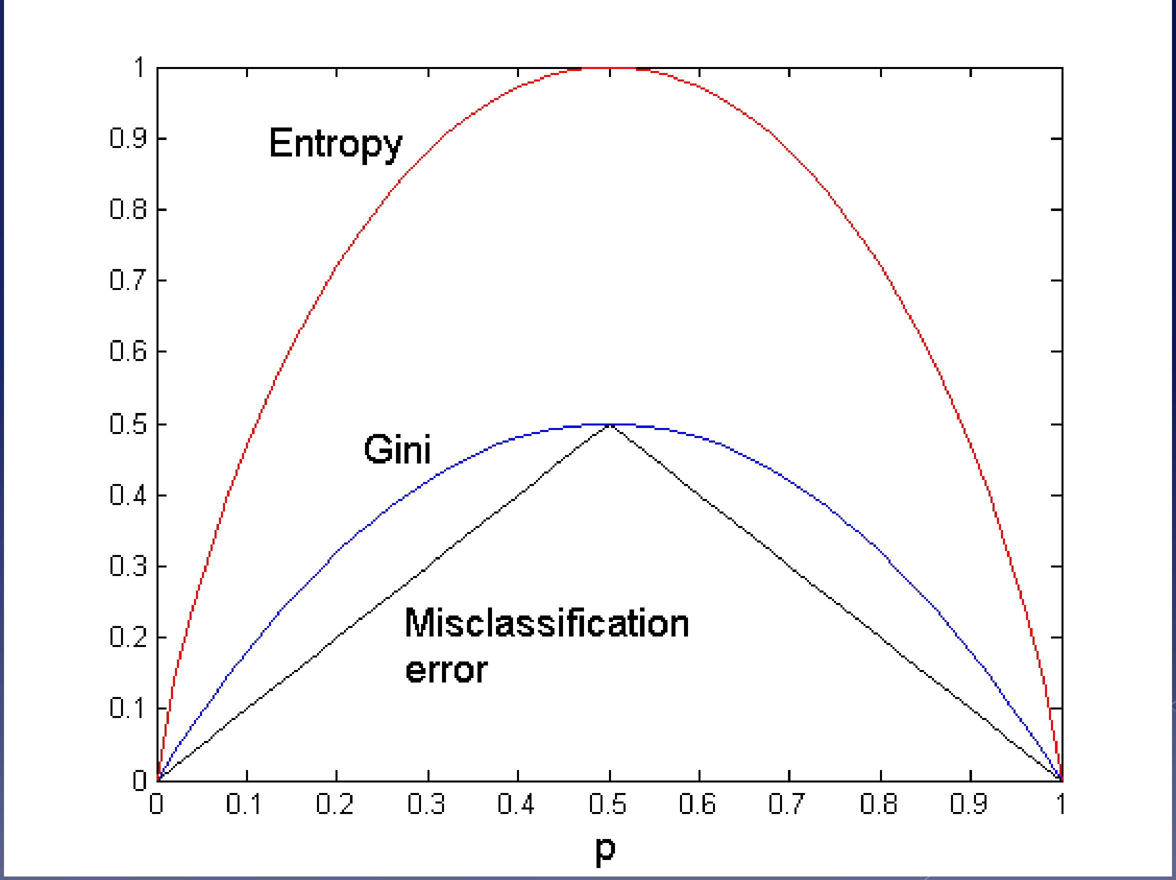
STEP 2: For each attribute compute Entropy/Classification error (Entropy split/attribute etc)

STEP 3: Calculate gains. Choose attribute with biggest gain and make it node….

… etc …

Comparing splitting criteria

- For a **2-class problem** **(Q: why 2-class?)**



Stopping criteria for Tree induction

- Stop expanding **when training set is empty**
- Stop expanding a node **when all the records belong to the same class**
- Stop expanding a node **when all the records have similar attribute values**
- Early termination

Note on evaluation metrics

Evaluation metrics

Note on terminology: in 2 class problems (binary class problems), classes are usually referred to as positive/negative. Could also use Class0/Class1 .

Precision

		Predicted class by model	
		Positive	**Negative**
True class it belongs to	**Positive**	True Positive	False Positive
	Negative	False Negative	True Negative

Recall

- **Accuracy**
  - Correctly identified categories (all)
- **Precision**
  - Proportion of positive class identification was correct = TP / (TP + FP) i.e.
- **Recall**
  - Proportion of actual positives was correct = TP / (TP + FN) i.e. how much of actual positive were identified.
- **F-measure (0 to 1) the greater the better.**
  - Single number for Precision and Recall: F-measure = (2*Precision*Recall)/(Precision + Recall) i.e. harmonic mean!

Important: **Above is a 2 class problem. Yet, can generalize Accuracy/Precision/Recall to problems with more than 2 classes.**

Evaluation metrics

		Predicted class by model	
		Positive	**Negative**
True class it belongs to	**Positive**	True Positive	False Positive
	Negative	False Negative	True Negative

- **When to use Precision, Recall, F-measure?**
  - When your classification dataset is imbalanced
    - When the distribution of rows in known classes is biased or skewed in the training set!
      - More clearly: don't have the same amount of obs in each class in your training set. Severe
  - Type of research
    - E.g. In medicine, *false negative* much more important than say *false positive*. Hence, **Recall** much more important. For YT recommendations, **Precision** better.
  - If balanced, accuracy is fine.

Decision Trees in R

Decision Trees in R

- In R, two ways of building and using Decision Trees
  - Using the **rpart package**
    - Recursive partitioning for classification
    - Documentation
      - See: **https://cran.r-project.org/web/packages/rpart/rpart.pdf**
  - Using the **tree package**
    - Example in R presented in next slides
      - Code in next slide(s) uses **Carseats** dataset of **package ISLR**, a simulated dataset **containing sales of child car seats at 400 different stores**

Decision Trees in R

- **Building/training** a decision tree in R (**tree library**)

```
#includes the Carseats dataset, a simulated dataset containing sales of child car seats at 400 different stores
library(ISLR)
#library for Classification and Regression Trees
library(tree)

#Add Carseats dataset to R's path making thus Carseats dataset available to R
attach(Carseats)
#Take a quick look at the data (peek at data)
head(Carseats)

# The aim of this example is to predict the value for the Sales attribute in the Carseats dataset (i.e. class=Sales). However,
# Sales attribute is continuous hence we have to transform it into a discrete (=categorical) variable.

#take a look at the values of attribute Sales in the Carseats dataset
range(Sales)
#Sales is a continuous attribute ranging from 0.00 up until 16.27. We make an assumption and say that if Sales >= 8 then Sales is High,
#otherwise not.
High = ifelse(Sales>=8, "Yes", "No")
# Now High is a list containing Yes/No values, one for each record in dataset
# We must now attach High to the Carseats dataset, increasing its dimension by 1.
Carseats = data.frame(Carseats, High)
# The Sales attribute is no longer needed, since we have transformed it into a discrete value as specified by High. Sales
# is the first attribute of the Carseats dataset, so get rid of it (i.e. remove it from the Carseats dataset). Dimension reduces by 1
Carseats = Carseats[,-1]
# We have made our class attribute a distinct value. Now, select randomly some records from Carseats in order to create the training
# dataset
set.seed(2) # initialize random number generator. Note: if you keep 2, the same records will always be selected
# Create training set. Get 200 random numbers from 1 to 400.
train = sample( 1:nrow(Carseats), nrow(Carseats)/2)
test = -train #the rest will be our testing data
# Train contains the indexes of the records in Carseats that will be included into the training set. Create the actual dataset
training_data = Carseats[train,]
# Create the decision tree using the training data. We want to predict High based on
# all other attributes of the Carseats dataset
tree_model = tree(High~., training_data)
# Decision tree built. Variable tree_model holds now our Decision tree as it has been created from the training set. Now visualize it
plot(tree_model)
# Plot does now show labels on Decision Tree. Plot tree with labels to make it easily understandable.
text(tree_model, pretty=0)
# Next, use the testing data to test the Decision tree referenced by tree_model
```

Decision Trees in R

- **Using Decision tree** to classify testing data in R (**tree library**)

```
#...continued from previous slide…
# Now that we have built/trained our decision tree, apply it on the testing dataset

# First, select records from Carseats that will comprise our testing dataset. Note:
# the testing dataset has the High attribute but we will not remove it. This is
# because we will need this to calculate accuracy and error rate. In addition, the
# tree library will ignore this attribute anyway.
testing_data = Carseats[test,]
# Peek at testing data
head(testing_data)
# Predict the class attribute (High) for the testing dataset. Apply testing dataset
tree_predict = predict(tree_model, testing_data, type="class")

# Prediction done. Now tree_predict is a one dimensional data structure (separate
# from testing dataset) that holds one value “Yes”/”No” for each record in testing
# set. I.e. the first value in tree_predict corresponds to the first record in
# testing set.

# Now, try to evaluate how well our testing data was classified by calculating the
# Confusion Matrix. There are two ways to do this:

# Using the 'table' command, which
# compares tree_predict and attribute High from testing dataset like this:
testingDataConfusionTable = table(tree_predict, testing_data$High)

# Ok, let's calculate some quality metrics of the model, in order to see how
# our model performed. We calculate accuracy and error rate of the classifier/mocel

# Calculate accuracy
modelAccuracy = sum( diag(testingDataConfusionTable)/sum(testingDataConfusionTable))

# Calculate Error rate (note could also use 1-modelAccuracy)
modelErrorRate = 1 - sum( diag(testingDataConfusionTable)/sum(testingDataConfusionTable))

# Print the result out nicely. We looooooooooove nice and clear responses.
sprintf("Model accuracy: %f, model error rate:%f", modelAccuracy,modelErrorRate )

# You can also use a different library (instead of table) for calculating the confusion matrix
library(caret) #needed for confusion matrix. Install this package (may take a while)
# Compare tree_predict and attribute High from testing dataset by showing the confuction
# matrix. Look at matrix and accuracy
confusionMatrix(tree_predict, testing_data$High)
```

Evaluation

Practical issues of classification

- Important aspects of classification using Decision Trees
  - **Underfitting and Overfitting**
  - How to cope with **missing values?**
  - **Cost** of classification

Underfitting and Overfitting

- There are two types of errors when dealing with decision trees
  - **Training errors** (aka resubstitution errors aka apparent error)
    - Number of misclassifications of records in the training set
  - **Generalization errors**
    - Expected errors of misclassifications when model is applied on unknown records (or the testing set)
- How to **measure misclassification?** (see confusion matrix)
  - Absolute number of records wrongly classified
  - Pct of records wrongly classified

Underfitting and Overfitting

- Misclassification on training set? Isn't this impossible?
  - Given **consistent training set**, will algorithms produce zero error on training set?
  - Considering termination conditions:
    - Training set empty: **zero error**
    - All records have the same class: **zero error**
    - No attributes left to split: **impossible** (consistent training set)
    - No attribute with positive information gain: **possible bur unusual**
  - So, in general, **under such circumstance it is impossible** (occasional possible)

Underfitting and Overfitting

- Misclassification on training set? Isn't this impossible?
  - Assume now **inconsistent training set**
  - Consider termination conditions
    - Training set empty: **zero error**
    - All instances have the same class: **zero error**
    - No attributes left to split on: **inconsistent class. Choosing most common class minimizes error**
    - No attribute has positive information gain: **possible but unusual**
  - So, **in such situations, it is possible**. You have a **minimum error** on the classification of the training set (training error)

Underfitting and Overfitting

- Reasons for **errors on the training set**
  - Training set is **not a good sample**
  - (consistent) **Noise** on some attributes
  - **Missing** attribute values
  - Some **class values present in small amounts**

Underfitting and Overfitting

- Are **training errors and generalization errors** –for a particular model-**related**?
- Is the following true?
  - *"Assume two models (also called Hypothesis) A and B over the same training set. If training_error(A) < training_error(B) then Generalization_error(A) < Generalization_error(B)"* ?
    - **No. That's WRONG!**

Underfitting

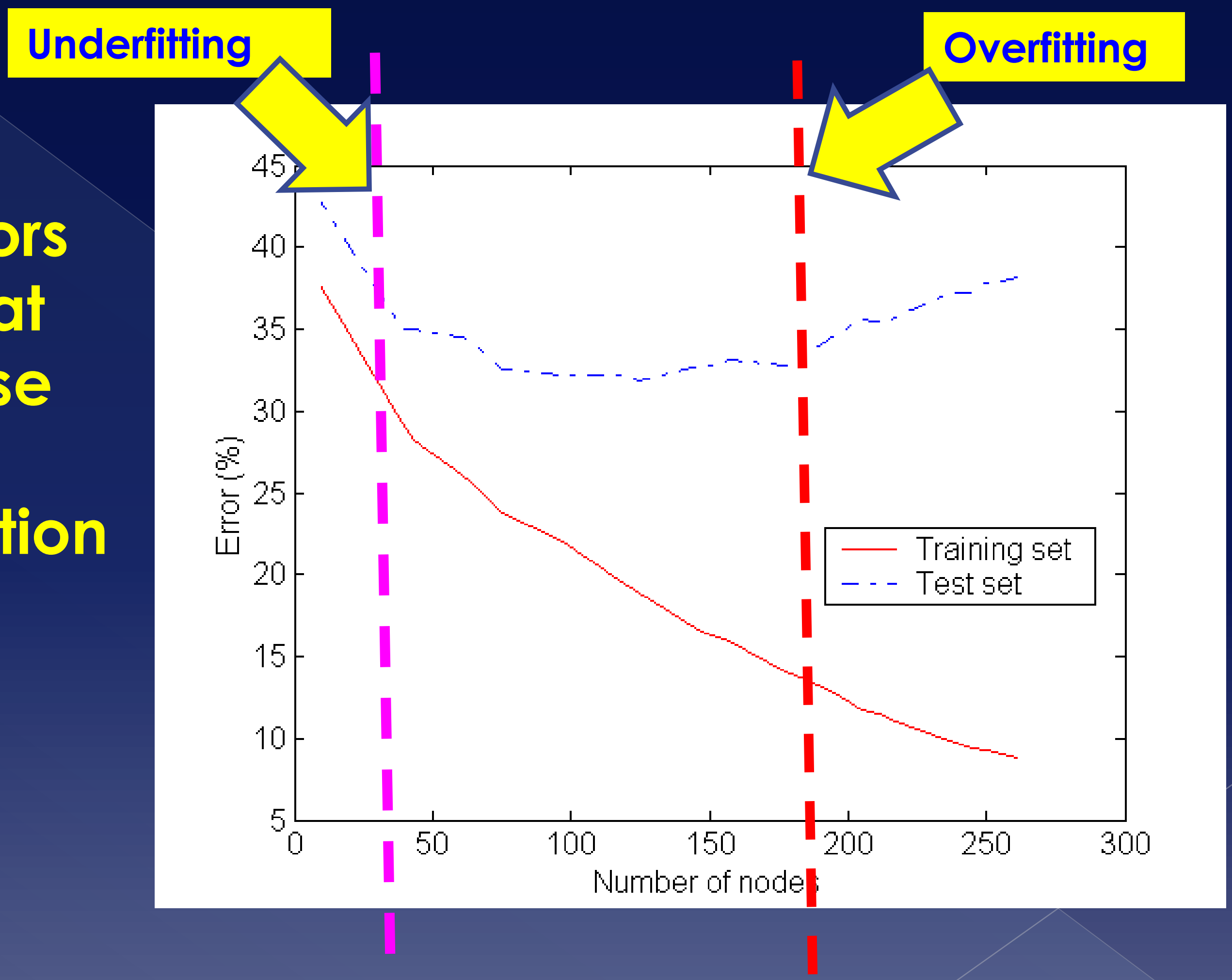
- Underfitting
  - When both **training errors and generalization errors are large**.
    - Happens when **Decision Tree is too simple** (i.e. small number of nodes)
    - When Decision tree has few nodes, it can’t deduce the structure/relationships of attributes
      - Some data will not fit. Hence errors.

Overfitting

- Overfitting
  - When **training error is very small** but **generalization errors are large**
    - Happens **when the Decision Tree adapts too well on the training data**. I.e. perfectly fits training data and hence **works only for training data !**
    - Happens when **Decision Tree grows large and complex (large number of nodes)**

Underfitting and Overfitting

- Overfitting
  - **Testing errors decrease at the expense of the generalization error!**



Notes on overfitting

- Overfitting results in decision trees that are **more complex than necessary**
  - **Complex? =>** More nodes
- Training error **no longer provides a good estimate** of how well the tree will perform on previously unseen records
- Need **new ways for estimating errors**
  - Important step to improve complex decision trees

Estimating generalization errors

- Assume
  - **e(t)** = Error on training set
  - **e'(t)** = Error on testing/unknown set (same model)
- Two approaches to estimating generalization errors
  - **Optimistic approach**
    - Assumes that **e(t) = e'(t)**
    - As discussed, rather not good estimation
      - Especially for complex decision trees due to overfitting

Estimating generalization errors

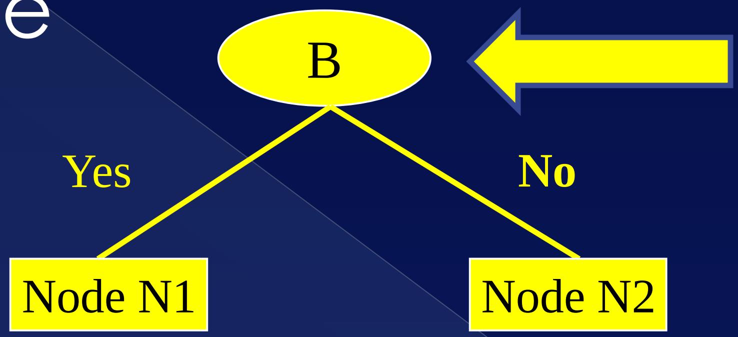
- Two approaches to estimating generalization errors (cont.)
  - **Pessimistic approach**
    - The idea: give complex decision trees a penalty and calculate a pessimistic/increased error due to this complexity
      - **Affects leaves of tree**
    - Pessimistic error for a decision tree (or subtree!) T, $e_g(T)$

$$e_g(T) = \frac{\sum_{i=1}^{k}[e(t_i)+\Omega(t_i)]}{\sum_{i=1}^{k} n(t_i)} = \frac{e(T)+cn_l}{N_t}$$

…where **e(T)** = generalization error of decision tree, **N_t** = number of records in training set, **n_l** = number of leaves in decision tree and **c** = penalty (usually 0.5)

Estimating generalization errors

- Example



	B
C1	6
C2	6
Assume error: 3/12	

	N1
C1	5
C2	2

	N2
C1	1
C2	4

Estimated pessimistic error for decision tree:

$e_g(T) = (3+0.5*2) / 12 = \underline{0.333}$

(**Note:** if there were 3 “children”/nodes (same #records and error): (3+0.5*3)/12 = $\underline{0.375}$ (increased penalty for complexity)

Addressing Overfitting

- Tree pruning
  - **Pruning?** Get rid/remove of some sections of the decision tree
  - Reduces the complexity of the tree (due to removal of nodes)
  - That way **improves accuracy** and **addresses overfitting**
- Two approaches to pruning
  - **Pre-pruning**
  - **Post-pruning**

How to address Overfitting

- **Pre-Pruning (Early Stopping Rule)**
  - Stop the algorithm **before it becomes a fully-grown tree** during the **training phase**
  - 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**
    - Apply χ^2 test to check if class distribution of instances are independent of the available features. If so, stop
    - Stop if **expanding the current node does not improve impurity measures** (e.g., Gini or information gain).

How to address Overfitting

- **Post-pruning**
  - **Grow** decision tree **to its entirety from the training set**
  - **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
    - Use of pessimistic error
  - **Class label of leaf node** is determined from **majority class** of instances in the sub-tree
  - Can use other methods e.g. MDL for post-pruning

Example of Post-Pruning

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

Assume fully grown tree from training set

Training Error (Before splitting) = 10/30

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

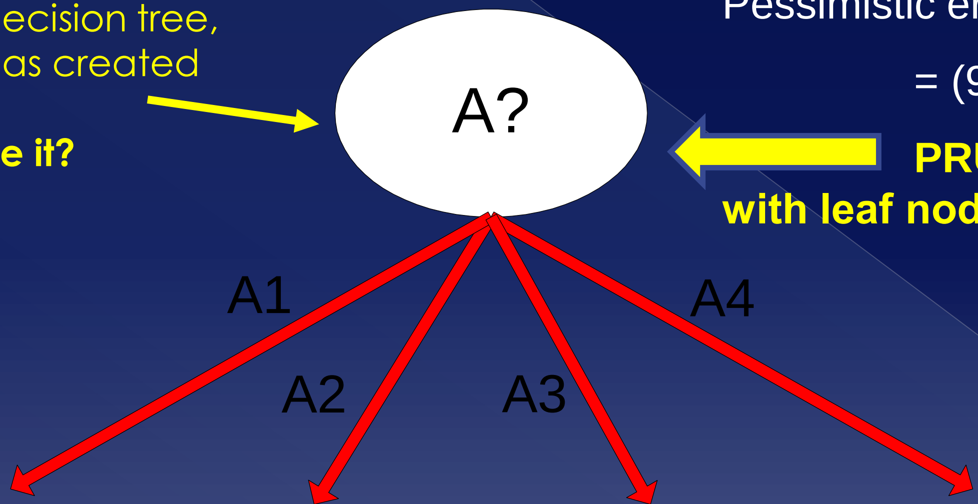
Training Error (After splitting) = 9/30

Pessimistic error (After splitting) =

= (9 + 4 * 0.5) / 30 = **11/30**

PRUNE THIS NODE! Replace with leaf node "Yes" (majority in subtree)

Node exists in decision tree, (forms subtree) as created from dataset.
Should we prune it?



Class = Yes	8
Class = No	4

Class = Yes	3
Class = No	4

Class = Yes	4
Class = No	1

Class = Yes	5
Class = No	1

Estimating generalization errors

- What does **penalty of 0.5 mean**?
  - Means that for binary split, a node should always be grown along children **if classification improves at least one record**
    - Is in general cheaper

Missing values

Handling missing values

- **Missing values** affect decision tree construction in three different ways:
  - Affects how **impurity measures are computed**
  - Affects how to **distribute instance with missing value to child nodes**
  - Affects how a **test instance with missing value is classified**

Missing values – computing impurity measure – Entropy Gain

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Class**
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	$\boxed{?}$	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)

= -0.3 log(0.3)-(0.7)log(0.7) = 0.8813

	Class=Yes	**Class=No**
Refund=Yes	**0**	**3**
Refund=No	**2**	**4**
Refund=?	**1**	**0**

Split on Refund:

Entropy(Refund=Yes) = 0

Entropy(Refund=No) =

= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

Entropy(Children)

= 0.3 (0) + 0.6 (0.9183) = 0.551

Probability of this gain

Entropy Gain = **0.9** * (0.8813 – 0.551) = **0.3303**

Missing values – How to distribute instances

Tid	Refund	Marital Status	Taxable Income	Class
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

Refund

Yes

Class=Yes	0
Class=No	3

No

Class=Yes	2
Class=No	4

Tid	Refund	Marital Status	Taxable Income	Class
10	?	Single	90K	Yes

Probability that Refund=Yes is 3/9

Probability that Refund=No is 6/9

Assign record with unknown Refund to the **left child with weight = 3/9** and to the **right child with weight = 6/9**

Refund

Yes

Class=Yes	**0+3/9**
Class=No	3

No

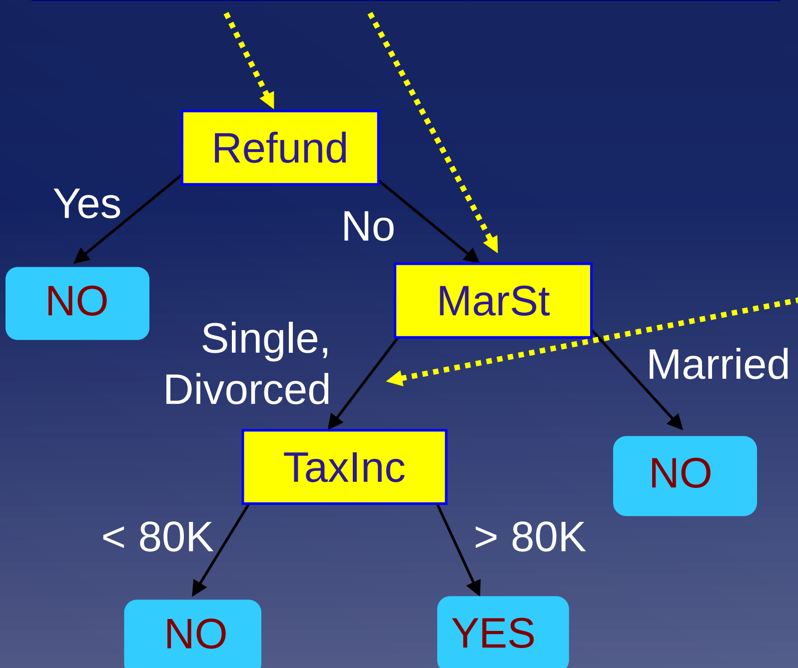
Class=Yes	**2+6/9**
Class=No	4

Missing values – Classifying unknown records

New/unknown record:

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Class**
11	No	**?**	85K	**?**

	Married	Single	Divorced	Total
Class=No	3	1	0	4
Class=Yes	6/9	1	1	2.67
Total	3.67	2	1	6.67



Probability that **Marital Status = Married** is 3.67/6.67

Probability that **Marital Status ={Single,Divorced}** is 3/6.67

Summary

Summary

- **Classification** is the problem of **predicting the value <u>of a categorical attribute</u>** (called the **class**) from a set of categorical/discrete/continuous attributes
- **Decision Trees** are one form of solving this problem
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap
- **Decision Trees** are **created (grown) from training sets**, where the class is known and **applied to testing and unknown data**
- Decision Trees have also problems though
  - **Most important one is Overfitting**

Appendices

Appendix A: Gini index of node

- Proof of the "**Gini Index of node" formula**:

Assume **m classes**, with **i ϵ {1,2,3,…,m}** denoting the class and $\mathbf{f_i}$ the fraction of items of a node labelled as belonging to class i .

Assume now a randomly selected record labelled as belonging to class i. The probability of this random record **to NOT belong to class i** is $\mathbf{(1-f_i)}$.

Since all records are **not drawn with equal probability**, the probability of selecting an item labelled as belonging to class i is f_i. Hence, *drawing a random record with label i, and the selected record not belonging to class i has probability* $\mathbf{P(i) = f_i(1-f_i)}$

where P(i) means the probability of selecting record labelled as belonging to i, and record does not belong to class I (i.e. is error)

The total probability of error will hence be:

$$P(1\ OR\ 2\ OR\ 3\ OR\ \ldots OR\ m) = P(1) + P(2) + P(3) + \ldots + P(m) =$$

$$= \sum_{i=1}^{m} f_i(1-f_i) = \sum_{i=1}^{m} \left(f_i - f_i^{\,2}\right) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^{\,2} = \mathbf{1} - \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{f_i^2}$$

Appendix B: Bibliography

- Classification and Regression Trees. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Wadsworth, Belmont, CA, 1984.
- C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning) by J. Ross Quinlan
- Learning Classification Trees, Wray Buntine, Statistics and Computation (1992), Vol 2, pages 63-73
- On the Boosting Ability of Top-Down Decision Tree Learning
- Algorithms. Kearns and Mansour,, STOC: ACM Symposium on Theory of Computing, 1996"