Data Mining

- **4. Data Mining Classification:**
- **Basic Concepts, Decision Trees, and Model Evaluation**

What is Classification?

- Goal: Previously unseen records should be assigned a class from a given set of classes as accurately as possible
- **Classification :** Classification is the task of learning a **target function** *f* that maps each attribute set **x** to one of the predefined class labels *y*.
- The target function is also known informally as a **classification model**.
- A classification model is useful for the following purposes.
  - Given a collection of records (*training set*)
  - Each record contains a set of *attributes*,
  - One of the attributes is the *class*.

- Find a *model* for class attribute as a function of the values of other attributes.
- 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.

- Variants:
  - Binary classification (e.g. fraud/no fraud or true/false)
  - Multi-class classification (e.g. low, medium, high)
  - Multi-label classification (more than one class per record, e.g. user interests)

Introduction to Classification

A Couple of Questions:

- **What is this?**
- **Why do you know?**
- **How have you come to that knowledge?**

Introduction to Classification

- Goal: Learn a model for recognizing a concept, e.g. trees
- Training data:

"tree"

"tree"

"tree"

"not a tree"

"not a tree"

"not a tree"

Introduction to Classification

– We (or the learning algorithm) look at positive and negative examples (training data)

– … and derive a model

e.g., "Trees are big, green plants that have a trunk and no wheels."

– Goal: Classification of unseen instances

Tree?

Tree?

Warning:
Models are only approximating examples!
Not guaranteed to be correct or complete!

- A decision tree is a flowchart-like structure.
- Each internal node represents a “test” on an attribute
  (e.g. whether a coin flip comes up heads or tails),
- Each branch represents the outcome of the test, and
- Each leaf node represents a class label
  (decision taken after computing all attributes).
- The paths from root to leaf represent classification rules.

- Tree based learning algorithms: one of the best and mostly used supervised learning methods.
- Tree based methods empower predictive models with high accuracy, stability and ease of interpretation.
- Unlike linear models, they map non-linear relationships quite well.
- They are adaptable at solving any kind of problem at hand (classification or regression).
- Decision Tree algorithms are referred to as **CART (Classification and Regression Trees)**.

- **Predictive Modeling:**

  A classification model can also be used to predict the class label of unknown records.

- Classification techniques are most suited for predicting or describing data sets with binary or nominal categories.

- They are less effective for ordinal categories (e.g., to classify a person as a member of high-, medium-, or low income group) because they do not consider the implicit order among the categories.

- Other forms of relationships, such as the subclass–superclass relationships among categories (e.g., humans and apes are primates, which in turn, is a subclass of mammals) are also ignored.

Common terms used with Decision trees:

- **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
- **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
- **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes, whereas, sub-nodes are the child of parent node.

Types of Decision Trees

- Types of decision tree is based on the type of target variable we have.
- Two types:
- **Categorical Variable Decision Tree:**

  Decision Tree which has categorical target variable then it called as categorical variable decision tree. E.g.:- the target variable “Student will play cricket or not” i.e. YES or NO.

- **Continuous Variable Decision Tree:**

  Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Decision Tree Algorithm Pseudocode

- **Decision Tree Algorithm Pseudocode**
- **The decision tree algorithm tries to solve the problem, by using tree representation.**
  - **Each internal node of the tree corresponds to an attribute, and**
  - **each leaf node corresponds to a class label.**

1. Place the best attribute of the dataset at the **root** of the tree.

2. Split the training set into **subsets**. Subsets should be made in such a way that each subset contains data with the same value for an attribute.

- Repeat step 1 and step 2 on each subset until you find **leaf nodes** in all the branches of the tree.

- In decision trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

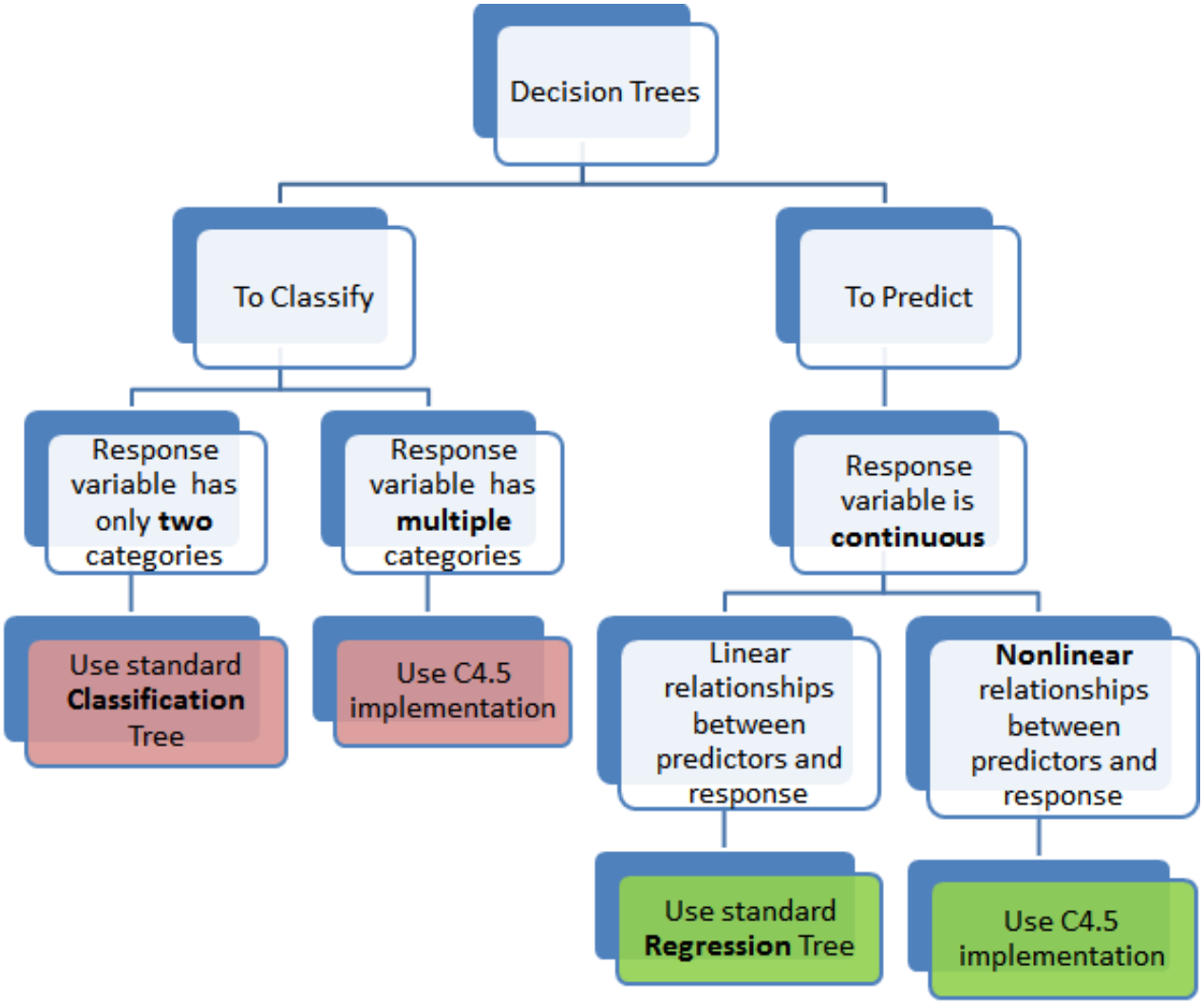
- We continue comparing our record's attribute values with other **internal nodes** of the tree until we reach **a leaf node** with predicted class value. The modeled decision tree can be used to predict the target class or the value.

Assumptions while creating Decision Tree

Some of the assumptions we make while using Decision tree:

- At the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Types of decision trees



Advantages of Decision Tree:

- **Easy to Understand**: Decision tree output is very easy to understand. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive.
- **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. It can also be used in data exploration stage. For e.g., we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.
- Decision trees implicitly perform variable screening or feature selection.
- Decision trees require relatively **little effort from users for data preparation**.
- **Less data cleaning required:** It requires less data cleaning. It is not influenced by outliers and missing values to a fair degree.
- **Data type is not a constraint:** It can handle both numerical and categorical variables. Can also *handle multi-output problems.*
- **Non-Parametric Method:** Decision tree is considered as non-parametric method. i.e decision trees have no assumptions about the space distribution and the classifier structure.
- Non-linear relationships between parameters do not affect tree performance.
- The number of hyper-parameters to be tuned is almost null.

Disadvantages of Decision Tree:

- **Over fitting:** Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning.
- **Not fit for continuous variables**: While working with continuous numerical variables, decision tree loses information, when it categorizes variables in different categories.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called **variance**, which needs to be lowered by methods like **bagging** and **boosting**.
- *Greedy* algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create *biased* trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.
- Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.
- Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
- Calculations can become complex when there are many class label.

Regression Trees vs Classification Trees

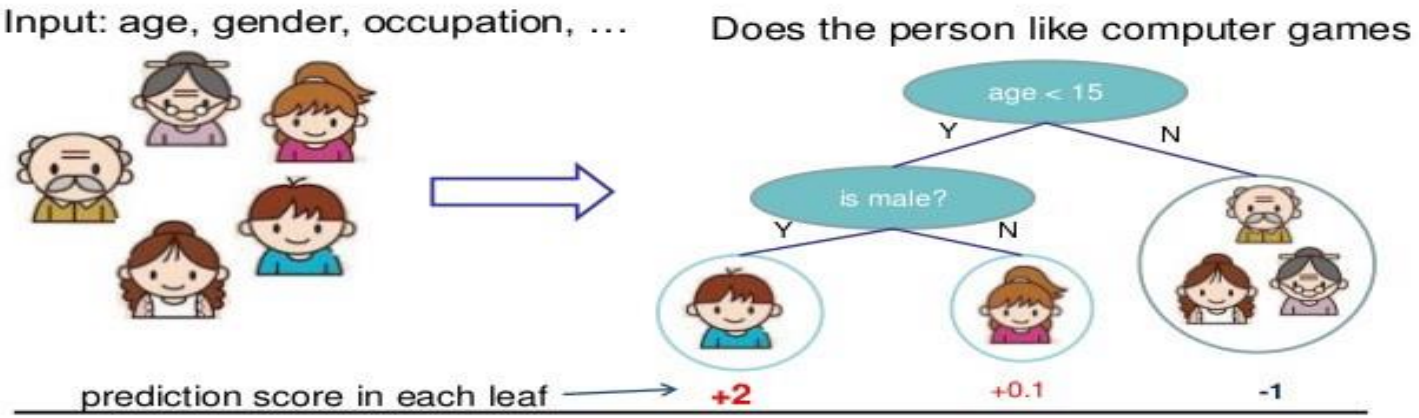
- The terminal nodes (or leaves) lies at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the bottom & roots are the tops.
- Both the trees work almost similar to each other.
- The primary differences and similarities between Classification and Regression Trees are:
- Regression trees are used when dependent variable is continuous. Classification Trees are used when dependent variable is categorical.
- In case of Regression Tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
- In case of Classification Tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
- Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions.
- Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as '*greedy*' because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.

This splitting process is continued until a user defined stopping criteria is reached. For e.g.: we can tell the algorithm to stop once the number of observations per node becomes less than 50.

In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to over fit data, leading to poor accuracy on unseen data. This bring 'pruning'. Pruning is one of the technique used tackle overfitting.

Regression Tree (CART)

- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value



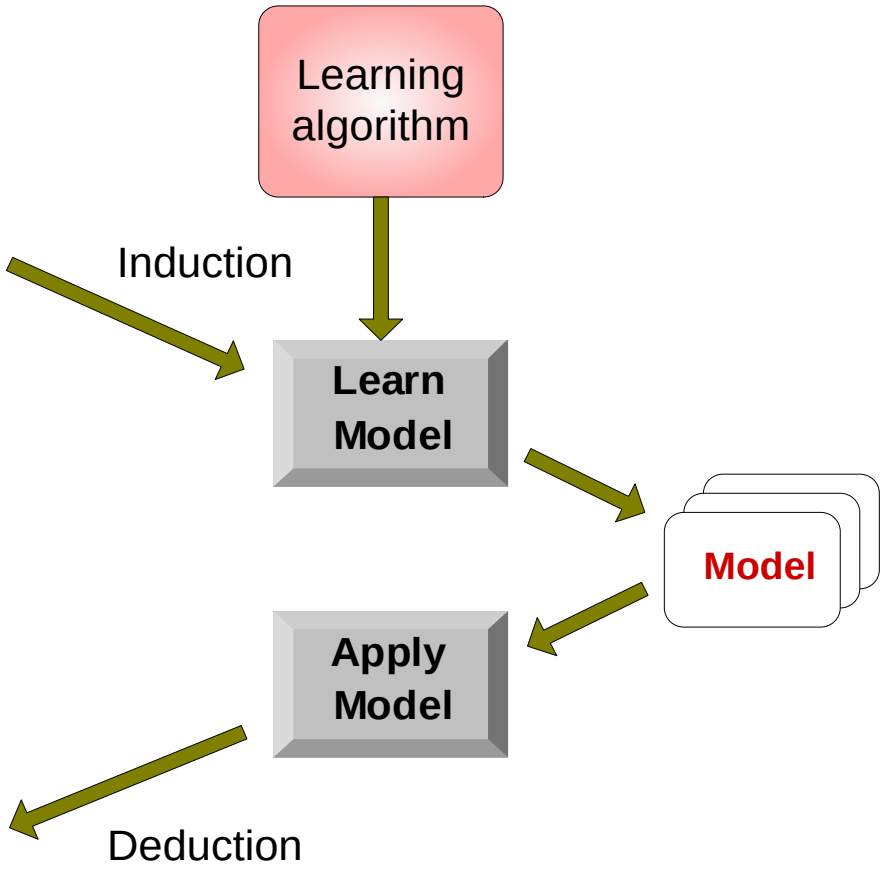
Illustrating Classification Task

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

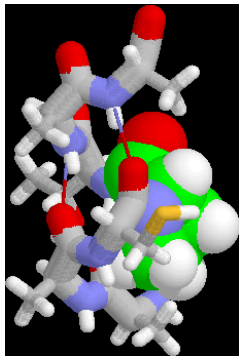


Examples of Classification Task

Credit Risk Assessment
- Attributes: your age, income, debts, …
- Class: are you getting credit by your bank?

Marketing
- Attributes: previously bought products, browsing behavior
- Class: are you a target customer for a new product?

SPAM Detection
- Attributes: words and header fields of an e-mail
- Class: regular e-mail or spam e-mail?



Identifying Tumor Cells
- Attributes: features extracted from x-rays or MRI scans
- Class: malignant or benign cells

Classifying secondary structures of protein
- as alpha-helix, beta-sheet, or random coil

Categorizing news stories as finance, weather, entertainment, sports…

Classification Techniques

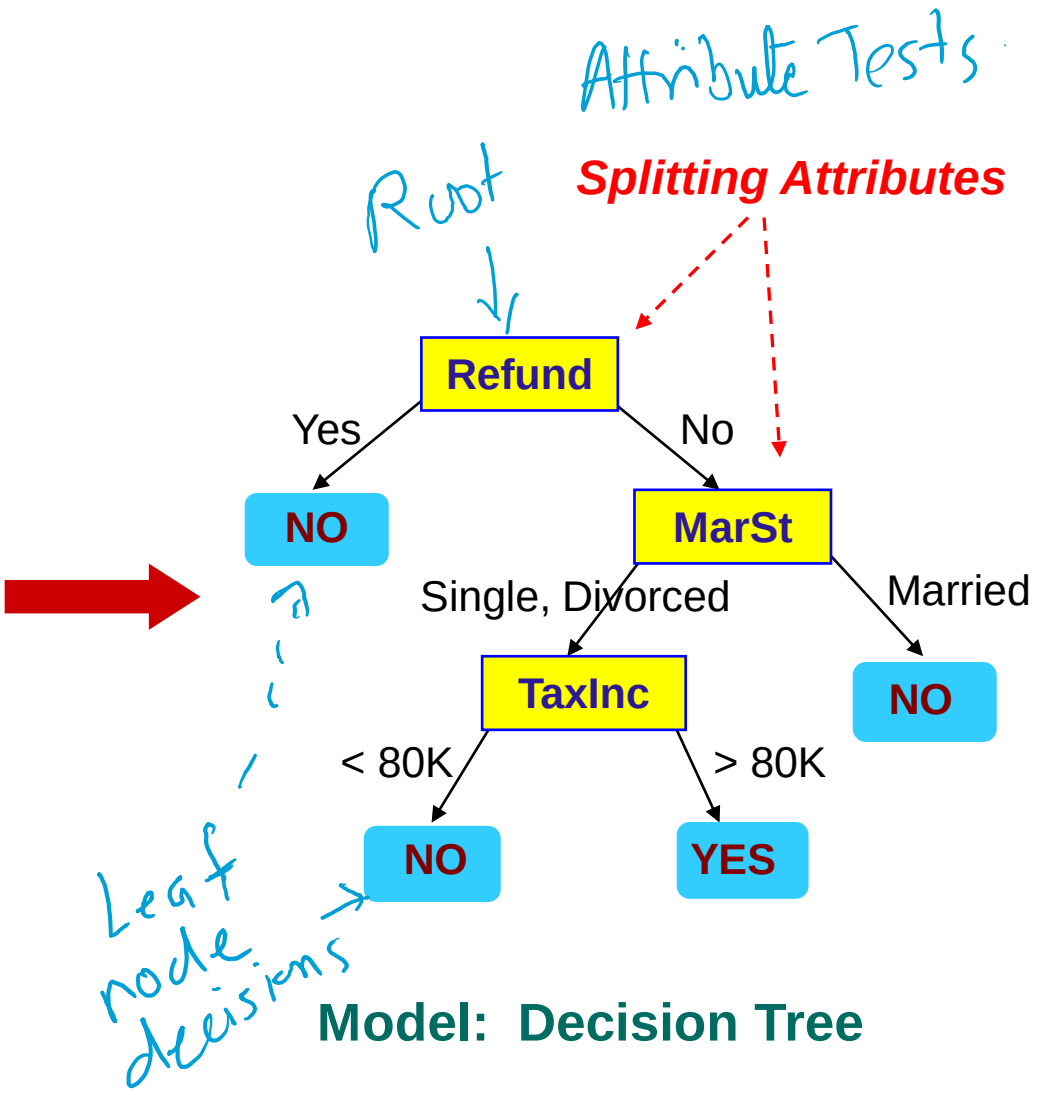
- Decision Tree based Methods
- Rule-based Methods
- Memory based reasoning
- Neural Networks
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines

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

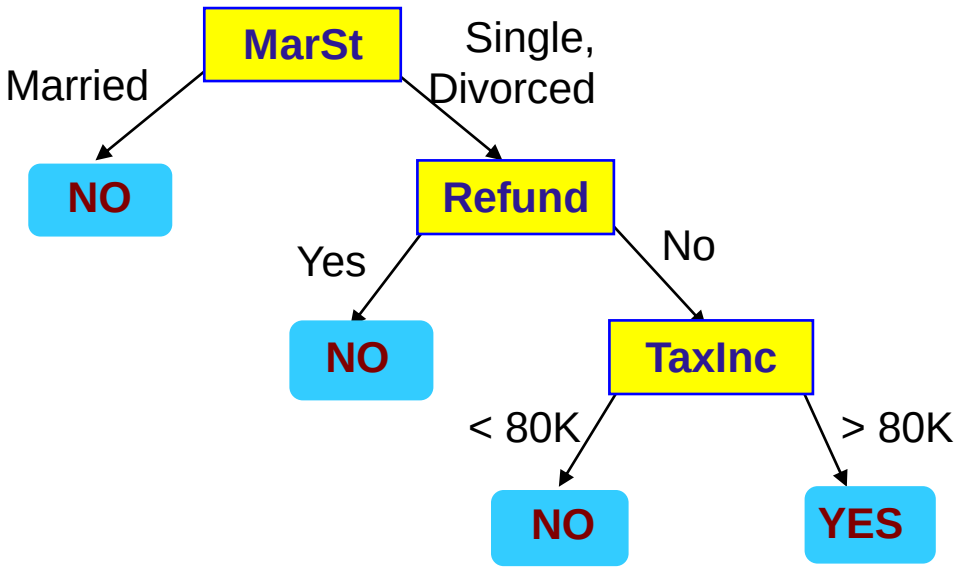
Training Data



Model: Decision Tree

Another Example of Decision Tree

Tid	Refund (categorical)	Marital Status (categorical)	Taxable Income (continuous)	Cheat (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	No	Single	90K	Yes



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

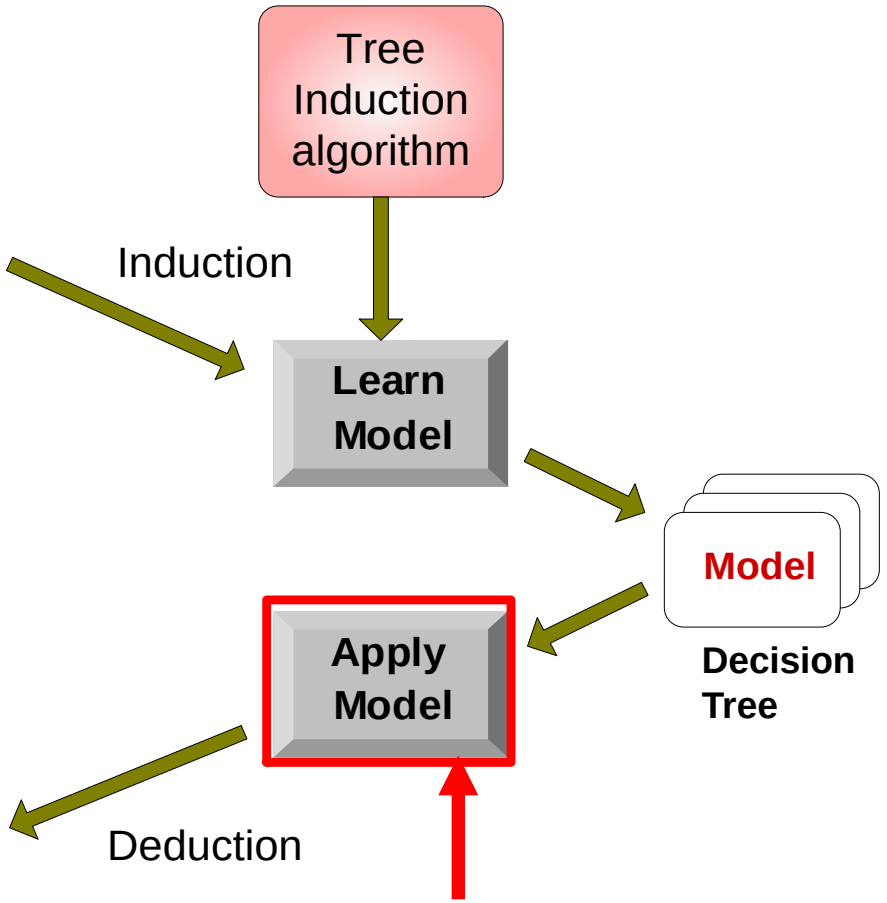
Decision Tree Classification Task

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

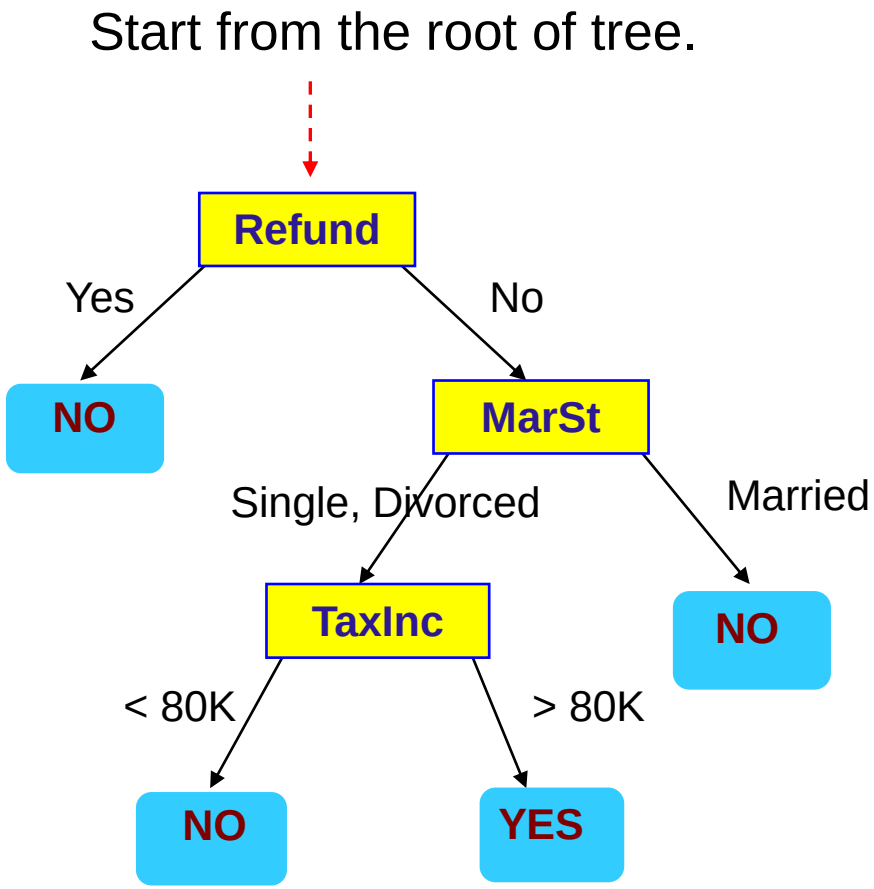
Training Set

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



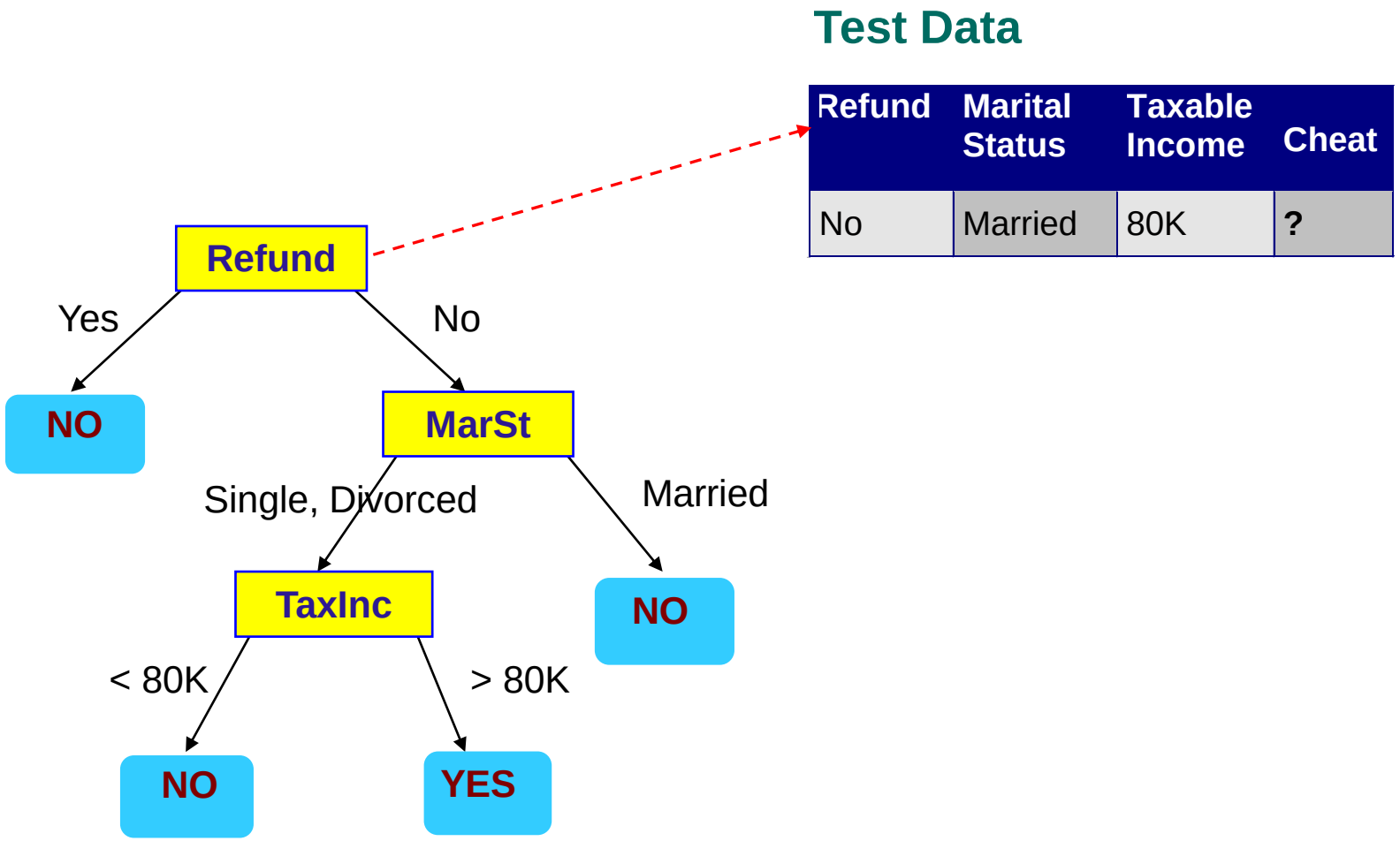
Apply Model to Test Data



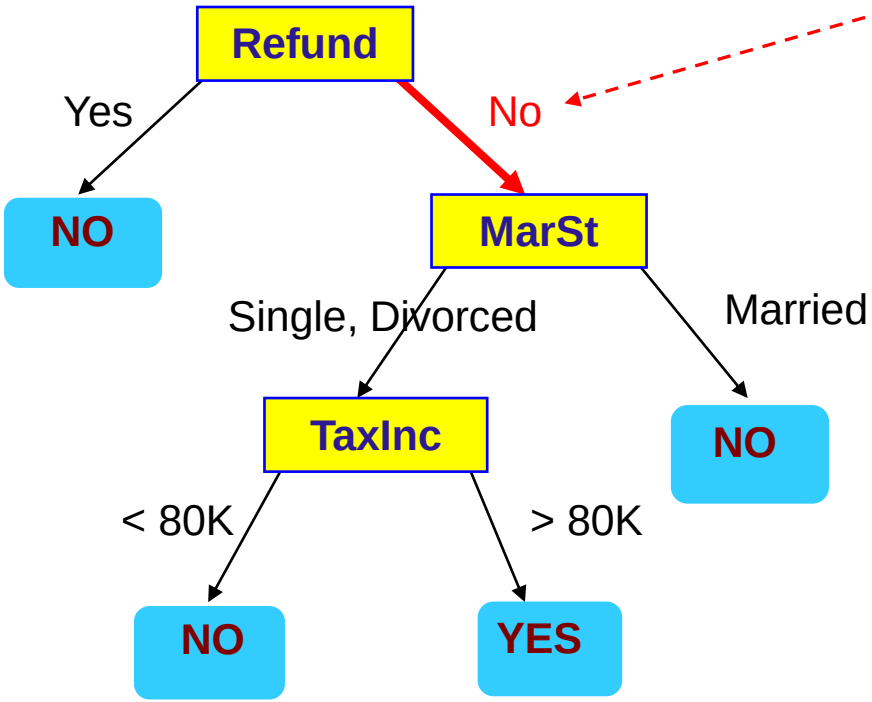
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data



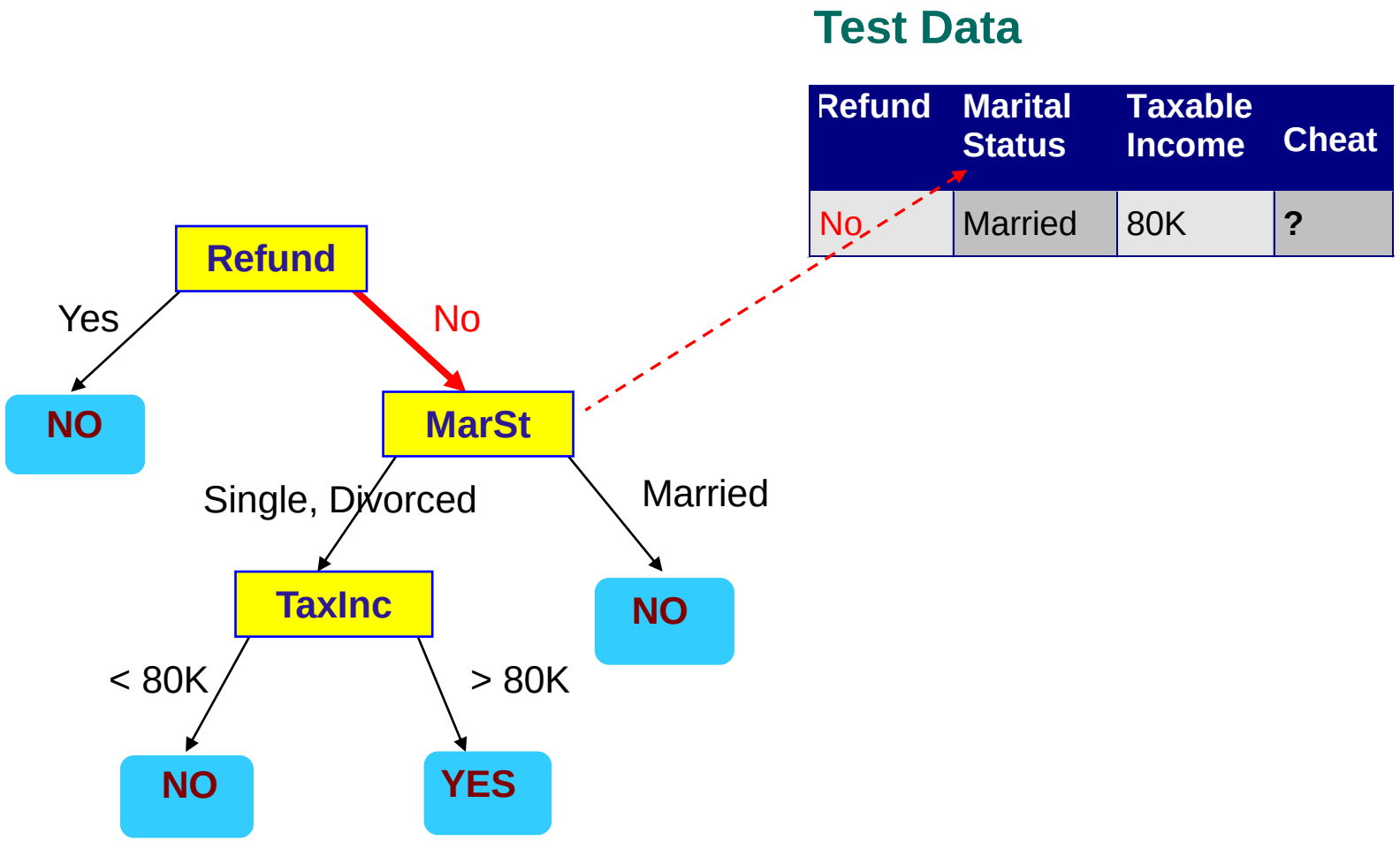
Apply Model to Test Data



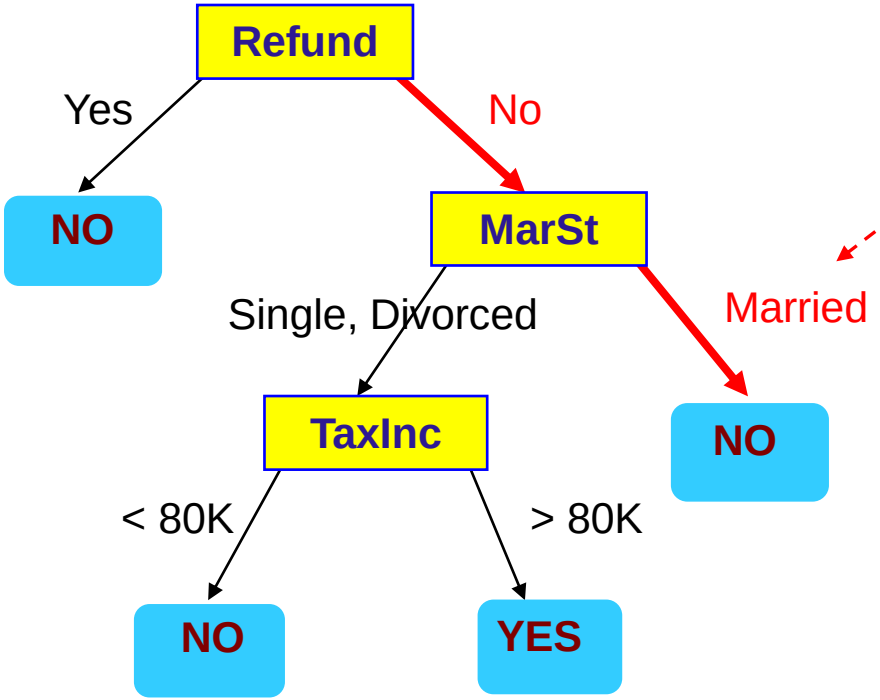
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data



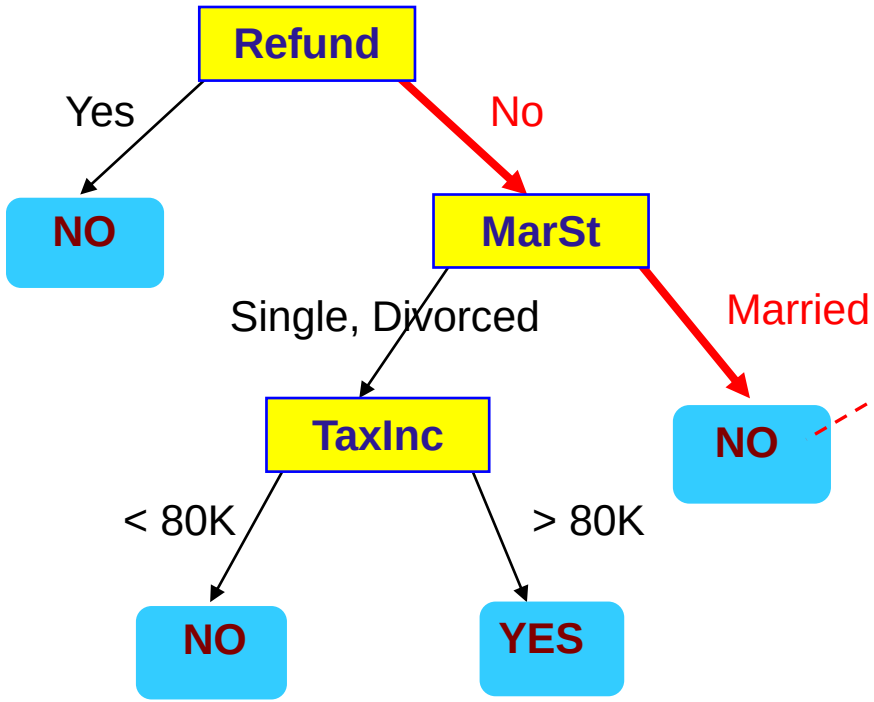
Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to “No”

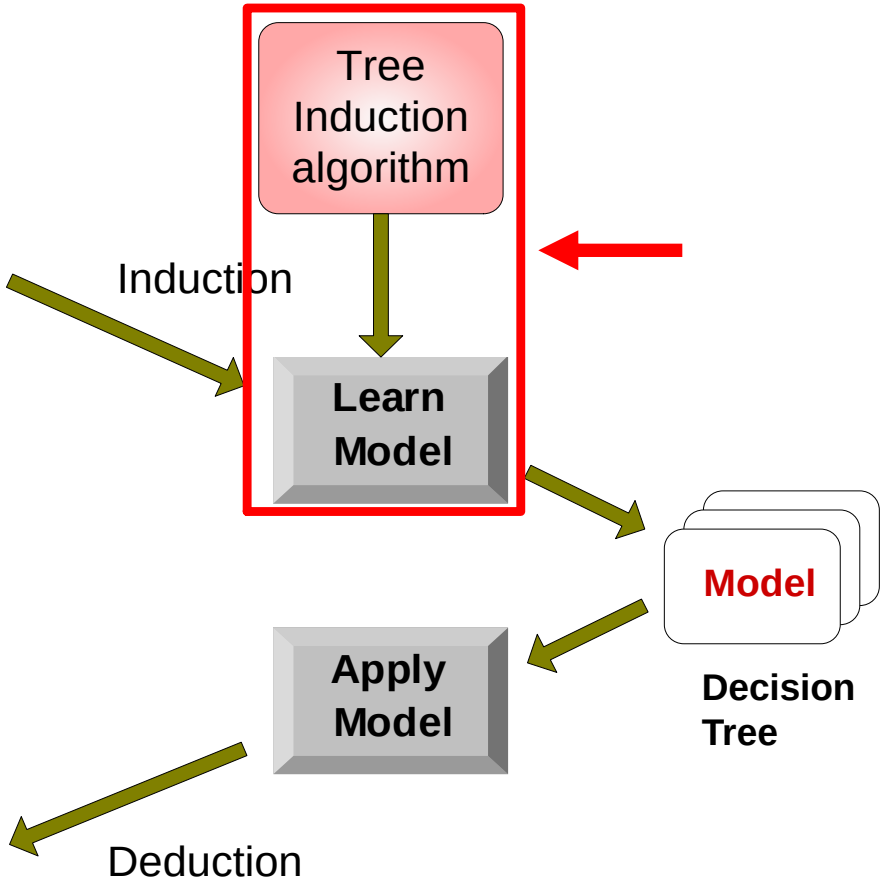
Decision Tree Classification Task

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



Decision Tree Induction

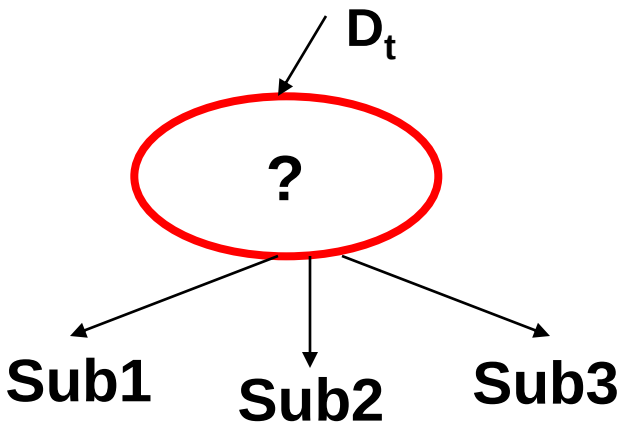
- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ,SPRINT

General Structure of Hunt's Algorithm

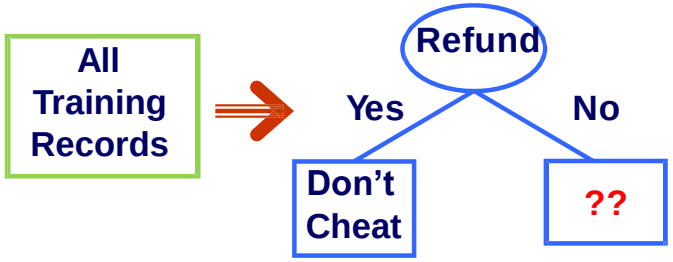
- Let *Dt* be the set of training records that are associated with node *t* and
- *y = {y1, y2, . . . , yc}* be the class labels.
- General Procedure:
- Generate leaf node or attribute test:
- if D_t only contains records that belong to the same class y_t, then t is a leaf node labeled as y_t
- if D_t contains records that belong to more than one class, use an attribute test to split the data into subsets having a higher purity.
  - for all possible tests: calculate purity of the resulting subsets
  - choose test resulting in highest purity

Recursively apply this 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



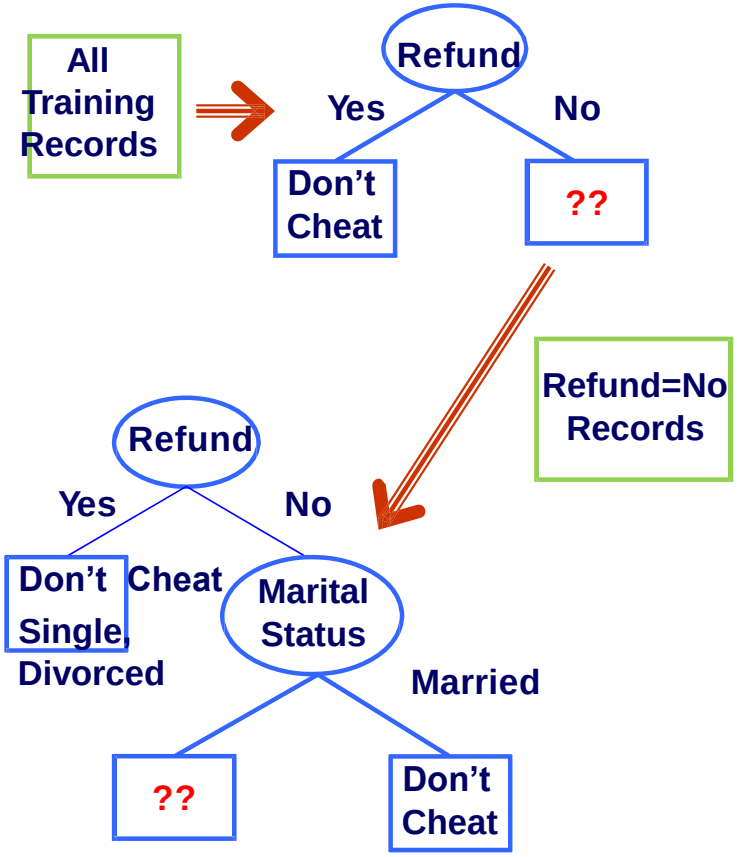
Hunt's Algorithm – Step 1



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

1. We calculate the purity of the resulting subsets for all possible splits
   - Purity of split on Refund
   - Purity of split on Marital Status
   - Purity of split on Taxable Income
2. We find the split on Refund to produce the purest subsets

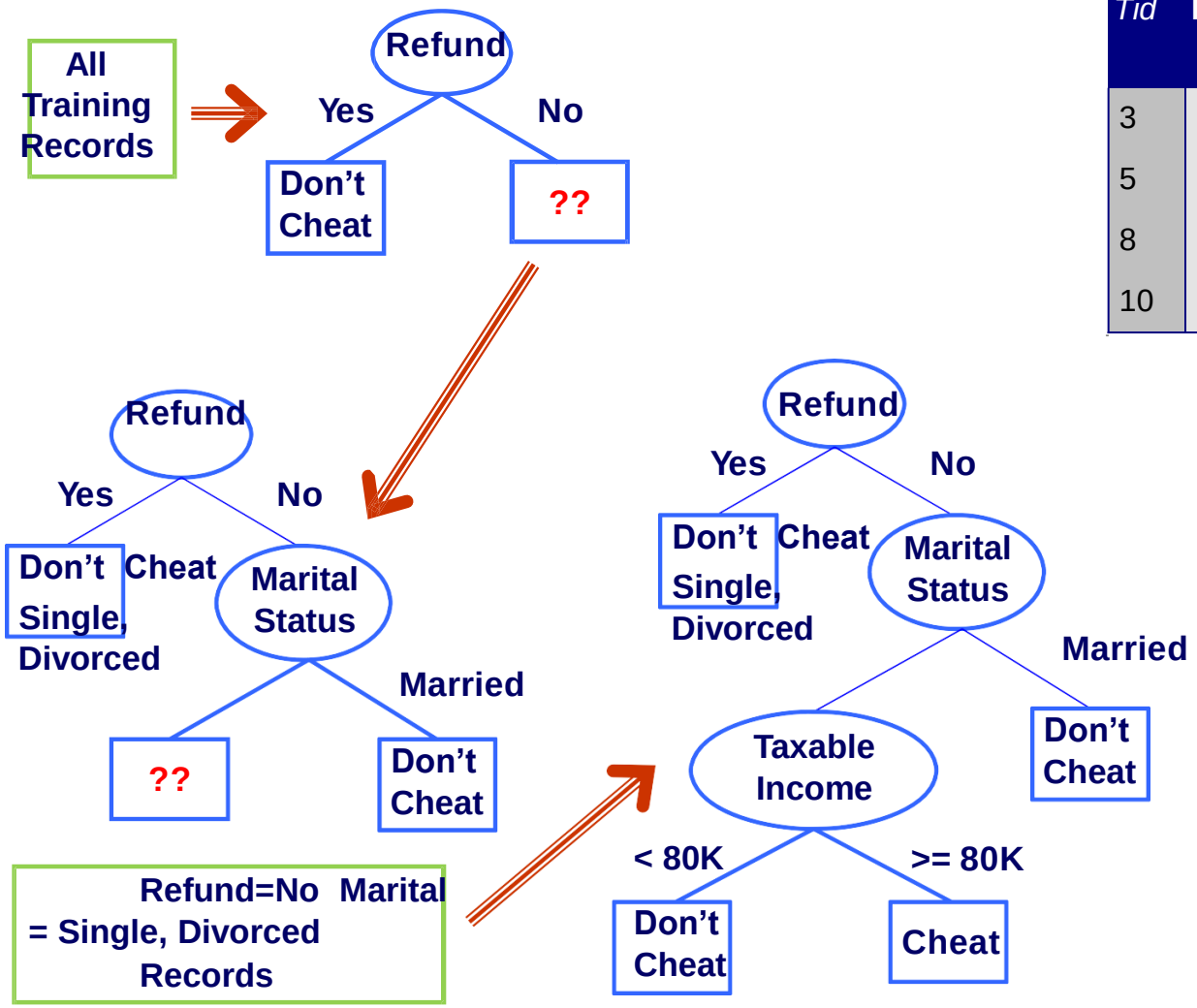
Hunt's Algorithm – Step 2



Tid	Refund	Marital Status	Taxable Income	Cheat
2	No	Married	100K	No
3	No	Single	70K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

1. We further examine the Refund=No records
2. Again, we test all possible splits
3. We find the split on Marital Status to produce the purest subsets

Hunt's Algorithm – Step 3



Tid	Refund	Marital Status	Taxable Income	Cheat
3	No	Single	70K	**No**
5	No	Divorced	95K	**Yes**
8	No	Single	85K	**Yes**
10	No	Single	90K	**Yes**

1. **We further examine the Marital Status=Single or =Divorced records**
2. **We find a split on Taxable Income to produce pure subsets**
3. **We stop splitting as no sets containing different classes are left**

Design Issues of Decision Tree Induction

A learning algorithm for inducing decision trees must address the following two issues.

1. **How should training records be split?**
   - **How to specify the attribute test condition?**
     - **Depends on number of ways to split: 2-way split, multi-way split**
     - **Depends on attribute data type: nominal, ordinal, continuous**
   - **How to determine the best split?**
     - **Different purity measures can be used**

Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.

Design Issues of Decision Tree Induction

2. When should the splitting procedure stop?

- **Shallow trees might generalize better to unseen records**
- **Fully grown trees might overfit training data**

A stopping condition is needed to terminate the tree-growing process.
A possible strategy is to continue expanding a node until either all the records belong to the same class or
All the records have identical attribute values.
Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier.
The advantages of early termination will be discussed later.

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

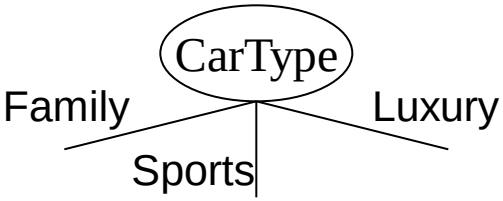
How to Specify Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

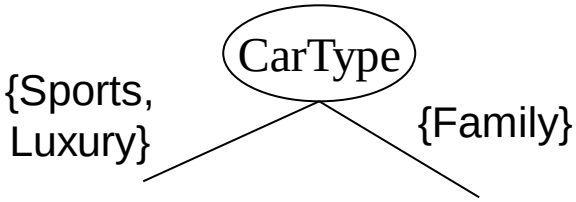
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

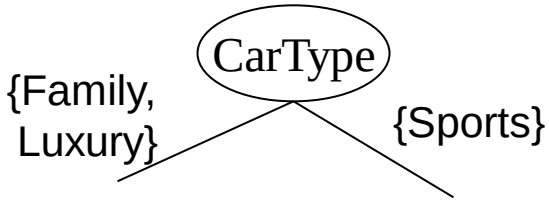
- Multi-way split: Use as many partitions as distinct values.



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

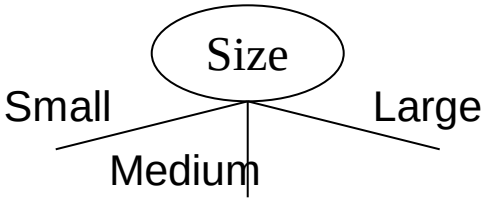


OR

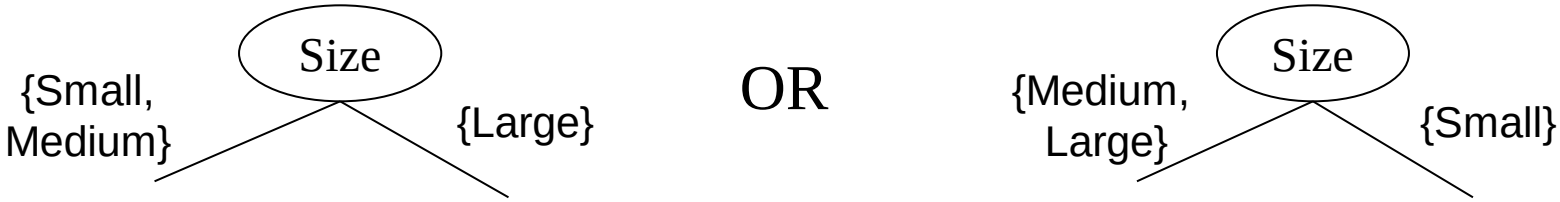


Splitting Based on Ordinal Attributes

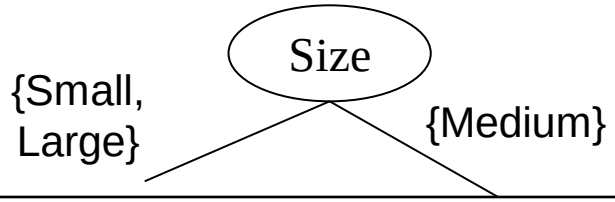
- Multi-way split: Use as many partitions as distinct values.



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



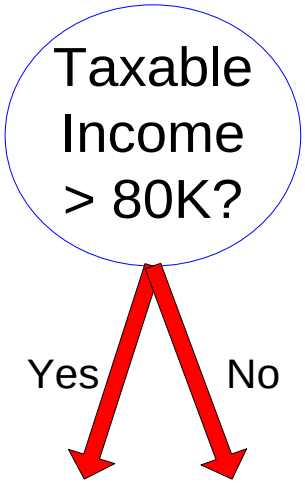
- What about this split?



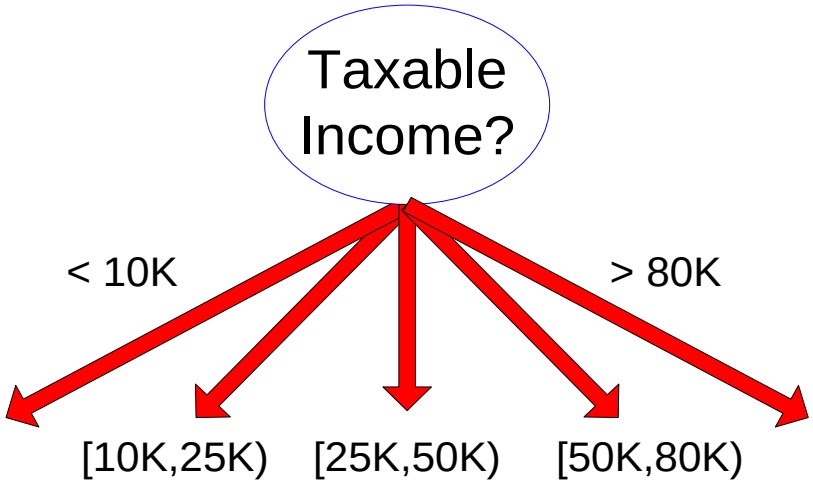
Splitting Based on Continuous Attributes

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

Splitting Based on Continuous Attributes



(i) Binary split



(ii) Multi-way split

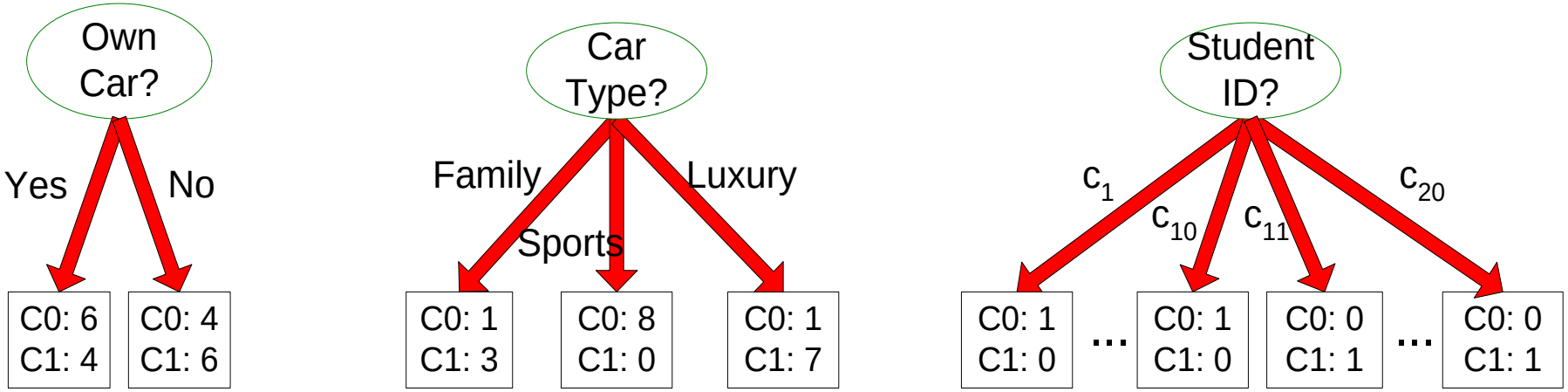
Discretization Example

- Values of the attribute, e.g., age of a person:
  - 0, 4, 12, 16, 16, 18, 24, 26, 28
- Equal-interval binning – for bin width of e.g., 10:
  - Bin 1: 0, 4 [-,10) bin
  - Bin 2: 12, 16, 16, 18 [10,20) bin
  - Bin 3: 24, 26, 28 [20,+) bin
    - denote negative infinity, + positive infinity
- Equal-frequency binning – for bin density of e.g., 3:
  - Bin 1: 0, 4, 12 [-, 14) bin
  - Bin 2: 16, 16, 18 [14, 21) bin
  - Bin 3: 24, 26, 28 [21,+] bin

How to determine the Best Split

In a two-class problem, the class distribution at any node is given by (p0,p1), where, p1=1-p0.

Before Splitting the dataset contains: **10 records of class 0, 10 records of class 1**



Which test condition is the best?

The degree of split is based on the degree of impurity of the child nodes.

Smaller degree of impurity: more skewed class distribution

Class distribution (0,1) : zero impurity; Class distribution (0.5,.0.5) : highest impurity

How to determine the Best Split

- Greedy approach: Test all possible splits and use the one that results in the most homogeneous (= pure) nodes
  - Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, **High degree of impurity**	**Homogeneous,** **Low degree of impurity**

Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error

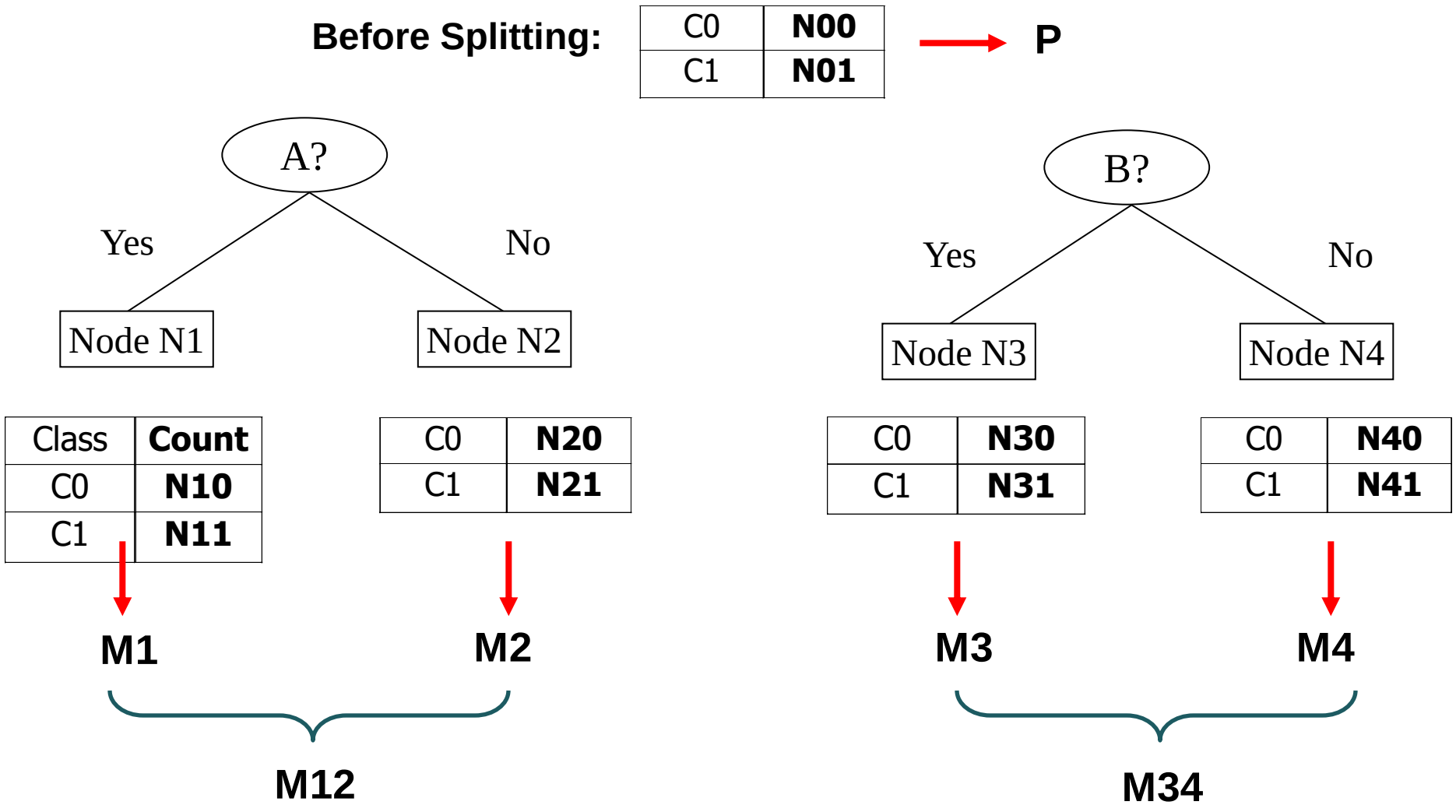
How to Find the Best Split?

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting for all possible splits
   - compute impurity measure of each child node
   - M is the weighted impurity of children
3. Choose the attribute test condition (split) that produces the highest purity gain

$$\text{Gain} = P - M$$

or equivalently, lowest impurity measure after splitting (M)

How to Find the Best Split



Measure of Impurity: GINI

- 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

Class	Count
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 Classification And Regression Tree (CART)
- Supervised Learning In Quest (SLIQ)
- Scalable Parallel Classifier (SPRINT)

When a node *p* is split into *k* partitions (children/subsets), the GINI index of each partition is weighted according to the partition's size

- The quality of split is computed as:

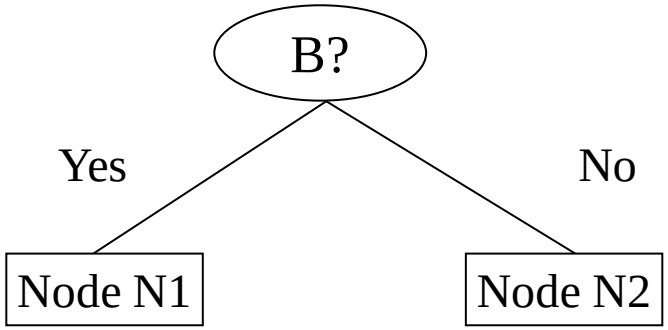
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

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

n = number of records at node *p*.

Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.



	Parent
C1	6
C2	6
Gini = 0.500	

Gini(N1)
= 1 – (5/7)² – (2/7)²
= 0.408

Gini(N2)
= 1 – (1/5)² – (4/5)²
= 0.32

	N1	N2
C1	5	1
C2	2	4
Gini=0.371		

Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.32
= 0.371

  - Purity Gain= 0.5 - 0.371 = 0.129

Categorical Attributes: Computing Gini Index

- 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		

Two-way split
(find best partition of values)

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

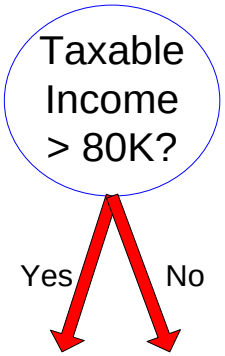
	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

The Multi-way split has smaller GINI index

Continuous Attributes: Computing Gini Index

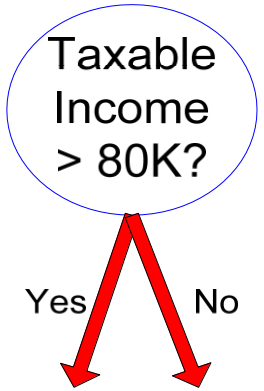
- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A < v and A ≥ v
- Simple method to choose best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

- How to find best binary split for a continuous attribute?
- For efficient computation: for each attribute,
  1. Sort the attribute on values (Annual income <=*v* *is used to split the training records)*
  2. Linearly scan these values, each time updating the count matrix and computing gini index
  3. 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	

Alternative Splitting Criteria based on INFO

- 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
  - Entropy based computations are similar to the GINI index computations

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

Splitting Based on INFO...

- 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 and C4.5
  - Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

Splitting Based on INFO...

- Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO} \qquad SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions;

n_i is number of records in partition i

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

Splitting Criteria based on Classification Error

- 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/nc) :

  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 for Computing Error

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

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

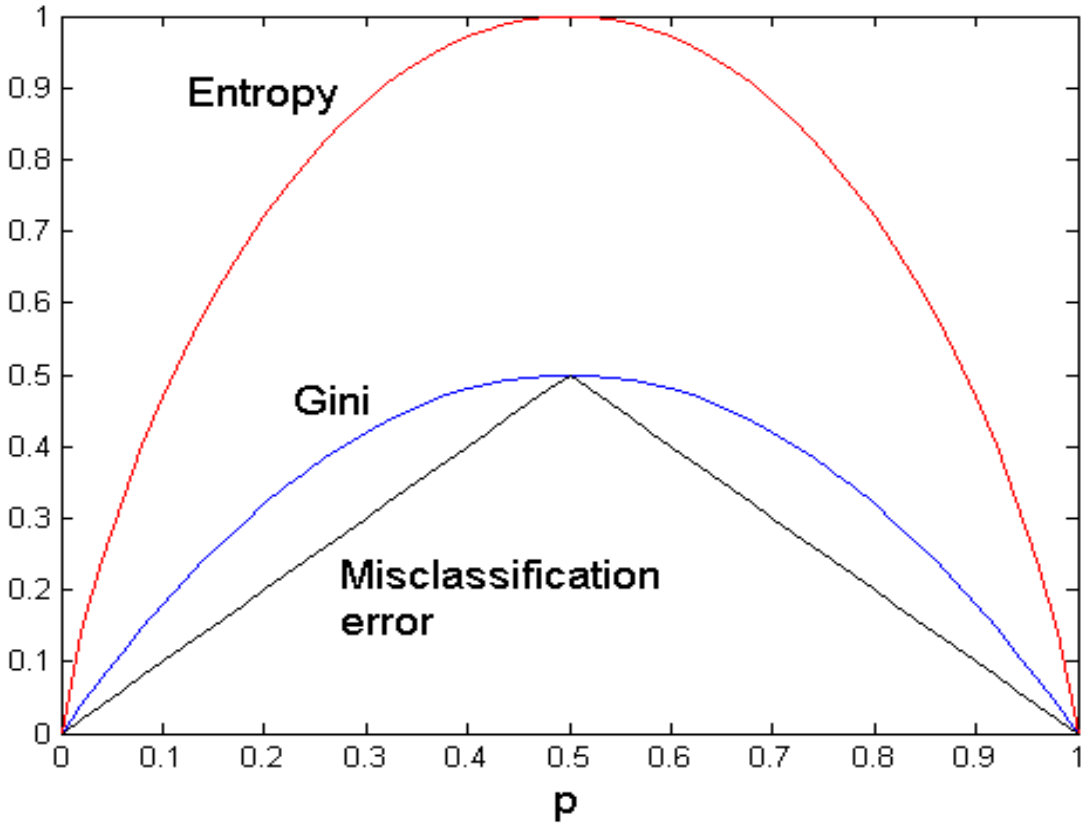
C1	2
C2	4

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

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

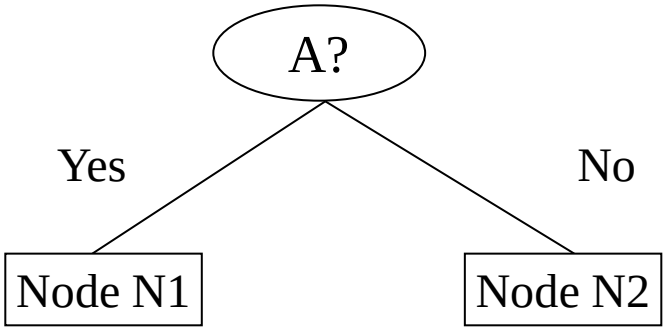
Comparison among Splitting Criteria

For a 2-class problem:



P : fraction of records that belong to one of the two classes.
P = 0.5: Three measures attain max value: uniform class distribution
P = 0 / 1: Three measures attain min value : all records belong to same class

Misclassification Error vs Gini



	Parent
C1	7
C2	3
Gini = 0.42	

Gini(N1)
= 1 – (3/3)² – (0/3)²
= 0

Gini(N2)
= 1 – (4/7)² – (3/7)²
= 0.489

	N1	N2
C1	3	4
C2	0	3
Gini=0.361		

Gini(Children)
= 3/10 * 0
+ 7/10 * 0.489
= 0.342

Gini improves !!

But Error increases!!

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

Stopping Criteria for Tree Induction

- 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 (to be discussed later)

Practical Issues of Classification

- Underfitting and Overfitting
- Missing Values
- Costs of Classification

Other Issues

- Data Fragmentation
- Search Strategy
- Expressiveness
- Tree Replication

Data Fragmentation

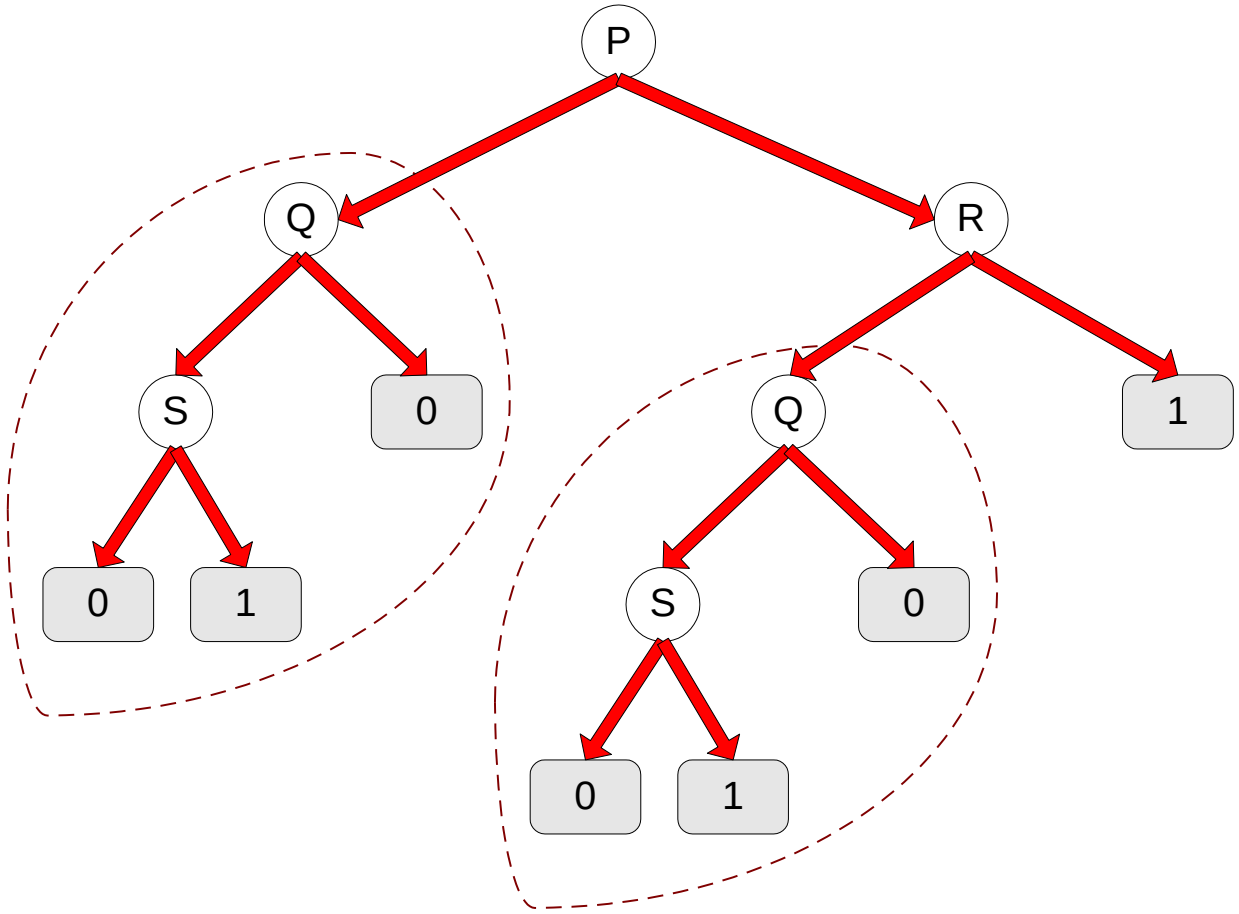
- Number of instances gets smaller as you traverse down the tree

- Number of instances at the leaf nodes could be too small to make any statistically significant decision

Search Strategy

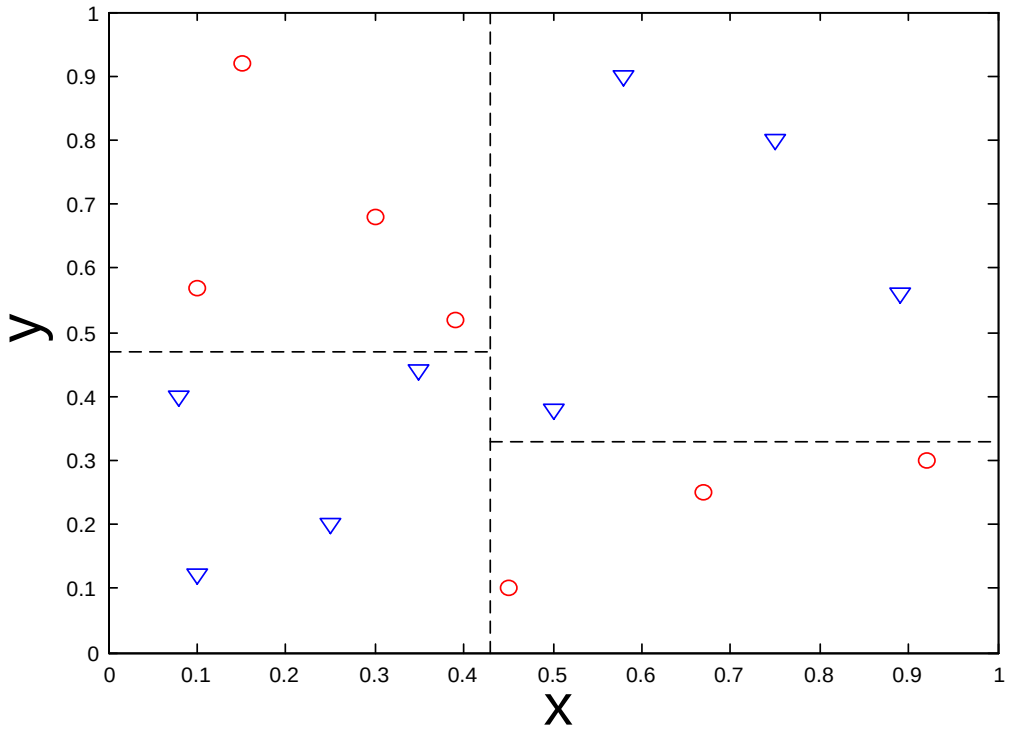
- Finding an optimal decision tree is NP-hard
- The algorithm presented so far uses a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
- Other strategies?
  - Bottom-up
  - Bi-directional

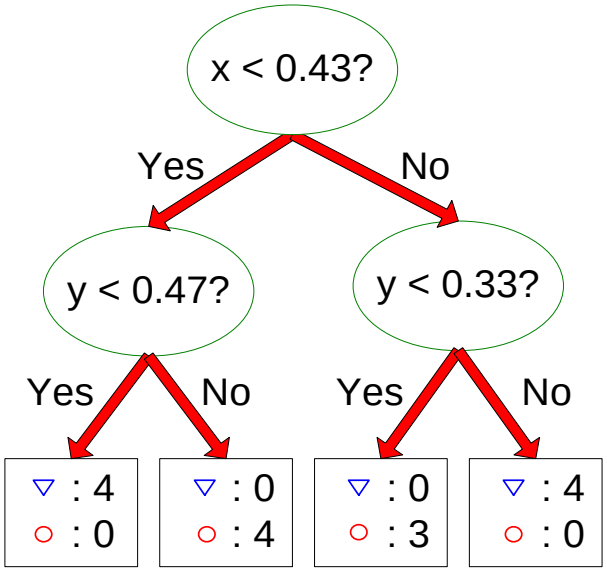
Tree Replication



- **Same subtree appears in multiple branches**

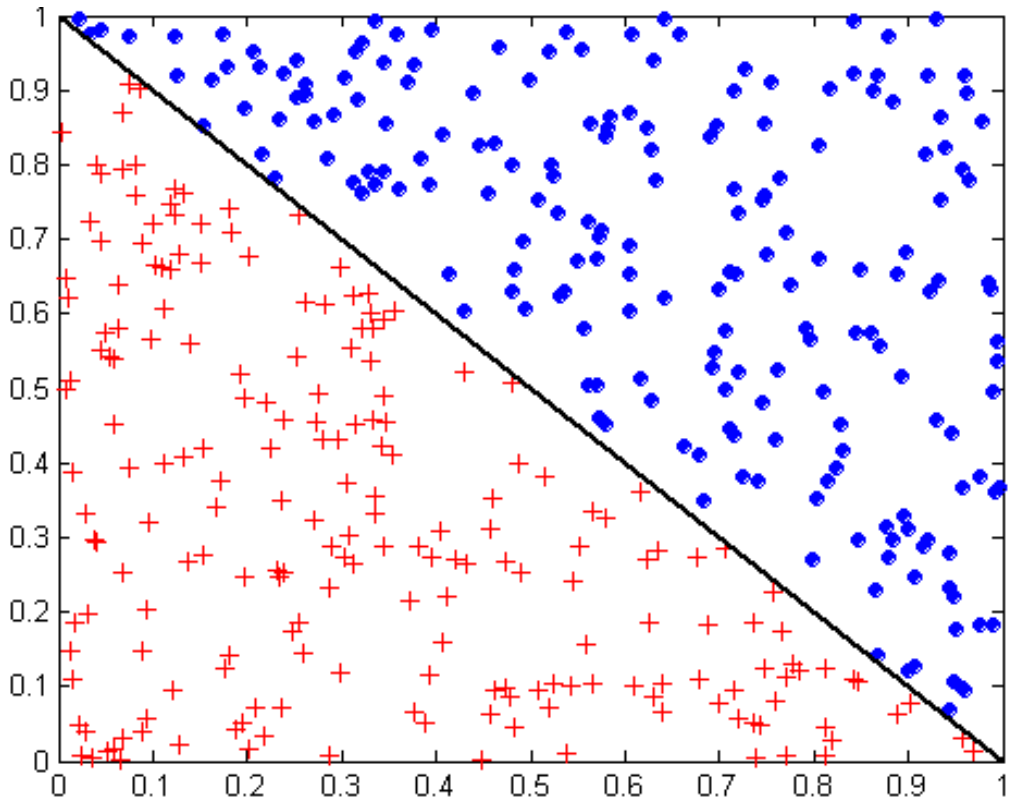
Decision Boundary

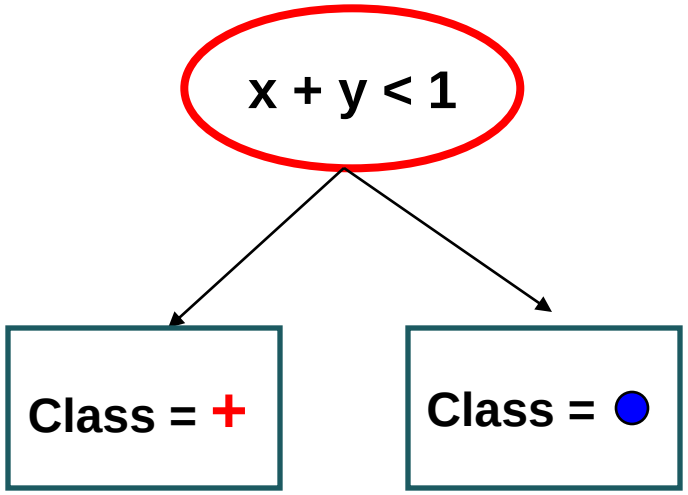




- **Border line between two neighboring regions of different classes is known as decision boundary**
- **Decision boundary is parallel to axes because test condition involves a single attribute at-a-time**

Oblique Decision Trees





- Test condition may involve multiple attributes
- More expressive representation
- Finding optimal test condition is computationally expensive

Discussion on Decision Trees

Advantages:

- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret by humans for small-sized trees (eager learning)
- Can easily handle redundant or irrelevant attributes
- Accuracy is comparable to other classification techniques for many low dimensional data sets (not texts and images)

Disadvantages:

- Space of possible decision tree is exponentially large
- Greedy approaches are often unable to find the best tree
- Trees do not take into account interactions between attributes.

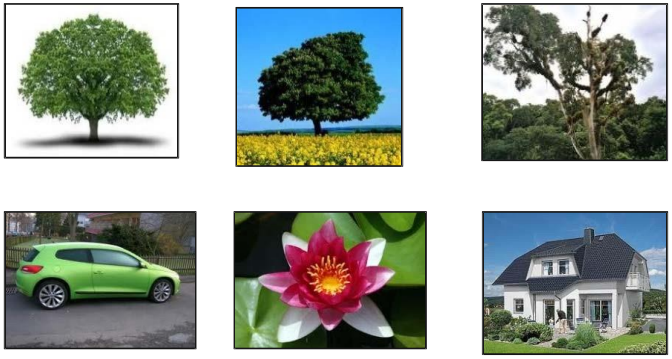
Overfitting

- We want to learn models that are good at classifying unseen records
- Overfitting: Learned models can fit the training data too closely and thus work poorly on unseen data
- Model perfectly fitting the training data:

"Trees are big, green plants that have a trunk and no wheels"

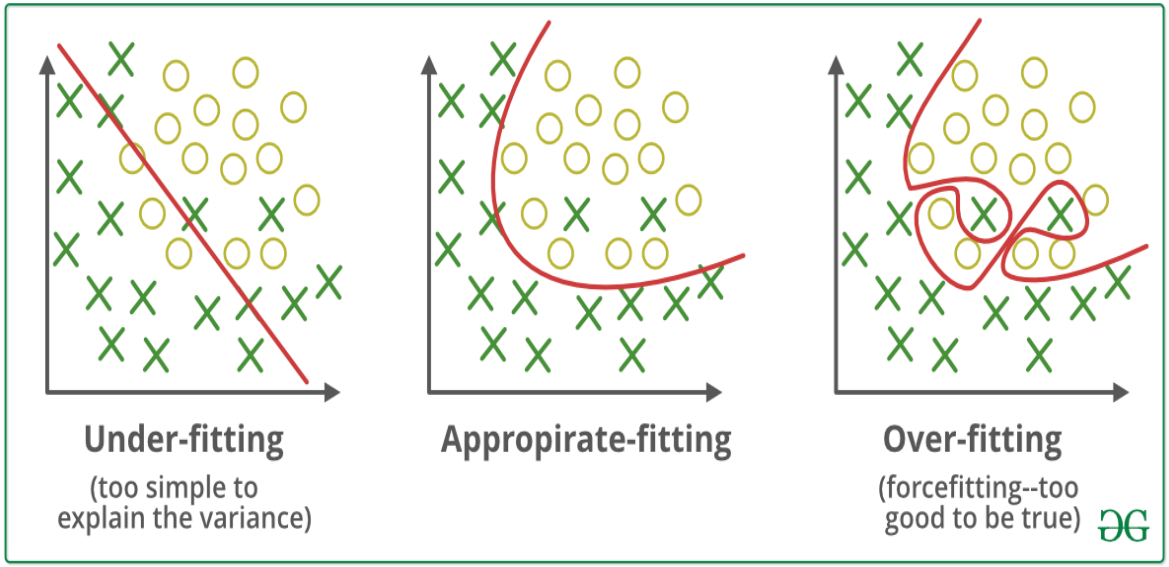
- Unseen example:

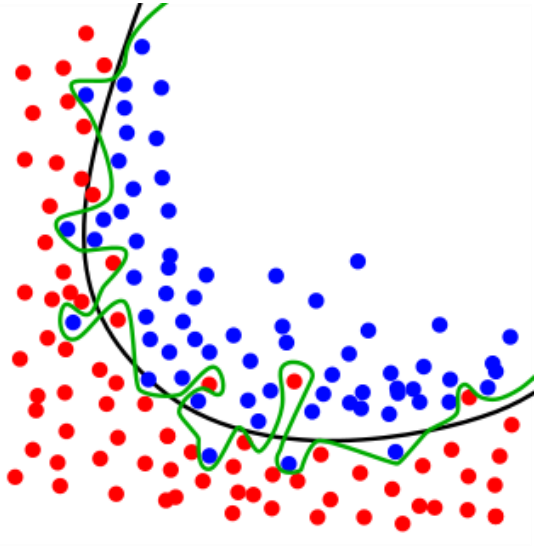
Training data



- Goal: Find good compromise between specificness and generality of the learned model

Underfitting and Overfitting





The black line fits the data well, the green line is overfitting.

The cause of the poor performance of a model in machine learning is either overfitting or underfitting the data.

Generalization in Data Mining /Machine Learning: refers to how well the concepts learned by a learning model generalizes to specific examples or data not yet seen by the model.

Overfitting vs Underfitting

Underfitting:

- **Occurs when a model is too simple.**
- **Informed by too few features or regularized too much - which makes it inflexible in learning from the dataset.**

- **Simple learners tend to have less variance in their predictions but more bias towards wrong outcomes.**
- **On the other hand, complex learners tend to have more variance in their predictions.**

- **Underfitted models are like those Engineers who wanted to be cricketers but forced by their parents to take up engineering.**

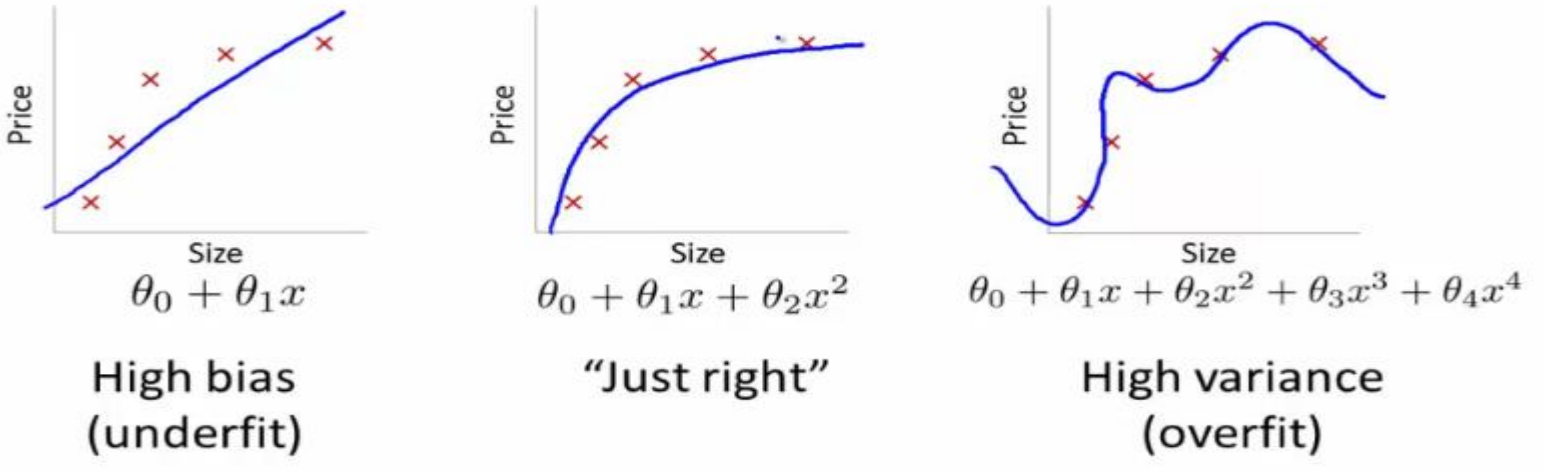
- **They will neither know engineering nor cricket pretty well. They never had their heart in what they did and have insufficient knowledge of everything.**

- **In terms of machine learning, we can state them as too little focus on the training set. Neither good for training not testing.**

Underfitting:

How to detect underfitting?

A model under fits when it is too simple with regards to the data it is trying to model.
One way to detect such a situation is to use the bias-variance approach.
Model is under fitted when you have a high bias.



How to avoid underfitting :

More data will not generally help. It will, in fact, likely increase the training error. Therefore we should increase more features. Because that expands the hypothesis space. This includes making new features from existing features. Same way more parameters may also expand the hypothesis space.

Overfitting:

Overfitted models are like subject matter experts

They know a lot about a particular field for example subject matter experts.

You ask them anything about the functionality of their tool (even in details), they'll probably be able to answer you and that too pretty precisely.

But when you ask them why the oil price fluctuate, they'll probably make an informed guess and say something peculiar.

In terms of machine learning, we can state them as too much focus on the training set (programmers) and learns complex relations which may not be valid in general for new data (test set).

Overfitting:

How to detect Overfitting?

A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it.

To address this, we can split our initial dataset into separate *training* and *test subsets*. This method can approximate how well our model will perform on new data.

If our model does much better on the training set than on the test set, then we're likely overfitting.

For example, it would be a big red flag if our model saw 95% accuracy on the training set but only 48% accuracy on the test set.

How to Prevent Overfitting:

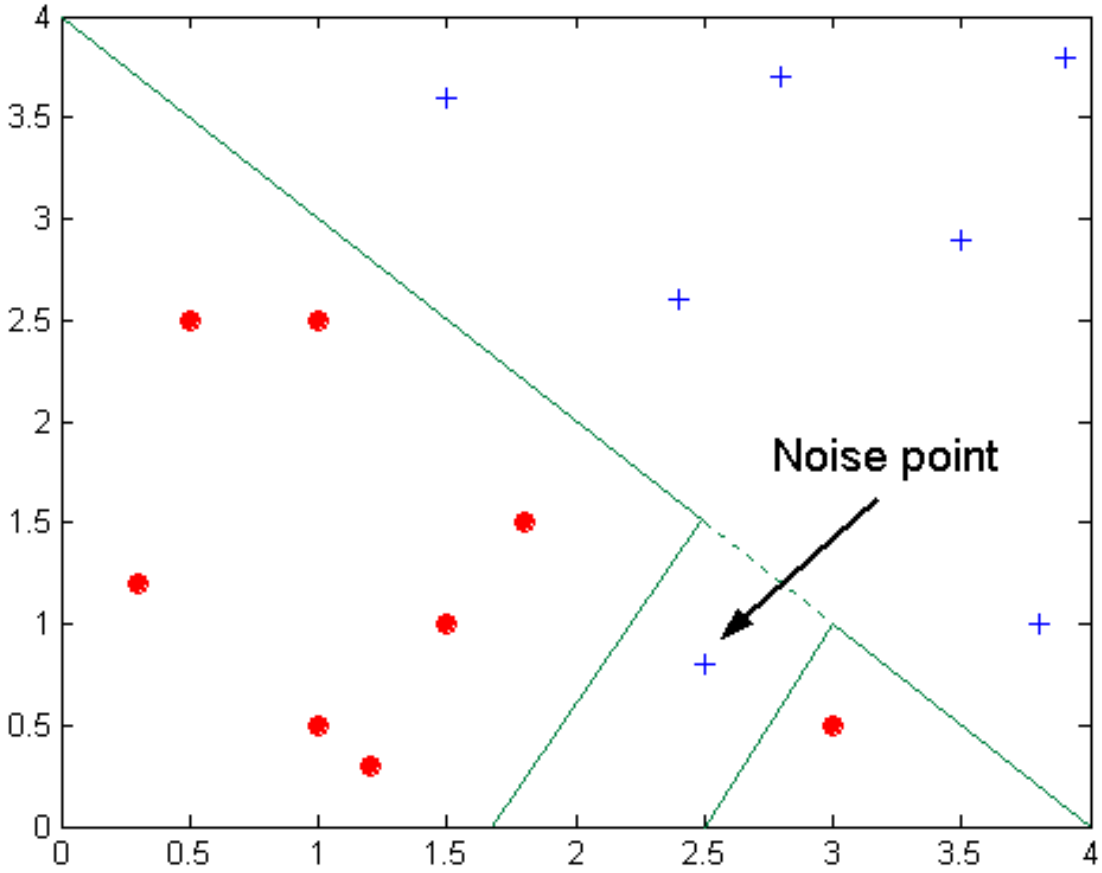
Detecting overfitting is useful, but it doesn't solve the problem. Fortunately, you have several options to try.

Here are a few of the most popular solutions for overfitting:
Cross Validation, Early Stopping, Pruning, Regularization, Remove Feature, Train with more data and Ensemble

Reasons for Model Overfitting

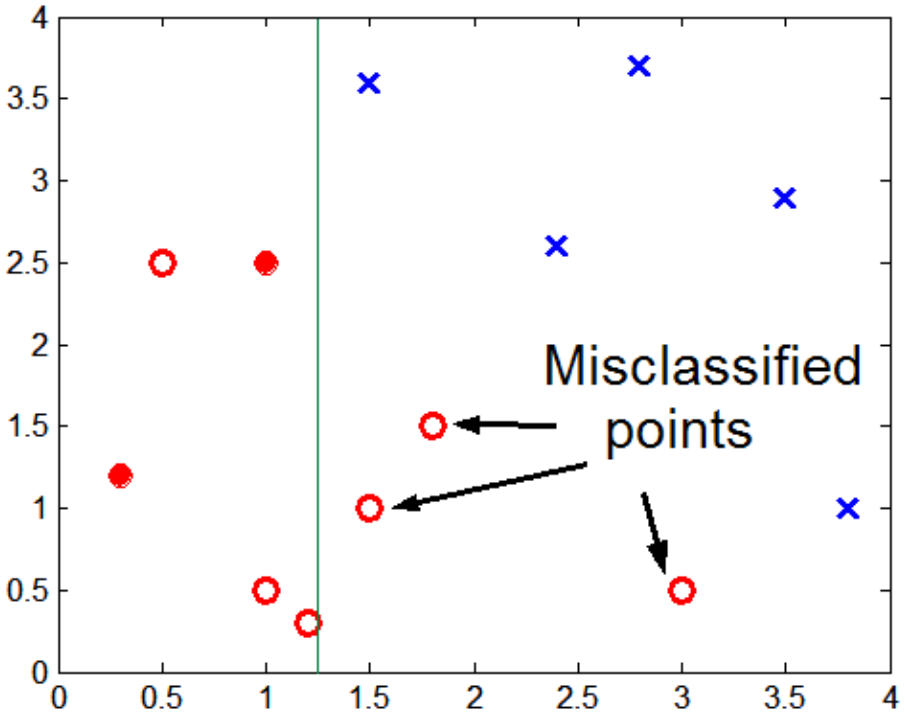
- Limited Training Size
- High Model Complexity
  - Multiple Comparison Procedure

Overfitting due to Presence of Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples, Lack of Representative samples



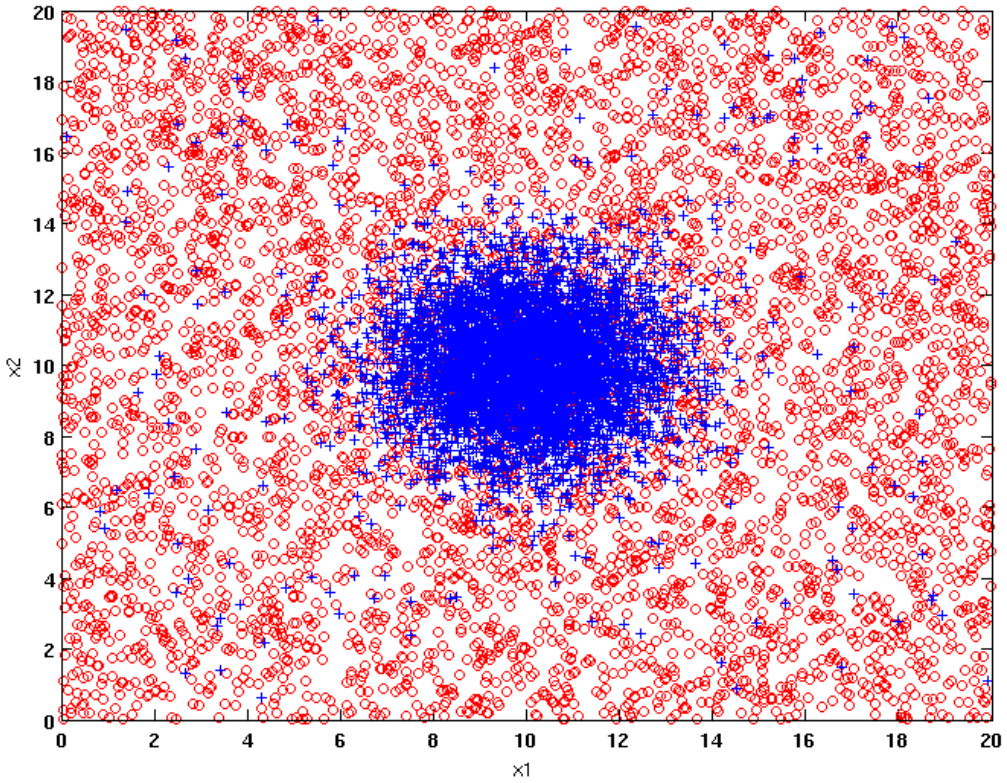
Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error does not provide a good estimate of how well the tree will perform on previously unseen records
- Need ways for estimating generalization errors

Example Data Set



Two class problem:

+ : 5200 instances

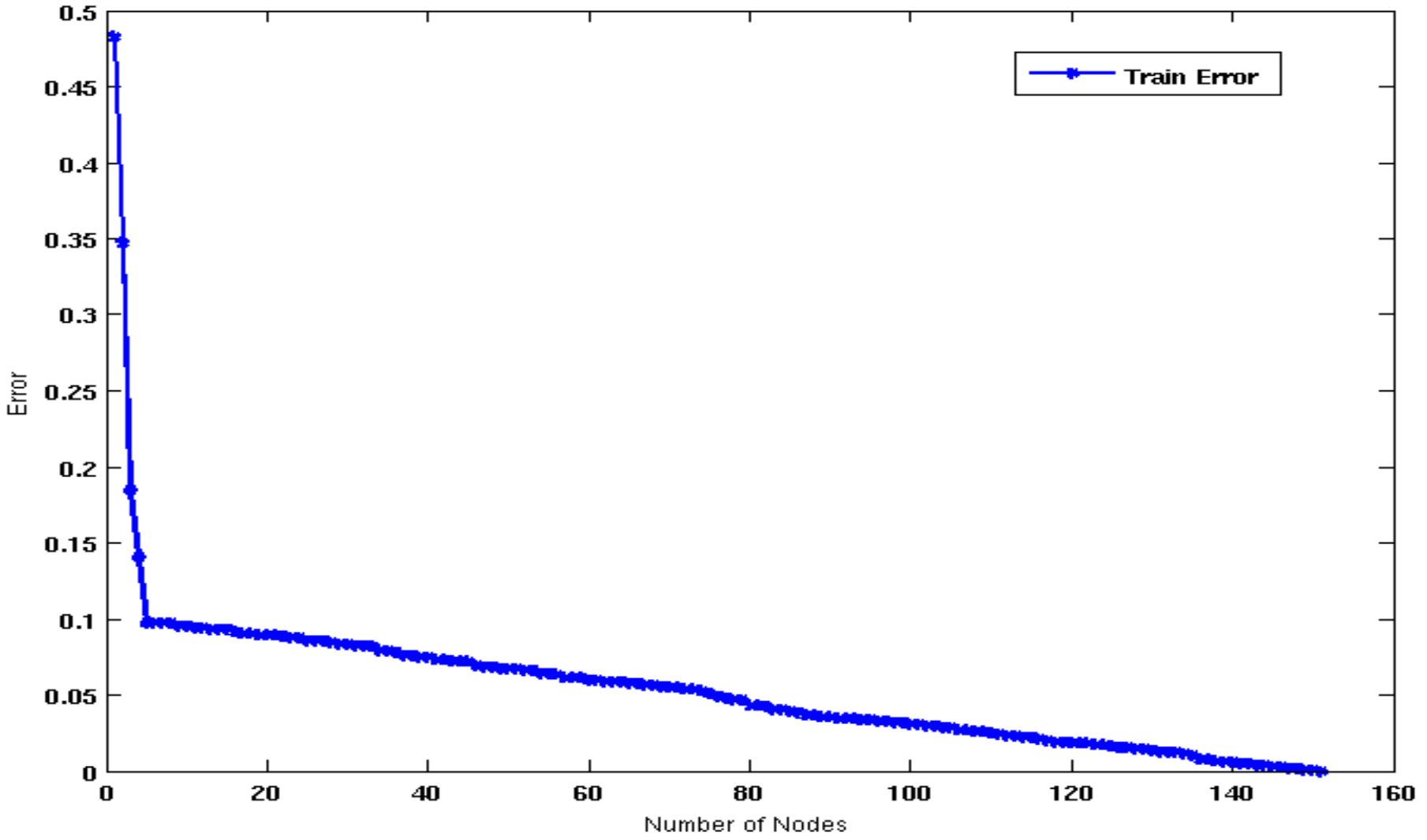
- **5000 instances generated from a Gaussian centered at (10,10)**
- **200 noisy instances added**

o : 5200 instances

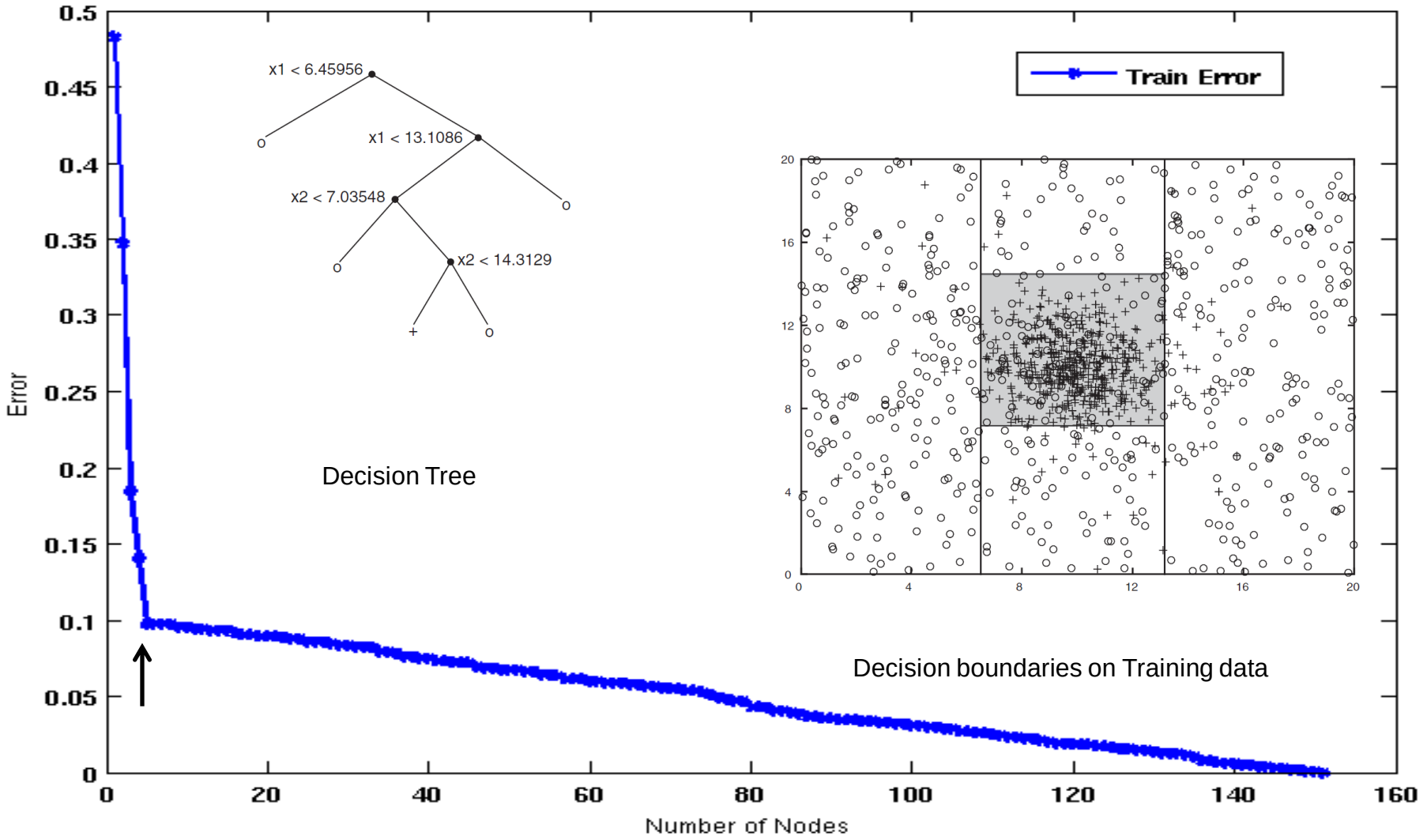
- **Generated from a uniform distribution**

10 % of the data used for training and 90% of the data used for testing

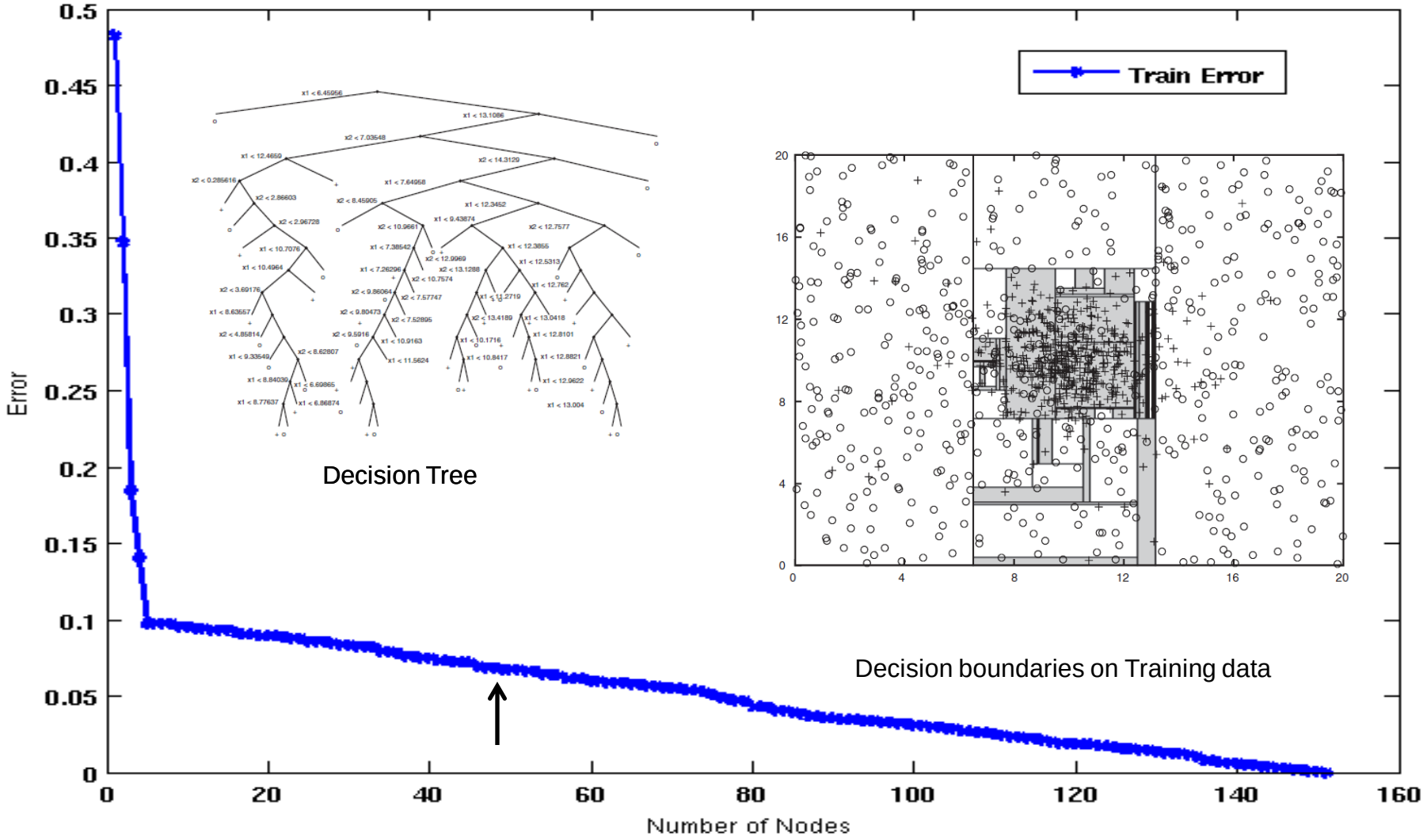
Increasing number of nodes in Decision Trees



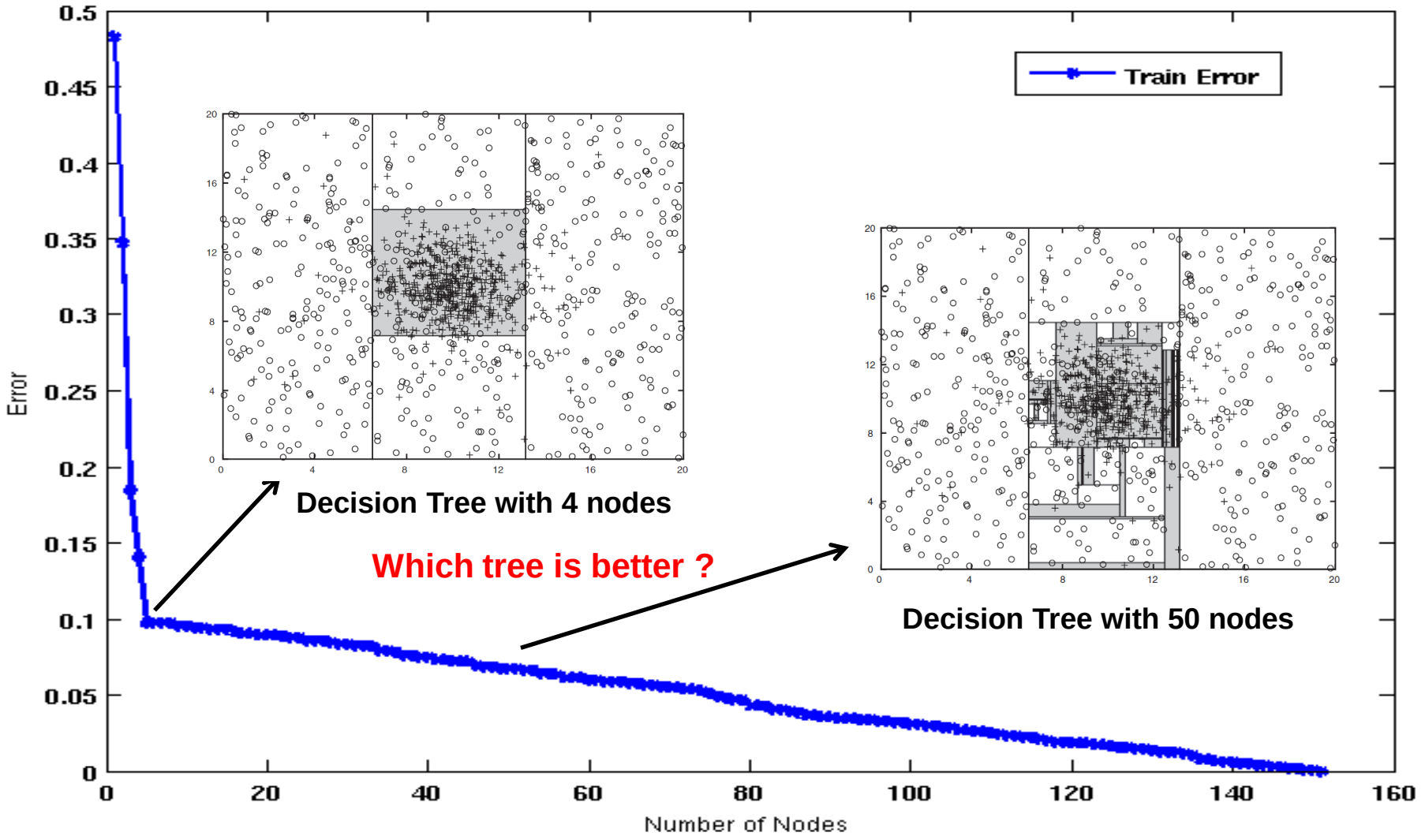
Decision Tree with 4 nodes



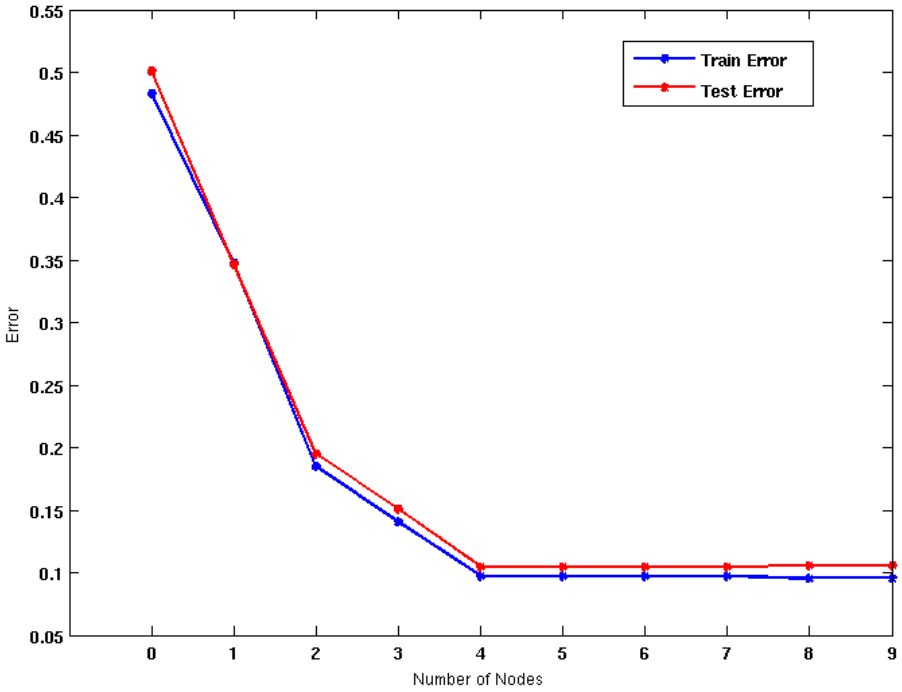
Decision Tree with 50 nodes

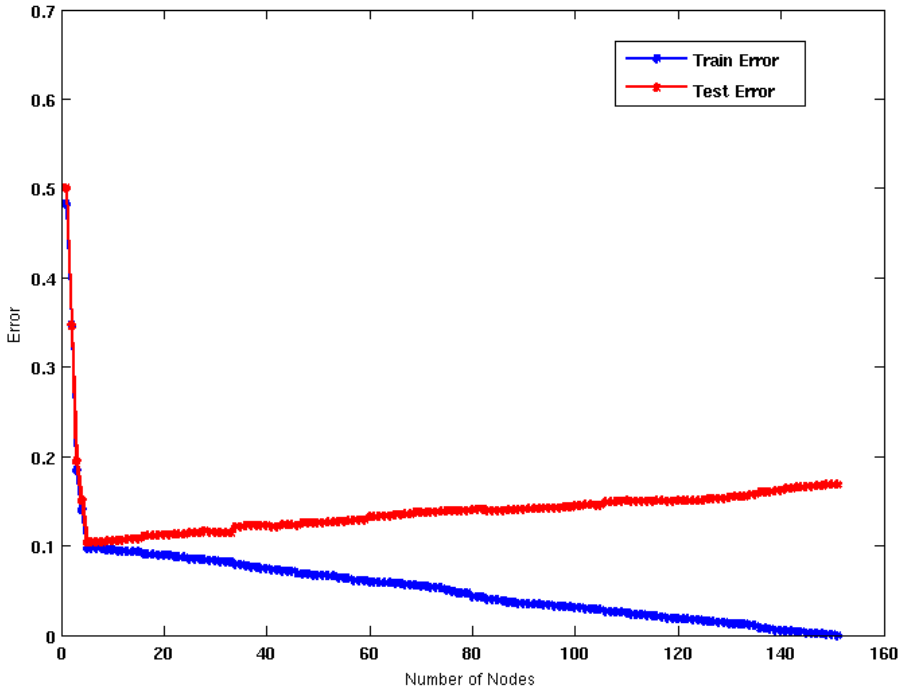


Which tree is better?



Model Overfitting

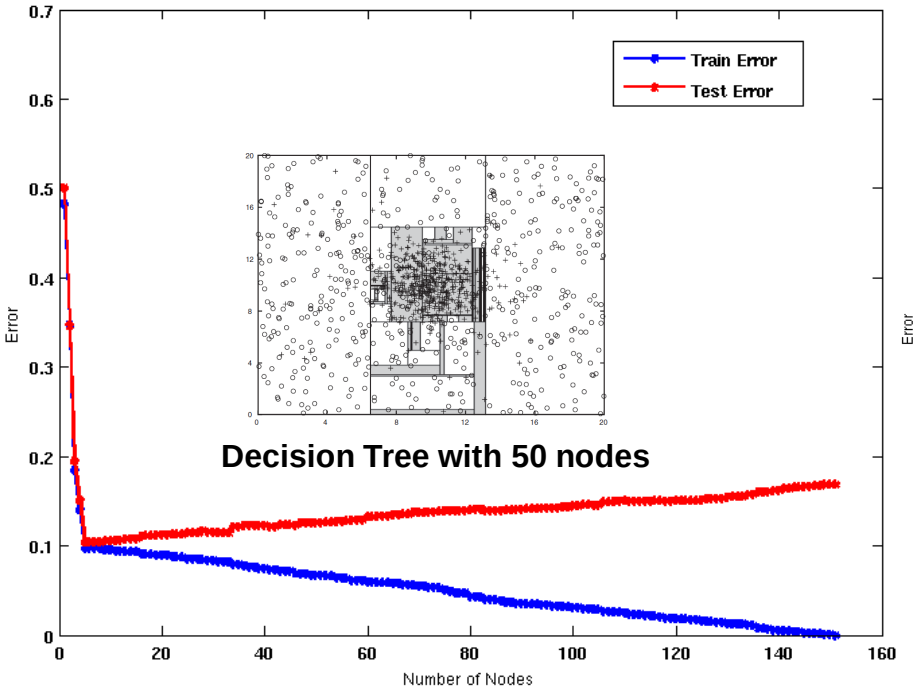


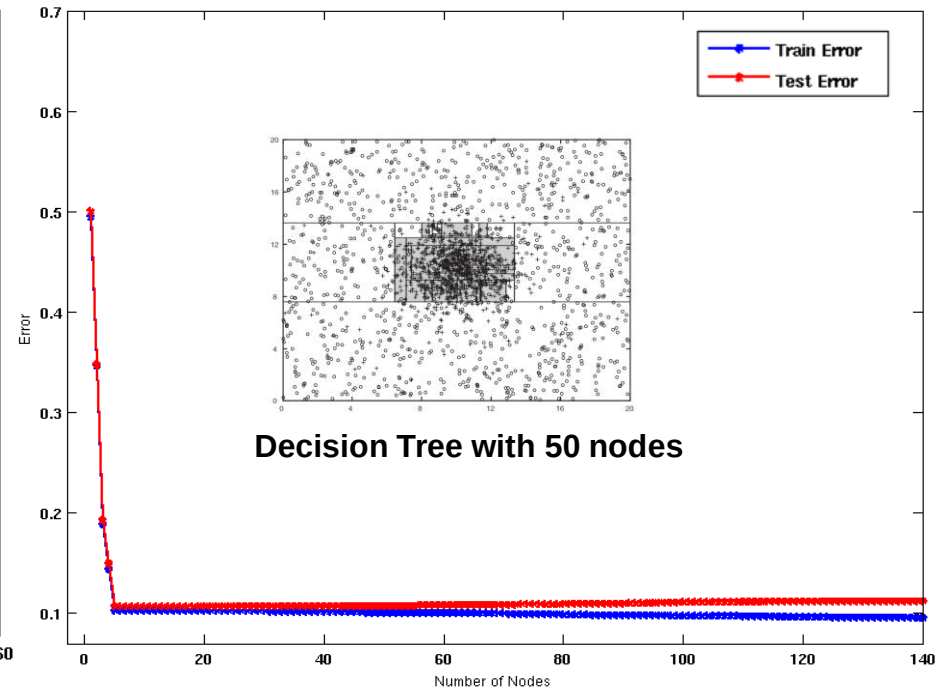


Underfitting: when model is too simple, both training and test errors are large

Overfitting: when model is too complex, training error is small, but test error is large

Model Overfitting





Using twice the number of data instances

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes

Effect of Multiple Comparison Procedure

- Consider the task of predicting whether stock market will rise/fall in the next 10 trading days

- Random guessing:

$$P(correct) = 0.5$$

- Make 10 random guesses in a row:

$$P(\#\,correct \geq 8) = \frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0547$$

Day 1	Up
Day 2	Down
Day 3	Down
Day 4	Up
Day 5	Down
Day 6	Down
Day 7	Up
Day 8	Up
Day 9	Up
Day 10	Down

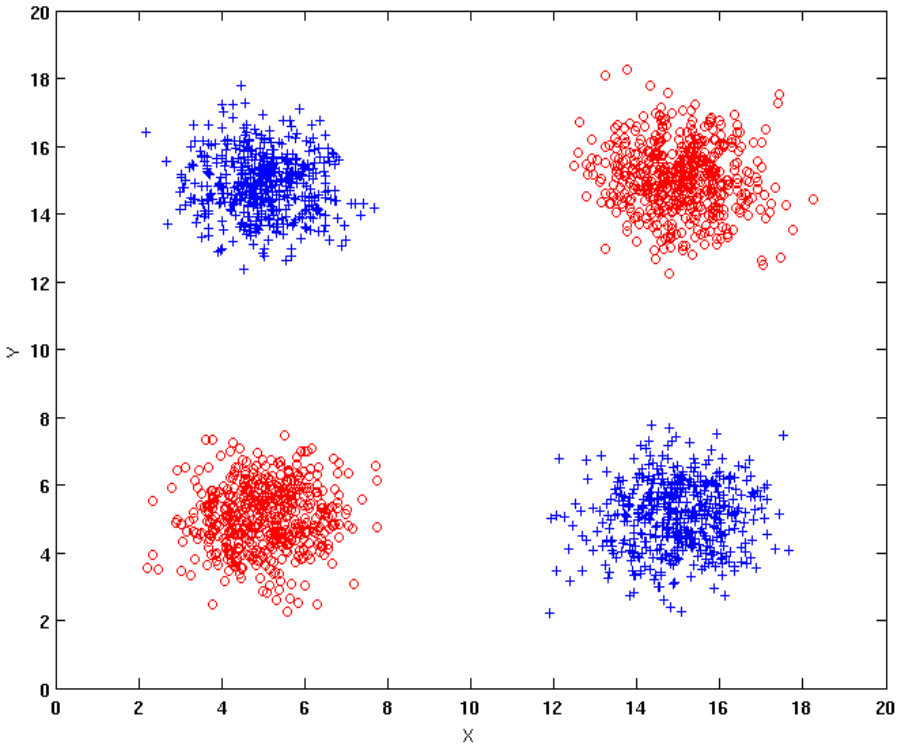
Effect of Multiple Comparison Procedure

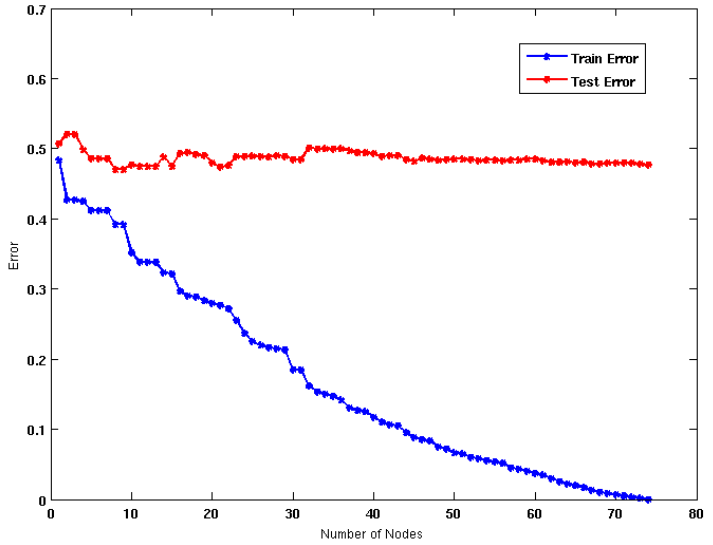
- Approach:
  - Get 50 analysts
  - Each analyst makes 10 random guesses
  - Choose the analyst that makes the most number of correct predictions

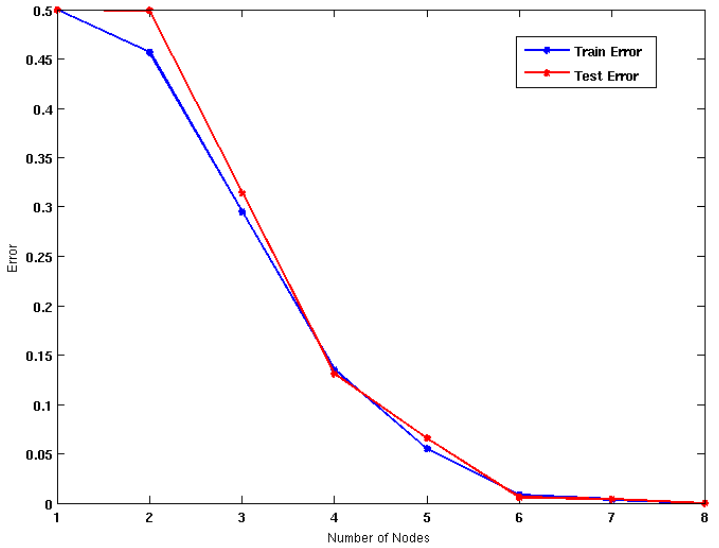
- Probability that at least one analyst makes at least 8 correct predictions

$$P(\#correct \geq 8) = 1 - (1 - 0.0547)^{50} = 0.9399$$

Effect of Multiple Comparison - Example







Use additional 100 noisy variables generated from a uniform distribution along with X and Y as attributes.

Use 30% of the data for training and 70% of the data for testing

Using only X and Y as attributes

Classification Errors

- Training errors (apparent errors)
  - Errors committed on the training set
- Test errors
  - Errors committed on the test set
- Generalization errors
  - Expected error of a model over random selection of records from same distribution

Estimating Generalization Errors

- Re-substitution errors: error on training data (Σ e(t))
- Generalization errors: error on testing data (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

- **Pessimistic Error Estimate**
- **Computes generalization error (on testing data) as the sum of training error and a penalty term for model complexity.**
- **The pessimistic error estimate of a decision tree *T*, *eg*(*T*), can be computed as follows:**

  ***eg*(*T*) = = [*e*(*T*) + Ω(*T*)] / *Nt* , Where,**

  ***e*(*T*) is the overall training error of the decision tree,**

  ***Nt* is the number of training records, and**

  Ω(*ti*) is the penalty term associated with each leaf node *ti*.

- **Thus out of many possible tree the one which has a better pessimistic error is selected.**
- **Also a node should not be expanded into its child nodes unless it reduces the misclassification error for more than one training record.**

- Pessimistic approach:
  - For each leaf node: e'(t) = (e(t)+0.5) (assume Ω(T)=0.5)
  - Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
  - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
  - Training error = 10/1000 = 1%

Generalization error = (10 + 30×0.5)/1000 = 2.5%

Occam's Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model

- For complex models, there is a greater chance that it was fitted accidentally by errors in data

- Therefore, one should include model complexity when evaluating a model

Incorporating Model Complexity

- Rationale: Occam's Razor
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model

Gen. Error(Model) = Train. Error(Model, Train. Data) +

α x Complexity(Model)

Estimating the Complexity of Decision Trees

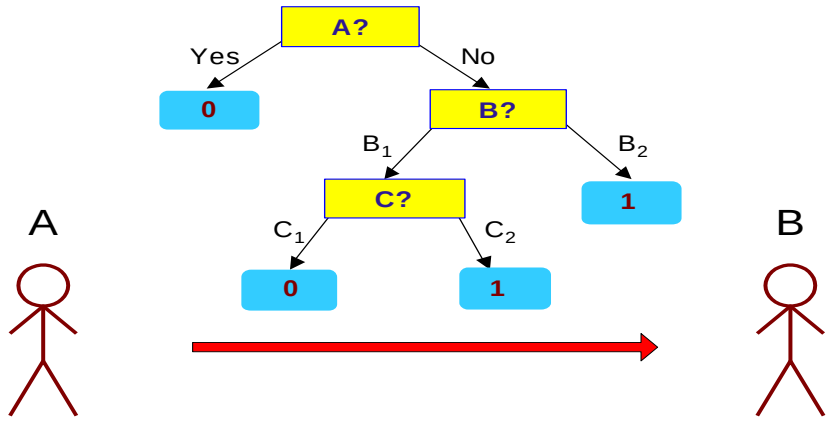
- **Pessimistic Error Estimate** of decision tree T with k leaf nodes:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

  - err(T): error rate on all training records
  - Ω: trade-off hyper-parameter (similar to α)
    - Relative cost of adding a leaf node
  - k: number of leaf nodes
  - N_{train}: total number of training records

Minimum Description Length (MDL)

X	y
X_1	1
X_2	0
X_3	0
X_4	1
…	…
X_n	1

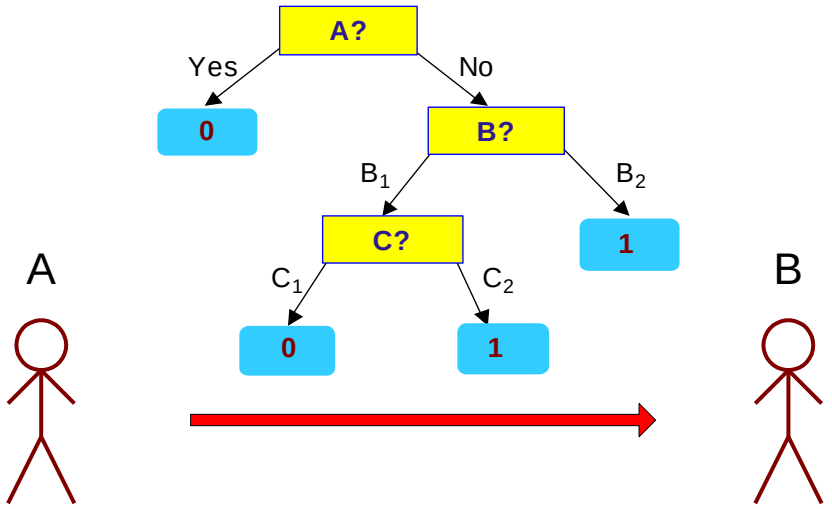


X	y
X_1	?
X_2	?
X_3	?
X_4	?
…	…
X_n	?

- Person A and B are given a set of records with known attribute values x.
- 'A' knows the exact class label for each record, while person 'B knows none of this information.
- 'B' can obtain the classification of each record by requesting that 'A' transmits the class labels sequentially.
- Such a message would require Θ(*n*) bits of information, where *n* is the total number of records.
- Alternatively, A may decide to build a classification model that summarizes the relationship between x and *y*.
- The model can be encoded in a compact form before being transmitted to B.
- If the model is 100% accurate, then the cost of transmission is equivalent to the cost of encoding the model.
- Otherwise, A must also transmit information about which record is classified incorrectly
- by the model.

Minimum Description Length (MDL)

X	y
X_1	1
X_2	0
X_3	0
X_4	1
…	…
X_n	1



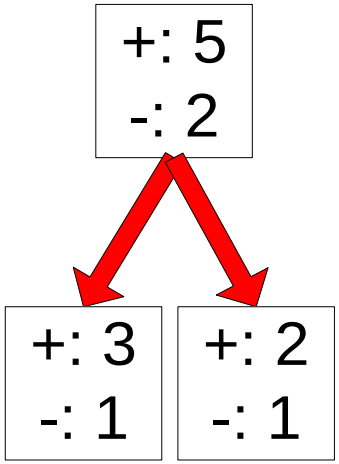
X	y
X_1	?
X_2	?
X_3	?
X_4	?
…	…
X_n	?

- Cost(Model,Data) = Cost(Data|Model) + α x Cost(Model)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- Cost(Data|Model) encodes the misclassification errors.
- Cost(Model) uses node encoding (number of children) plus splitting condition encoding.

Among all equally good classifier you would like to pick up one which requires least number of bits to describe it. So the description length should be as small as possible given that the classifier has some acceptable level of performance. (Classifier description: no of support vectors in SVM, no of weight and other parameters in ANN etc. and encoding them for communication or so.)

Small error requires less number of bits. If classifier is complex we need lots of bits to describe it but error is minimum and number of bits required to specify error are minimum. Trade off between classifier description and error description). We need to select based on some trade off criterion.

4.4.4.Estimating Statistical Bounds



$$e'(N, e, \alpha) = \frac{e + \frac{z^2_{\alpha/2}}{2N} + z_{\alpha/2}\sqrt{\frac{e(1-e)}{N} + \frac{z^2_{\alpha/2}}{4N^2}}}{1 + \frac{z^2_{\alpha/2}}{N}}$$

Before splitting: e = 2/7, e'(7, 2/7, 0.25) = 0.503

e'(T) = 7 × 0.503 = 3.521

After splitting:

$e(T_L)$ = 1/4, e'(4, 1/4, 0.25) = 0.537

$e(T_R)$ = 1/3, e'(3, 1/3, 0.25) = 0.650

e'(T) = 4 × 0.537 + 3 × 0.650 = 4.098

Therefore, do not split

Model Selection:

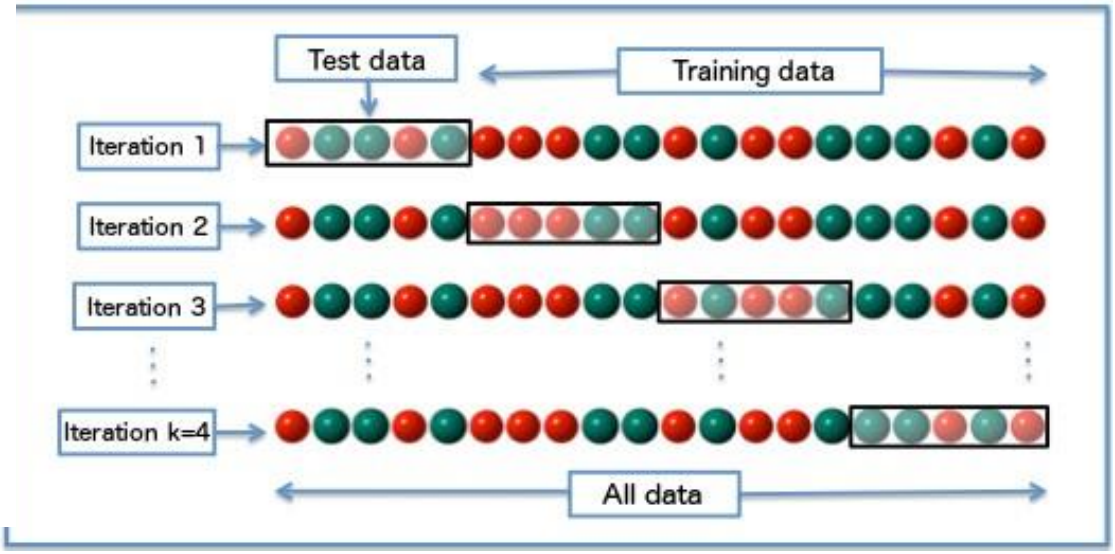
Using Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set

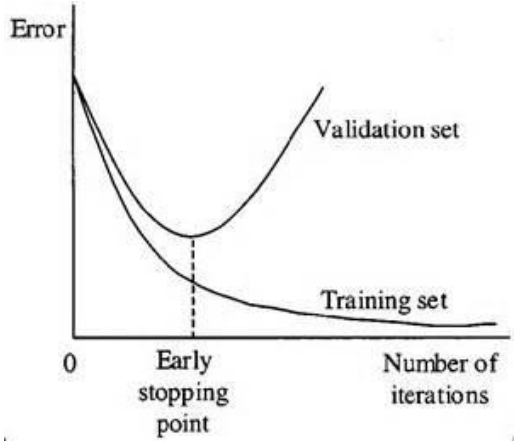
- Drawback:
  - Less data available for training

Here are a few of the most popular solutions for overfitting:

•Cross-Validation: A standard way to find out-of-sample prediction error is to use 5-fold cross-validation.



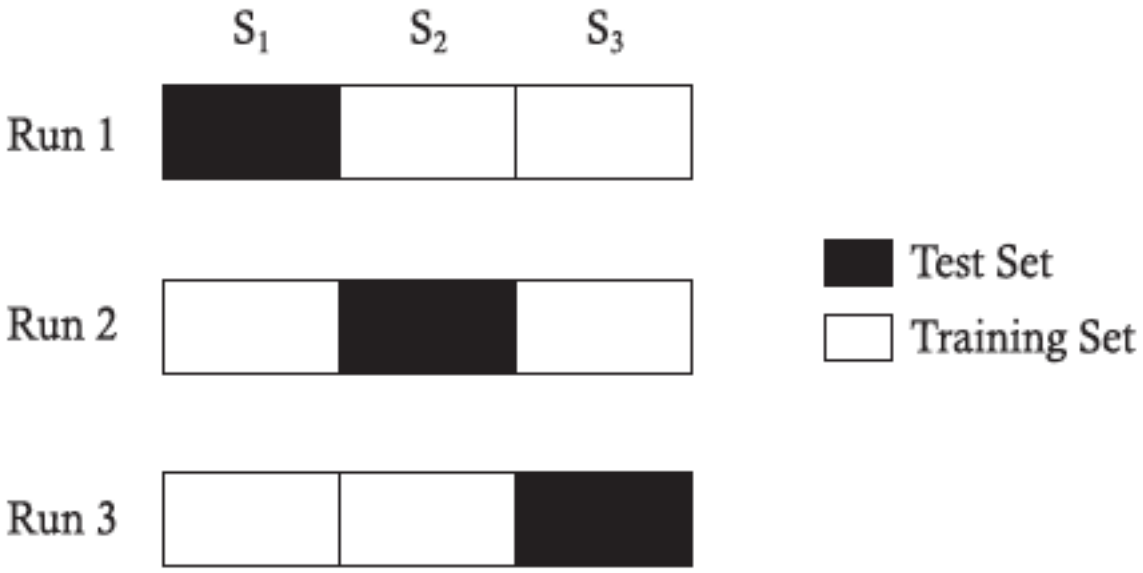
•Early Stopping: Its rules provide us with guidance as to how many iterations can be run before the learner begins to over-fit.



- **Pruning:** Pruning is extensively used while building related models. It simply removes the nodes which add little predictive power for the problem in hand.
- **Regularization:** It introduces a cost term for bringing in more features with the objective function. Hence it tries to push the coefficients for many variables to zero and hence reduce cost term.
- **Remove features:** Some algorithms have built-in feature selection. For those that don't, you can manually improve their generalizability by removing irrelevant input features. An interesting way to do so is to tell a story about how each feature fits into the model.
- **Train with more data:** It won't work every time, but training with more data can help algorithms detect the signal better.
- If we just add more noisy data, this technique won't help. That's why you should always ensure your data is clean and relevant.

Cross-validation Example

- 3-fold cross-validation



How to Address Overfitting

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree

    shallower tree potentially generalizes better (Occam's razor)

    - Typical stopping conditions for a node (no pruning):
      - 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 (High threshold leads to underfitting problem, Low threshold leads to overfitting problem)
      - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
      - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

How to Address Overfitting…

Post-pruning

1. Grow decision tree to its entirety
2. Trim the nodes of the decision tree in a bottom-up fashion(using a validation set)
3. Estimate generalization error before and after trimming
4. If generalization error improves after trimming
   - replace sub-tree by a leaf node or
   - replace subtree by most frequently used branch

Class label of leaf node is determined from majority class of instances in the sub-tree

Example of Post-Pruning

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

Training Error (Before splitting) = 10/30

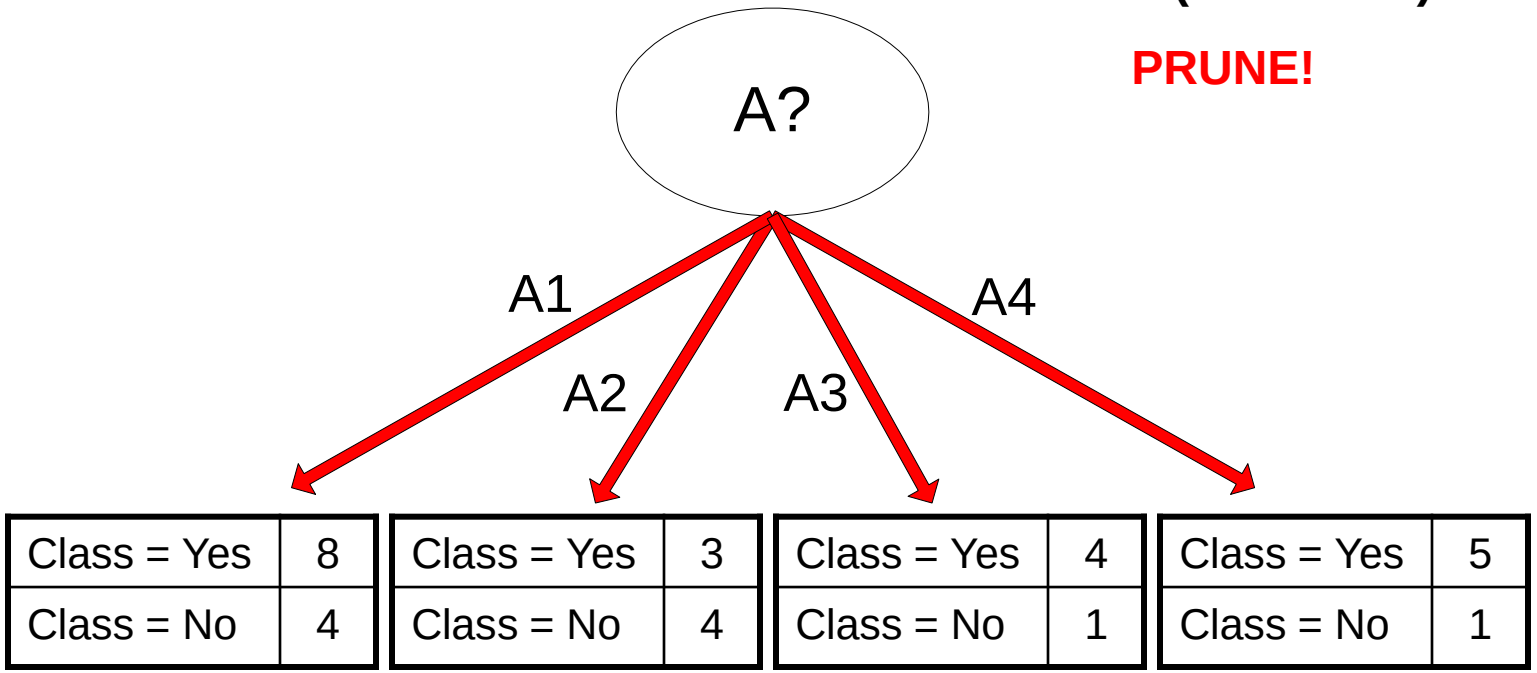
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 × 0.5)/30 = 11/30

PRUNE!



Class = Yes	8
Class = No	4

Class = Yes	3
Class = No	4

Class = Yes	4
Class = No	1

Class = Yes	5
Class = No	1

Handling Missing Attribute 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

Computing Impurity Measure

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	?	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

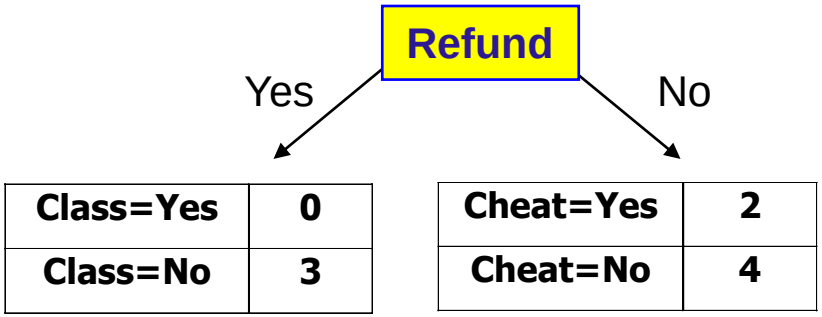
Entropy(Refund=No)
= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

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

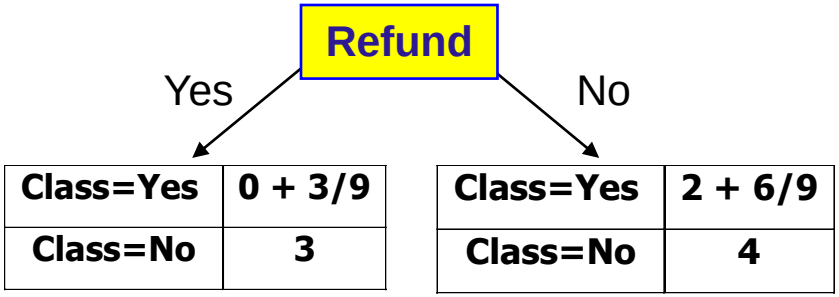
Gain = 0.9 × (0.8813 – 0.551) = 0.3303

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



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



Probability that Refund=Yes is 3/9

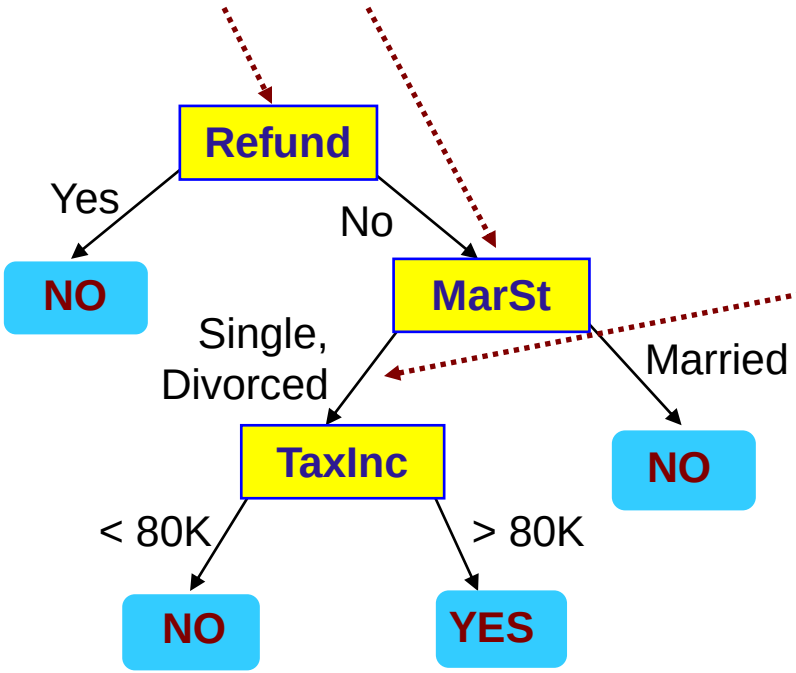
Probability that Refund=No is 6/9

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

Classify Instances

New 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

Model Evaluation

- How good is a model at classifying unseen records?
- Metrics for Performance Evaluation
  - How to evaluate/measure the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Model Evaluation

- Purpose:
  - To estimate performance of classifier on previously unseen data (test set)
- Holdout
  - Reserve k% for training and (100-k)% for testing
  - Random subsampling: repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- A **confusion matrix** is a matrix (table) that can be used to measure the performance of an Classifier (machine learning algorithm), usually in a supervised learning mode on a set of test data for which the true values are known.
- It allows the visualization of the performance of an algorithm.
- The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
- It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made

Confusion Matrix

- Positive (P) (Yes) (+) (T): Observation is positive (for example: is an apple).
- Negative (N) (No) (-) (F) : Observation is not positive (for example: is not an apple)

		PREDICTED CLASS	
		Class= Yes	Class= No
ACTUAL CLASS / Observation	Class=Yes	a (TP)	b (FN) Type II error
	Class=No	c (FP) Type I error	d (TN)

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

True +ve: Observation is positive, and is predicted to be positive.

If the person is actually having cancer (Actual class = Yes) and we predict correctly as **Yes** he is actually having a cancer (Predicted Class = Yes)

False -ve: (Type 2 Error) Observation is positive, but is predicted negative

If the person is actually having a cancer(Actual class = Yes) and we predict wrong as **No**, he is actually having a cancer (Predicted class = No)

False +ve: (Type 1 Error) Observation is negative, but is predicted positive

if the person is not having a cancer (Actual class = No) and we predict as **Yes**,he is actually not having cancer but we predict as he is having cancer, which is wrong (Predicted class = Yes)

True -ve: Observation is negative, and is predicted to be negative.

If the person is not having a cancer (Actual class = No) and we predict correctly as **No,** that means we predict correctly as the person is not having cancer (Predicted Class = No)

Class Imbalance

Model performance for classification models is usually debatable in terms of which model performance is most relevant, especially when the dataset is imbalanced.

The usual model performance measures for evaluating a classification model are accuracy, sensitivity or recall, specificity, precision, KS statistic and Area under the curve (AUC).

The Class Imbalance Problem

- Sometimes, classes have very unequal frequency
- Fraud detection: 98% transactions OK, 2% fraud
- E-commerce: 99% surfers don't buy, 1% buy
- Intruder detection:99.99% of the users are no intruders
- Security:>99.99% of Americans are not terrorists

The Class Imbalance Problem

- Consider a 2-class problem : The class of interest is commonly called Positive Class
  - Number of Class 0 (Negative) examples = 9990
  - Number of Class 1 (Positive) examples = 10

- If model predicts everything to be class 0, accuracy is 9990+0/10000 = 99.9 %
- Accuracy is misleading because model does not detect any class 1 example
- In this case we get accuracy as 99.9% but we cannot evaluate the performance of a model on the basis of accuracy because we were not able to predict class 1 examples.

- If we see carefully the proportion of Class 0 examples is high which is 9990 and the proportion of class 1 examples is very low which is 10 for the 2-class problem.
- Hence in the above case accuracy will not be a correct measure to evaluate the performance of model.

Metrics for Performance Evaluation…

Accuracy: Widely-used metric: -Treat every class as equally important

$$Accuracy = \frac{Correct\ Prediction}{Al\ Predictions}$$

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

It may be defined as the number of correct predictions made by our ML model.

Accuracy is the number of correct (True) predictions made by the model by the total number of records. The best accuracy is 100% indicating that all the predictions are correct.

- For an imbalanced dataset, accuracy is not a valid measure of model performance.
- For a dataset where the default rate is 5%, even if all the records are predicted as 0, the model will still have an accuracy of 95%. It assumes equal costs for both kinds of errors.
- The model may ignore all the defaults and can be very detrimental to the business.

Alternate: Use performance metrics from information retrieval which are biased towards the positive class by ignoring TN

Recall or Sensitivity:

Recall may be defined as the number of positives returned by our ML model. When it's actually yes, how often does it predict yes?

Recall is the ratio of the total number of correctly classified positive examples divide to the total number of positive examples.

High Recall indicates the class is correctly recognized (small number of FN).

$$Recall = \frac{TP}{TP + FN}$$

Precision:

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. When it predicts yes, how often is it correct?

High Precision indicates an example labeled as positive is indeed positive (small number of FP).

Precision is given by the relation:

$$Precision = \frac{TP}{TP + FP}$$

- sensitivity, recall, hit rate, or true positive rate (TPR)
- specificity, selectivity or true negative rate (TNR)
- precision or positive predictive value (PPV)
- negative predictive value (NPV)
- miss rate or false negative rate (FNR)
- fall-out or false positive rate (FPR)
- false discovery rate (FDR)
- false omission rate (FOR)
- Threat score (TS) or Critical Success Index (CSI)

KS statistic

KS statistic is a measure of degree of separation between the positive and negative distributions.

KS value of 100 indicates that the scores partition the records exactly such that one group contains all positives and the other contains all negatives.

In practical situations, a KS value higher than 50% is desirable.

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives (high FP).

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

Recall or sensitivity gives us information about a model's performance on false negatives (incorrect prediction of customers who will default), while precision gives us information of the model's performance of false positives.

Based on what is predicted, precision or recall might be more critical for a model.

The cost function comes into play in deciding which of the incorrect predictions can be more detrimental — the false positive or the false negative (in other words, which performance measure is important — precision or recall).

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time.

F-measure:

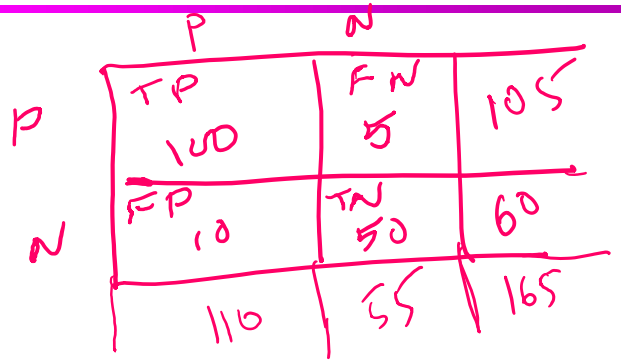
Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.
The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$F\text{-}measure = \frac{2*Recall*Precision}{Recall + Precision}$$

- Let's consider an example now, in which we have infinite data elements of class B and a single element of class A and the model is predicting class A against all the instances in the test data.
  Here, Precision : 0.0, Recall : 1.0
- Now: Arithmetic mean: 0.5 , Harmonic mean: 0.0
  When taking the arithmetic mean, it would have 50% correct. Despite being the worst possible outcome! While taking the harmonic mean, the F-measure is 0.

Example

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	



- **Accuracy:** Overall, how often is the classifier correct?
  - (TP+TN)/total = (100+50)/165 = 0.91
- **Misclassification Rate:** Overall, how often is it wrong?
  - (FP+FN)/total = (10+5)/165 = 0.09
  - equivalent to 1 minus Accuracy also known as "Error Rate"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
  - TP/actual yes = 100/105 = 0.95 also known as "Sensitivity" or "Recall"
- **False Positive Rate:** When it's actually no, how often does it predict yes?
  - FP/actual no = 10/60 = 0.17
- **True Negative Rate:** When it's actually no, how often does it predict no?
  - TN/actual no = 50/60 = 0.83
  - equivalent to 1 minus False Positive Rate also known as "Specificity"
- **Precision:** When it predicts yes, how often is it correct?
  - TP/predicted yes = 100/110 = 0.91
- **Prevalence:** How often does the yes condition actually occur in our sample?
  - actual yes/total = 105/165 = 0.64

$R = \frac{TP}{TP+FN} = \frac{100}{105} = 0.95$

$F = \frac{2 \times R \times P}{R+P} = \frac{2 \times 0.95 \times 0.91}{0.95+0.91} = \frac{1.729}{1.86} = 92.95$

How to Calculate a Confusion Matrix

- Here, is step by step process for calculating a confusion Matrix in data mining
- Step 1) First, you need to test dataset with its expected outcome values.
- Step 2) Predict all the rows in the test dataset.
- Step 3) Calculate the expected predictions and outcomes:
  - The total of correct predictions of each class.
  - The total of incorrect predictions of each class.
- After that, these numbers are organized in the below-given methods:
  - Every row of the matrix links to a predicted class.
  - Every column of the matrix corresponds with an actual class.
- The total counts of correct and incorrect classification are entered into the table.
  - The sum of correct predictions for a class go into the predicted column and expected row for that class value.
  - The sum of incorrect predictions for a class goes into the expected row for that class value and the predicted column for that specific class value.

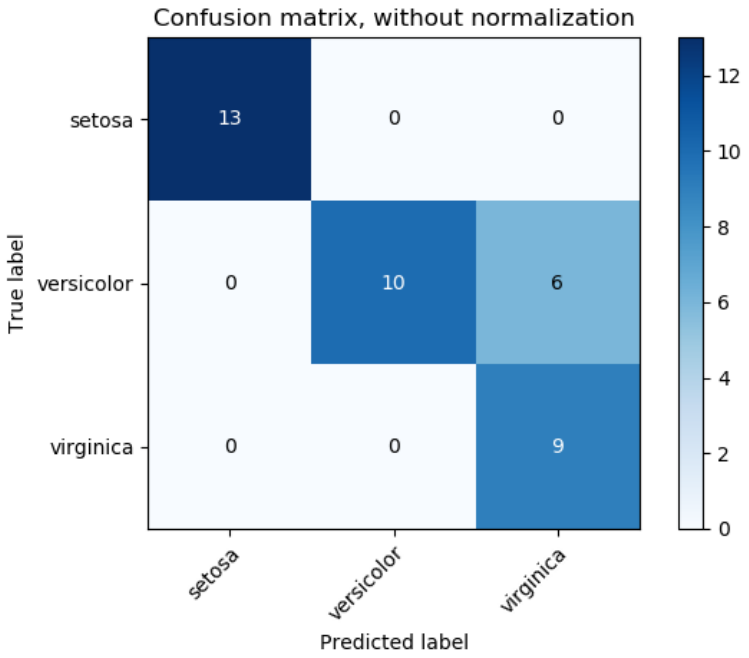
- **Null Error Rate:**
- This is how often you would be wrong if you always predicted the majority class. (In our example, the null error rate would be 60/165=0.36 because if you always predicted yes, you would only be wrong for the 60 "no" cases.)
- This can be a useful baseline metric to compare your classifier against.
- However, the best classifier for a particular application will sometimes have a higher error rate than the null error rate
- **Cohen's Kappa:** (Range 0-1; ideal=1)
- This is essentially a measure of how well the classifier performed as compared to how well it would have performed simply by chance.
- In other words, a model will have a high Kappa score if there is a big difference between the accuracy and the null error rate.

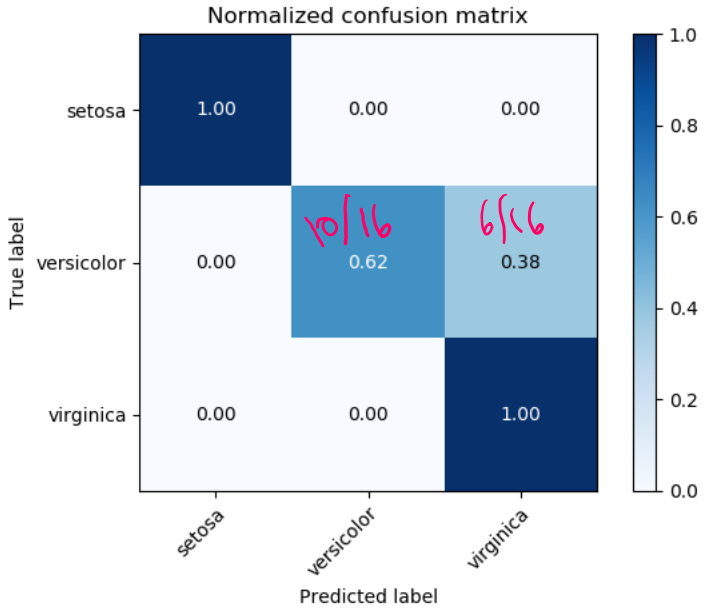
Why you need Confusion matrix?

- Pros/benefits of using a confusion matrix:
- It shows how any classification model is confused when it makes predictions.
- Confusion matrix not only gives you insight into the errors being made by your classifier but also types of errors that are being made.
- This breakdown helps you to overcomes the limitation of using classification accuracy alone.
- Every column of the confusion matrix represents the instances of that predicted class.
- Each row of the confusion matrix represents the instances of the actual class.

Confusion matrix with and without normalization by class support size (number of elements in each class). T

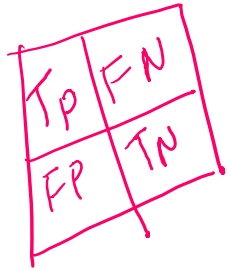
This kind of normalization can be interesting in case of class imbalance to have a more visual interpretation of which class is being misclassified.





Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)



C(i|j): Cost of misclassifying class j example as class i

Cost matrix is similar to the confusion matrix except the fact that we are calculating the cost of wrong prediction or right prediction.

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i|j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 150+250/150+250+40+60=80%

Cost = 150(-1)+40(100)+60(1)+250(0)=3910

Accuracy = 250+200/250+200+45+5=90%

Cost = 250(-1)+45(100)+5(1)+200(0)=4255

Cost vs Accuracy

Count	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

Cost	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	p	q
	Class=No	q	p

Accuracy is proportional to cost if

1. C(Yes|No)=C(No|Yes) = q
2. C(Yes|Yes)=C(No|No) = p

$N = a + b + c + d$

$Accuracy = (a + d)/N$

$$\begin{aligned} Cost &= p\,(a + d) + q\,(b + c) \\ &= p\,(a + d) + q\,(N - a - d) \\ &= q\,N - (q - p)(a + d) \\ &= N\,[q - (q-p) \times Accuracy] \end{aligned}$$

Two models Model M1 and M2 both of which are having correct predictions and wrong predictions.

Accuracy for Model M2 is higher compare to Model M1, however the cost for Model M2 is higher compare to Model M1.

So it depends on what kind of problem statement we are facing.

If we are focusing on accuracy then we will go with the Model M2 (In this case we need to compromise on cost) , however if we are focusing on cost then we will go with the Model M1 (In this case we need to compromise on accuracy).

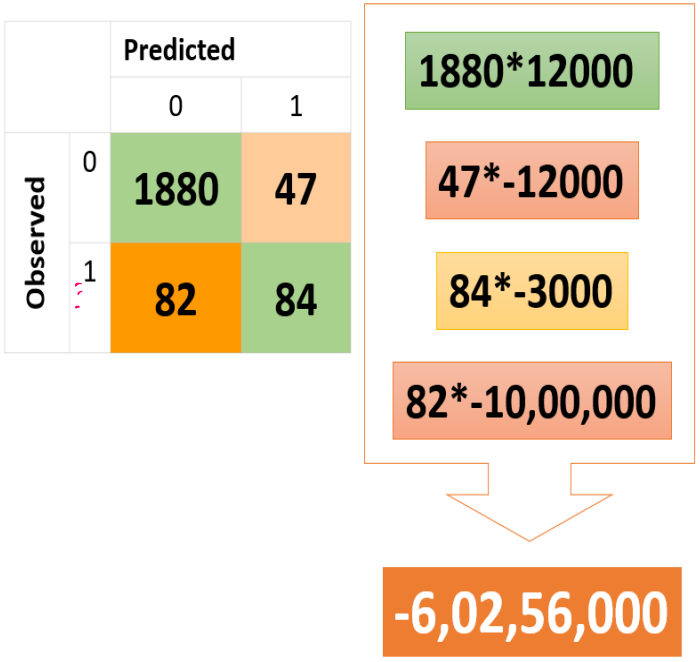
Let us understand some model performance measures based on an example of predicting loan default.

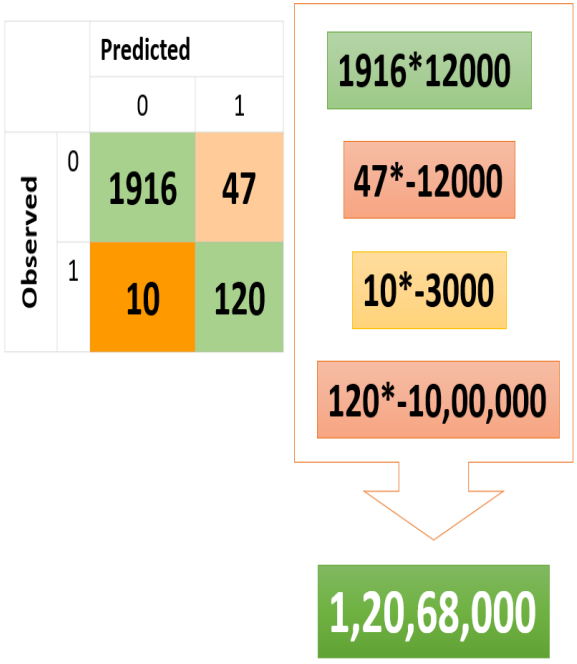
A loan default dataset is a typical example of an imbalanced dataset where the two classes are Loan default Y and Loan default N.

The number of loan defaulters is usually a very small fraction of the total dataset — not more than 7–8%.

This provides a classical imbalanced dataset to understand why cost functions are critical is deciding on which model to use.

Since the false negative cost is the highest, the most ***optimal model will be the one with the minimum false negatives***. In other words, a model with higher sensitivity / recall will fetch a higher net revenue compared to other models. Now that we have the method to calculate the net revenue, let us compare 2 models based on their confusion matrix:





Cost-Sensitive Measures

$$\text{Precision (p)} = \frac{a}{a+c}$$

$$\text{Recall (r)} = \frac{a}{a+b}$$

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Methods for Performance Evaluation

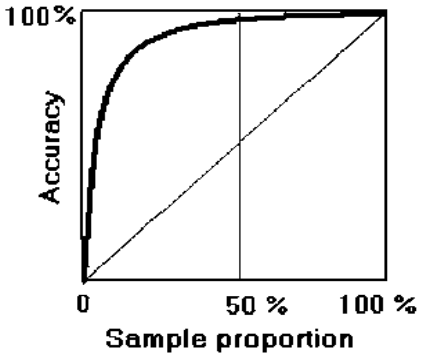
- How to obtain a reliable estimate of performance?

- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

Learning Curve

ACCURACY AS A FUNCTION OF SAMPLE SIZE

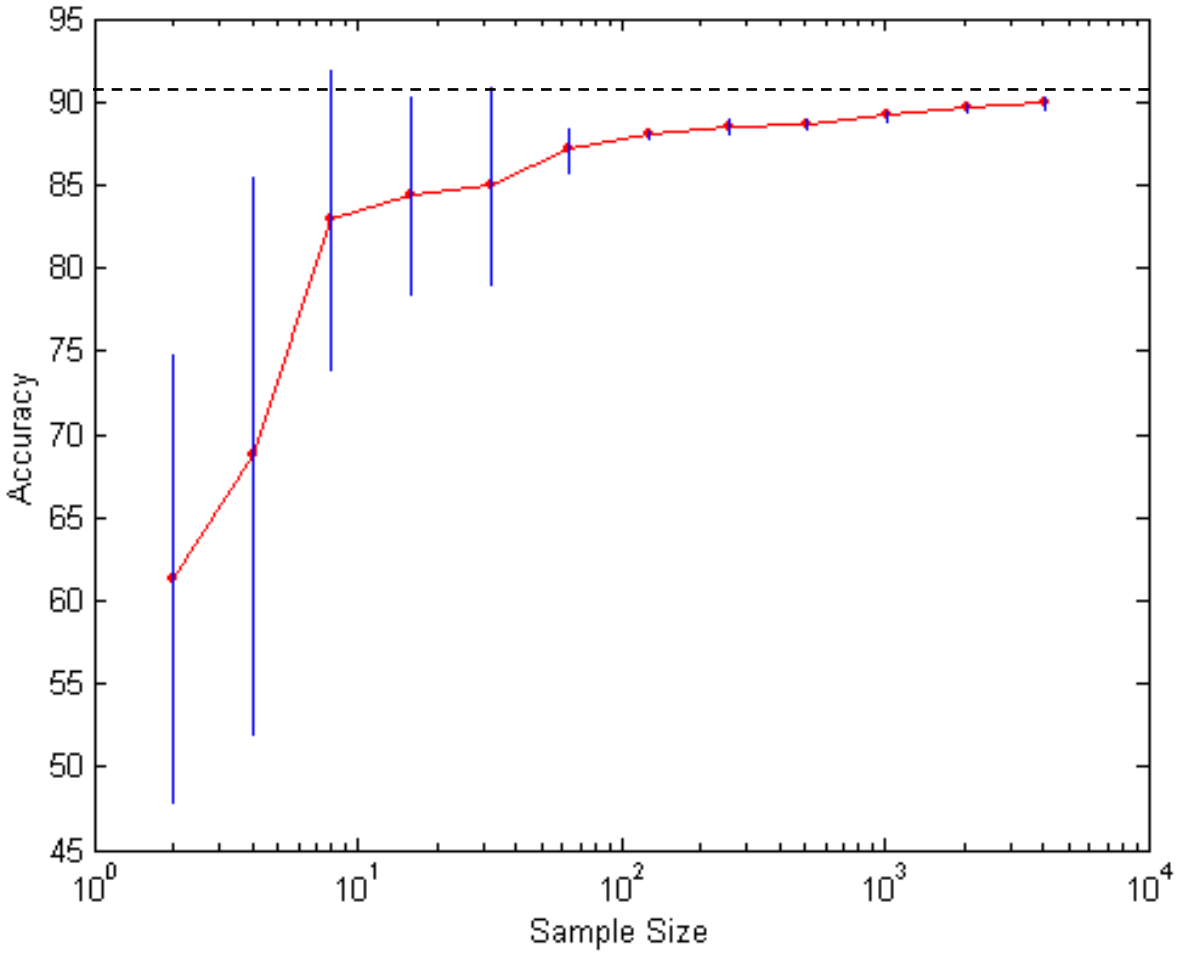
The following diagram illustrates the pattern of accuracy growth when sample size increases.



To be noted that:

- Accuracy is 100% when the entire population has been examined (as in the case of a census).
- The pattern of accuracy growth is not linear. The accuracy of a sample equal to half the data population size is not 50% but very near to 100%.
- Good accuracy levels can be achieved at relatively small sample sizes, provided that the samples are representative.
- The result of this relationship is that beyond a certain sample size the gains in accuracy are negligible, while sampling costs increase significantly

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve:
  - Arithmetic sampling (Langley, et al)
  - Geometric sampling (Provost et al)

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

Methods of Estimation

- Holdout
  - Reserve 2/3 for training and 1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation (CV)
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- Stratified sampling (in hold out if no representation of the class- ensure at least equal representation from every class)
  - oversampling vs undersampling
- Bootstrap
  - Sampling with replacement (CV with replacement)

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms
- ROC curve plots TP (on the y-axis) against FP (on the x-axis)
- Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point

ROC chart & Area under the curve (AUC)

ROC chart is a plot of (1-specificity) (FP) in the X axis and sensitivity (TP) in the Y axis.

Area under the ROC curve is a measure of model performance.

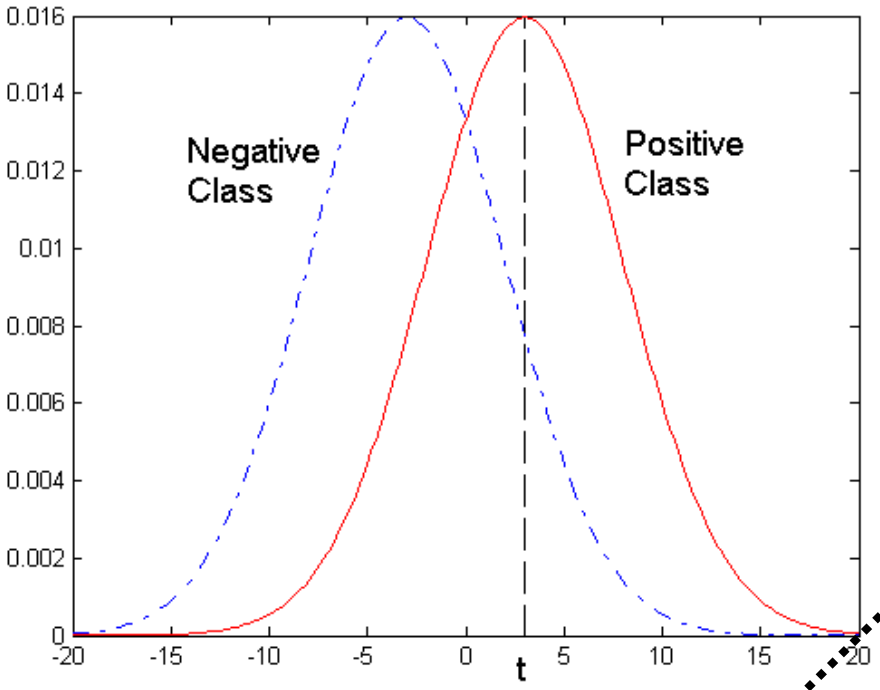
The AUC of a random classifier is 50% and that of a perfect classifier is 100%.

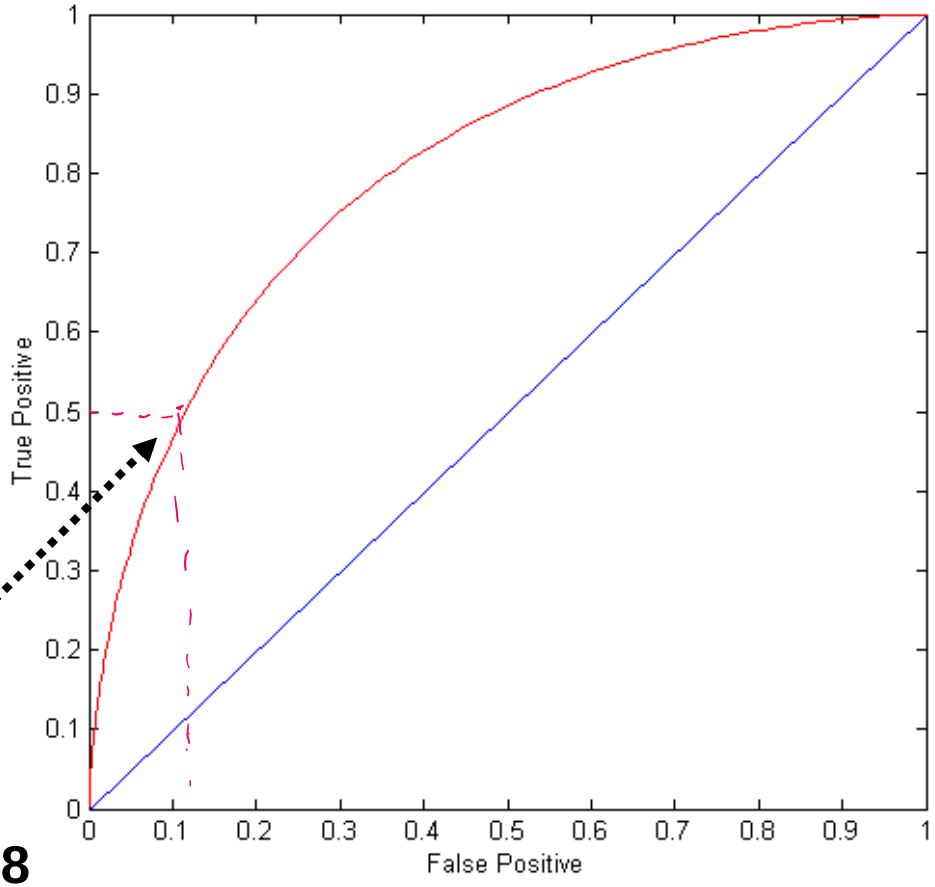
For practical situations, an AUC of over 70% is desirable.

ROC Curve

- 1-dimensional data set containing 2 classes (positive and negative)

- any points located at x > t is classified as positive





At threshold t:

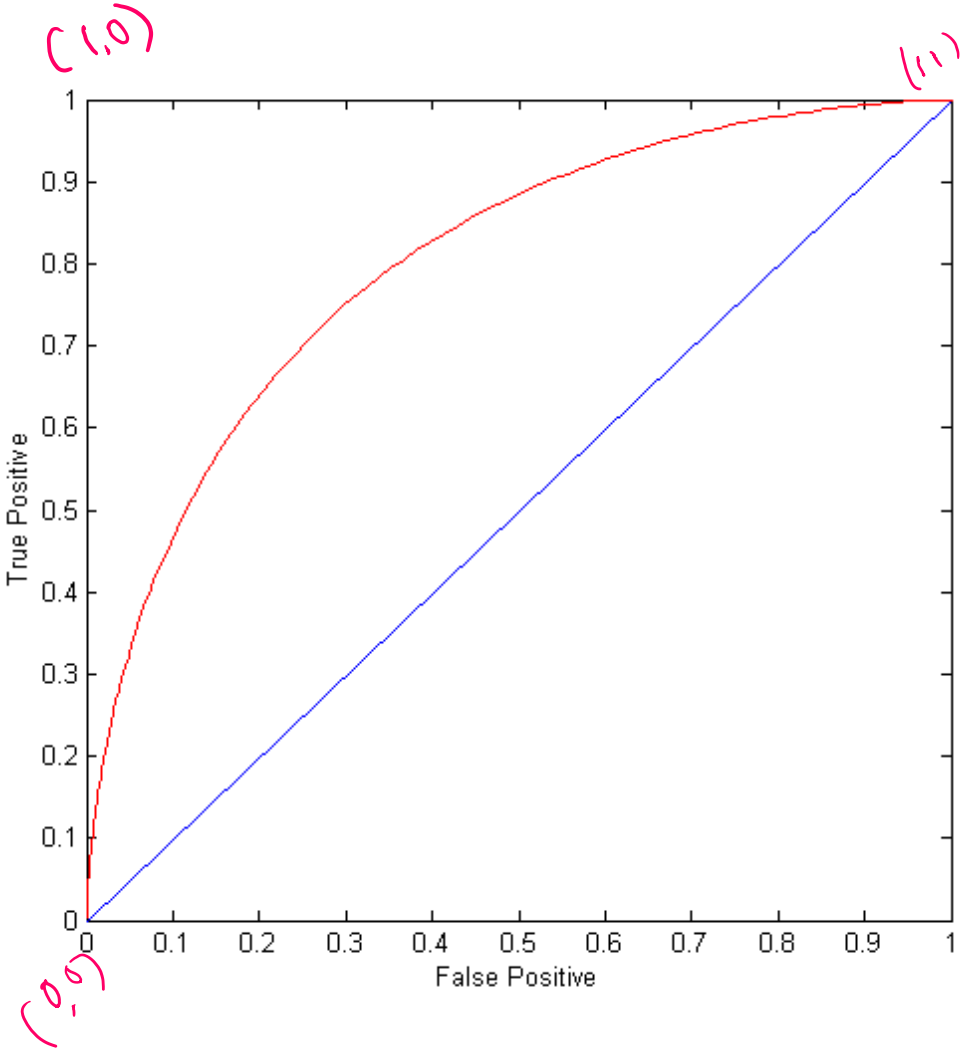
TP=0.5, FN=0.5, FP=0.12, TN=0.88

ROC Curve

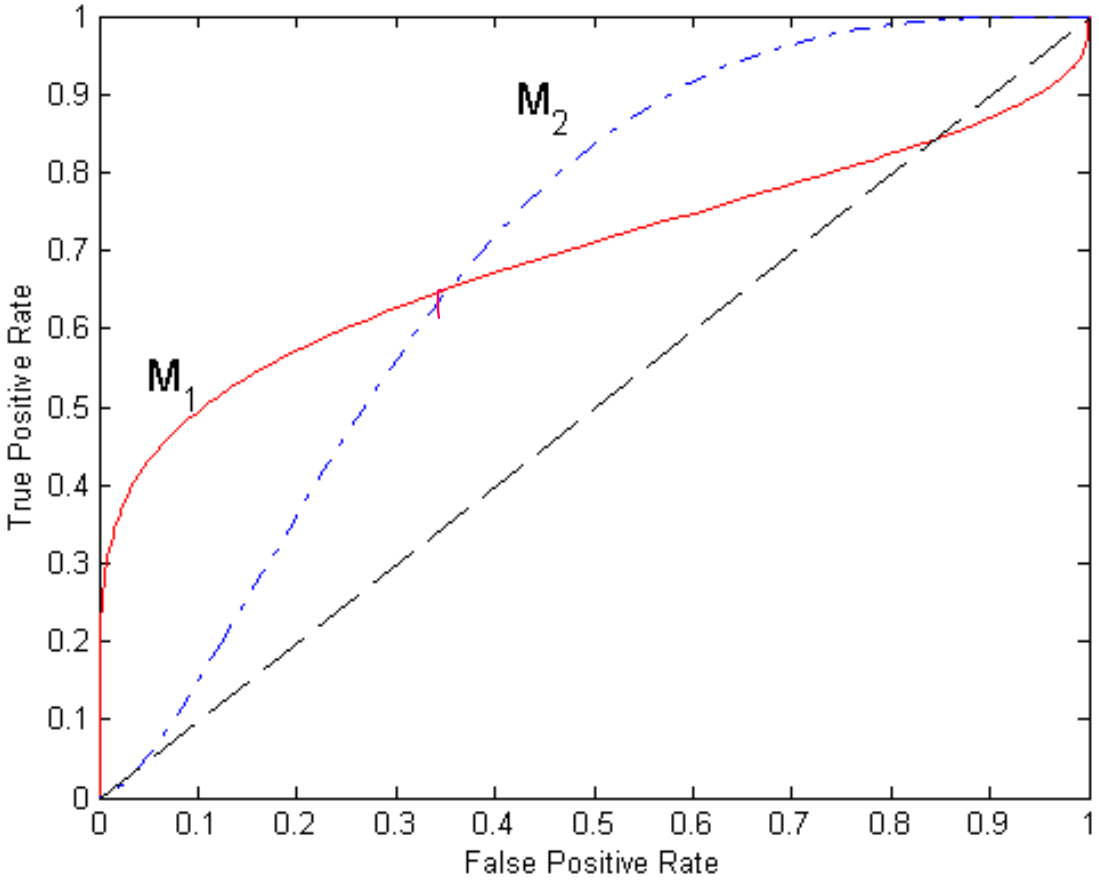
(TP,FP):

- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (1,0): ideal

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



Using ROC for Model Comparison



- No model consistently outperform the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5

How to Construct an ROC curve

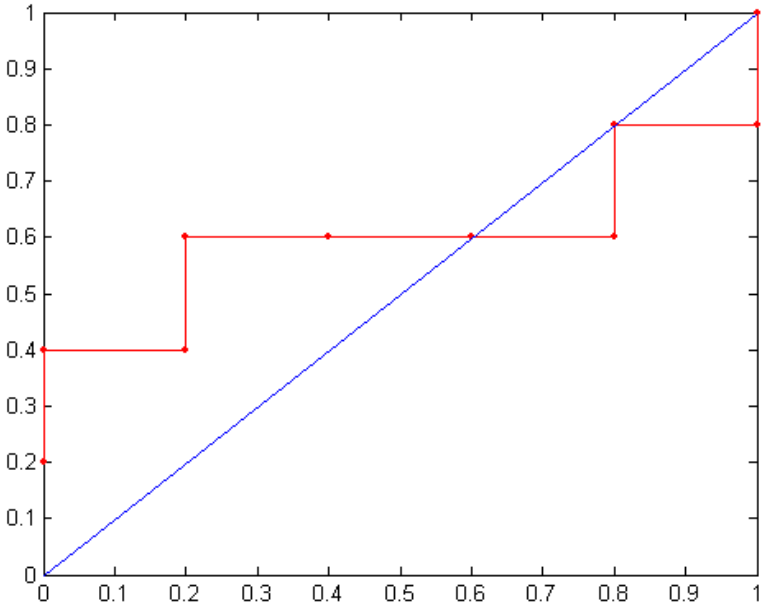
Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/(TP+FN)
- FP rate, FPR = FP/(FP + TN)

How to construct an ROC curve

	Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=		0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
	TP	5	4	4	3	3	3	3	2	2	1	0
	FP	5	5	4	4	3	2	1	1	0	0	0
	TN	0	0	1	1	2	3	4	4	5	5	5
	FN	0	1	1	2	2	2	2	3	3	4	5
→	TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
→	FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

ROC Curve:



Test of Significance

- Given two models:
  - Model M1: accuracy = 85%, tested on 30 instances
  - Model M2: accuracy = 75%, tested on 5000 instances

- Can we say M1 is better than M2?
  - How much confidence can we place on accuracy of M1 and M2?
  - Can the difference in performance measure be explained as a result of random fluctuations in the test set?