Classification

Concepts and Techniques

Classification: Definition

Data is a collection of records (training set)

Each **record** is by characterized by a **tuple** $(\mathbf{x},y)$, where $\mathbf{x}$ is the *attribute* set and y is the *class* label

$\mathbf{x}$: attribute, predictor, independent variable → input

y: class, response, dependent variable → output

Task: Learn a model that maps each attribute set $\mathbf{x}$ into one of the predefined class labels

A classification model is an abstract representation of the relationship between the attribute set and the class label

Examples of Classification Task

Binary classification

problems, in which each data instance can be categorized into one of two classes

Multiclass classification

problems, in which each data instance can be categorized into one of multiple classes

Task	Attribute set **(x)**	Class label (y)	*Classification type*
Categorizing email messages	Features extracted from email message header and content	spam or non-spam	Binary
Identifying tumor cells	Features extracted from x-rays or MRI scans	malignant or benign cells	Binary
Cataloging galaxies	Features extracted from telescope images	Elliptical, spiral, or irregular-shaped galaxies	Multiclass

Classification Models

Roles of Classification Models:

- **Predictive Model**: Classifies previously unlabeled instances.

- Descriptive Model: Identifies distinguishing characteristics between different classes.

  Example: Medical diagnosis, where justifying predictions is crucial.

Key Requirements:

- *Accuracy*: Predict outcomes correctly.

- *Efficiency*: fast response times.

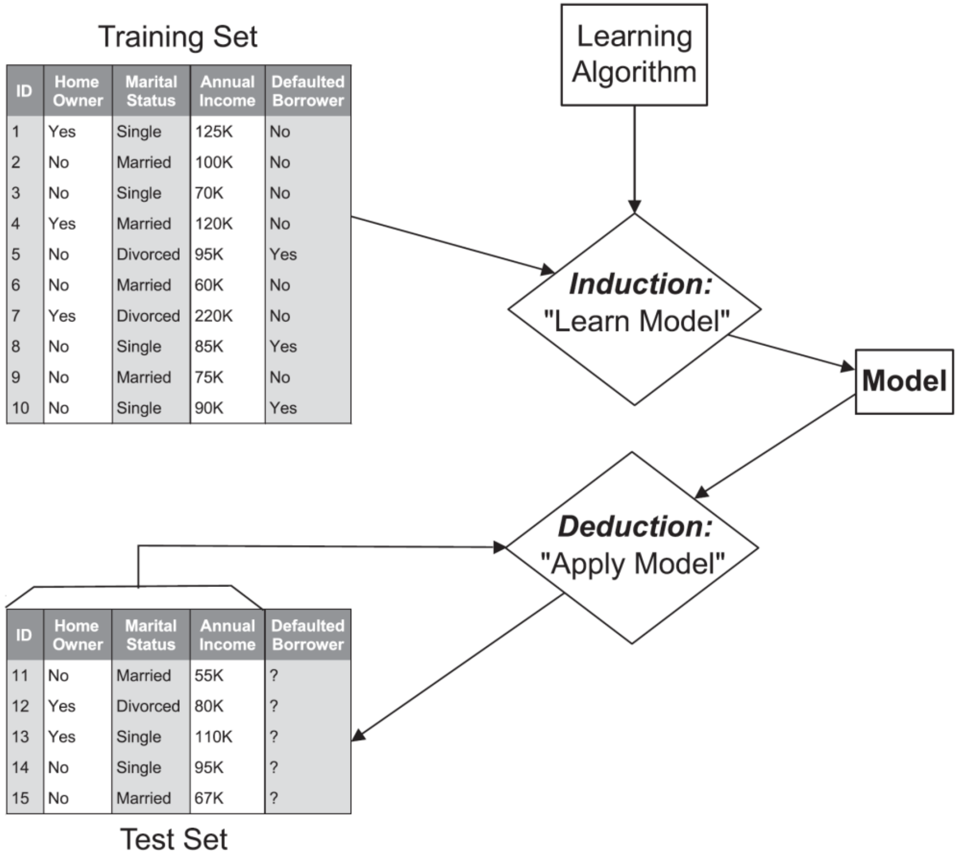
General Framework for Classification

Key Concepts:

- **Classifier**: A model used to perform classification.
- **Training Set**: A set of instances with attribute values and class labels used to build the model.
- **Learning Algorithm**: The systematic approach used to create the model from the training set. This process is called **Induction** (learning/building a model).
- **Deduction**: Applying the learned model to new, unseen test instances to predict their class labels.

Steps in Classification:

- **Induction (training)**: Learn the model from training data.
- **Deduction (inference)**: Apply the model to classify new instances.



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
11	No	Married	55K	?
12	Yes	Divorced	80K	?
13	Yes	Single	110K	?
14	No	Single	95K	?
15	No	Married	67K	?

Figure 3.3. General framework for building a classification model.

Classification performance

Pivotal concepts

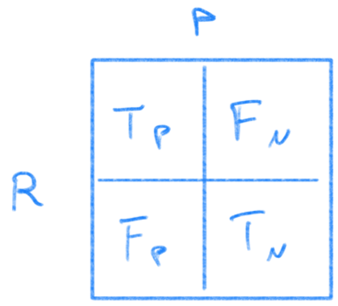
Independence of Training and Test Sets: Ensures the model can predict class labels for unseen instances.

Generalization Performance: A model with good generalization can accurately predict labels for new, unseen data.

Model Performance Evaluation: Compare **predicted labels** against **true labels** to evaluate accuracy.

Model Performance Evaluation: Compare **predicted labels** against **true labels** to evaluate accuracy.

- **Confusion Matrix**



- **Accuracy**: number of correct over total
- **Error Rate**: number of wrong over total

Classification Techniques

Base Classifiers

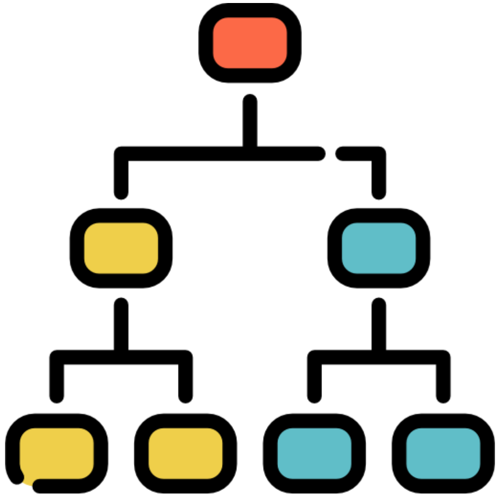
- Decision Tree based Methods
- Nearest-neighbor
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines
- Neural Networks, Deep Neural Nets

Ensemble Classifiers

- Boosting, Bagging, Random Forests

Decision

Tree



Decision Tree

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It models decisions and their possible consequences in a tree-like structure, where:

Root node: node with no incoming link

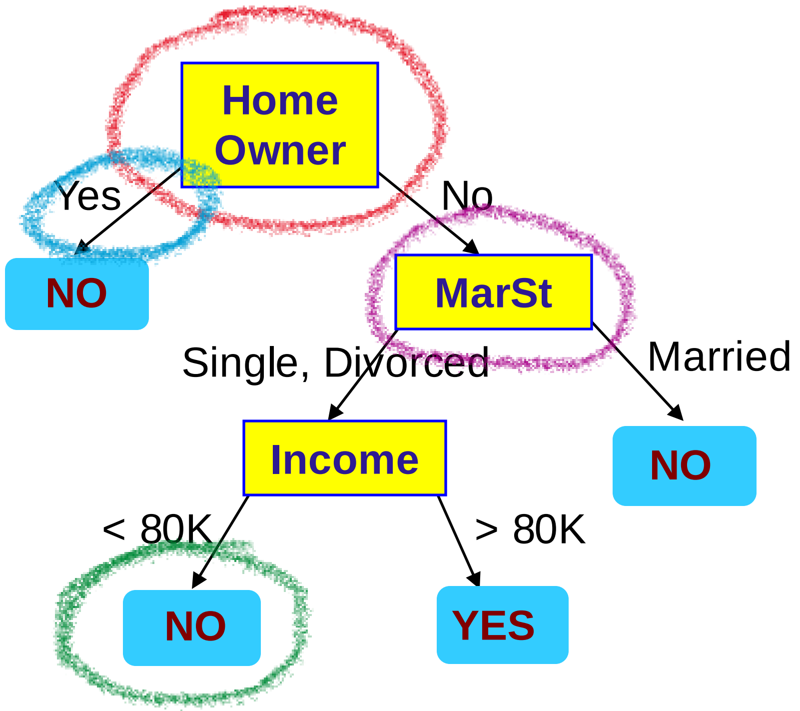
Nodes: represent decisions or tests on attributes (features).

Edges: represent the outcome of the test, leading to either more nodes or leaf nodes.

Leaf or terminal nodes: represent the final outcome or class label in classification tasks.

The tree is built by splitting the data based on feature values that best separate the classes or predict the target variable.

The goal is to create a tree that accurately classifies new instances by following the decision rules in the tree structure.

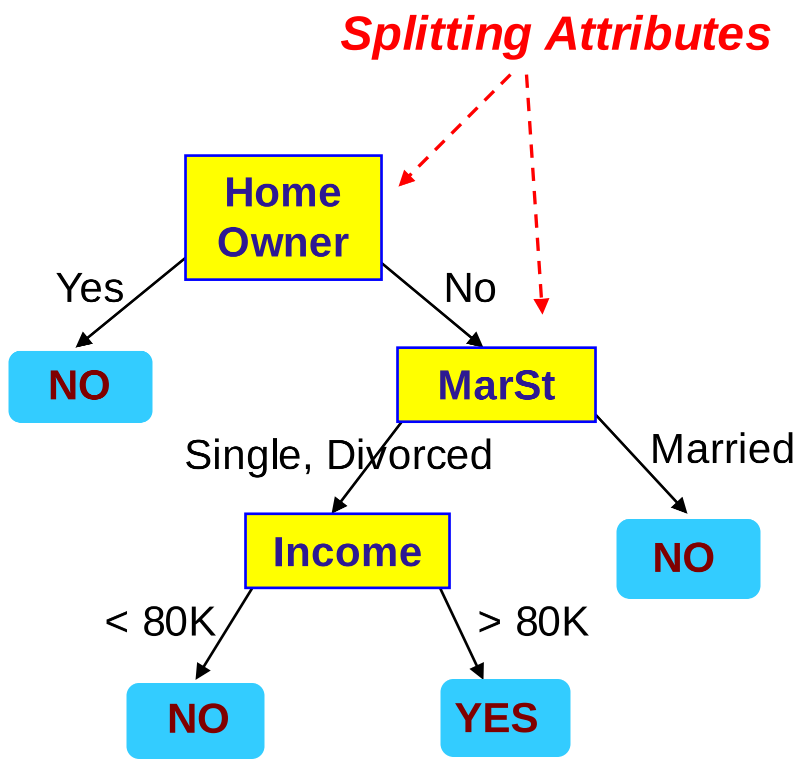


Example of a Decision Tree

categorical categorical continuous class

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

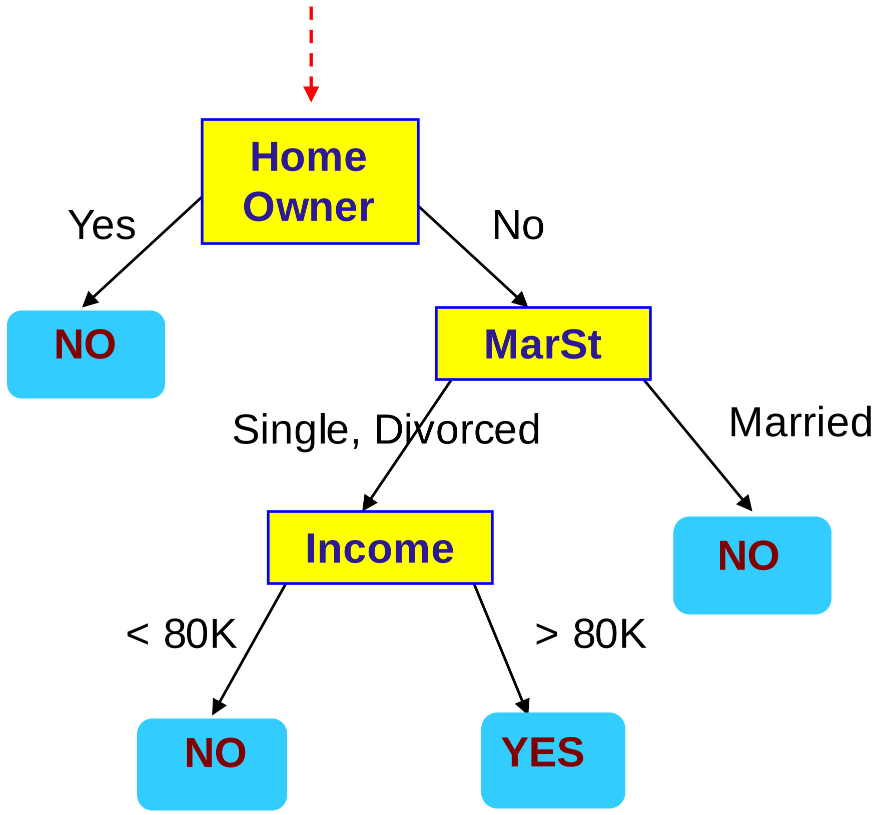
Training Data



Model: Decision Tree

Apply Model to Test Data

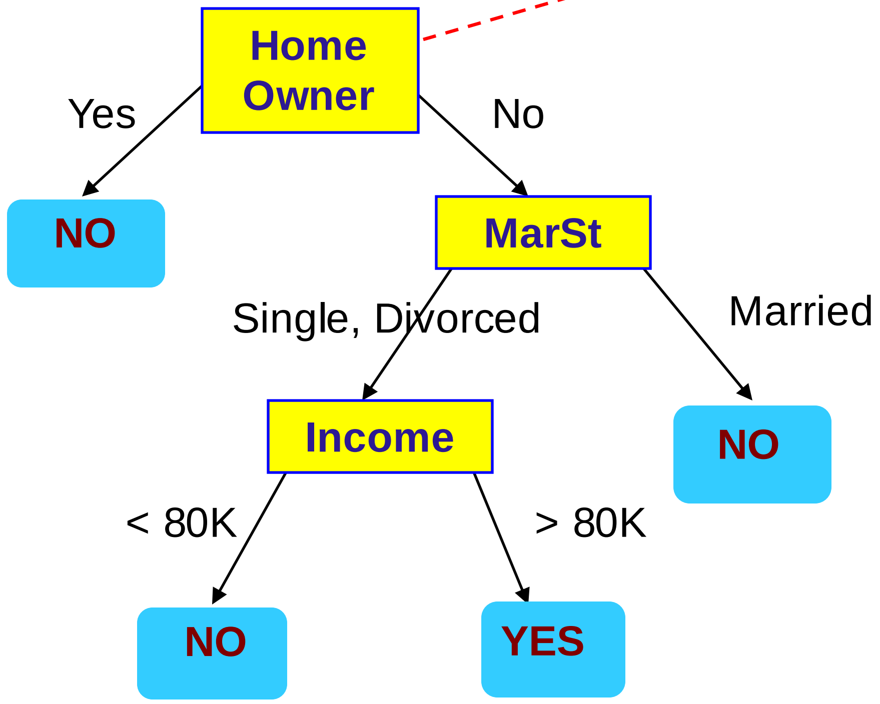
Start from the root of tree.



Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Apply Model to Test Data



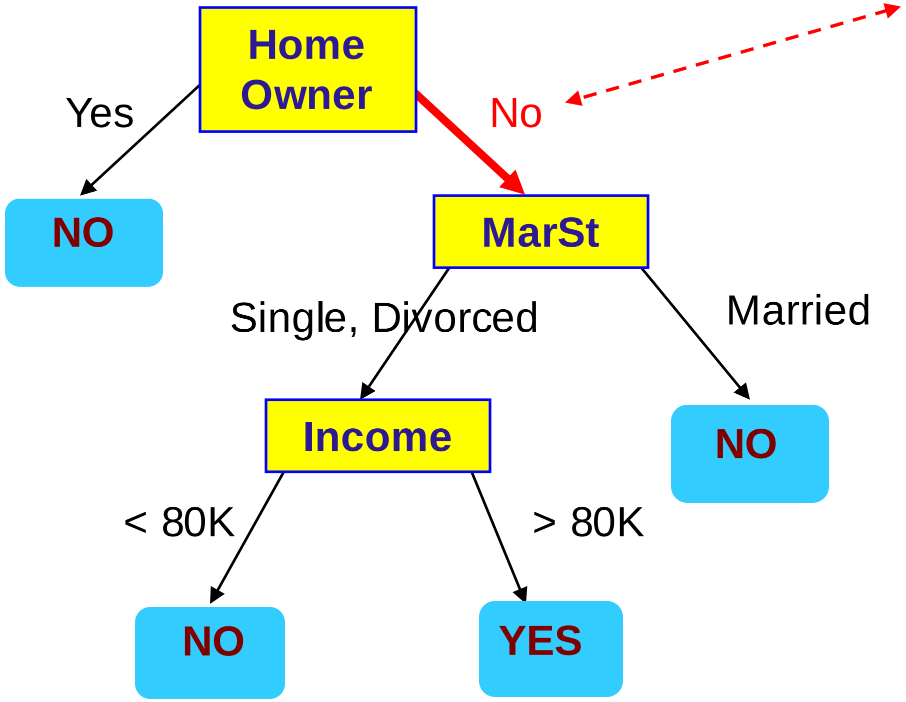
Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

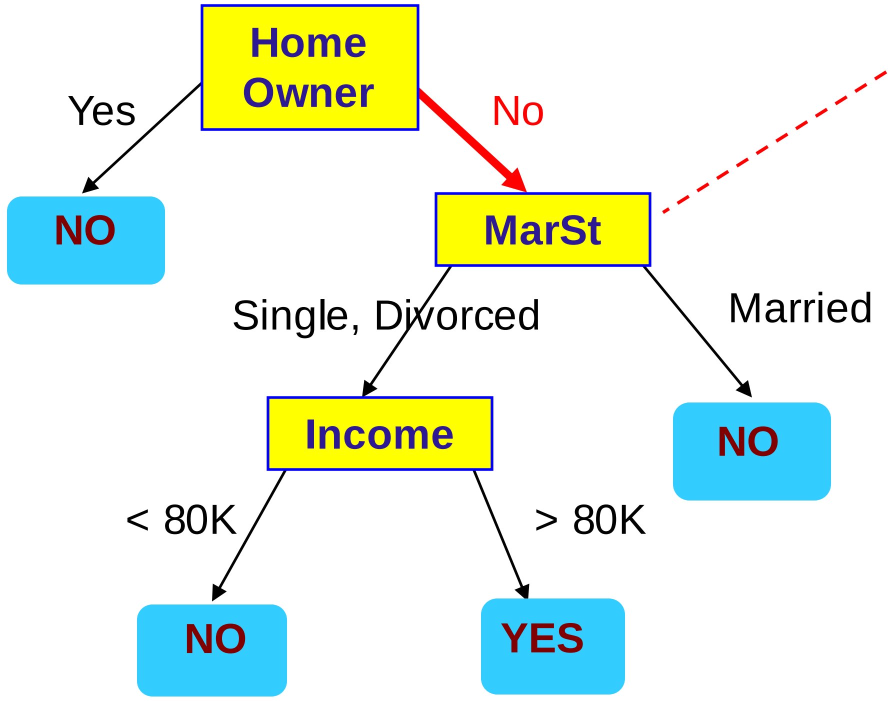
Apply Model to Test Data

Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



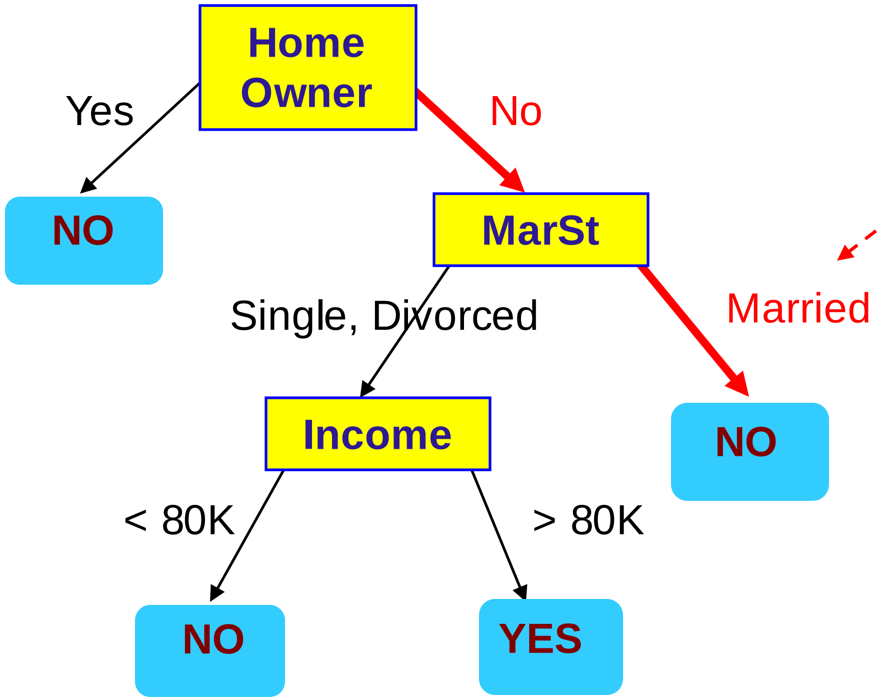
Apply Model to Test Data



Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

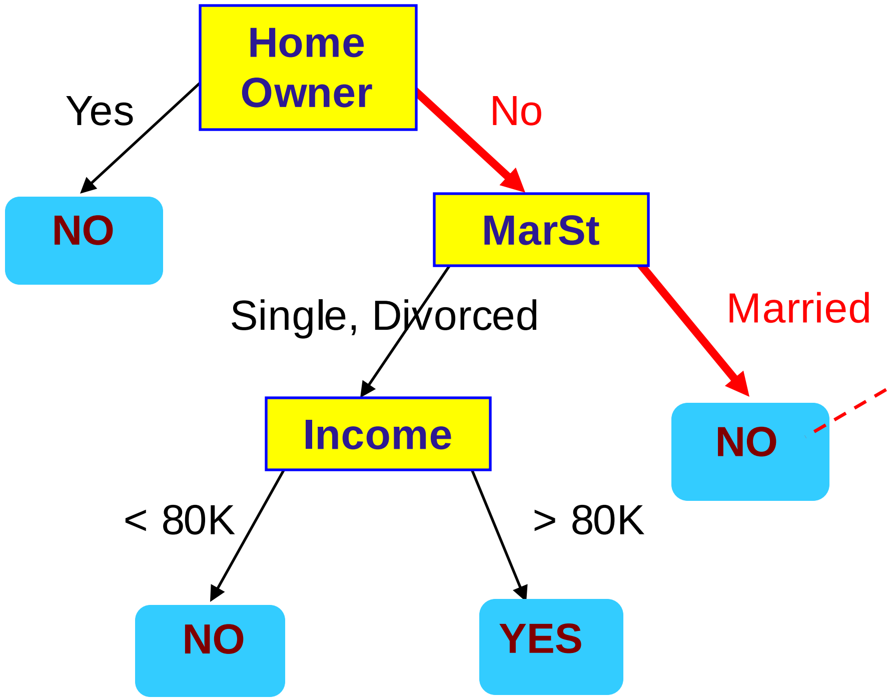
Apply Model to Test Data



Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Apply Model to Test Data



Test Data

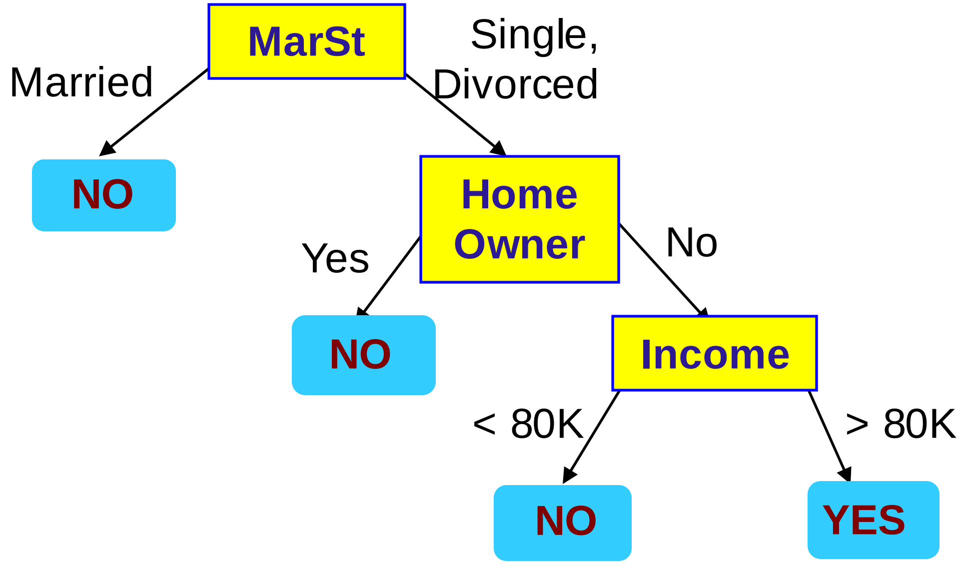
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Assign Defaulted to "No"

Another Example of Decision Tree

categorical categorical continuous class

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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 Induction

Many Algorithms:

- Hunt’s Algorithm (one of the earliest)
- CART
- ID3, C4.5
- SLIQ,SPRINT

General Structure of Hunt's Algorithm

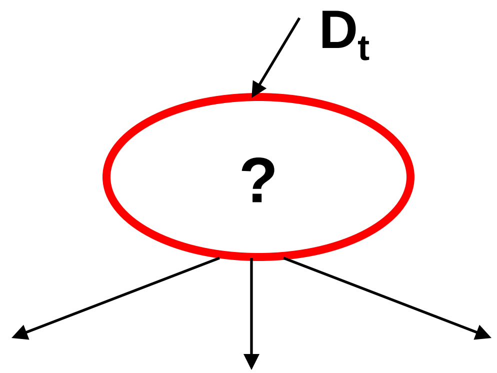
Let D_t be the set of training records that reach a node t

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 contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



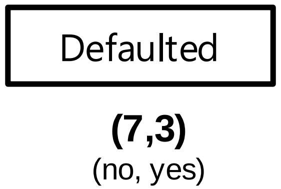
Hunt's Algorithm

Defaulted

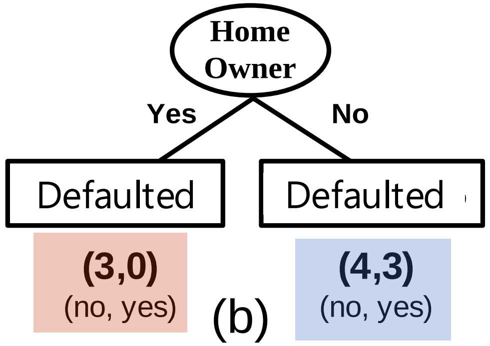
(7,3)
(no, yes)
(a)

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



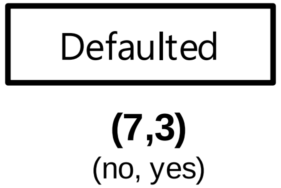
(a)



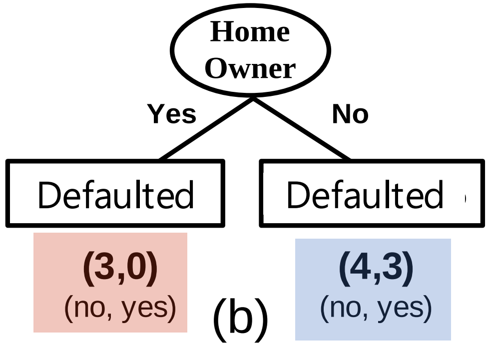
(b)

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

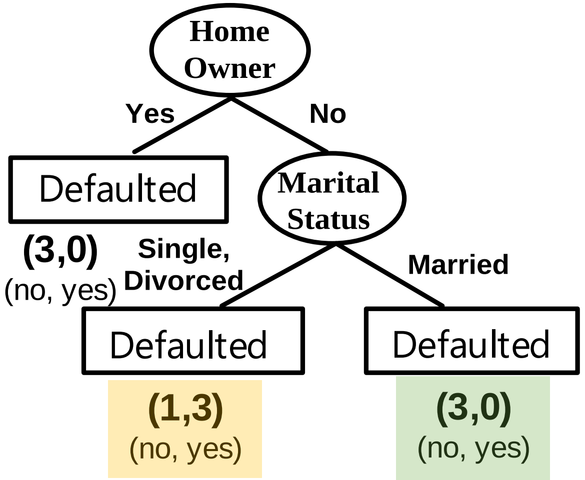
Hunt’s Algorithm



(a)



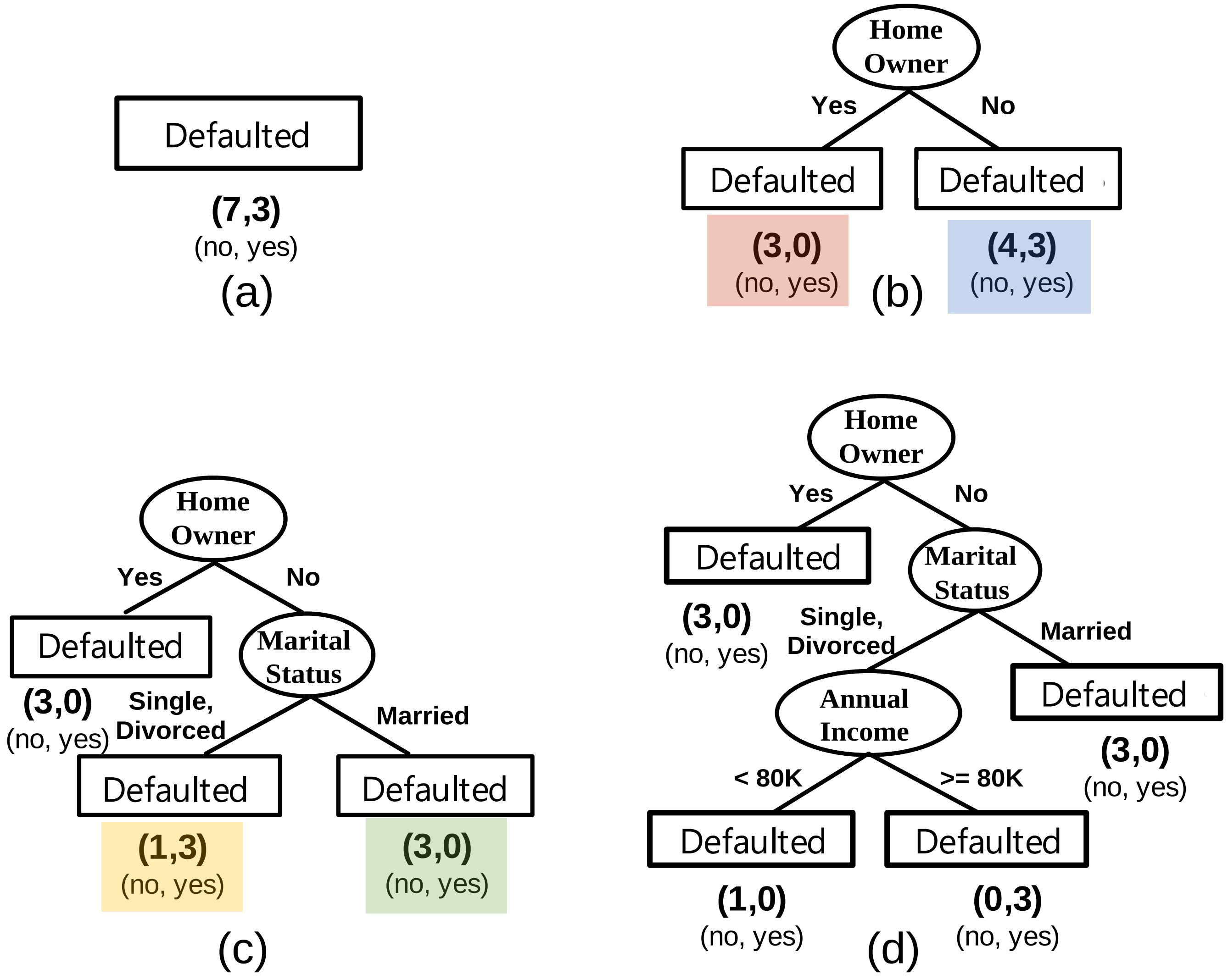
(b)



(c)

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Design Issues of Decision Tree Induction

How should training records be **split**?

– Method for expressing test condition depending on *attribute types*

– Measure for **evaluating the goodness** of a test condition

How should the splitting procedure **stop**?

– Stop splitting if all the records belong to the **same class** or have **identical attribute** values

– Early termination

Methods for Expressing Test Conditions

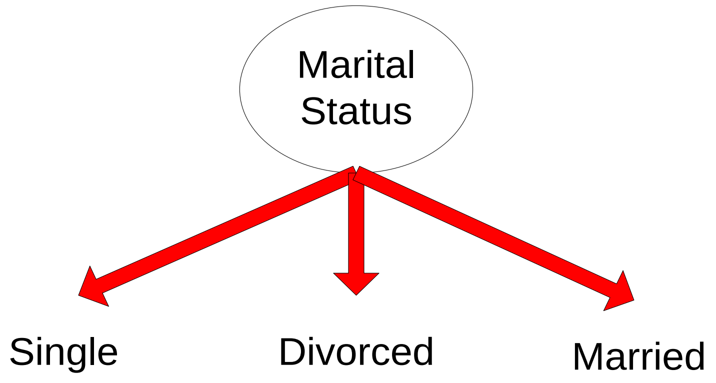
Depends on attribute types

- Binary
- Nominal
- Ordinal
- Continuous

Test Condition for Nominal Attributes

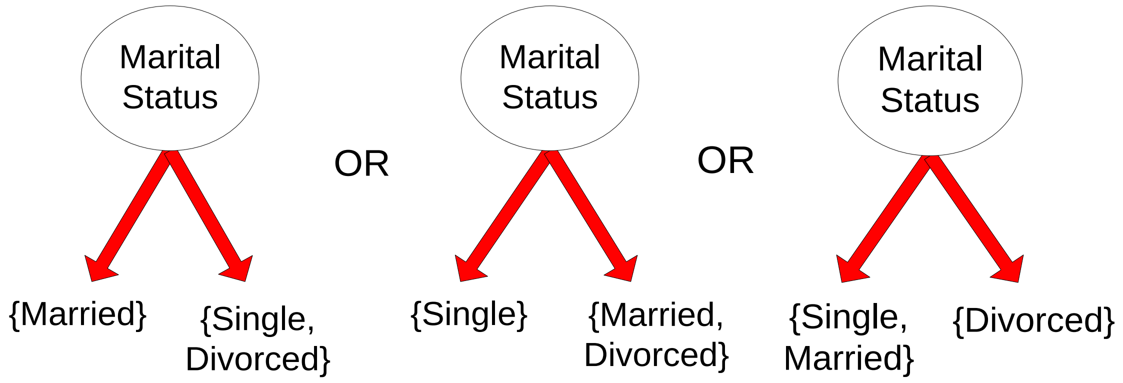
Multi-way split:

– Use as many partitions as distinct values.



Binary split:

– Divides values into two subsets



Test Condition for Ordinal Attributes

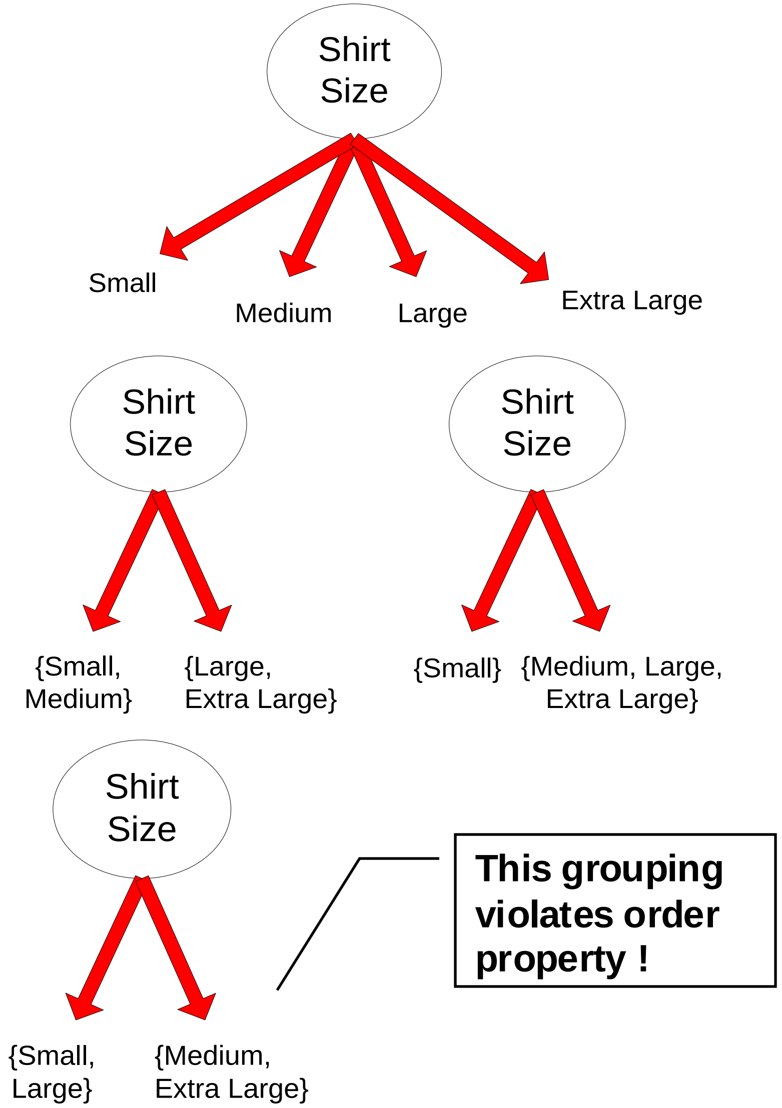
Multi-way split:

– Use as many partitions as distinct values

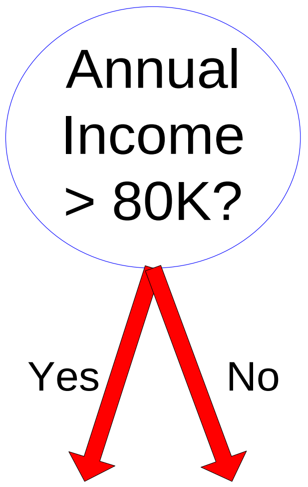
Binary split:

– Divides values into two subsets

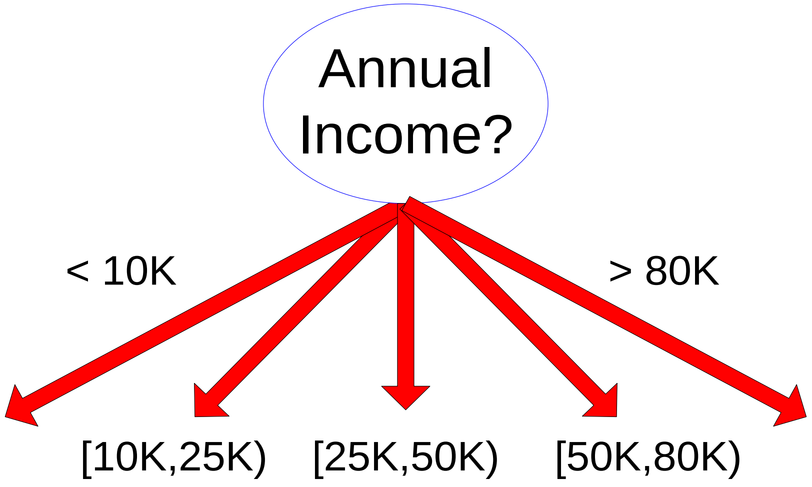
– Preserve order property among attribute values



Test Condition for Continuous Attributes



(i) Binary split



(ii) Multi-way split

Splitting Based on Continuous Attributes

Different ways of handling

- Discretization to form an ordinal categorical attribute

  Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.

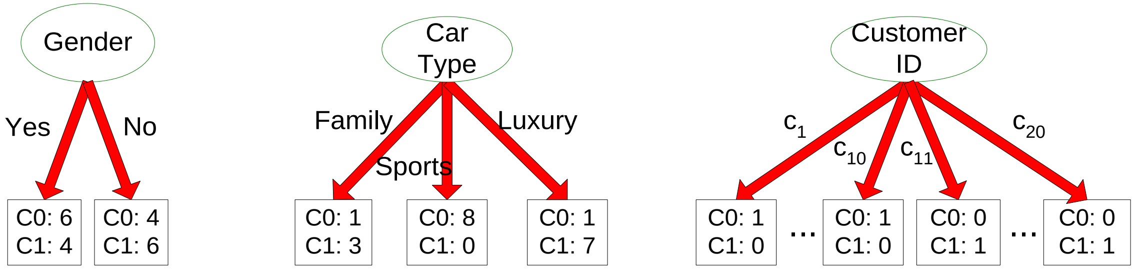
  - Static – discretize once at the beginning
  - Dynamic – repeat at each node

- Binary Decision: ($A < v$) or ($A \geq v$)
  - consider all possible splits and finds the best cut
  - can be more compute intensive

How to determine the Best Split

Goodness of an attribute test condition: many measures try to give preference to attribute test conditions that partition the training instances into **purer subsets** in the child nodes, which mostly have the same class labels

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



Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

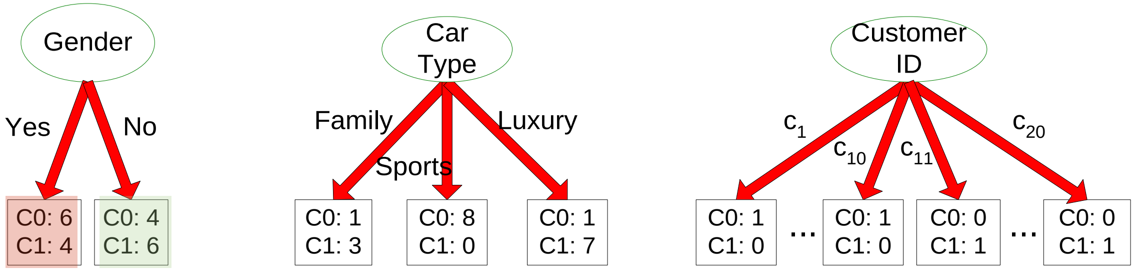
Which test condition is the best?

NOTE: Larger trees are less desirable as they are more susceptible to model overfitting

How to determine the Best Split

Goodness of an attribute test condition: many measures try to give preference to attribute test conditions that partition the training instances into **purer subsets** in the child nodes, which mostly have the same class labels

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



Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Which test condition is the best?

NOTE: Larger trees are less desirable as they are more susceptible to model overfitting

How to determine the Best Split

Greedy approach: Nodes with purer class distribution are preferred

We need a measure of node impurity !

C0: 5 C1: 5	**High degree of impurity**
C0: 9 C1: 1	**Low degree of impurity**

Let's define the ***RELATIVE FREQUENCY*** $p_i(t)$

$C_0 = 5$ $\quad P_0(t) = \frac{5}{10}$

$C_1 = 5$ $\quad P_1(t) = \frac{5}{10}$

$C_0 = 1$ $\quad P_0(t) = \frac{1}{10}$

$C_1 = 9$ $\quad P_1(t) = \frac{9}{10}$

Measures of **Single Node Impurity**

Gini Index

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node **t**, and $\boldsymbol{c}$ is the total number of classes

Entropy

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

$2^5 = 32$

$log_2(32) = 5$

Misclassification error

$$Classification\ error = 1 - \max[p_i(t)]$$

Compute **Single Node Impurity**

C0: 5
C1: 5

High degree of impurity

$$C_0 = 5 \qquad P_0(t) = \frac{5}{10}$$

$$C_1 = 5 \qquad P_1(t) = \frac{5}{10}$$

$$GI = 1 - \left(\left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^2\right) = 0{,}5$$

$$E = -\left(\frac{5}{10}\log_2\frac{5}{10} + \frac{5}{10}\log_2\frac{5}{10}\right) = 1$$

C0: 9
C1: 1

Low degree of impurity

$$C_0 = 1 \qquad P_0(t) = \frac{1}{10}$$

$$C_1 = 9 \qquad P_1(t) = \frac{9}{10}$$

$$GI = 1 - \left(\left(\frac{1}{10}\right)^2 + \left(\frac{9}{10}\right)^2\right) = 0{,}18$$

$$E = -\left(\frac{1}{10}\log_2\frac{1}{10} + \frac{9}{10}\log_2\frac{9}{10}\right) = 0{,}469$$

Compute Collective impurity of **Child** nodes

Once a decision rule **attribute test condition** have been applied it is possible to compute the collective impurity of child nodes

Consider an **attribute test condition** that splits a node containing N training instances into k children,
{v1, v2, · · · , vk}

Let N(vj) be the number of training instances associated with a child node vj, whose impurity value is I(vj)

The **weighted impurity** measure of its **children**

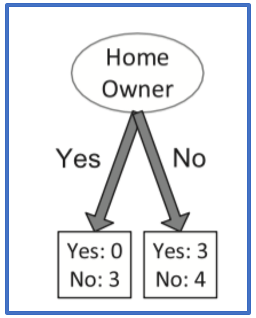
$$I(\text{children}) = \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j),$$

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

$$I(\texttt{Home Owner = yes}) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$$

$$I(\texttt{Home Owner = no}) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985$$

$$I(\texttt{Home Owner}) = \frac{3}{10} \times 0 + \frac{7}{10} \times 0.985 = 0.690$$

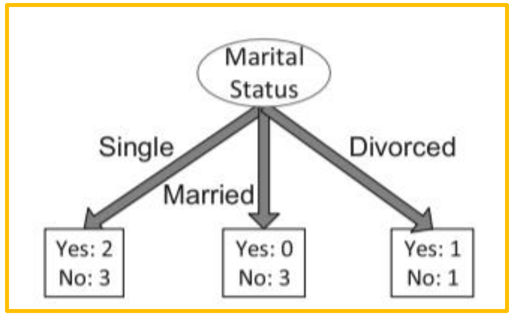


$$I(\texttt{Marital Status = Single}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$

$$I(\texttt{Marital Status = Married}) = -\frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3} = 0$$

$$I(\texttt{Marital Status = Divorced}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1.000$$

$$I(\texttt{Marital Status}) = \frac{5}{10} \times 0.971 + \frac{3}{10} \times 0 + \frac{2}{10} \times 1 = 0.686$$



Finding the Best Split

1. Compute impurity measure **I(parent)** before splitting

2. Compute impurity measure **I(children)** after splitting

    - Compute impurity measure of each child node

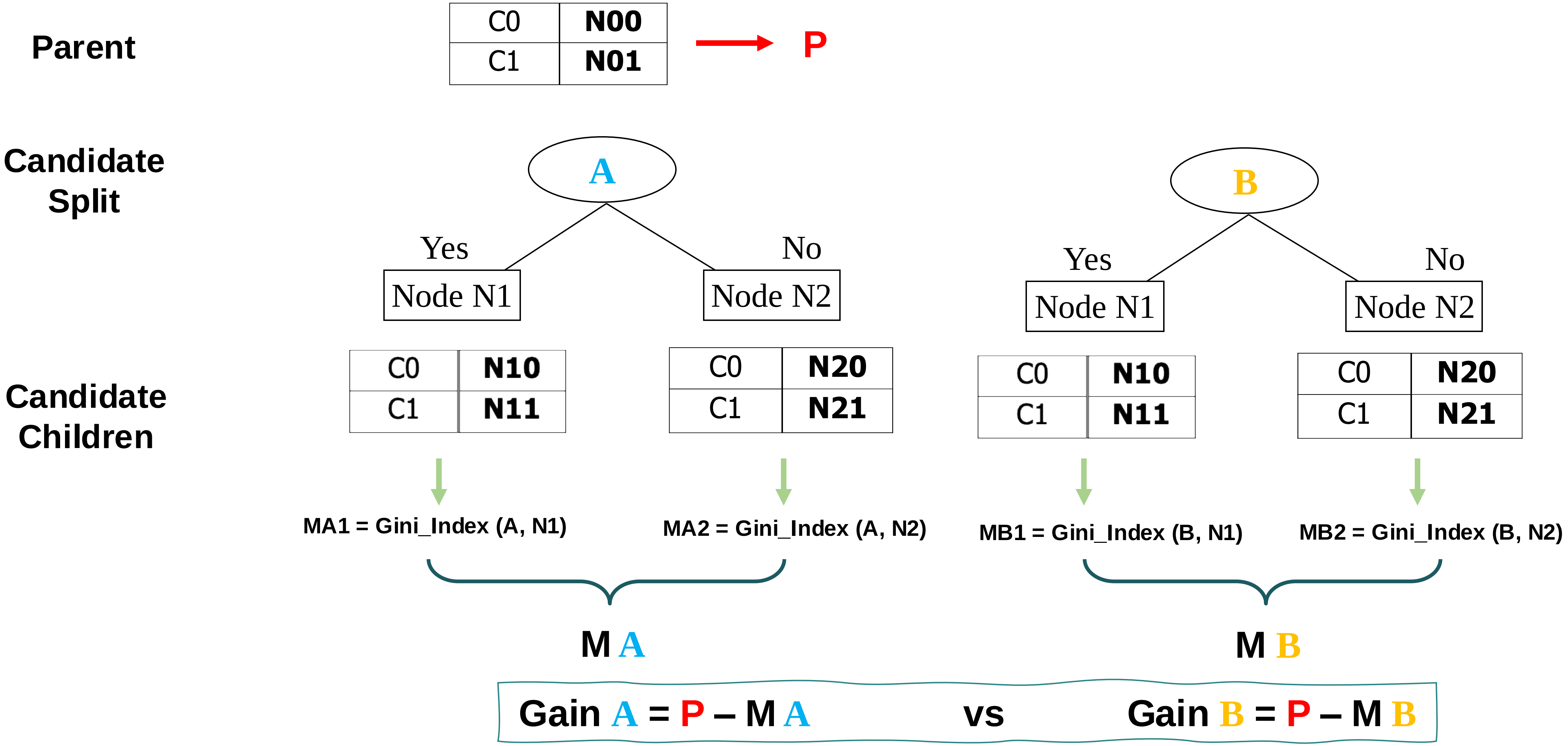
    - **I(children)** is the weighted impurity of **child** nodes

3. Choose the attribute test condition that produces the highest gain

$$\Delta = I(\text{parent}) - I(\text{children}),$$

or equivalently, lowest impurity measure after splitting

Finding the Best Split

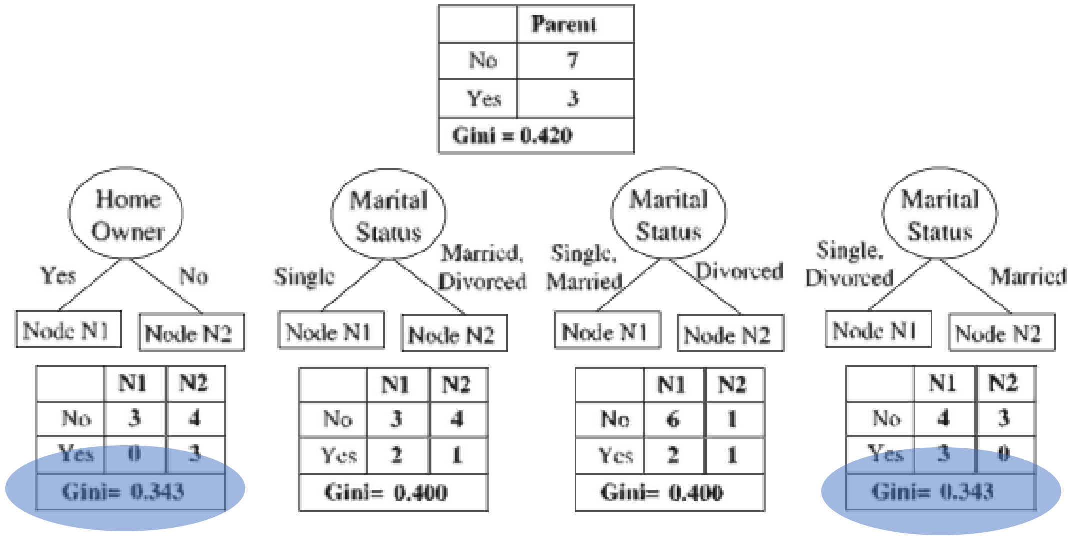


Finding the Best Split

It can be shown that the gain is non-negative since I(parent) ≥ I(children) for any reasonable measure

The higher the gain, **the purer are the classes** in the child nodes relative to the parent node.

The splitting criterion in the decision tree learning algorithm selects the attribute test condition that shows the maximum gain (minimizing the weighted impurity measure of its children)



$$\Delta = I(\text{parent}) - I(\text{children}),$$

Home Owner and *Marital Status (with the highlited association)* shows the minimum weighted impurity

$$I(\text{children}) = \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j),$$

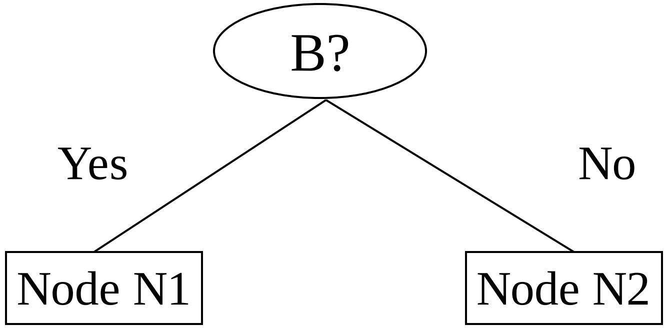
N(vj): number of training instances associated with the child node vj

Binary Attributes: Computing GINI Index

Splits into two partitions (child nodes)

Effect of Weighing partitions:

- Larger and purer partitions are sought



	Parent
C1	**7**
C2	**5**
Gini = 0.486	

Gini(N1)
$= 1 - (5/6)^2 - (1/6)^2$
$= 0.278$

Gini(N2)
$= 1 - (2/6)^2 - (4/6)^2$
$= 0.444$

	N1	N2
C1	**5**	**2**
C2	**1**	**4**
Gini=0.361		

Weighted Gini of N1 N2
= 6/12 * 0.278 +
6/12 * 0.444
= 0.361

Gain = 0.486 – 0.361 = 0.125

Categorical Attributes: Computing Gini Index

For each distinct value, gather counts for each class in the dataset

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

Two-way split (find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

Which of these is the best?

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.

ID	Home Owner	Marital Status	Annual Income	Defaulted
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

Annual Income ?

	≤ 80	> 80
Defaulted Yes	0	3
Defaulted No	3	4

Continuous Attributes: Computing Gini Index

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

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220

Continuous Attributes: Computing Gini Index

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

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual 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
	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >

Continuous Attributes: Computing Gini Index

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

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual 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														
No							3	4														
Gini							0.343															

Continuous Attributes: Computing Gini Index

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

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual 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	1	2												
No							3	4	3	4												
Gini							0.343		0.417													

Continuous Attributes: Computing Gini Index

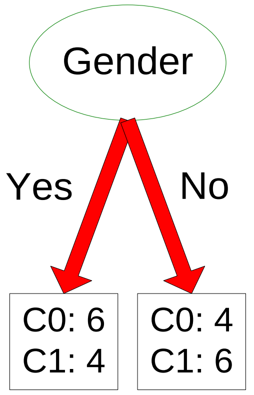
- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

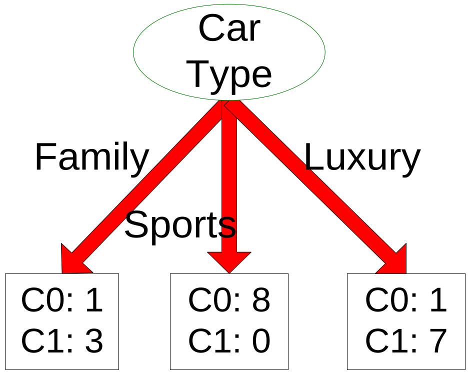
Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values	60	70	75	85	90	95	100	120	125	220

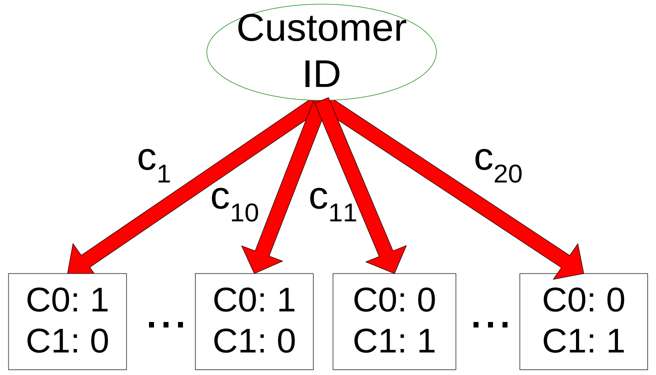
Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Problem with large number of partitions

Node **impurity measures** tend to **prefer** splits that result in **large number of partitions**, each being small but pure







Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Customer ID has highest information gain because *Gini* or *Entropy* is **0** for all the children

Gain Ratio

The number of children produced by the splitting attribute must be taken into consideration while deciding the best attribute test condition

A measure known as **Gain Ratio** is used to compensate for attributes that produce a large number of child nodes

$$\boldsymbol{Gain\ Ratio} = \frac{\Delta}{Split\ Info}$$

The **Split Information** measures the entropy of splitting a node into its child nodes and evaluates if the split results in a larger number of equally-sized child nodes or not

$$Split\ Info = -\sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

Gain Ratio

$$Gain\ Ratio = \frac{\Delta}{Split\ Info} \qquad Split\ Info = \sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	**0.163**		

SplitINFO = 1.52

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	**0.468**	

SplitINFO = 0.72

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	**0.167**	

SplitINFO = 0.97

Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

Decision Tree Based Classification

Advantages:

- Relatively inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant attributes
- Can easily handle irrelevant attributes (unless the attributes are interacting)

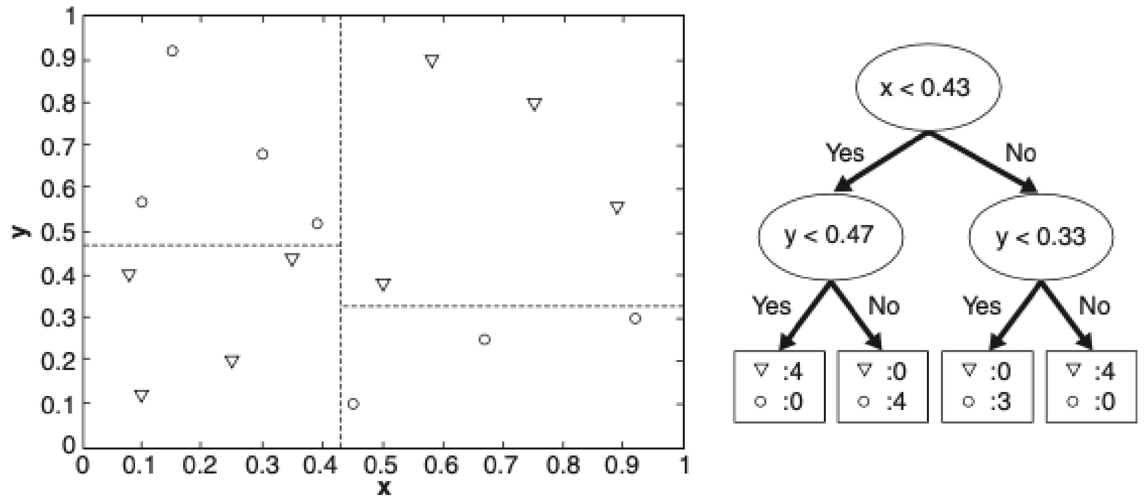
Disadvantages: .

- Due to the greedy nature of splitting criterion, interacting attributes (that can distinguish between classes together but not individually) may be passed over in favor of other attributed that are less discriminating.
- Each decision boundary involves only a single attribute

Handling interactions

Rectilinear Decision Boundaries:

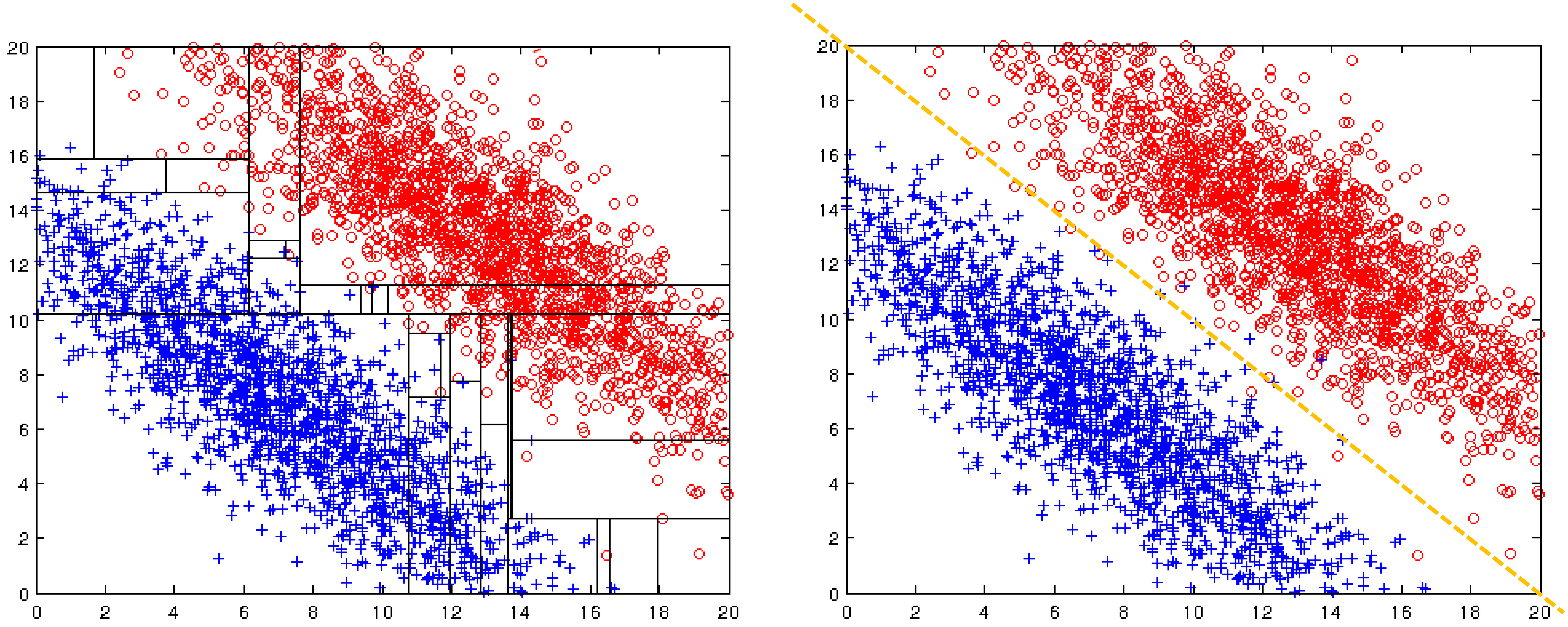
- **Single Attribute Testing**: Trees partition the attribute space based on one attribute at a time.
- **Rectilinear Boundaries**: The decision boundary is parallel to coordinate axes.
- **Limitation**: Restricts decision trees in representing complex boundaries, especially with continuous attributes.



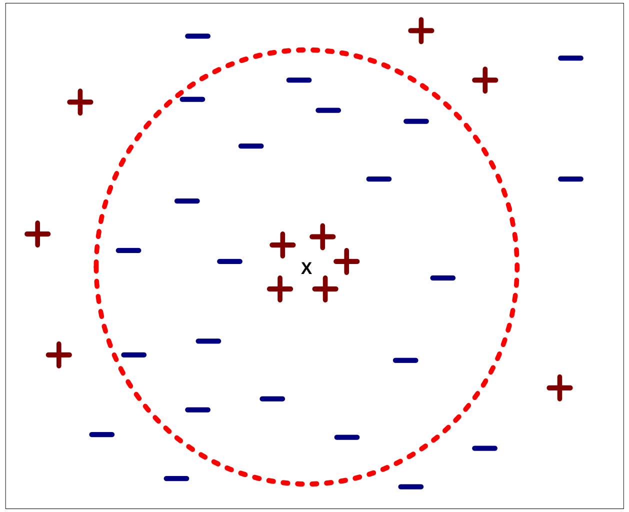
Oblique Decision Trees:

- **Improvement**: Allow tests using multiple attributes (e.g., x + y < 20) for more flexible decision boundaries.
- **Result**: Can better represent complex, non-rectilinear boundaries.

Limitations of single attribute-based decision boundaries



Nearest-Neighbor Classifiers



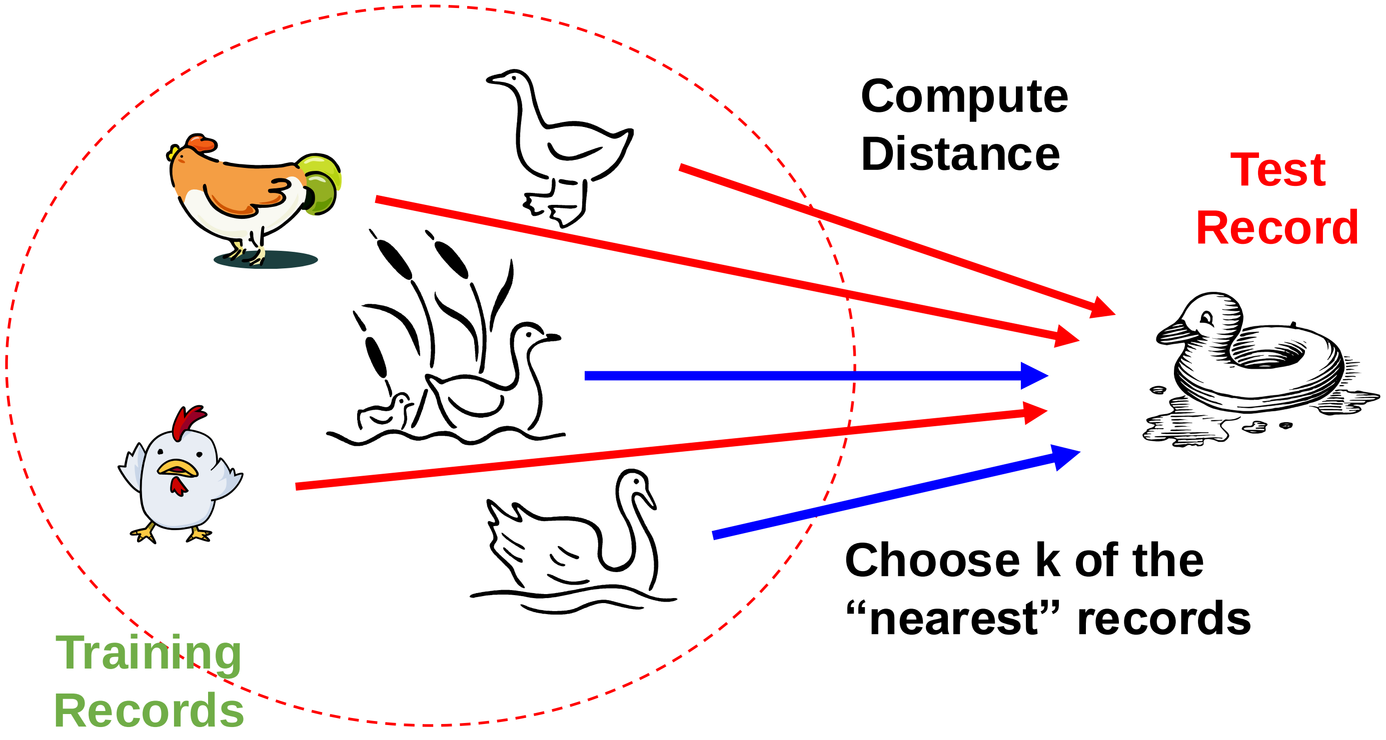
Nearest-Neighbor Classifiers

Basic idea: ***If it walks like a duck, quacks like a duck, then it's probably a duck***

A *nearest neighbor classifier* represents each example as a data point in a d-dimensional space (d the number of attributes)

Given a test instance, we compute its proximity to the training instances according to one of the proximity measures

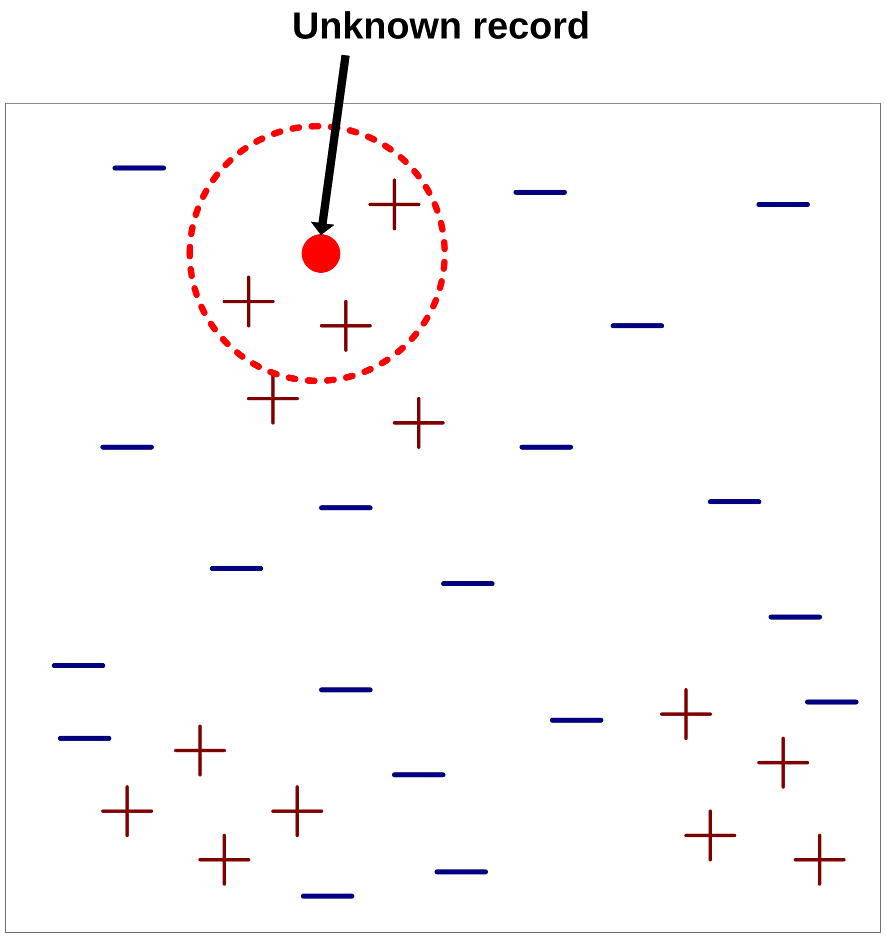
The k-nearest neighbors of a given test instance z refer to the k training examples that are closest to z.



Nearest-Neighbor Classifiers

Requires the following:

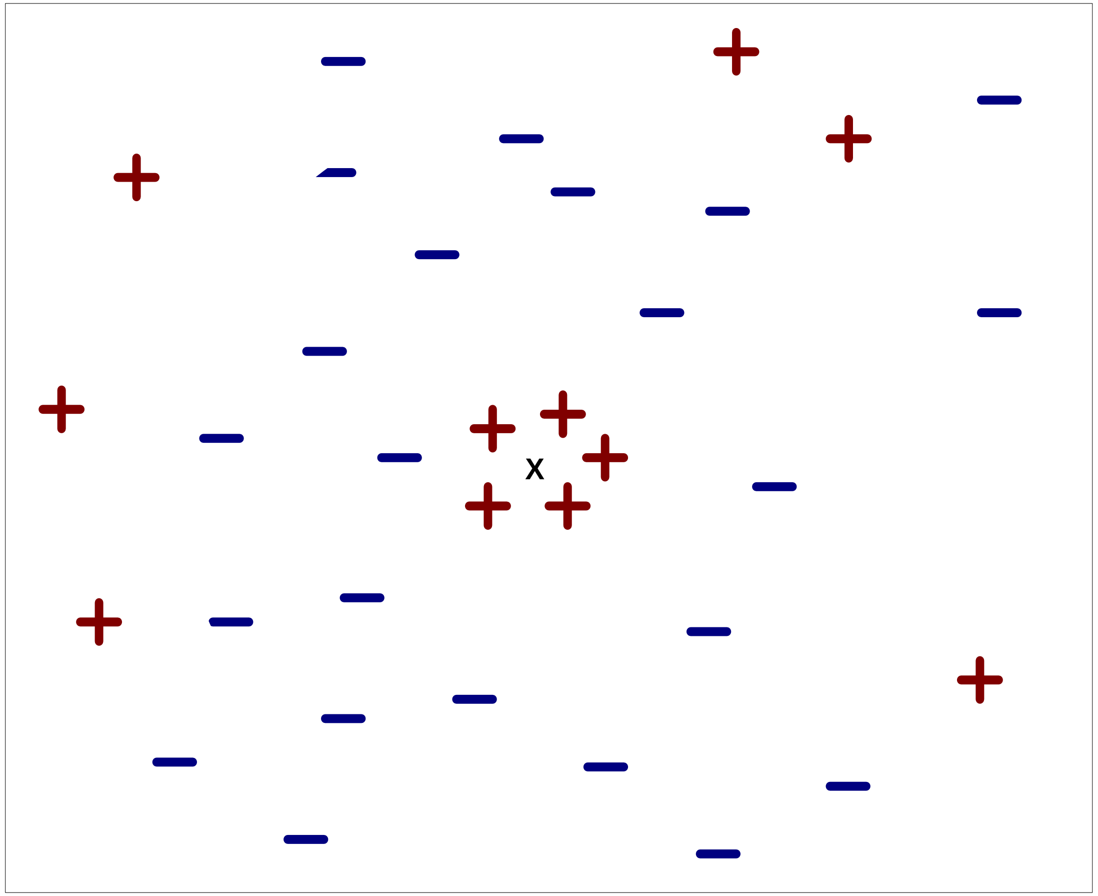
- A set of labeled records
- Proximity metric to compute distance/similarity between a pair of records e.g., Euclidean distance
- The value of *k*, the number of nearest neighbors to retrieve
- A method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)



Nearest-Neighbor Classifiers

Choosing the value of k:

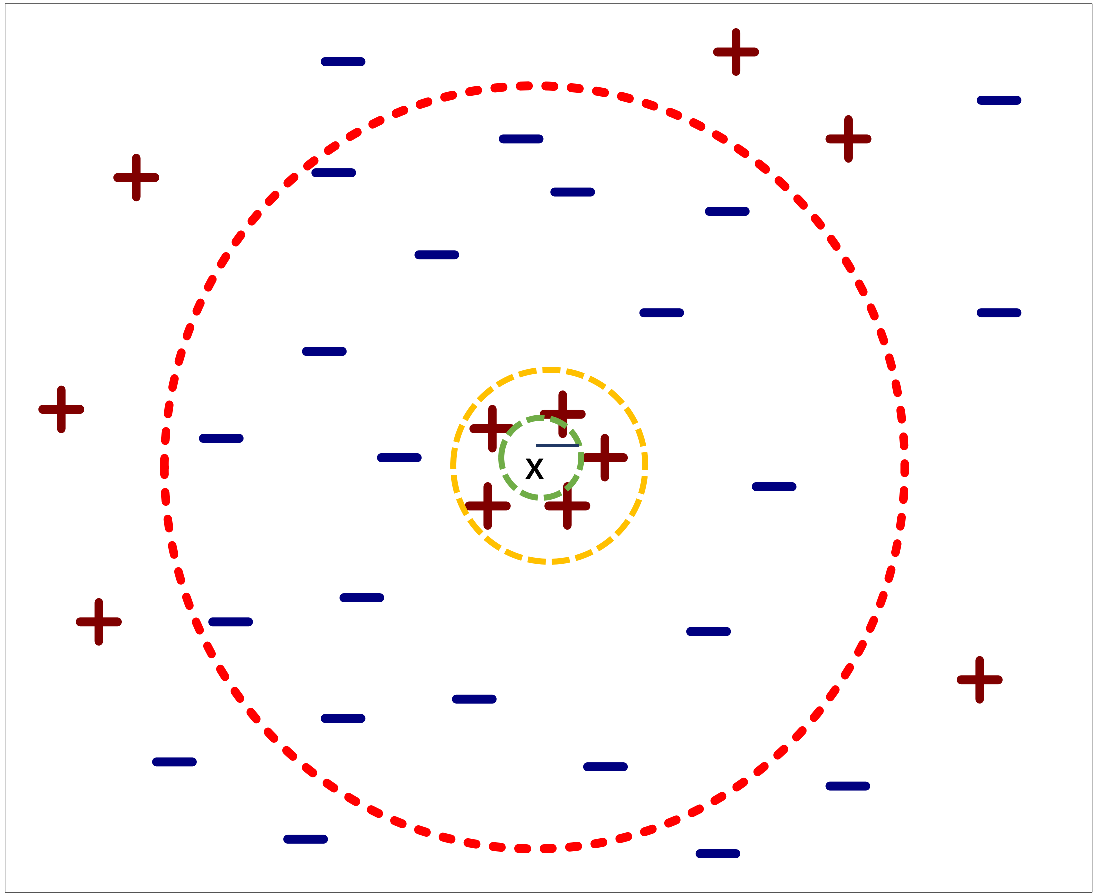
- If k is **too small**, sensitive to noise points (overfitting)
- If k is **too large**, neighborhood may include points from other classes
- An example of well balanced k



Nearest-Neighbor Classifiers

Choosing the value of k:

- If k is **too small**, sensitive to noise points (overfitting)
- If k is **too large**, neighborhood may include points from other classes
- An example of well balanced k



Nearest-Neighbor Classifiers

How to Determine the class label of a Test Sample?

Let k be the number of nearest neighbors and D be the set of training examples.
for each test instance $z = (\mathbf{x}', y')$ **do**
    Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example, $(\mathbf{x}, y) \in D$.
    Select $D_z \subseteq D$, the set of k closest training examples to z.
    $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$
end for

Take the **majority vote** of class labels among the k-nearest neighbors

Weight the vote according to distance

- weight factor, $w = 1/d^2$

Nearest-Neighbor Classifiers

- v is a **class label**
- y_i is the **class label** for **one** of the **nearest neighbors**
- $I(\cdot)$ is an **indicator function** that returns the value 1 if its argument is true and 0 otherwise

Majority Voting

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i),$$

Distance-Weighted Voting

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i).$$

weight factor, $w = 1/d^2$

Nearest-Neighbor Classifiers

Data preprocessing is often required

Attributes may have to be **scaled** to prevent *distance measures* from being *dominated* by one of the attributes

- Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

- It is often useful to **standardize** data to have 0 means a standard deviation of 1

Nearest-Neighbor Classifiers

How to handle missing values in training and test sets?

- Proximity computations normally require the presence of all attributes
- Some approaches use the subset of attributes present in two instances
    - This may not produce good results since it effectively uses different proximity measures for each pair of instances
    - Thus, proximities are not comparable

How to handle Irrelevant and Redundant Attributes

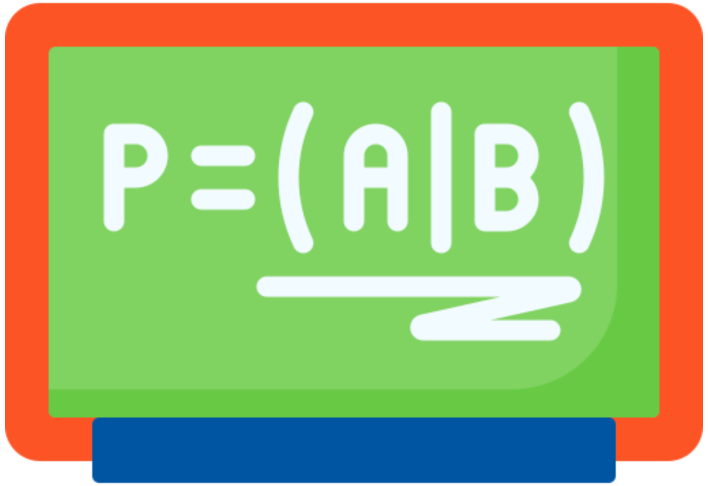
- Irrelevant attributes add noise to the proximity measure
- Redundant attributes bias the proximity measure towards certain attributes

Nearest-Neighbor Classifiers

Characteristics

- Nearest-Neighbor classification is part of a more general technique known as instance-based learning
- **Classifying** a test instance can be quite **expensive** because we need to compute the proximity values individually (lazy learners)
- Nearest neighbor classifiers can produce **decision boundaries** of **arbitrary shape**
- Nearest neighbor classifiers have **difficulty handling missing values** in both the training and test sets since proximity computations normally require the presence of all attributes
- The presence of irrelevant attributes can distort commonly used proximity measures, especially when the number of irrelevant attributes is large

Bayesian Classifiers



Bayesian Classifiers

Many classification problems involve **uncertainty**

- Observed attributes and class labels may be unreliable due to **imperfections** in the measurement process
- Set of attributes may not be fully representative of the target class, resulting in **uncertain** predictions
- Classification model learned over a finite training set may not be able to fully capture the true relationships in the overall data

In the presence of **uncertainty** we need provide *predictions* of class labels but a *measure of confidence*

Probability theory offers a systematic way for *quantifying* and *manipulating* **uncertainty** in data

Classification models that make use of *probability* are known as **probabilistic classification model**

Bayesian Classifiers

Introduction to probability

Consider a variable $\boldsymbol{X}$, which can take any discrete value from the set $\{x_1, x_2, \ldots, x_k\}$

X has the value x_i for n_i data objects

Relative frequency of event $X = x_i$ is $\frac{n_i}{N}$ where $\mathrm{N} = \sum_{i=1}^{k} n_i$

The **probability** of an event e, e.g., $\boldsymbol{P(X = x_i)}$, measures how likely it is for the event x_i to occur

A probability is always a number between 0 and 1

The sum of probability values of all possible events (outcomes of a variable X) is equal to 1

Variables that have probabilities associated with each possible outcome (values) are known as random variables

Bayesian Theorem

$P(Y|X)$ denote the **conditional probability** of observing the random variable Y whenever the random variable X takes a particular value

Bayes theorem provides a relationship between the conditional probabilities $P(Y|X)$ and $P(X|Y)$

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{P(X|Y)P(Y)}{\sum_{i=1}^{k} P(X|y_i)P(y_i)}$$

Bayesian Classifiers

Using Bayes Theorem for Classification

For the purpose of classification, we are interested in computing the probability of observing a *class label* $\boldsymbol{y}$ for a data instance given its *set of attribute* $\boldsymbol{x}$

$P(y|\boldsymbol{x})$ is the posterior probability

$$P(y|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|y)P(y)}{P(\boldsymbol{x})}$$

where

$P(x|y)$ is the class-conditional probability the likelihood of observing $\boldsymbol{x}$ from the distribution of instances belonging to y.

$P(y)$ is the prior probability (prior beliefs about the distribution of class labels) [expert knowledge or distribution]

$P(x)$ can be computed as $P(x) = \sum_{i=1}^{k} P(\boldsymbol{x}|y_i)P(y_i)$

Bayesian Classifiers

Naïve Bayes Assumption for Classification

Naïve Bayes classifier assumes that the class-conditional probability $P(\boldsymbol{x}|y)$ of all attributes x can be factored as a product of class-conditional probabilities $P(x|y)$

$$P(\boldsymbol{x}|y) = \prod_{i=1}^{d} P(x_i|y)$$

Where data instance $\boldsymbol{x}$ the set $\{x_1, x_2, \dots, x_d\}$

The basic assumption behind the previous equation is that the attribute values

x_i are conditionally independent

Naïve Bayes Assumption for Classification

The Naïve Bayes classifier computes the posterior probability for a test instance $\boldsymbol{x}$ by using the following equation

$$P(y|\boldsymbol{x}) = P(y)\prod_{i=1}^{d} P(x_i|y)$$

Using Bayes Theorem for Classification

- Consider each attribute and class label as *random variables*
- Given a record with attributes $\{x_1, x_2, \ldots, x_d\}$ the goal is to predict class y
- Specifically, we want to find the value of Y that maximizes $P(y|x_1, x_2, \ldots, x_d)$

Can we estimate $P(y|x_1, x_2, \ldots, x_d)$

directly from data?

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Evade**
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**

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$P(y|\boldsymbol{x}) = P(y)\prod_{i=1}^{d} P(x_i|y)$$

- We need to estimate

  $P(y = yes|\boldsymbol{x})$ and $P(y = no|\boldsymbol{x})$

In the following we will replace
y = Yes by Yes, and
y = No by No

Tid	Refund	Marital Status	Taxable Income	Evade
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

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$P(yes|\boldsymbol{x}) = P(yes)\prod_{i=1}^{d} P(x_i|yes)$$

$$P(no|\boldsymbol{x}) = P(no)\prod_{i=1}^{d} P(x_i|no)$$

Tid	Refund	Marital Status	Taxable Income	Evade
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

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$P(yes) = 3/10$$

$$P(no) = 7/10$$

Tid	Refund	Marital Status	Taxable Income	Evade
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

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$\prod_{i=1}^{d} P(x_i|yes)$$

P(X | Yes) =

P(Refund = No | Yes) x

P(Divorced | Yes) x

P(Income = 120K | Yes)

$$\prod_{i=1}^{d} P(x_i|no)$$

P(X | No) =

P(Refund = No | No) x

P(Divorced | No) x

P(Income = 120K | No)

Tid	Refund	Marital Status	Taxable Income	Evade
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

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$\prod_{i=1}^{d} P(x_i|yes)$$

P(X | Yes) =

P(Refund = No | Yes) x

P(Divorced | Yes) x

P(Income = 120K | Yes)

$$\prod_{i=1}^{d} P(x_i|no)$$

P(X | No) =

P(Refund = No | No) x

P(Divorced | No) x

P(Income = 120K | No)

Tid	Refund	Marital Status	Taxable Income	Evade
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

Categorical

Continuous

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

Tid	Refund	Marital Status	Taxable Income	Evade
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

For categorical attributes:

$P(x_i = c|yes) = n_c/n$ = n_c/ n

where n_c is number of instances having attribute value $x_i = c$ and belonging to class y

–Examples:

P(Status=Married|No) = 4/7
P(Refund=Yes|Yes) = 0

Bayesian Classifiers: Example

For continuous attributes:

- Discretization: Partition the range into bins:
  - Replace continuous value with bin value
    - Attribute changed from continuous to ordinal
- Probability density estimation:
  - Assume attribute follows a normal distribution
  - Use data to estimate parameters of distribution (e.g., mean and standard deviation)
  - Once probability distribution is known, use it to estimate the conditional probability $P(X_i|Y)$

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

Normal distribution:

$$P(X_i \mid Y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$

– One for each (X_i, Y_i) pair

For (Income, Class=No):

– If Class=No

sample mean = 110

sample variance = 2975

Tid	Refund	Marital Status	Taxable Income	Evade
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

$$P(Income = 120 \mid No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

Tid	Refund	Marital Status	Taxable Income	Evade
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

Naïve Bayes Classifier:

P(Refund = Yes | No) = 3/7
P(Refund = No | No) = 4/7
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/7
P(Marital Status = Divorced | No) = 1/7
P(Marital Status = Married | No) = 4/7
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0

For Taxable Income:
If class = No: sample mean = 110
sample variance = 2975
If class = Yes: sample mean = 90
sample variance = 25

- P(X | No) = P(Refund=No | No) $\times$ P(Divorced | No) $\times$ P(Income=120K | No) = 4/7 $\times$ 1/7 $\times$ 0.0072 = 0.0006

- P(X | Yes) = P(Refund=No | Yes) $\times$ P(Divorced | Yes) $\times$ P(Income=120K | Yes) = 1 $\times$ 1/3 $\times$ 1.2 $\times$ 10^{-9} = 4 $\times$ 10^{-10}

Bayesian Classifiers: Example

Given a *Test Record*:

$$X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$$

$$P(y|\boldsymbol{x}) = P(y)\prod_{i=1}^{d} P(x_i|y)$$

$$P(no|x) = P(no)\ P(\boldsymbol{x}|no) \mathbf{= 0.0004}$$

$$P(yes|x) = P(yes)\ P(\boldsymbol{x}|yes) \mathbf{= 1.2\ 10^{-10}}$$

P(X | No) = P(Refund=No | No)
× P(Divorced | No)
× P(Income=120K | No)
= 4/7 × 1/7 × 0.0072 = 0.0006

$$P(no) = 7/10$$

P(X | Yes) = P(Refund=No | Yes)
× P(Divorced | Yes)
× P(Income=120K | Yes)
= 1 × 1/3 × 1.2 × 10^{-9} = 4 × 10^{-10}

$$P(yes) = 3/10$$

Bayesian Classifiers: Issue

Naïve Bayes Classifier can make decisions with partial information about attributes in the test record

Naïve Bayes Classifier:

P(Refund = Yes | No) = 3/7
P(Refund = No | No) = 4/7
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/7
P(Marital Status = Divorced | No) = 1/7
P(Marital Status = Married | No) = 4/7
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0

For Taxable Income:
If class = No: sample mean = 110
sample variance = 2975
If class = Yes: sample mean = 90
sample variance = 25

P(Yes) = 3/10   P(No) = 7/10

If we only know that marital status is Divorced, then:

P(Yes | Divorced) = 1/3 x 3/10 / P(Divorced)

P(No | Divorced) = 1/7 x 7/10 / P(Divorced)

If we also know that Refund = No, then

P(Yes | Refund = No, Divorced) = 1 x 1/3 x 3/10   P(Divorced, Refund = No)

P(No | Refund = No, Divorced) = 4/7 x 1/7 x 7/10 P(Divorced, Refund = No)

If we also know that Taxable Income = 120, then

P(Yes | Refund = No, Divorced, Income = 120) =
1.2 x10^{-9} x 1 x 1/3 x 3/10 P(Divorced, Refund = No, Income = 120)

P(No | Refund = No, Divorced Income = 120) =
0.0072 x 4/7 x 1/7 x 7/10 P(Divorced, Refund = No, Income = 120)

Bayesian Classifiers: Issue

Given a Test Record: **X = (Married)**

Naïve Bayes Classifier:

P(Refund = Yes | No) = 3/7
P(Refund = No | No) = 4/7
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/7
P(Marital Status = Divorced | No) = 1/7
P(Marital Status = Married | No) = 4/7
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0

For Taxable Income:
If class = No: sample mean = 110
sample variance = 2975
If class = Yes: sample mean = 90
sample variance = 25

P(Yes) = 3/10

P(No) = 7/10

P(Yes | Married) = 0 x 3/10 / P(Married)

P(No | Married) = 4/7 x 7/10 / P(Married)

Bayesian Classifiers: Issue

Consider the table with Tid = 7 deleted

Tid	Refund	Marital Status	Taxable Income	Evade
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
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Naïve Bayes Classifier:

P(Refund = Yes | No) = 2/6
P(Refund = No | No) = 4/6
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/6
P(Marital Status = Divorced | No) = 0
P(Marital Status = Married | No) = 4/6
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0/3
For Taxable Income:
If class = No: sample mean = 91
sample variance = 685
If class = No: sample mean = 90
sample variance = 25

Given X = (Refund = Yes, Divorced, 120K)

P(X | No) = 2/6 X 0 X 0.0083 = 0

P(X | Yes) = 0 X 1/3 X 1.2 X 10^{-9} = 0

Naïve Bayes will not be able to classify X as Yes or No!

Bayesian Classifiers: Issue

If one of the conditional probabilities is zero, then the entire expression becomes zero

Need to use other estimates of conditional probabilities than simple fractions

Probability estimation:

$$\text{original: } P(X_i = c|y) = \frac{n_c}{n}$$

$$\text{Laplace Estimate: } P(X_i = c|y) = \frac{n_c + 1}{n + v}$$

$$\text{m} - \text{estimate: } P(X_i = c|y) = \frac{n_c + mp}{n + m}$$

n: number of training instances belonging to class y

n_c: number of instances with $X_i = c$ and $Y = y$

v: total number of attribute values that X_i can take

p: initial estimate of $(P(X_i = c|y))$ known apriori

m: hyper-parameter for our confidence in p

Bayesian Classifiers: Issue

Robust to isolated noise points

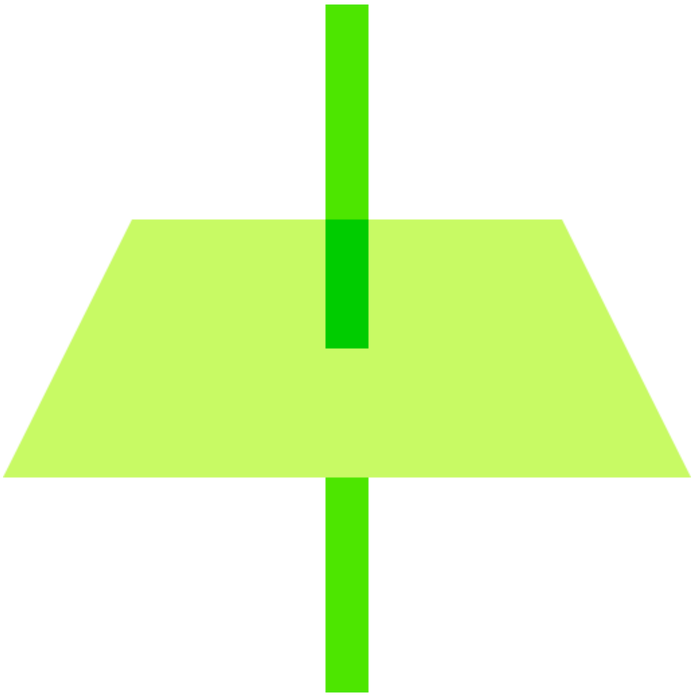
Handle missing values by ignoring the instance during probability estimate calculations

Robust to irrelevant attributes

Redundant and correlated attributes will violate class conditional assumption

- Use other techniques such as Bayesian Belief Networks (BBN)

Support Vector Machines



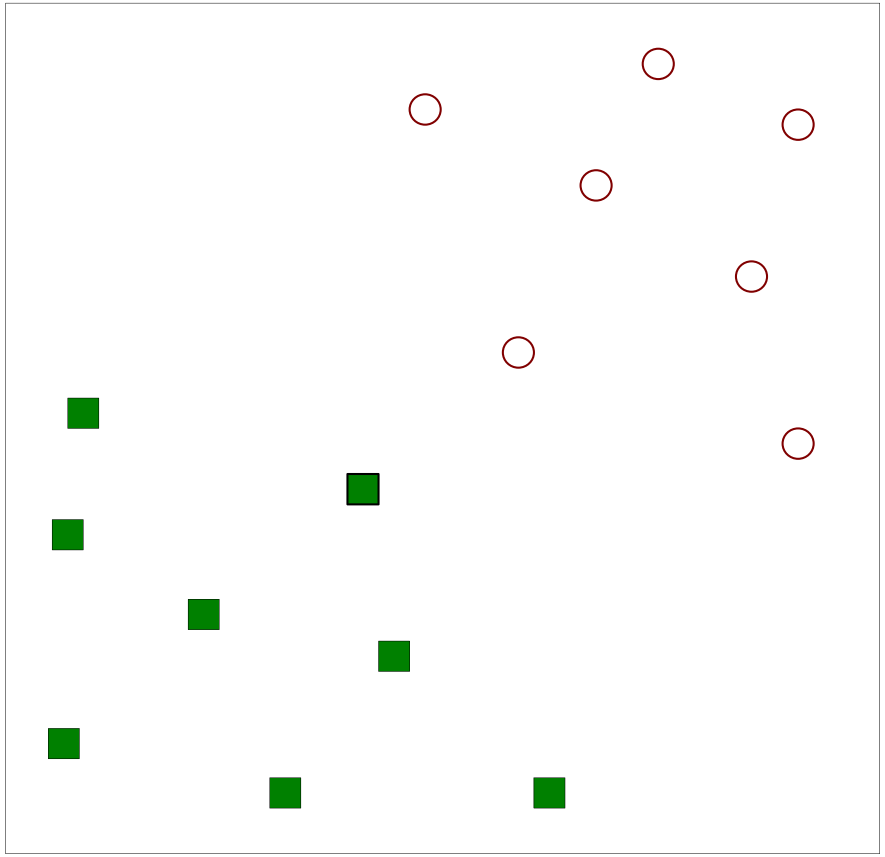
Support Vector Machines

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM

- strong regularization capabilities.
- good generalization performance.
- learn highly expressive models without suffering from overfitting.

Find a linear hyperplane (decision boundary) that will separate the data



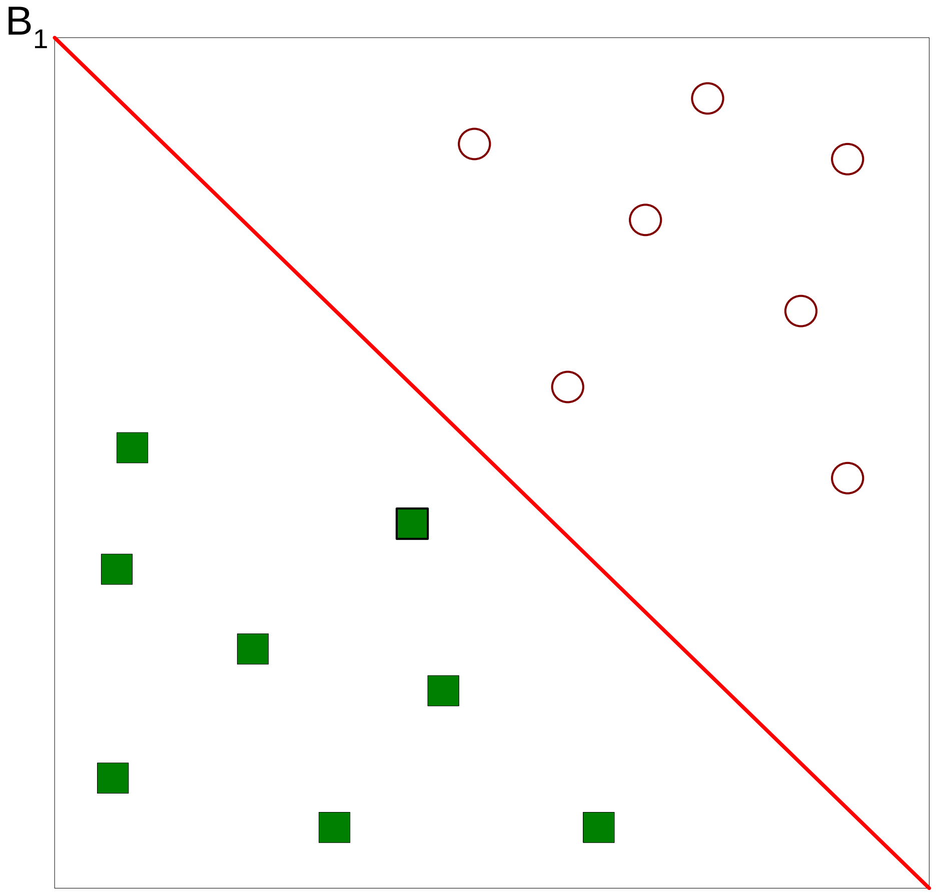
Support Vector Machines

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM

- **strong regularization capabilities.**
- **good generalization performance.**
- **learn highly expressive models without suffering from overfitting.**

One Possible Solution



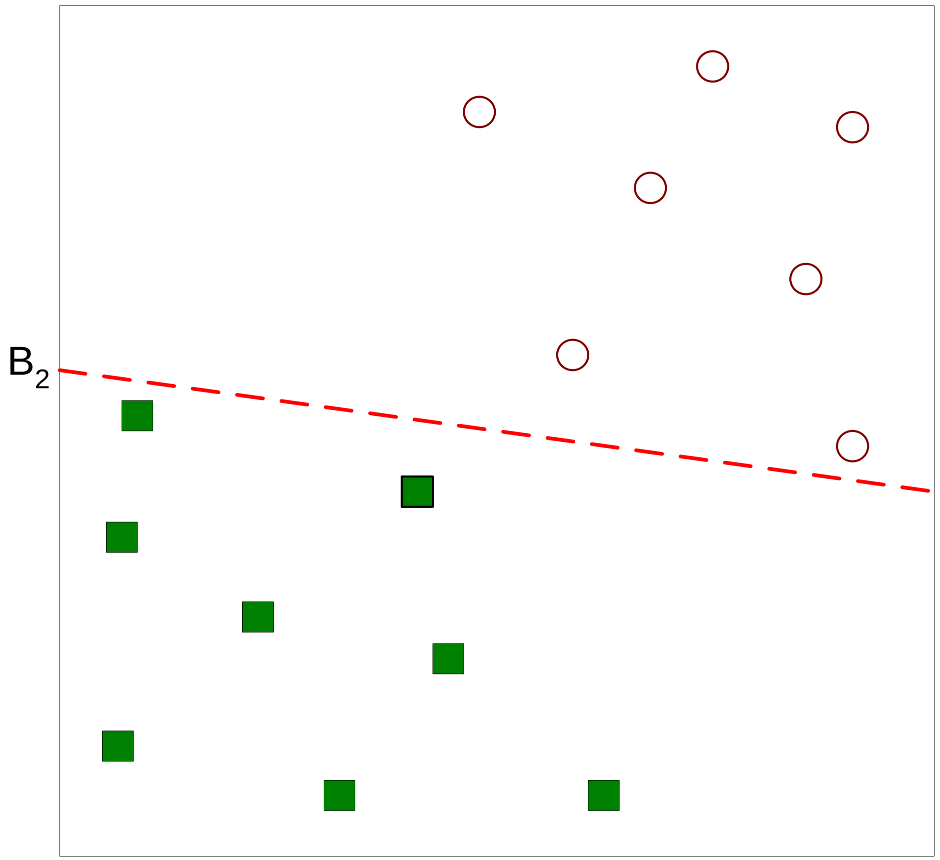
Support Vector Machines

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM

- **strong regularization capabilities.**
- **good generalization performance.**
- **learn highly expressive models without suffering from overfitting.**

Another Possible Solution



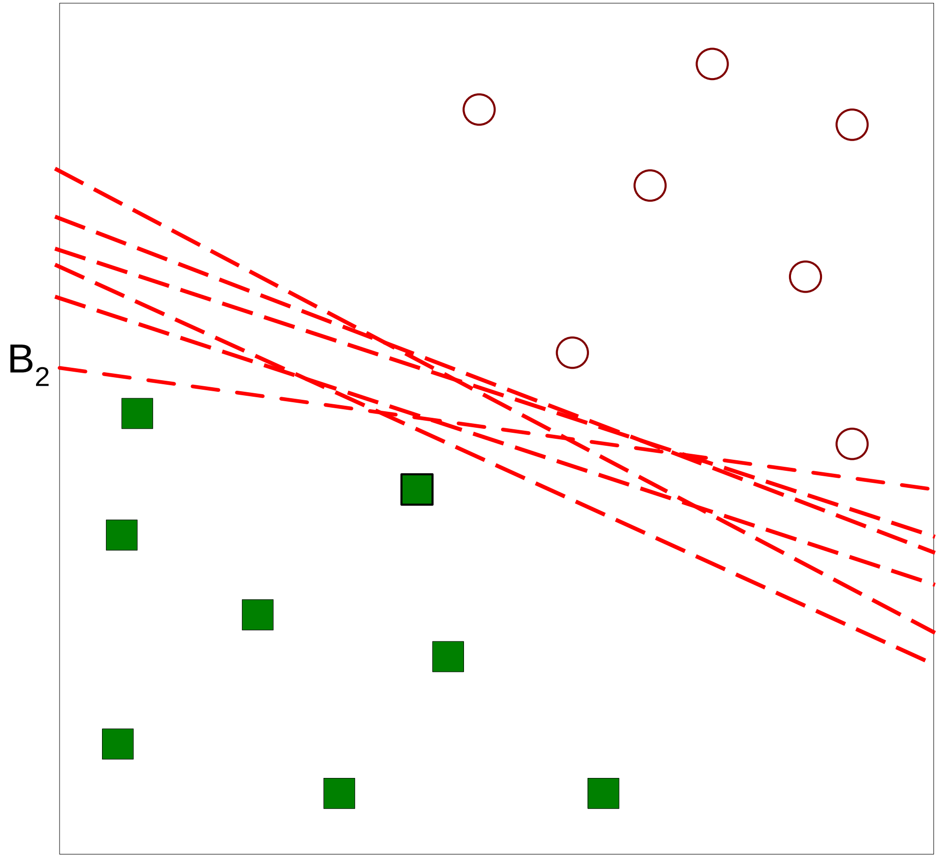
Support Vector Machines

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM

- **strong regularization capabilities.**
- **good generalization performance.**
- **learn highly expressive models without suffering from overfitting.**

Other Possible Solutions

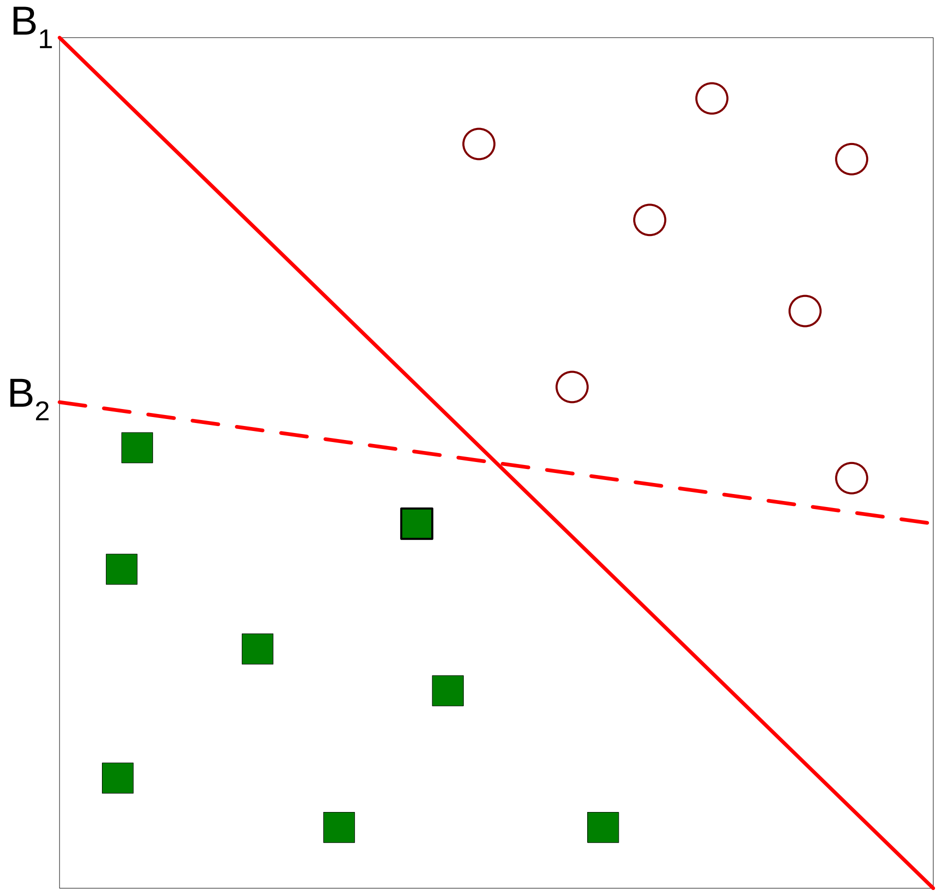


Support Vector Machines

A support vector machine (SVM) is a discriminative classification model that learns linear or nonlinear decision boundaries in the attribute space to separate the classes.

SVM

- **strong regularization capabilities.**
- **good generalization performance.**
- **learn highly expressive models without suffering from overfitting.**

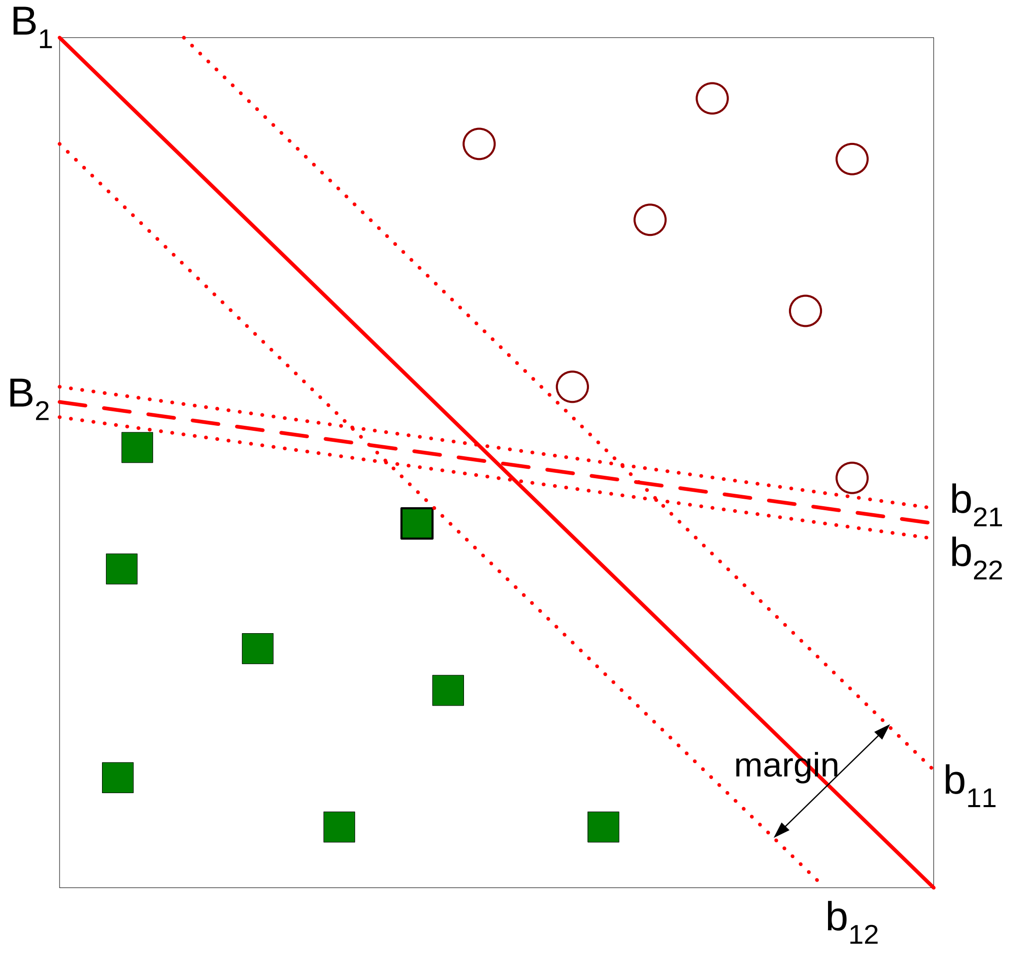


Which one is better? B1 or B2?

How do you define better?

Support Vector Machines

The point is now about the margin of a separating hyperplane and the rationale for choosing such a hyperplane with maximum margin.



Find hyperplane maximizes the margin

B1 is better than B2

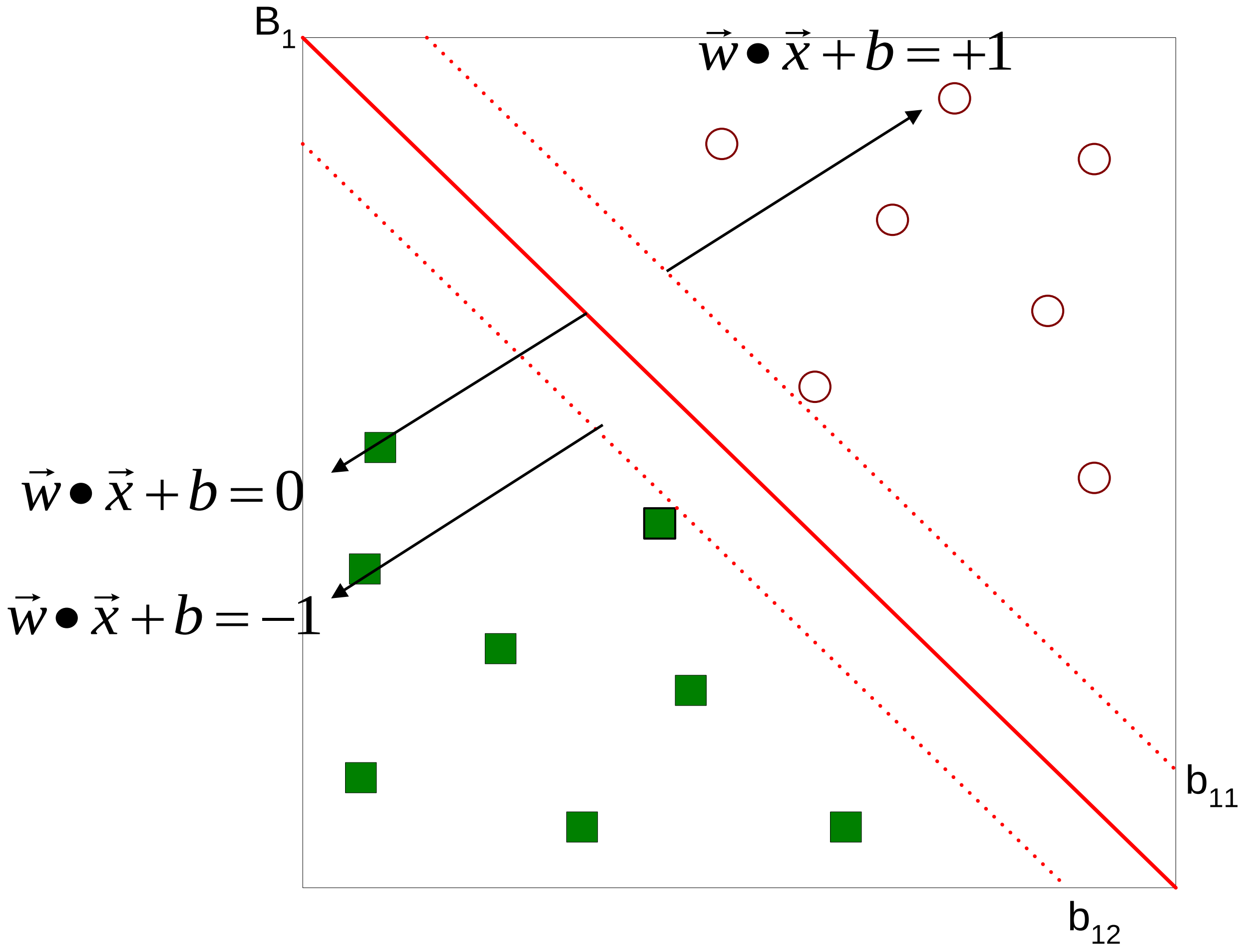
Support Vector Machines

The generic equation of a separating hyperplane can be written as

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0$$

where x represent the attributes and (w, b) represent the hyperplane.

A data instance x_i can belong to either side of the hyperplane depending on the sign of $w^T x + b$



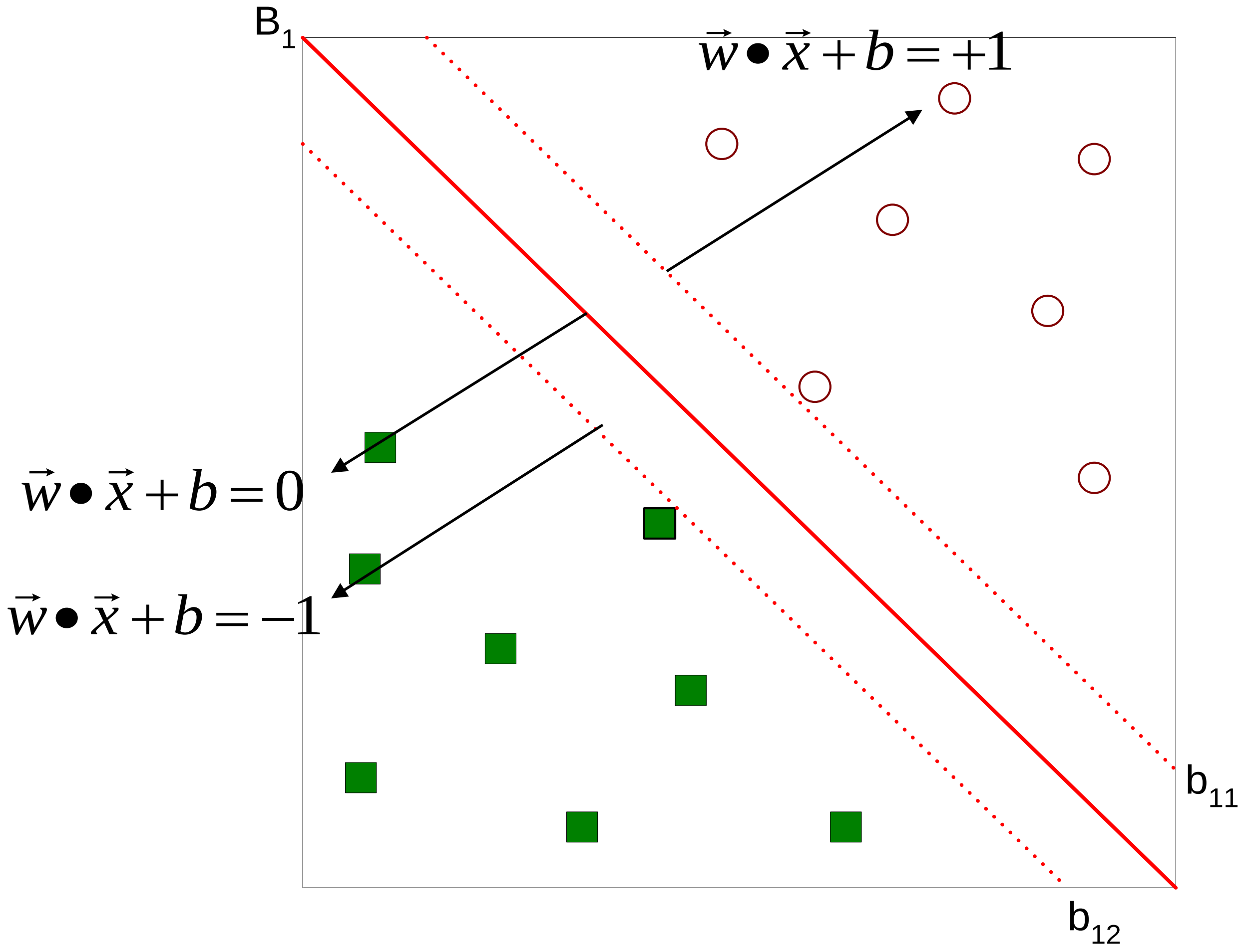
Support Vector Machines

The hyperplane parameters (w, b) **working on this condition:**

$$\boldsymbol{w}^T\boldsymbol{x_i} + b > 0 \quad if\ y_i = 1$$
$$\boldsymbol{w}^T\boldsymbol{x_i} + b < 0 \quad if\ y_i = -1$$

That led to:

$$\min_{w,b} \frac{\|\boldsymbol{w}\|^2}{2}$$
$$subject\ to \quad y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) > 1$$



Support Vector Machines

SVM to learn linear hyperplanes even in situations where the classes are not linearly separable.

To do this, SVM must consider the trade-off between the width of the margin and the number of training errors committed by the linear hyperplane.

The hyperplane parameters (w, b) working on this condition:

$$\boldsymbol{w}^T \boldsymbol{x_i} + b > 1 - \xi_i$$

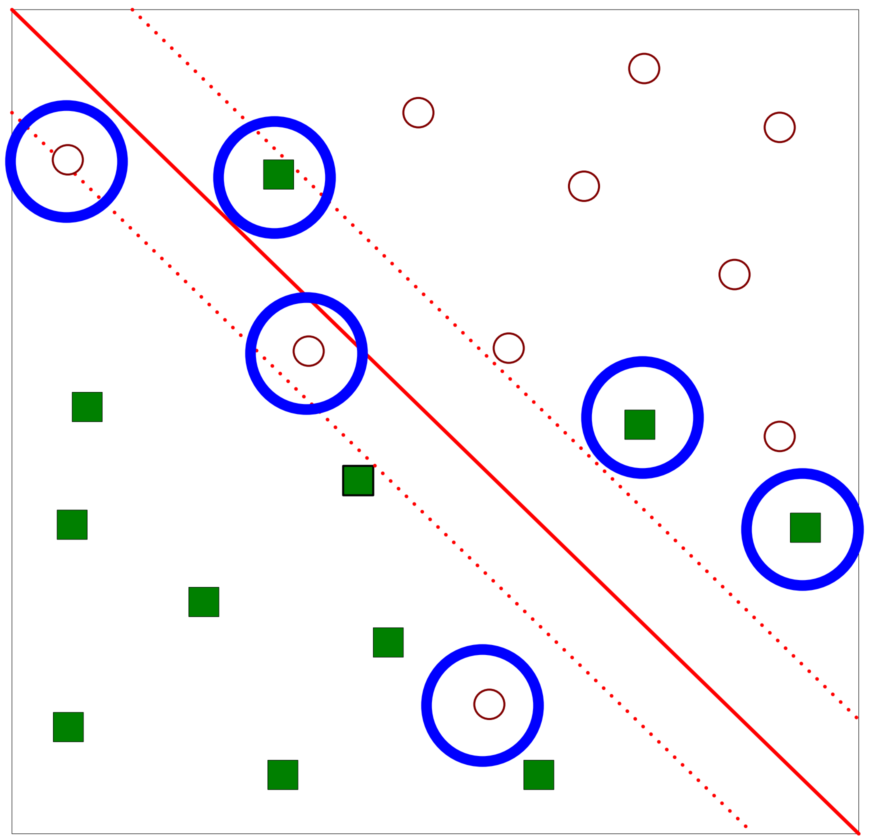
That led to:

$$\min_{w,b,\xi_i} \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=1}^{n} \xi_i$$

$$subject\ to \quad \begin{array}{c} y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) > 1 - \xi_i \\ \xi_i > 0 \end{array}$$

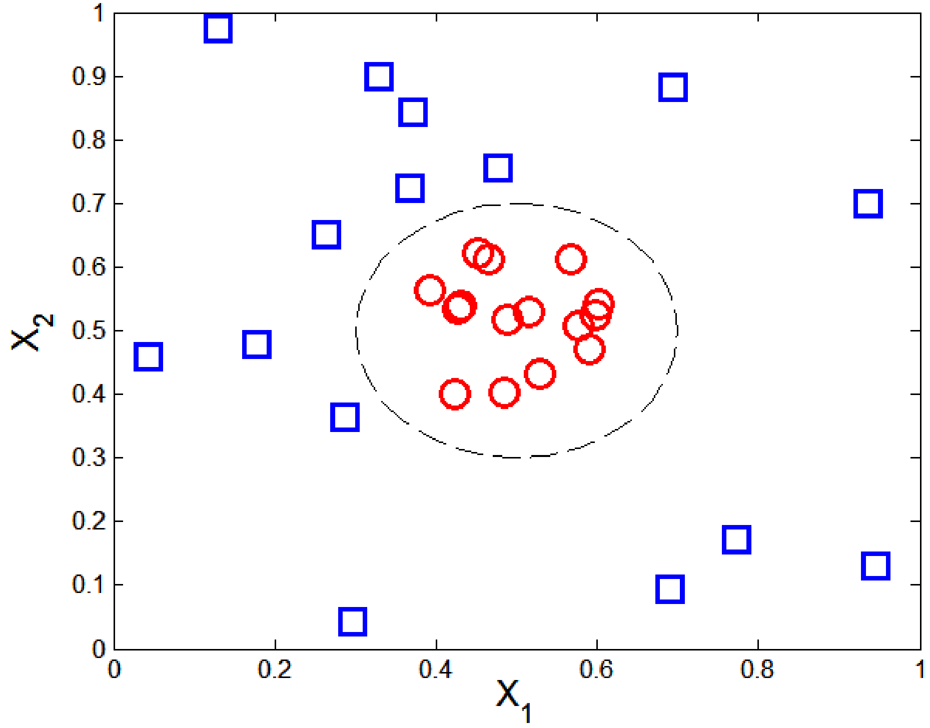
C is a hyper-parameter that makes a trade-off between *maximizing* the *margin* and *minimizing* the *training error*.

A large value of C pays more emphasis on minimizing the training error than maximizing the margin



Nonlinear Support Vector Machines

$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$



What if decision boundary is not linear?

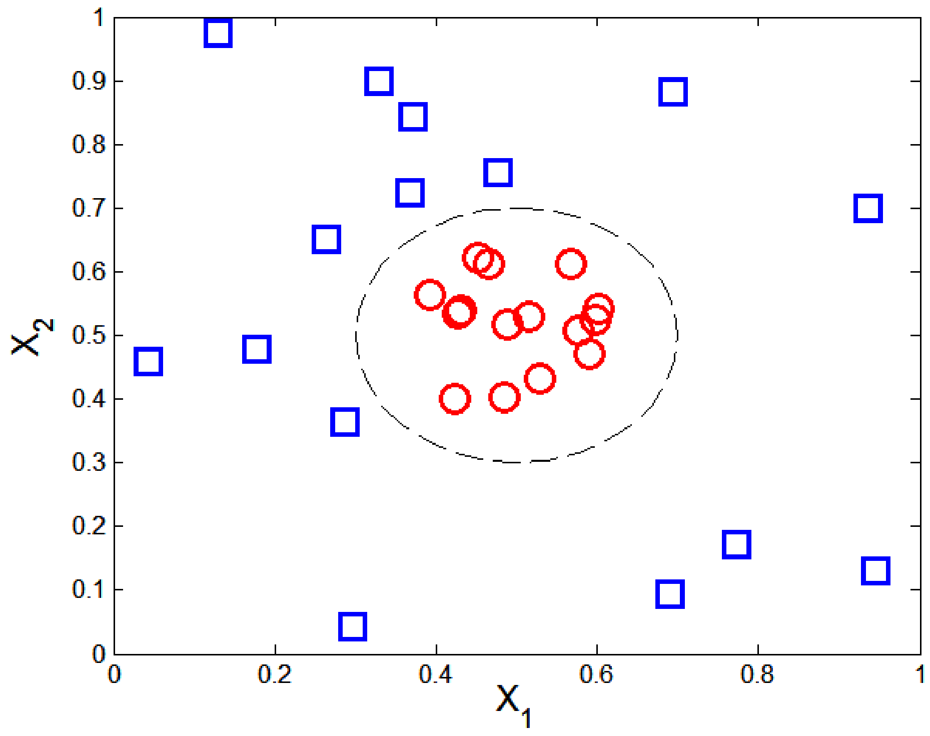
A nonlinear transformation $\boldsymbol{\varphi}$ is needed to map the data from its original attribute space into a new **linear separable** space

Transform the data from its original attribute space in $\boldsymbol{x}$ into a new space $\boldsymbol{\varphi}(\boldsymbol{x})$ so that a linear hyperplane

Learned hyperplane can then be projected back to the original attribute space, resulting in a nonlinear decision boundary.

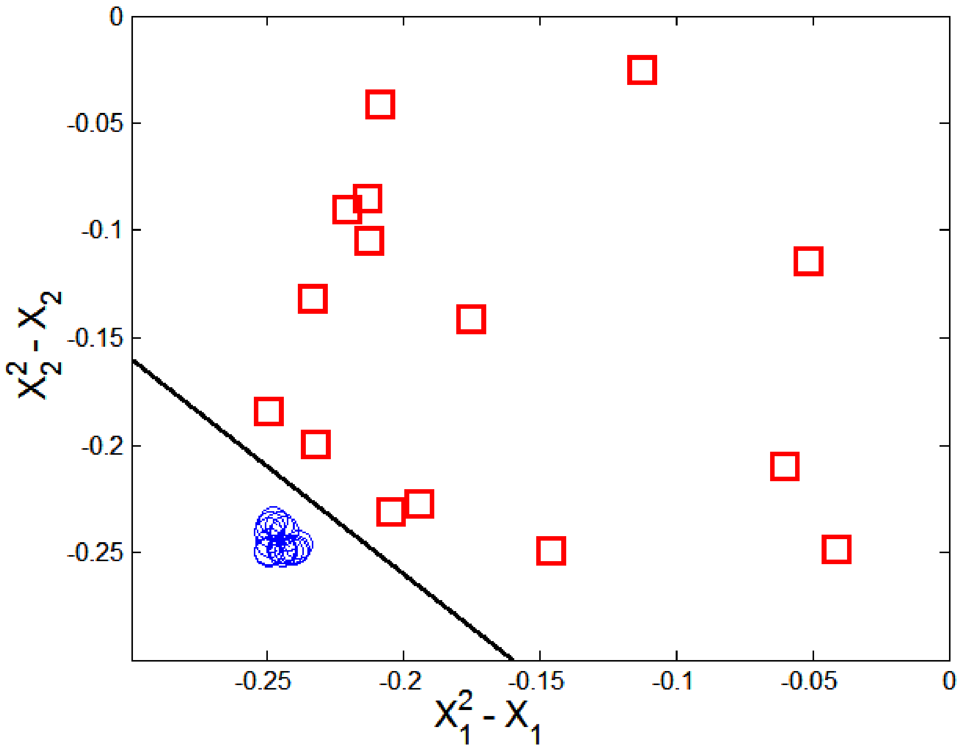
Nonlinear Support Vector Machines

$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$



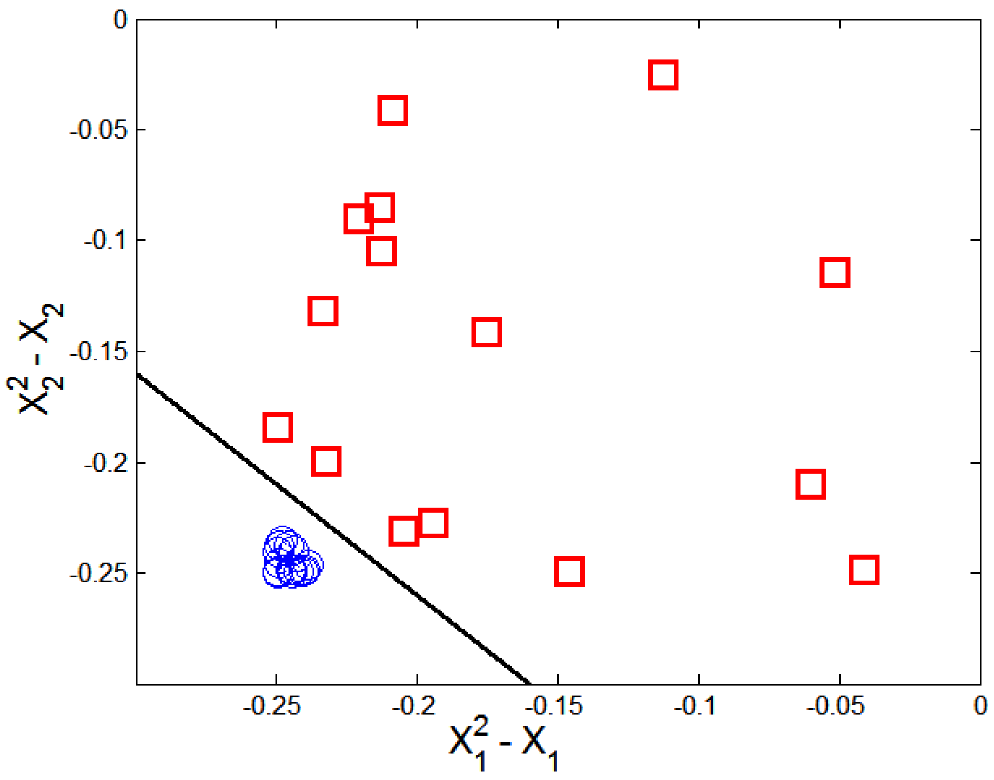
The transformation required is:

$$\varphi: (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$



Nonlinear Support Vector Machines

$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$



Learning not linear model

The new decision boundary is:

$$\boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x_i}) + b = 0$$

that results in the new learning

$$\min_{w,b,\xi_i} \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=1}^{n} \xi_i$$

$$\textit{subject to} \quad y_i(\boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x_i}) + b) > 1 - \xi_i$$
$$\xi_i > 0$$

We need only inner products of $\varphi(x)$

- What type of mapping function φ should be used?
- How to do the computation in high dimensional space?

Learning Nonlinear SVM

The **kernel trick** is a method for computing this similarity as a *function of the original attribute*

$$\mathrm{K}(\mathrm{u}, \mathrm{v}) = \langle \boldsymbol{\varphi}(\boldsymbol{u}), \boldsymbol{\varphi}(\boldsymbol{v}) \rangle = f(u, v)$$

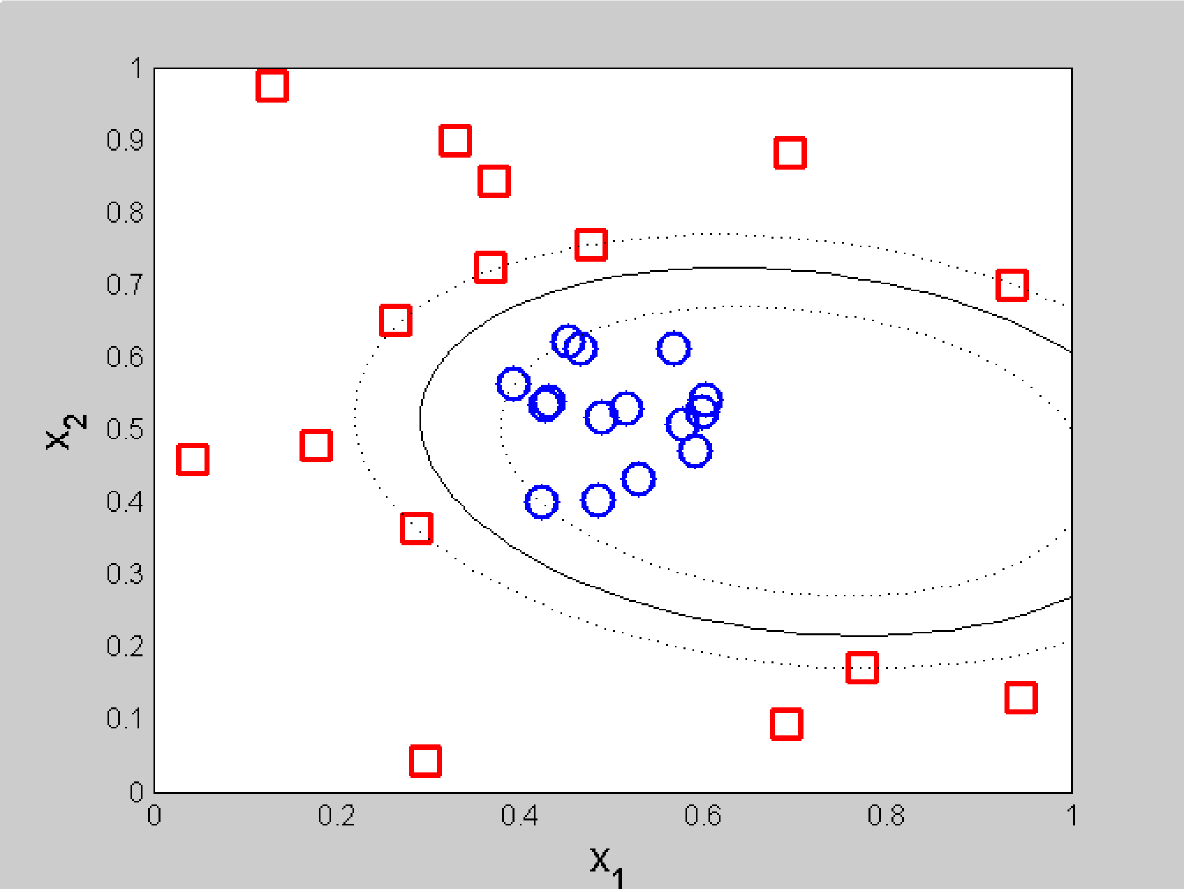
where f() is a function that follows certain conditions as stated by the Mercer's Theorem

$$\begin{array}{rl} \text{Polynomial kernel} & K(\mathbf{u}, \mathbf{v}) = (\mathbf{u}^T\mathbf{v} + 1)^p \\ \text{Radial Basis Function kernel} & K(\mathbf{u}, \mathbf{v}) = e^{-\|\mathbf{u}-\mathbf{v}\|^2/(2\sigma^2)} \\ \text{Sigmoid kernel} & K(\mathbf{u}, \mathbf{v}) = \tanh(k\mathbf{u}^T\mathbf{v} - \delta) \end{array}$$

By using a kernel function, we can directly work with inner products in the transformed space without dealing with the exact forms of the nonlinear transformation function

High-dimensional transformations, performing calculations only in the original attribute space

Example of Nonlinear SVM



SVM with polynomial degree 2 kernel

Learning Nonlinear SVM

Advantages of using kernel:

- Don't have to know the mapping function $\boldsymbol{\varphi}$
- Computing dot product $\langle \boldsymbol{\varphi}(\boldsymbol{u}), \boldsymbol{\varphi}(\boldsymbol{v}) \rangle$ in the original space avoids curse of dimensionality

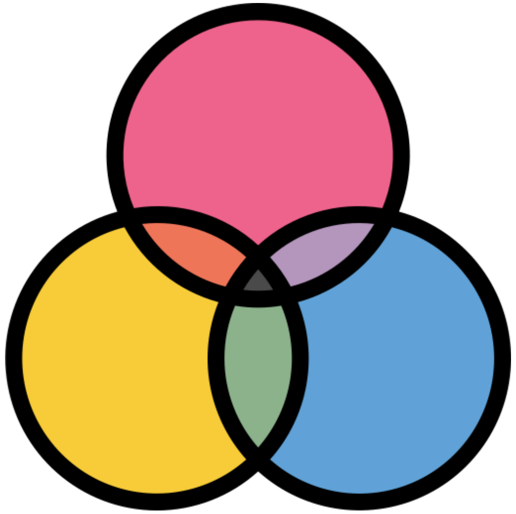
Not all functions can be kernels

- Must make sure there is a corresponding Φ in some high-dimensional space
- Mercer's theorem (see textbook)

Characteristics of SVM

- Robust to noise
- Overfitting is handled by maximizing the margin of the decision boundary,
- SVM can handle irrelevant and redundant attributes better than many other techniques
- The user needs to provide the type of kernel function and cost function
- Difficult to handle missing values

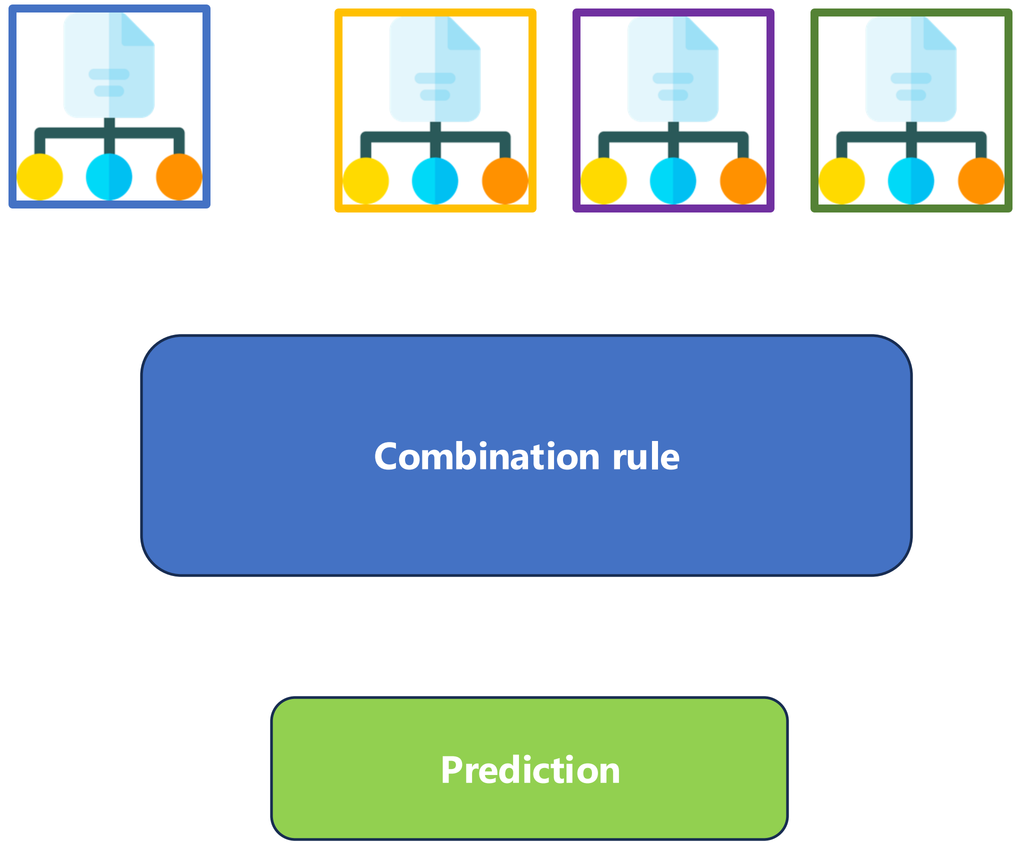
Ensemble Techniques



Ensemble Methods

Construct a **set** of base classifiers learned from the training data

Predict class label of test records by **combining the predictions** made by multiple classifiers (e.g., by taking majority vote)

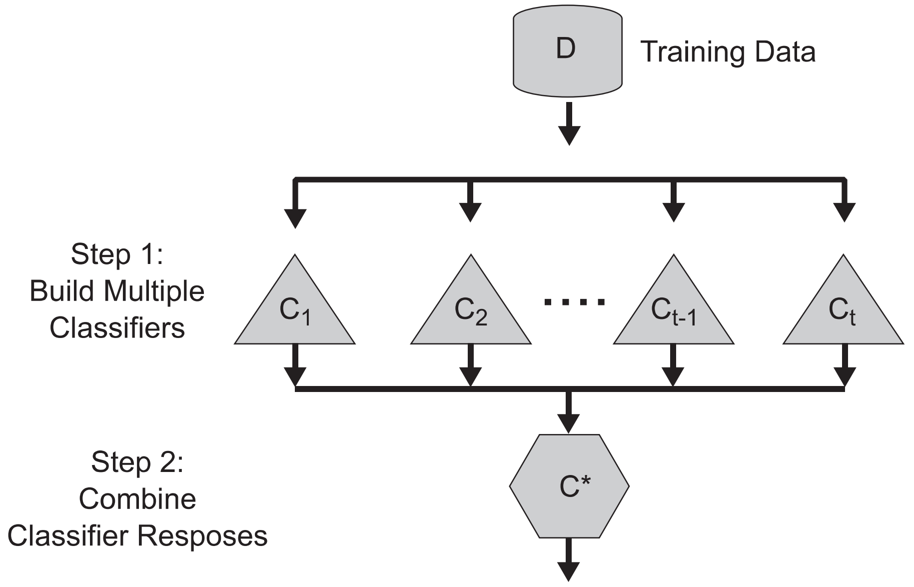


General Approach of Ensemble Learning

The ensemble of classifiers can be constructed in many ways:

- By manipulating the training set
- By manipulating the input features
- By manipulating the class labels
- By manipulating the learning algorithm

The first three approaches are generic methods that are applicable to any classifier, whereas the fourth approach depends on the type of classifier used.



Using majority vote or weighted majority vote
(weighted according to their accuracy or relevance)

$$C\ (x) = f(C_1(x), C_2(x), \ldots, C_k(x))$$

Example: Why Do Ensemble Methods Work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, ϵ = 0.35
  - Majority vote of classifiers used for classification
  - If all classifiers are identical:
    - Error rate of ensemble = ϵ (0.35)
  - If all classifiers are independent (errors are uncorrelated):
    - Error rate of ensemble = probability of having more than half of base classifiers being wrong

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

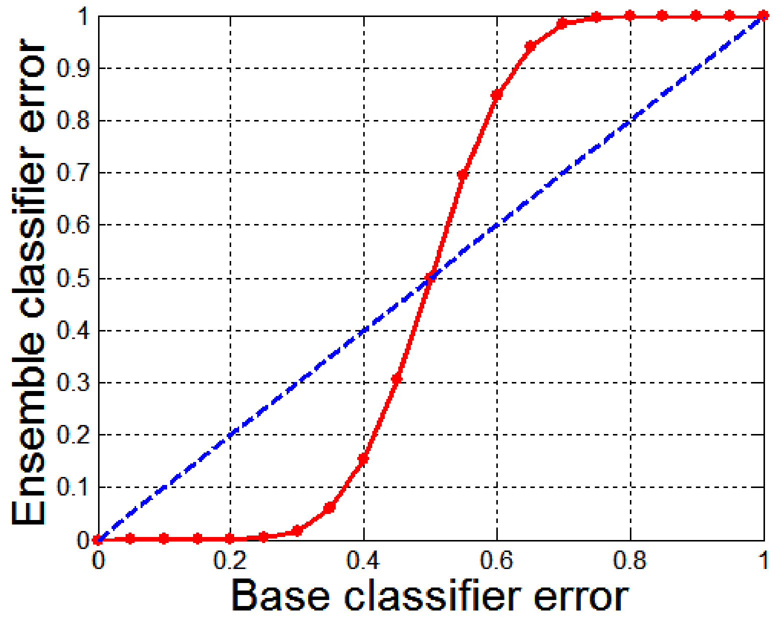
Necessary Conditions for Ensemble Methods

Ensemble Methods work better than a single base classifier if:

1. All base classifiers are independent of each other

2. All base classifiers perform better than random guessing (error rate < 0.5 for binary classification)

Classification error for an ensemble of 25 base classifiers, assuming their errors are uncorrelated.



Rationale for Ensemble Learning

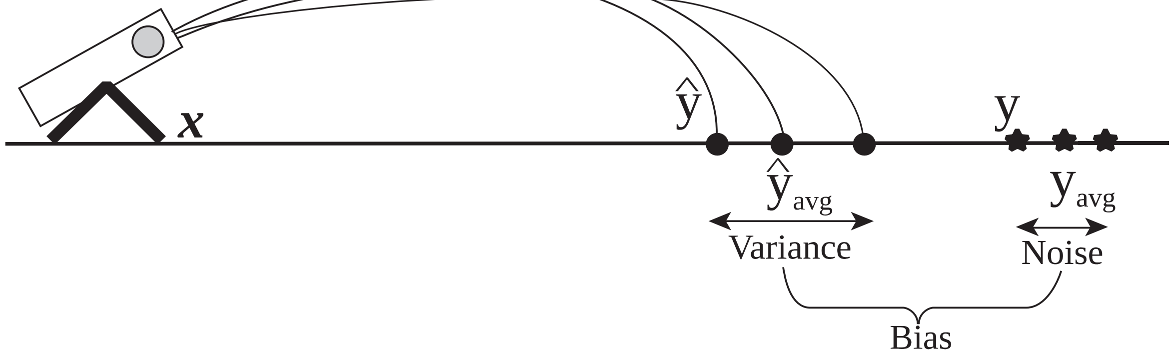
Ensemble Methods work best with **unstable base classifiers**

- Classifiers that are sensitive to minor perturbations in training set, due to *high model complexity*
- Examples: Unpruned decision trees, ANNs, …

Bias-Variance Decomposition

Bias-variance decomposition is a formal method for analyzing the generalization error of a predictive model

- Noise term is intrinsic to the target class
- Bias and **Variance** terms depend on the choice of the classification model
  - **Bias** of a model represents how close the average prediction of the model is to the average target
  - **Variance** of a model captures the stability of its predictions

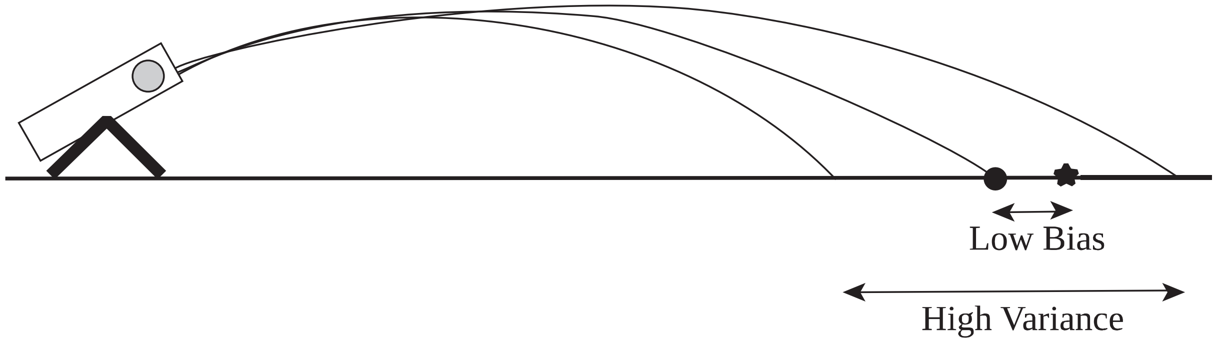


For classification, the generalization error of model m can be given by:

$$gen.error(m) = c_1 + bias(m) + c_2 \times variance(m)$$

Bias-Variance Trade-off and Overfitting

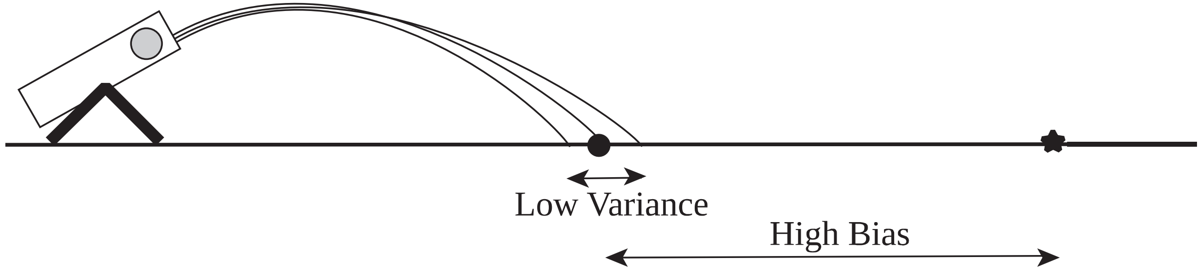
Ensemble methods try to reduce the variance of complex models (with low bias) by ***aggregating*** responses of multiple base classifiers (even help in reducing the bias)



Overfitting

Low bias but high variance, it can become susceptible to overfitting,

Bias-Variance trade-off can be used to explain why ensemble learning improves the generalization performance of unstable classifiers



Underfitting

Bagging (Bootstrap AGGregatING)

Bagging (AKA bootstrap aggregating), is a technique that repeatedly *samples* (*with replacement*) from a data set according to a uniform probability distribution

- Each bootstrap sample has the same size as the original data
- Some instances may **appear** several times in the same training set
- Probability of a training instance being selected in a bootstrap sample is:
  - *$1 - (1 - 1/n)^n$ (n: number of training instances)*
  - *~0.632 when n is large*

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Build classifier on each bootstrap sample

Bagging Algorithm

Algorithm 4.5 Bagging algorithm.

```
Let k be the number of bootstrap samples.
for i = 1 to k do
    Create a bootstrap sample of size N, D_i.
    Train a base classifier C_i on the bootstrap sample D_i.
end for
C*(x) = argmax_y Σ_i δ(C_i(x) = y).
{δ(·) = 1 if its argument is true and 0 otherwise.}
```

Let k be the number of bootstrap samples.

for $i = 1$ to k **do**

    Create a bootstrap sample of size N, D_i.

    Train a base classifier C_i on the bootstrap sample D_i.

end for

$C^*(x) = \underset{y}{\operatorname{argmax}} \sum_i \delta\big(C_i(x) = y\big)$.

$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise.$\}$

Bagging Example

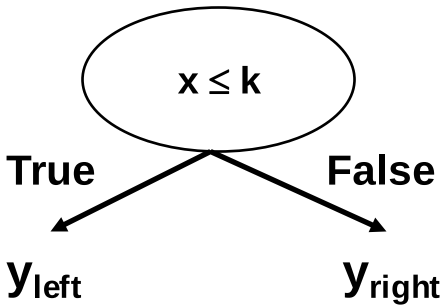
Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Classifier is a ***decision stump*** (decision tree of size 1)

- Decision rule: $x \leq k$ versus $x > k$
- Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

Summary of Trained Decision Stumps:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

Use majority vote (sign of sum of predictions) to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging can also increase the complexity (representation capacity) of simple classifiers such as decision stumps

Bagging Example

Use majority vote (sign of sum of predictions) to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging can also increase the complexity (representation capacity) of simple classifiers such as decision stumps

Bagging Example

Use majority vote (sign of sum of predictions) to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging can also increase the complexity (representation capacity) of simple classifiers such as decision stumps

Boosting

Boosting is an iterative procedure used to adaptively change the distribution of training examples for learning base classifiers so that they increasingly focus on examples that are hard to classify

Boosting assigns a weight to each training example and may adaptively change the weight at the end of each boosting round

1. They can be used to inform the sampling distribution used to draw a set of bootstrap samples from the original data.

2. They can be used to learn a model that is biased toward examples with higher weight.

Examples that are classified incorrectly will have their weights increased, while those that are classified correctly will have their weights decreased

This forces the classifier to focus on examples that are difficult to classify in subsequent iterations

AdaBoost

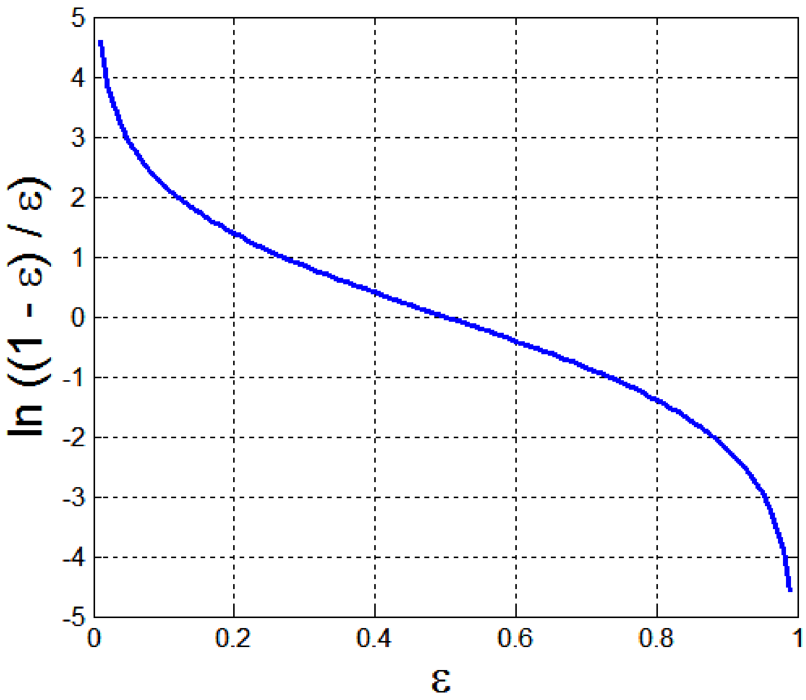
Consider a dataset $\{(x_i, y_i) | j = 1,2, \dots, N\}$

In **Adaboost** the ***importance*** of a classifier C_i *depends* on the error rate :

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j^{(i)} \, \delta(C_i(x_j) \neq y_j)$$

The ***importance*** of a base classifier is **given** by

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



AdaBoost Algorithm

Weight update:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

Where Z_i is the normalization factor

If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

$$C^*(x) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

AdaBoost Algorithm

Algorithm 4.6 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4:   Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5:   Train a base classifier C_i on D_i.
6:   Apply C_i to all examples in the original training set, D.
7:   $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \; \delta\big(C_i(x_j) \neq y_j\big)\right]$ {Calculate the weighted error.}
8:   **if** $\epsilon_i > 0.5$ **then**
9:     $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10:     Go back to Step 4.
11:   **end if**
12:   $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13:   Update the weight of each example according to Equation 4.103.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta\big(C_j(\mathbf{x}) = y\big)$.

AdaBoost Example

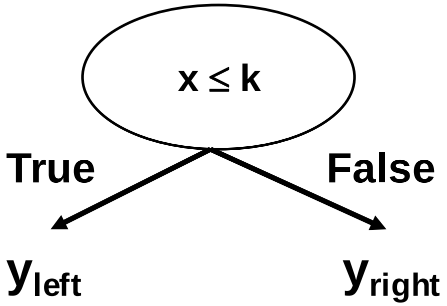
Consider 1-dimensional data set:

Original Data:

x	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
y	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**1**	**1**	**1**

Classifier is a ***decision stump*** (decision tree of size 1)

- Decision rule: $x \leq k$ versus $x > k$
- Split point k is chosen based on entropy



AdaBoost Example

Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

AdaBoost Example

Alpha

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Random Forest Algorithm

Construct an ensemble of decision trees by manipulating training set as well as features

- Use bootstrap sample to train every decision tree (similar to Bagging)
- Use the following tree induction algorithm:
  - At every internal node of decision tree, randomly sample p attributes for selecting split criterion
  - Repeat this procedure until all leaves are pure (unpruned tree)

Random Forest Algorithm

Construct an ensemble of decision trees by manipulating training set as well as features

- Base classifier are unpruned trees and hence are unstable classifier
- Base classifier are uncorrelated (due to randomization in training and features)
- Random forest reduce variance of unstable classifier without impacting the bias