Some More Classifiers: Naïve Bayes Classifier, Decision Trees, Support Vector Machines

CS277

Slide Materials

- Tan, Steinbach and Kumar's book on “An Introduction to Data Mining”
- Gareth James, Daniella Witten, Trevor Hastie and Robert Tibshirani, “An Introduction to Statistical Learning with Applications in R”
- Some of the materials are used from the tutorial of Prof. Andreas Tilevik, University of Skövde, Sweden

Bayes Theorem

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

Bayes Theorem

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

$$posterior = \frac{likelihood * prior}{evidence}$$

Naïve Bayes Classification

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

$$posterior = \frac{likelihood * prior}{evidence}$$

Training Naïve Bayes

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

Class Feature

Training Naïve Bayes: The Naïve Assumption

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

- Difficult to calculate joint probabilities produced by expanding for all features

$$P(C|X) = P(X_1\ X_2\ \dots\ X_n|C) * P(C)$$
$$P(C|X) = P(X_1|X_2\ \dots\ X_n\ C) * P(X_2\ \dots\ X_n|C) * P(C)$$
$$P(C|X) = \dots$$

Training Naïve Bayes: The Naïve Assumption

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

- **Solution:** assume all features independent of each other

$$P(C|X) = P(X_1|C) * P(X_2|C) * \cdots * P(X_n|C) * P(C)$$

Training Naïve Bayes: The Naïve Assumption

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

- **Solution:** assume all features independent of each other

$$P(C|X) = P(X_1|C) * P(X_2|C) * \cdots * P(X_n|C) * P(C)$$

- This is the "naïve" assumption

$$P(C|X) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

Training Naïve Bayes

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

- Class assignment is selected based on *maximum a posteriori* (MAP) rule

$$\underset{k \in \{1, \dots K\}}{argmax} P(C_k) \prod_{i=1}^{n} P(X_i|C_k)$$

Training Naïve Bayes

- For each class (C), calculate probability given features (X)

$$P(C|X) = P(X|C) * P(C)$$

- Class assignment is selected based on *maximum a posteriori* (MAP) rule

$$\underset{k \in \{1, \ldots K\}}{argmax} P(C_k) \prod_{i=1}^{n} P(X_i|C_k)$$

Means select potential class with largest value

The Log Trick

- Multiplying many values together causes computational instability (underflows)

$$\underset{k \in \{1, \ldots K\}}{argmax} P(C_k) \prod_{i=1}^{n} P(X_i | C_k)$$

The Log Trick

- Multiplying many values together causes computational instability (underflows)

$$\underset{k \in \{1, \dots K\}}{argmax} P(C_k) \prod_{i=1}^{n} P(X_i | C_k)$$

- Work with log values and sum the results

$$\log(P(C_k)) \sum_{i=1}^{n} \log(P(X_i | C_k))$$

Example: Predicting Tennis With Naïve Bayes

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example: Training Naïve Bayes Tennis Model

P(Play=Yes) = 9/14 P(Play=No) = 5/14

Create probability lookup tables based on training data

Example: Training Naïve Bayes Tennis Model

P(Play=Yes) = 9/14 P(Play=No) = 5/14

Outlook	Play=Yes	Play=No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=Yes	Play=No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=Yes	Play=No
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=Yes	Play=No
Strong	3/9	3/5
Weak	6/9	2/5

Create probability lookup tables based on training data

Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

x'=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

$$P(yes|sunny, cool, high, strong) = P(sunny|yes) \quad P(cool|yes) * P(high|yes) * P(strong|yes) * P(yes)$$

$$P(no|sunny, cool, high, strong) = P(sunny|no) \quad P(cool|no) * P(high|no) * P(strong|no) * P(no)$$

Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

x’=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

Feature	Play=Yes	Play=No
Outlook=Sunny		
Temperature=Cool		
Humidity=High		
Wind=Strong		
Overall Label		

Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

x’=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

Feature	Play=Yes	Play=No
Outlook=Sunny	2/9	3/5
Temperature=Cool	3/9	1/5
Humidity=High	3/9	4/5
Wind=Strong	3/9	3/5
Overall Label	**9/14**	**5/14**

Example: Predicting Tennis With Naïve Bayes

Predict outcome for the following:

x’=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

Feature	Play=Yes	Play=No
Outlook=Sunny	2/9	3/5
Temperature=Cool	3/9	1/5
Humidity=High	3/9	4/5
Wind=Strong	3/9	3/5
Overall Label	**9/14**	**5/14**
Probability	**0.0053**	**0.0206**

Laplace Smoothing

Problem: categories with no entries result in a value of "0" for conditional probability

$$P(C|X) = \boxed{0}\ P(X_1|C) * P(X_2|C) * P(C)$$

Solution: add “k" to numerator and denominator to avoid 0 likelihood

$$P(X_i|C) = \frac{\#(X_i, C) + k}{\#(C) + k|X_i|}$$

K is smoothing parameter

Number of different values of X_i

Combining Feature Types

Problem

- Model features contain different data types (continuous and categorical)

Solutions

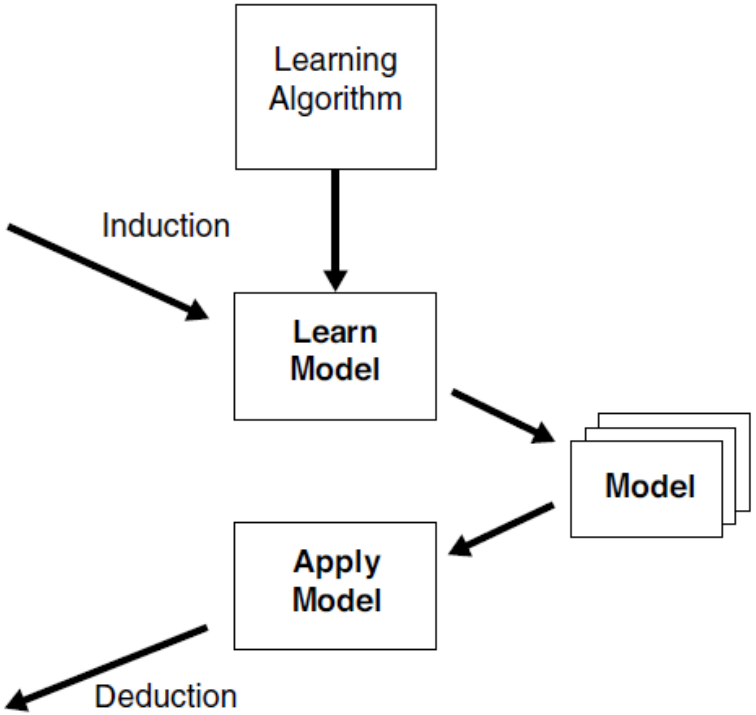
Option 1: Bin continuous features to create categorical ones and fit multinomial model

Option 2: Fit Gaussian model on continuous features and multinomial on categorical features; combine to create "meta model"

Classification Framework

Training Set

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



Test Set

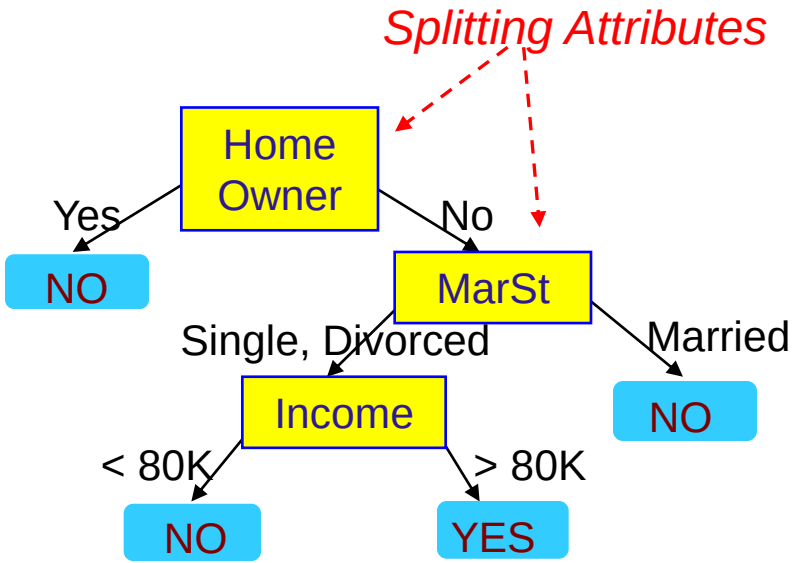
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

"Introduction to Data Mining" by Tan, Steinbach, Kumar

Example of a Decision Tree

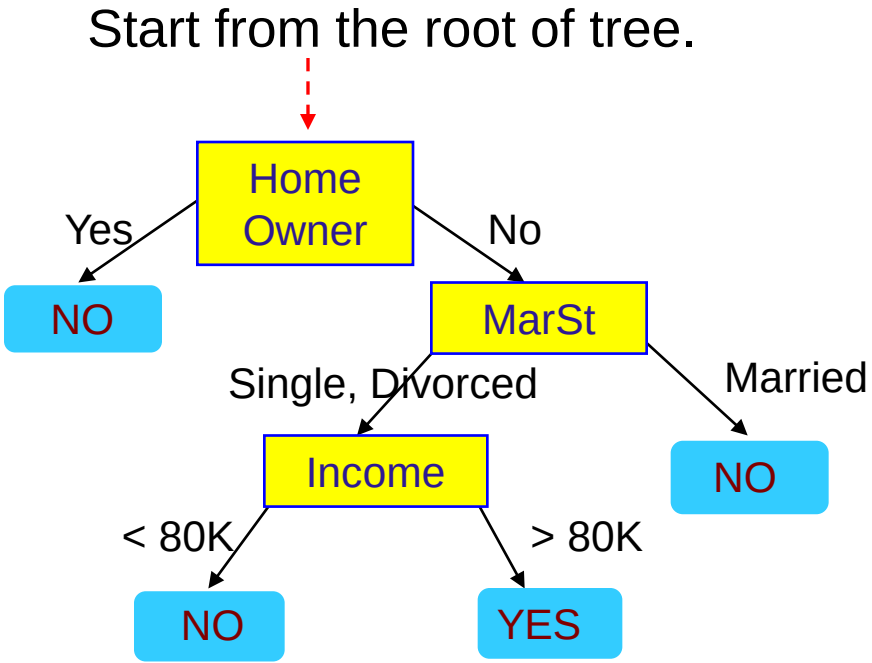
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
	binary	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

Apply Model to Test Data



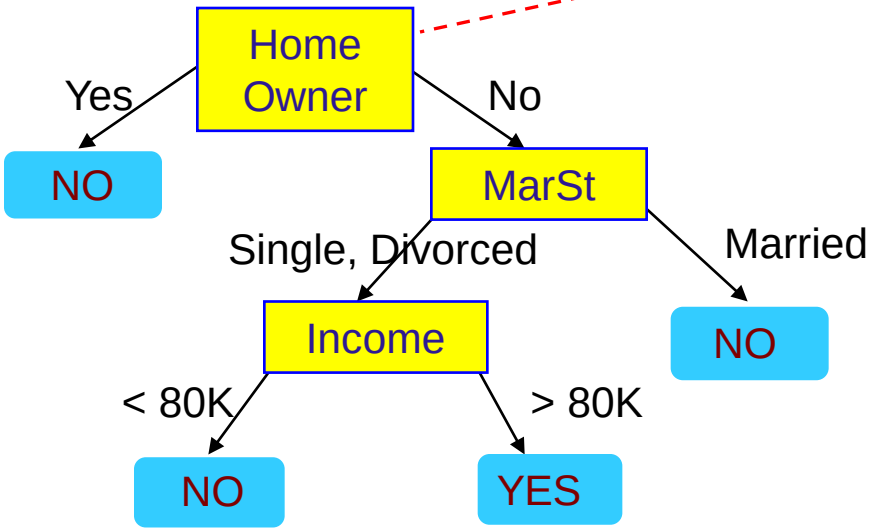
Test Data

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

Apply Model to Test Data

Test Data

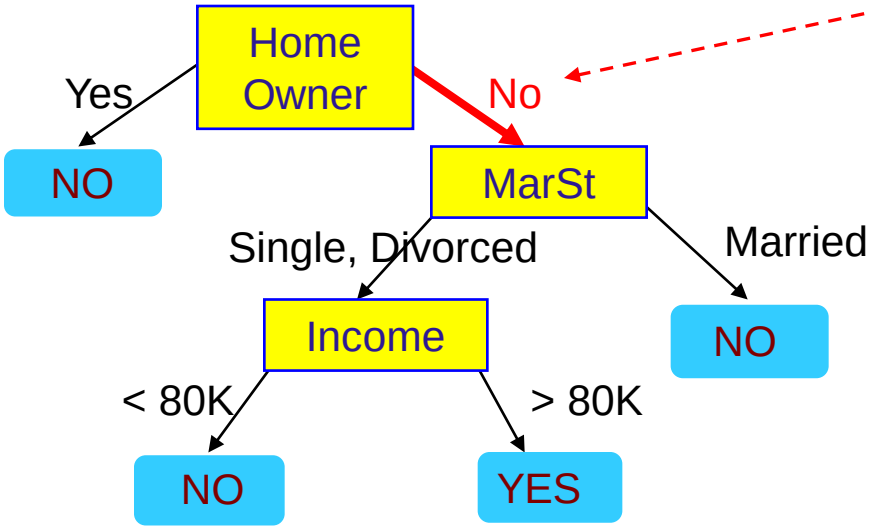
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

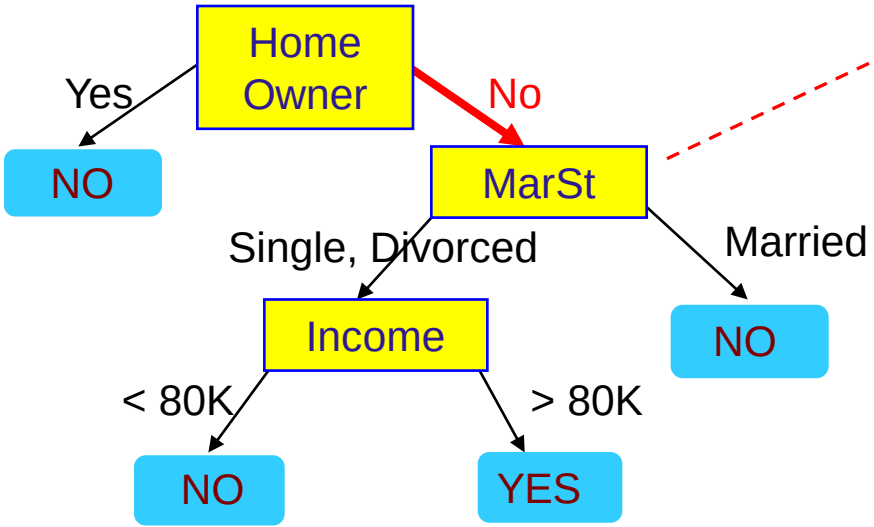
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

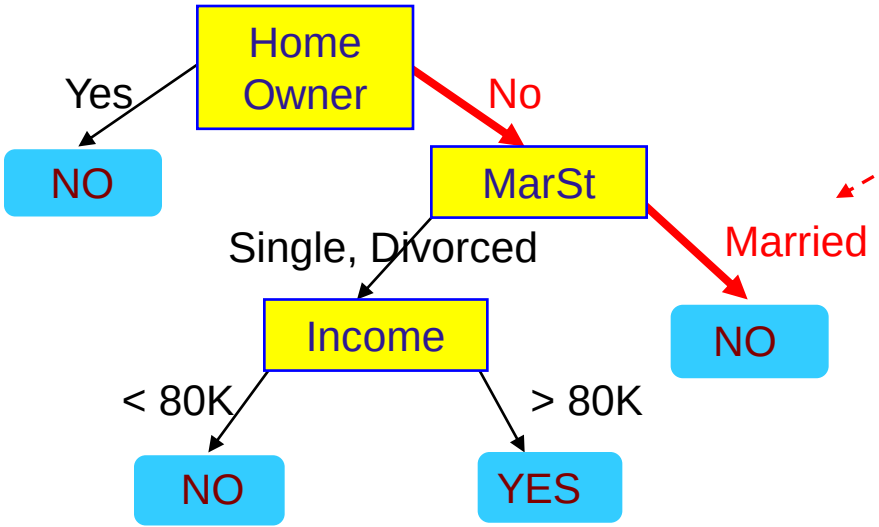
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

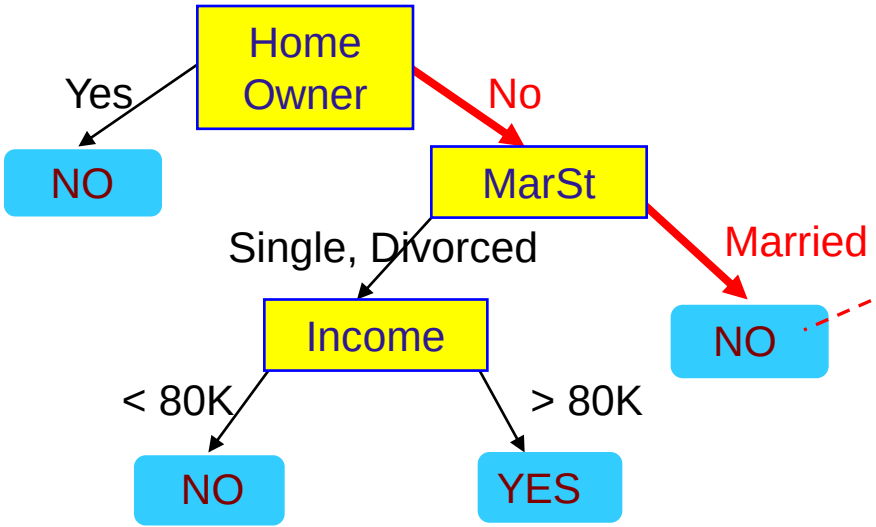
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

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

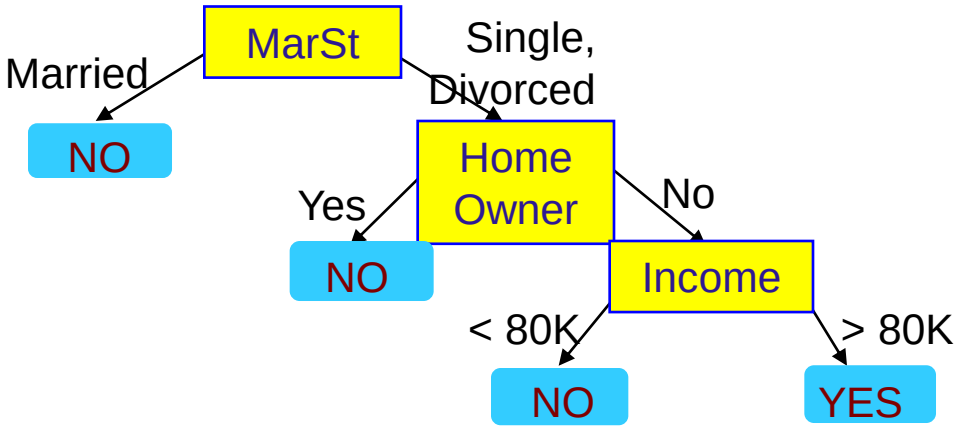


Assign Defaulted to "No"

Another Example of Decision Tree

binary 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!

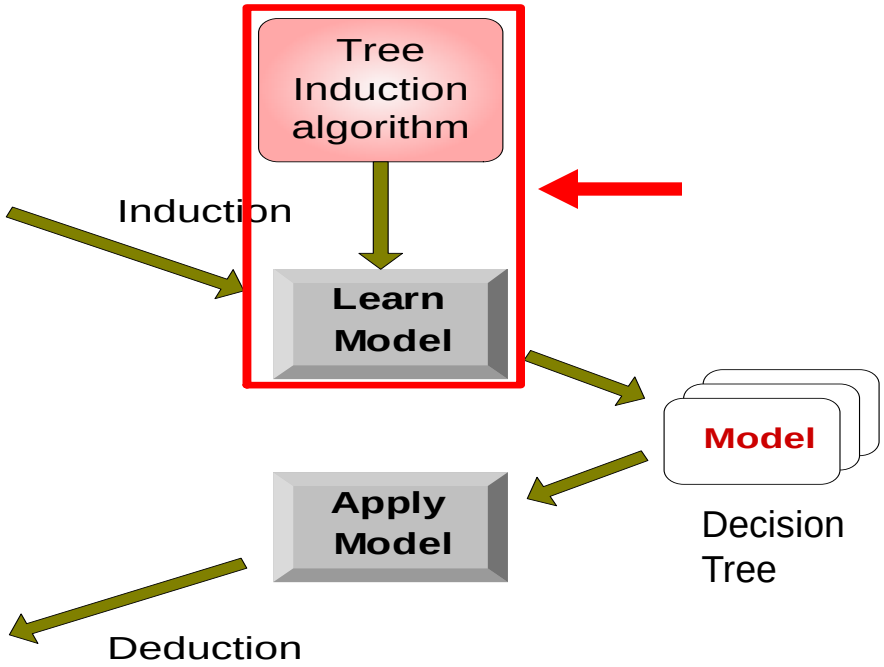
Decision Tree Classification Task

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



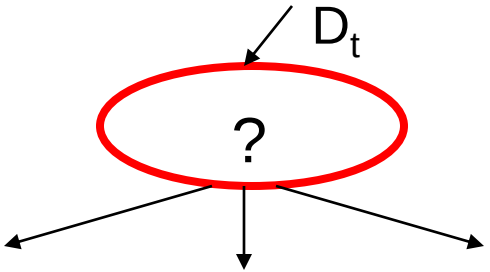
Decision Tree Induction

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

General Structure of Hunt's Algorithm

- Let 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

(7,3)

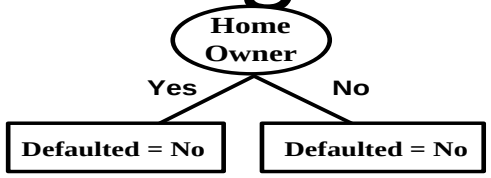
(a)

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

(7,3)

(a)

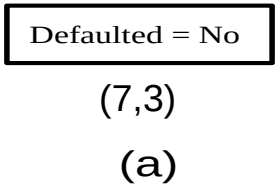


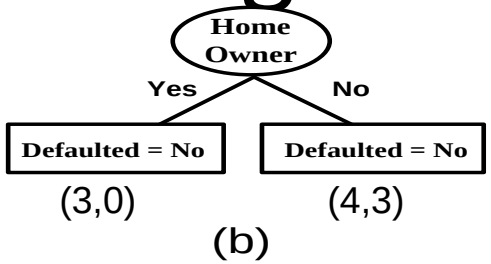
(3,0) (4,3)

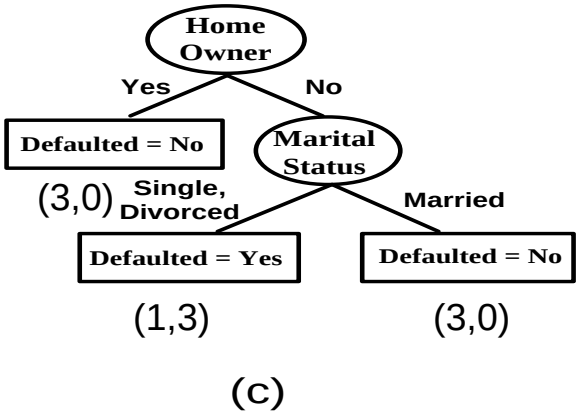
(b)

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





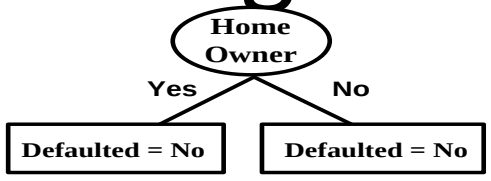


ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Hunt's Algorithm

(7,3)

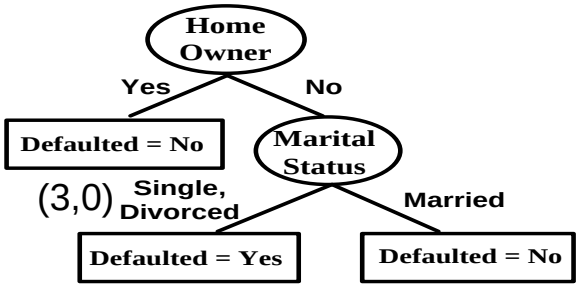
(a)



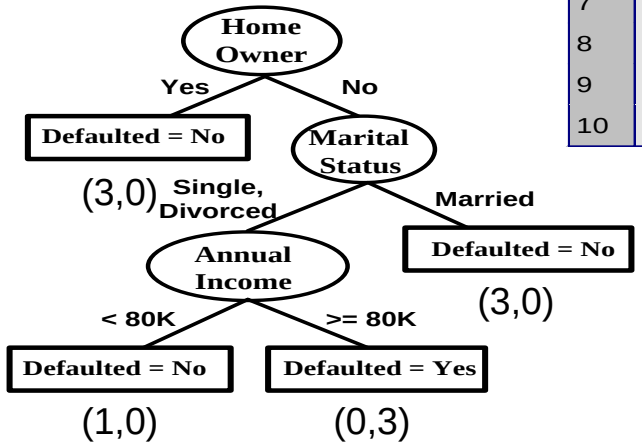
(3,0) (4,3)

(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



(c)



(d)

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

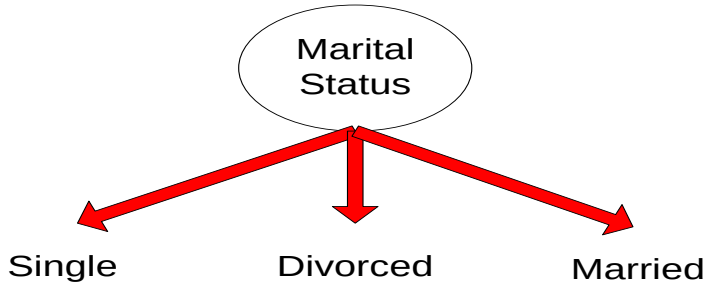
- How should the splitting procedure stop?
  - Stop splitting if all the records belong to the same class or have identical attribute values
  - Early termination

Methods for Expressing Test Conditions

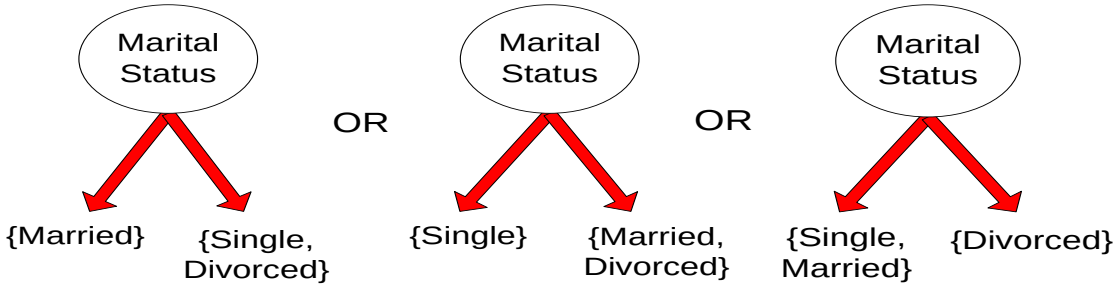
- Depends on attribute types
  - Binary
  - Nominal
  - Ordinal
  - Continuous

Test Condition for Nominal Attributes

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

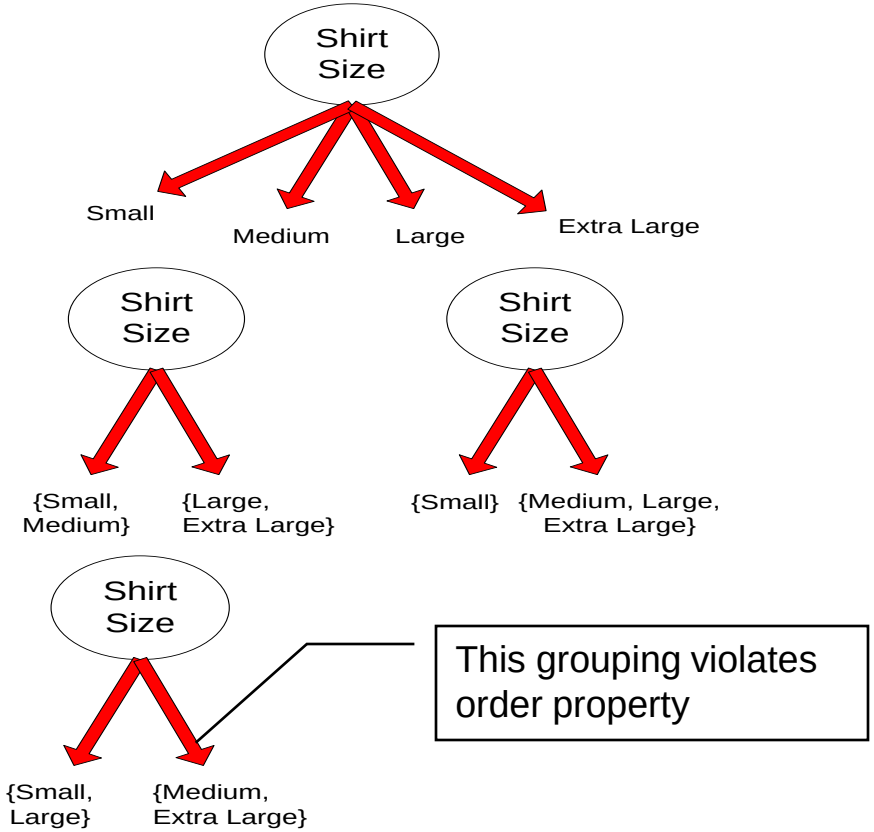


- Binary split:
  - Divides values into two subsets

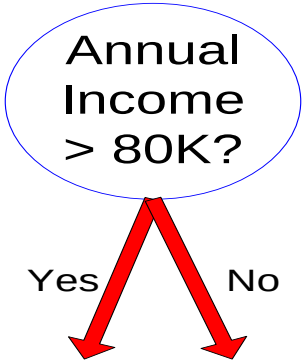


Test Condition for Ordinal Attributes

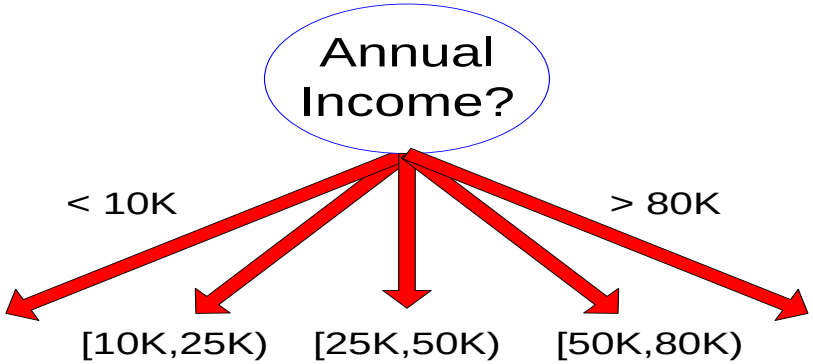
- Multi-way split:
  - Use as many partitions as distinct values
- Binary split:
  - Divides values into two subsets
  - Preserve order property among attribute values



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

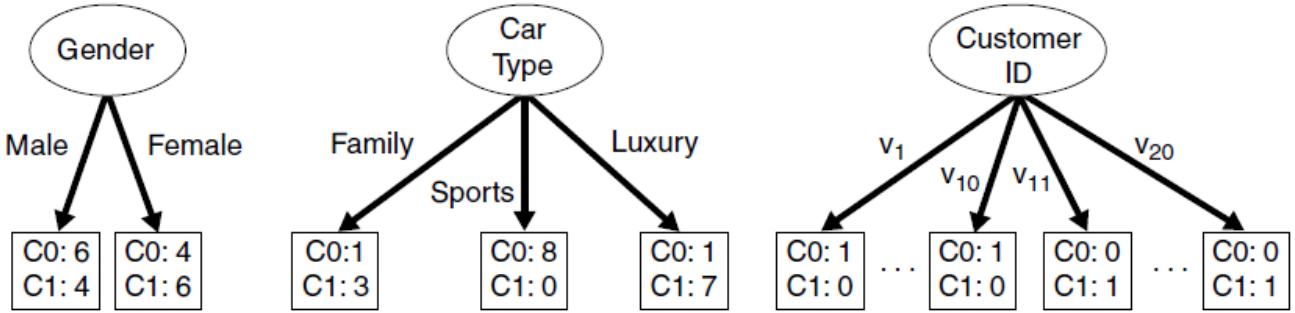
    Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
    - Static – discretize once at the beginning
    - Dynamic – repeat at each node

  - 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?

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

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



Which test condition is the best?

How to determine the Best Split

- Greedy approach:
  - Nodes with purer class distribution are preferred

- Need a measure of node impurity:

C0: 5 C1: 5

High degree of impurity

C0: 9 C1: 1

Low degree of impurity

Measures of 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

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

- Misclassification error

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

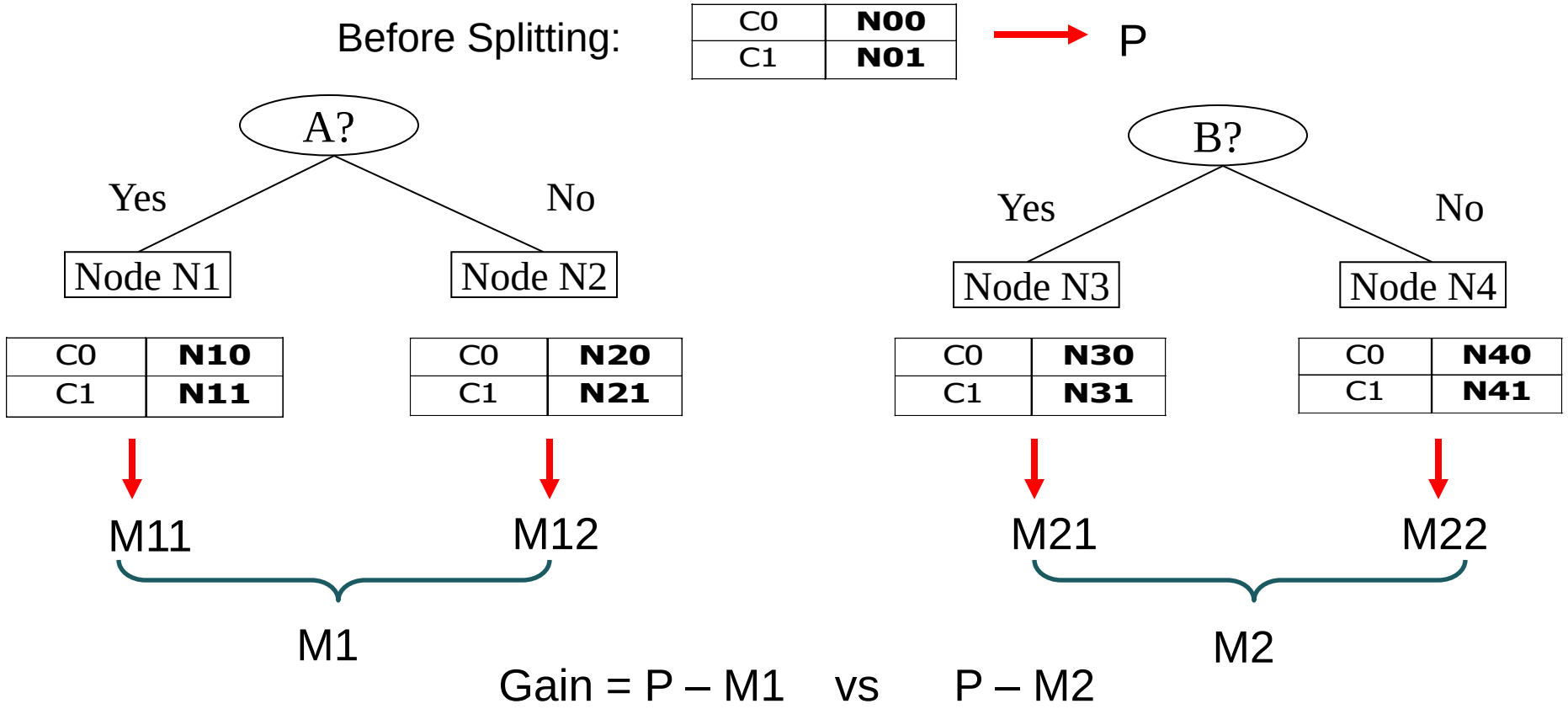
Finding the Best Split

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting
   - Compute impurity measure of each child node
   - M is the weighted impurity of child nodes
3. Choose the attribute test condition that produces the highest gain

Gain = P - M

or equivalently, lowest impurity measure after splitting (M)

Finding the Best Split



Measure of Impurity: GINI

- Gini Index for a given node $\boldsymbol{t}$

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

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

  - Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
  - Minimum of 0 when all records belong to one class, implying the most beneficial situation for classification
  - Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT

Measure of Impurity: GINI

- Gini Index for a given node t :

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

  - For 2-class problem (p, 1 – p):
    - GINI = $1 - p^2 - (1 - p)^2 = 2p\ (1\text{-}p)$

Computing Gini Index of a Single Node

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

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

Computing Gini Index for a Collection of Nodes

- When a node p is split into k partitions (children)

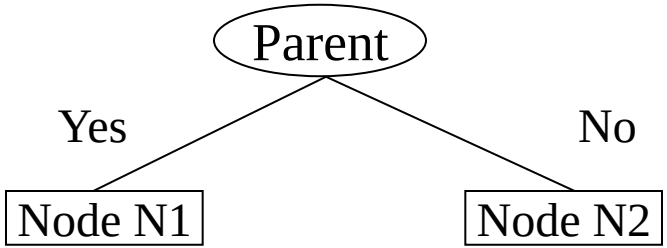
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at parent node p.

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(N1)

	N1	N2
C1	5	2
C2	1	4

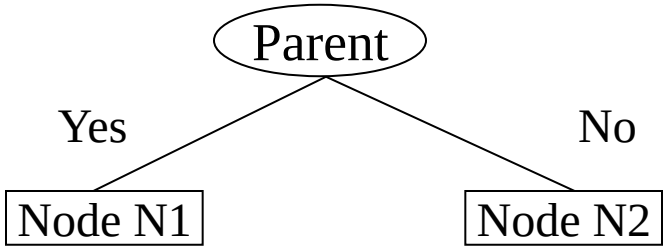
Weighted Gini of N1 N2

Gini(N2)

Gain :

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
- Use the count matrix to make decisions

Multi-way split

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

Two-way split
(find best partition of values)

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

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

Compute the Gini values

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	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	

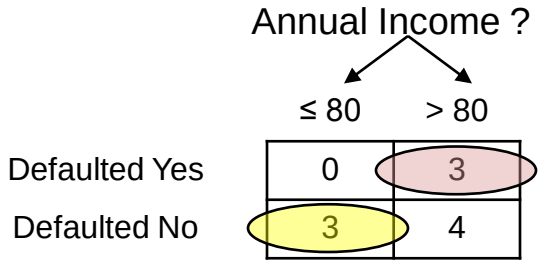
	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

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



Continuous Attributes: Computing Gini Index...

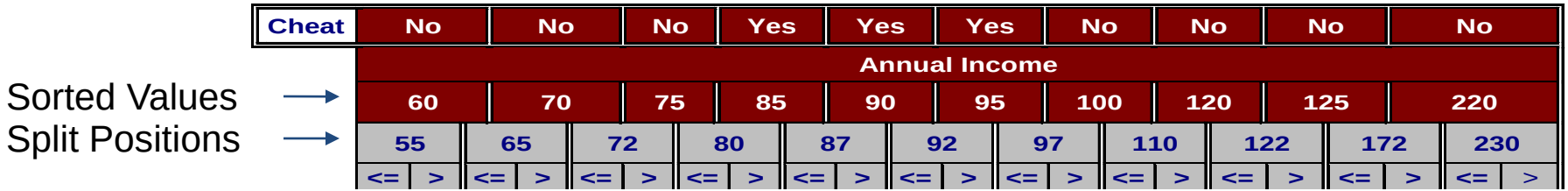
- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Sorted Values →

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

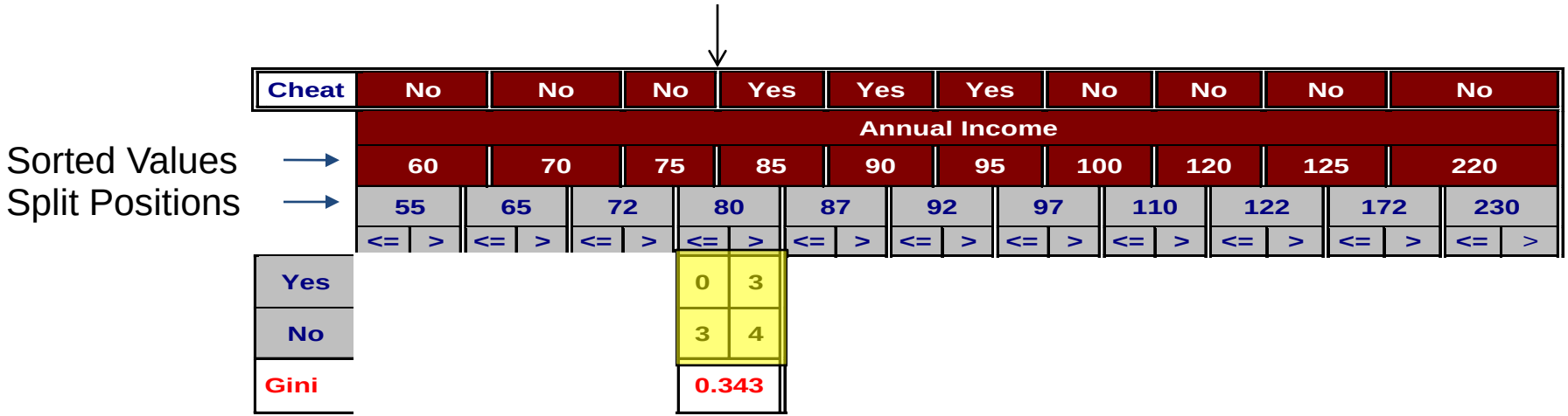
Continuous Attributes: Computing Gini Index...

- 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



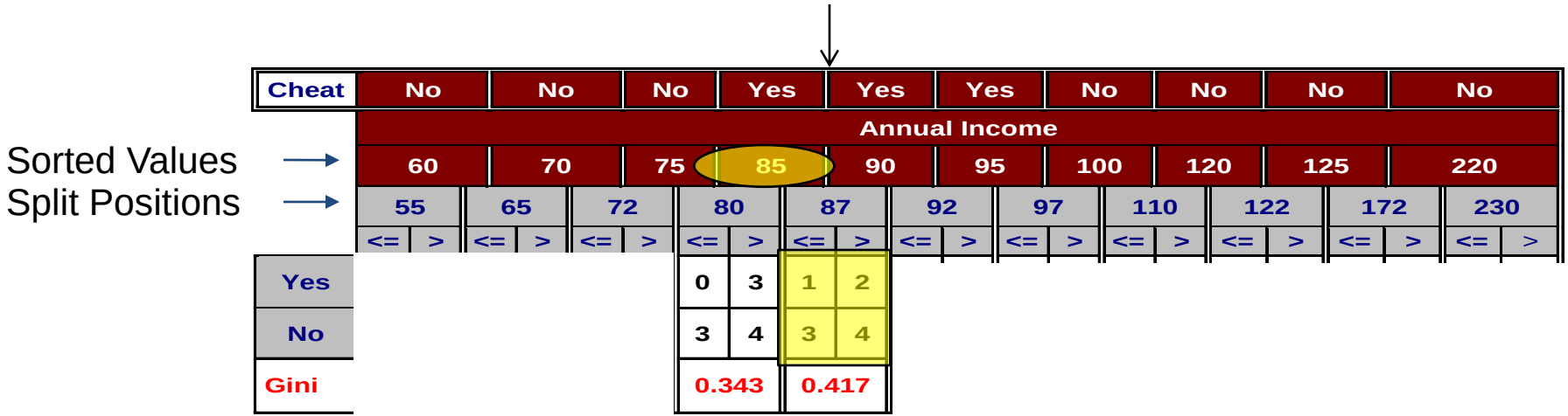
Continuous Attributes: Computing Gini Index...

- 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



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



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	

Measure of Impurity: Entropy

- Entropy at a given node $\boldsymbol{t}$

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

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

- Maximum of $\log_2 c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
- Minimum of 0 when all records belong to one class, implying most beneficial situation for classification

– Entropy based computations are quite similar to the GINI index computations

Computing Entropy of a Single Node

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

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

Computing Information Gain After Splitting

- Information Gain:

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

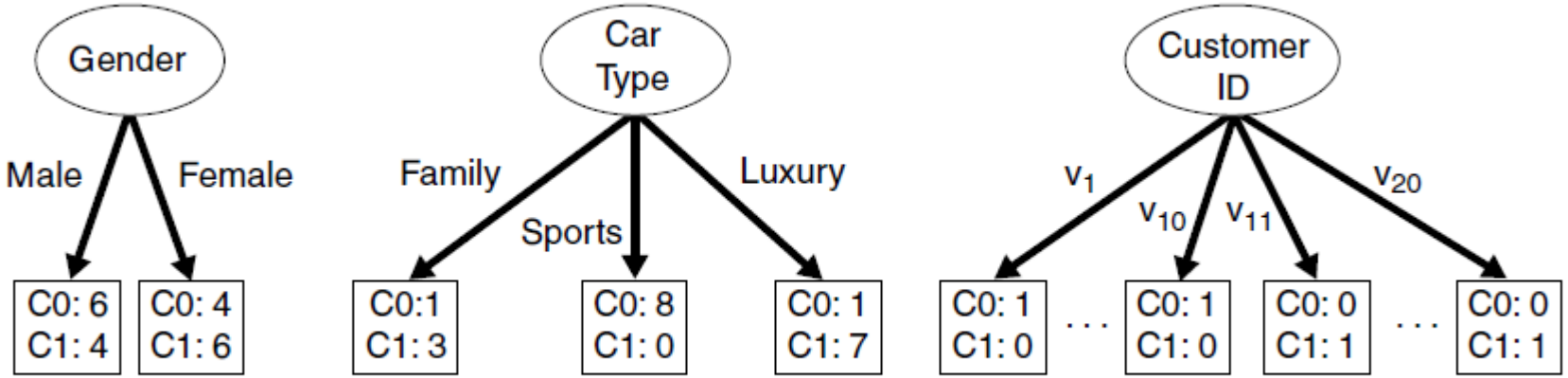
Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

  - Choose the split that achieves most reduction (maximizes GAIN)
  - Used in the ID3 and C4.5 decision tree algorithms
  - Information gain is the mutual information between the class variable and the splitting variable

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 has highest information gain because entropy for all the children is zero

Gain Ratio

- Gain Ratio:

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

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

n_i is number of records in child node i

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

Gain Ratio

- Gain Ratio:

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

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

n_i is number of records in child node i

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

SplitINFO

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

SplitINFO

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

SplitINFO

Gain Ratio

- Gain Ratio:

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

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

n_i is number of records in child node i

	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

Measure of Impurity: Classification Error

- Classification error at a node t

$$Error(t) = 1 - \max_i [p_i(t)]$$

  - Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least interesting situation
  - Minimum of 0 when all records belong to one class, implying the most interesting situation

Computing Error of a Single Node

$$Error(t) = 1 - \max_i [p_i(t)]$$

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

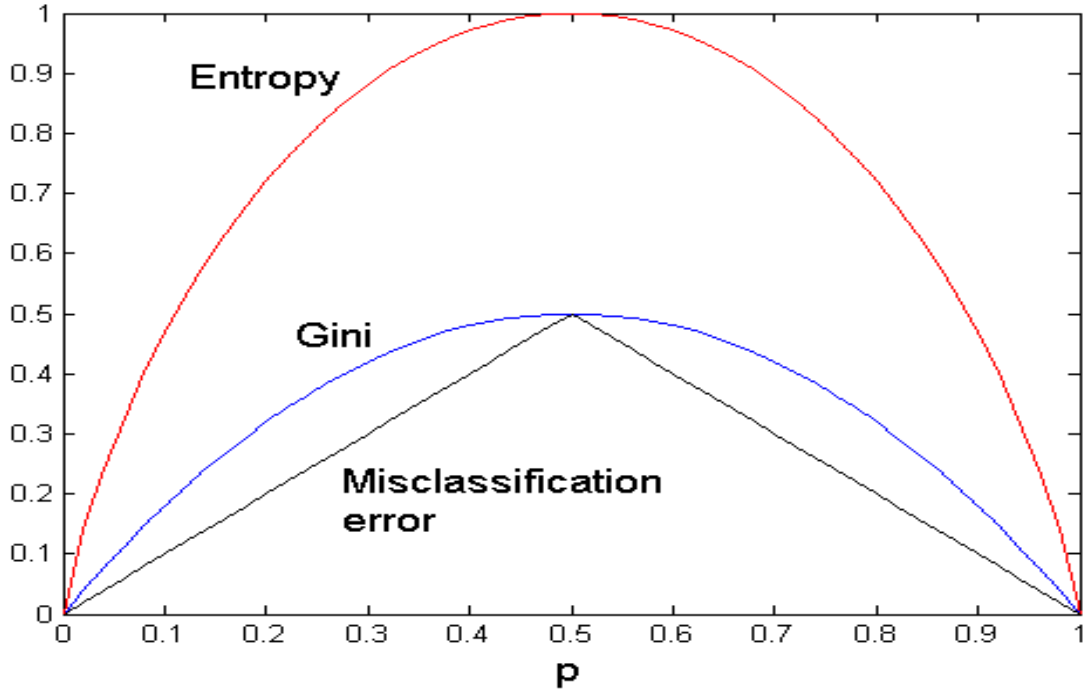
C1	2
C2	4

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

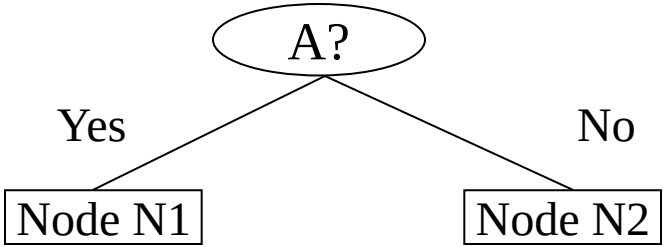
Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Impurity Measures

For a 2-class problem:



Misclassification Error vs Gini Index



	Parent
C1	7
C2	3
Gini = 0.42	

Gini(N1)
= 1 – $(3/3)^2$ – $(0/3)^2$
= 0

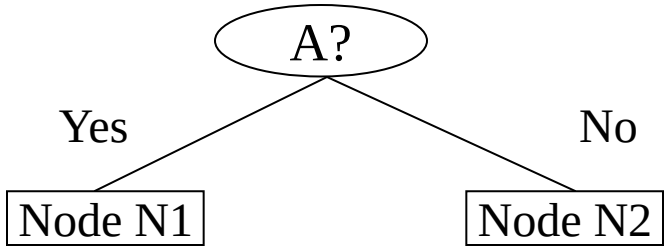
Gini(N2)
= 1 – $(4/7)^2$ – $(3/7)^2$
= 0.489

	N1	N2
C1	3	4
C2	0	3
Gini=0.342		

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

Gini improves but error remains the same!!

Misclassification Error vs Gini Index



	Parent
C1	7
C2	3
Gini = 0.42	

	N1	N2
C1	3	4
C2	0	3
Gini=0.342		

	N1	N2
C1	3	4
C2	1	2
Gini=0.416		

Misclassification error for all three cases = 0.3 !

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

Tree Pruning

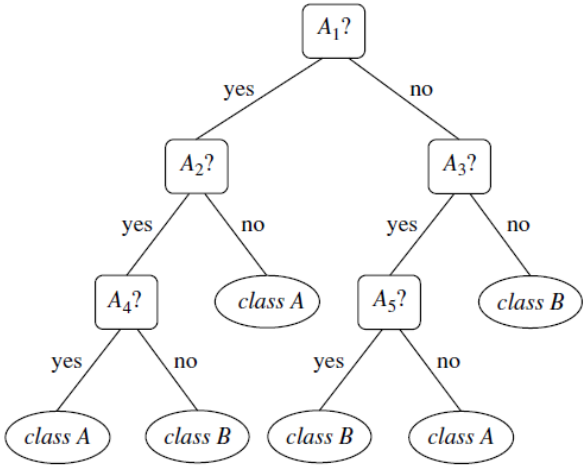
- The training samples may contain noises
- Some branches may reflect anomalies
- Tree pruning may address the problem of overfitting
- Methods
  - Prepruning
  - Postpruning

Tree Prepruning

- A tree is “pruned” by halting its construction early
- Upon halting, the node becomes a leaf.
- The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.
- When to halt?
  - If partitioning the tuples at a node would result in a split that falls below a pre-specified threshold, then further partitioning of the given subset is halted.

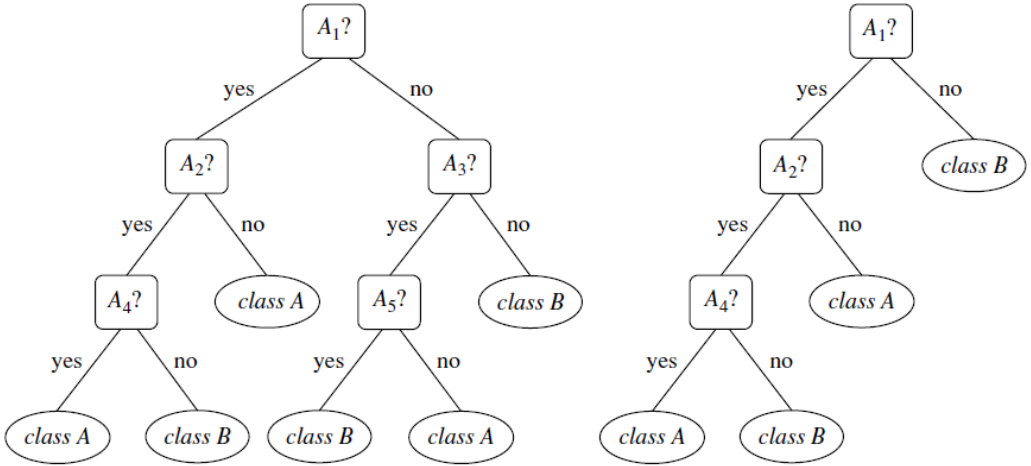
Tree Postpruning

- Removes subtrees from a “fully grown” tree
- A subtree at a given node is pruned by removing its branches and replacing it with a leaf.
- The leaf is labeled with the most frequent class among the subtree being replaced



Tree Postpruning

- Removes subtrees from a “fully grown” tree
- A subtree at a given node is pruned by removing its branches and replacing it with a leaf.
- The leaf is labeled with the most frequent class among the subtree being replaced



- The **cost complexity** pruning algorithm used in CART is an example of the postpruning approach
- This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the **error rate** is the percentage of tuples misclassified by the tree)
- It starts from the bottom of the tree
- For each internal node, *N*, it computes the cost complexity of the subtree at *N*, and the cost complexity of the subtree at *N* if it were to be pruned (i.e., replaced by a leaf node)
- The two values are compared If pruning the subtree at node *N* would result in a smaller cost complexity, then the subtree is pruned. Otherwise, it is kept.

Pruning Set

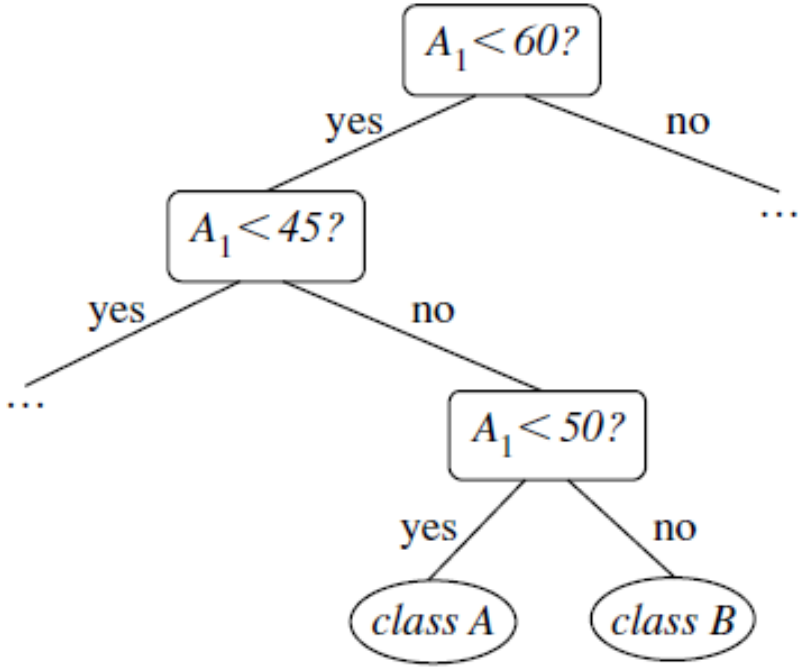
- A **pruning set** of class-labeled tuples is used to estimate cost complexity.
- This set is independent of the training set used to build the unpruned tree and of any test set used for accuracy estimation.
- The algorithm generates a set of progressively pruned trees.
- In general, the smallest decision tree that minimizes the cost complexity is preferred

Pessimistic Pruning

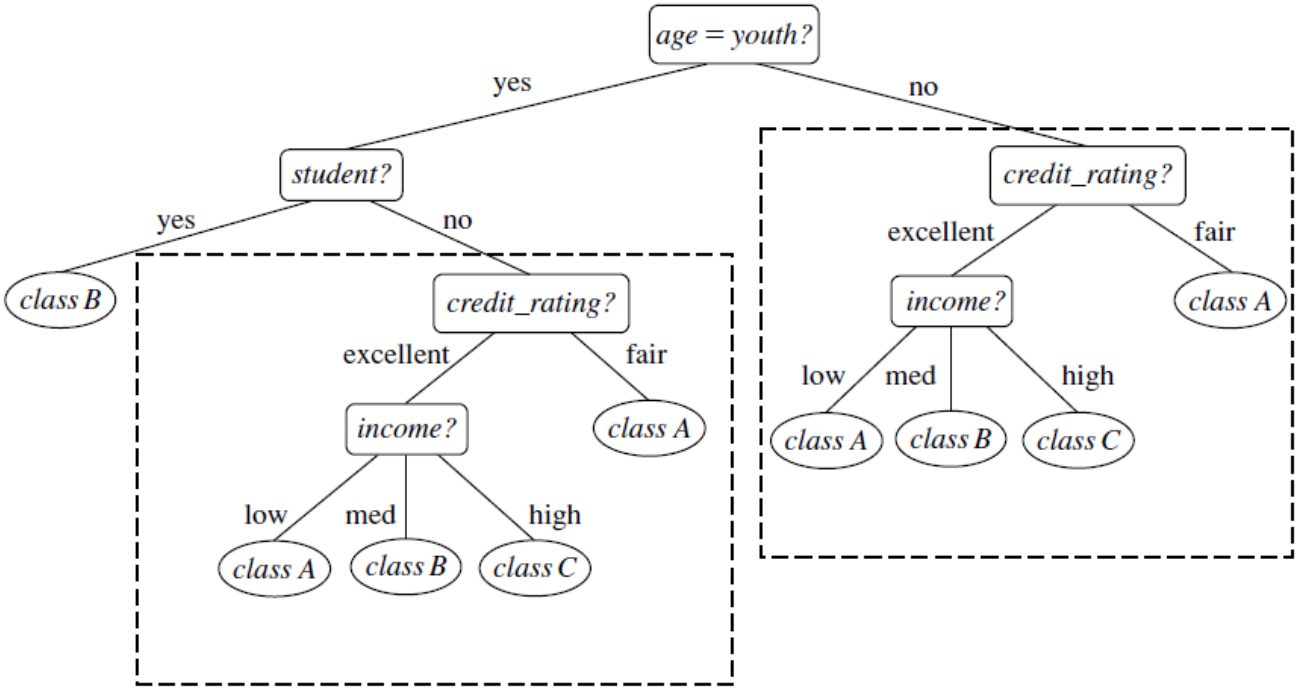
- C4.5 uses a method called **pessimistic pruning**, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning.
- Pessimistic pruning, however, does not require the use of a prune set.
- Instead, it uses the training set to estimate error rates.
- An estimate of accuracy or error based on the training set is overly optimistic and, therefore, strongly biased
- The pessimistic pruning method therefore adjusts the error rates obtained from the training set by adding a penalty, so as to counter the bias incurred

Decision tree can also suffer from repetition and replication

Repetition



Replication



Support Vector Machines

Here we approach the two-class classification problem in a direct way:

> *We try and find a plane that separates the classes in feature space.*

If we cannot, we get creative in two ways:

- We soften what we mean by "separates", and
- We enrich and enlarge the feature space so that separation is possible.

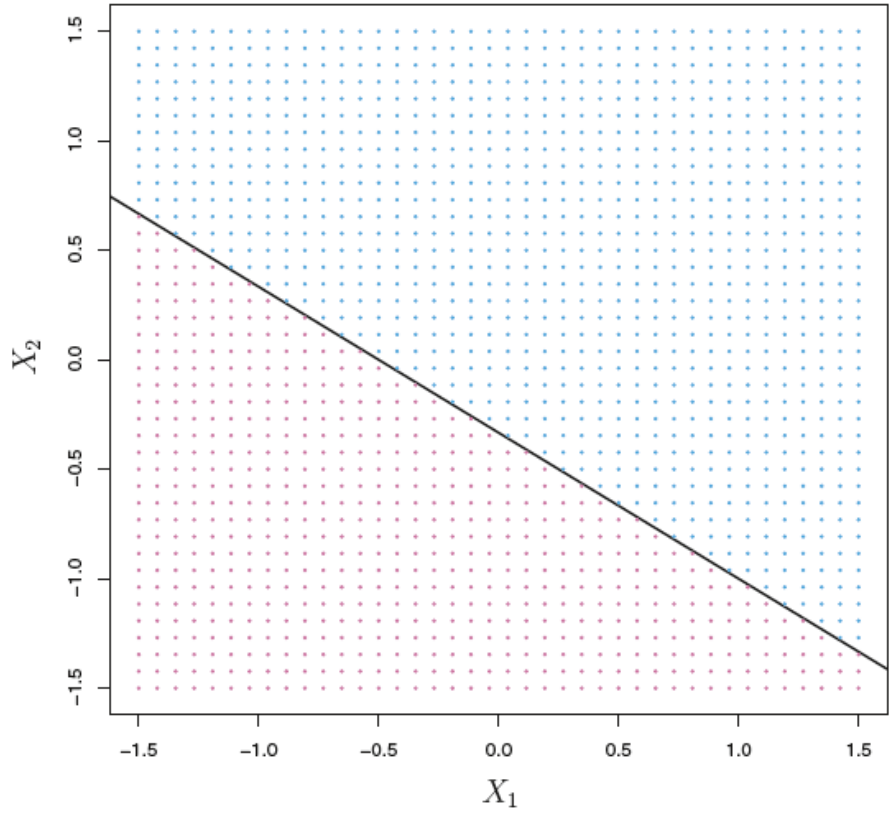
What is a Hyperplane?

- A hyperplane in p dimensions is a flat affine subspace of dimension $p - 1$.
- In general the equation for a hyperplane has the form

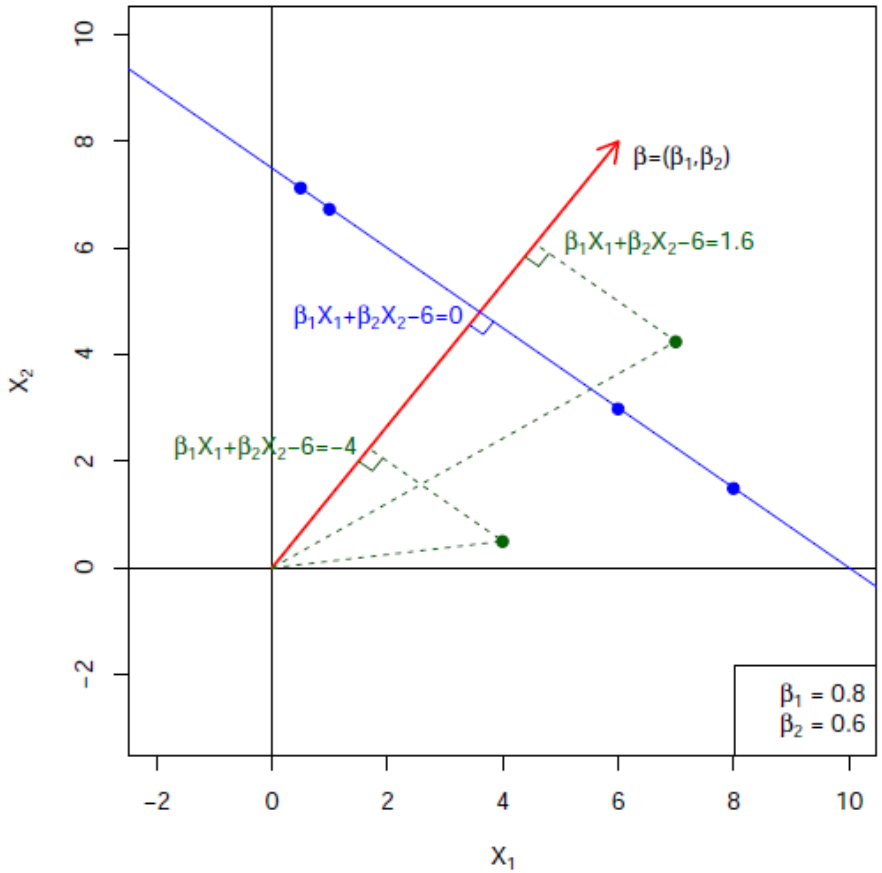
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$

- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.

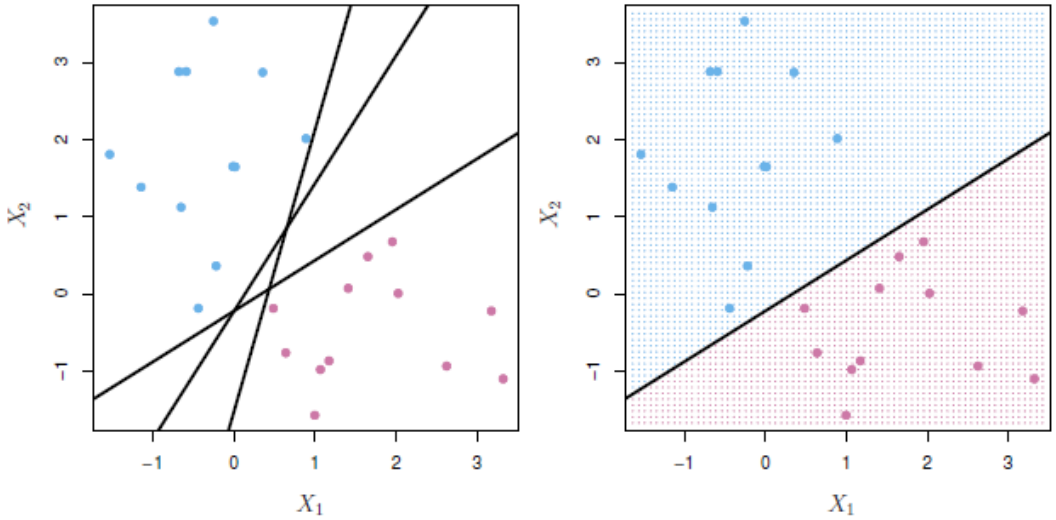
Hyperplane



Hyperplane in 2 Dimensions



Separating Hyperplanes

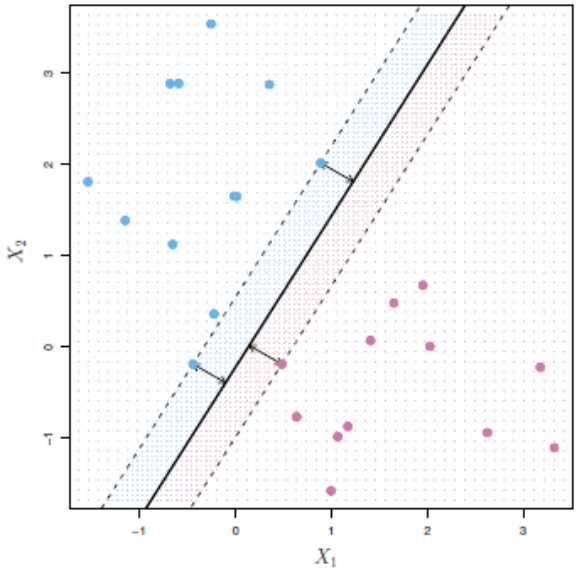


- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.

Aim to maximize the marginal distance



Constrained optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

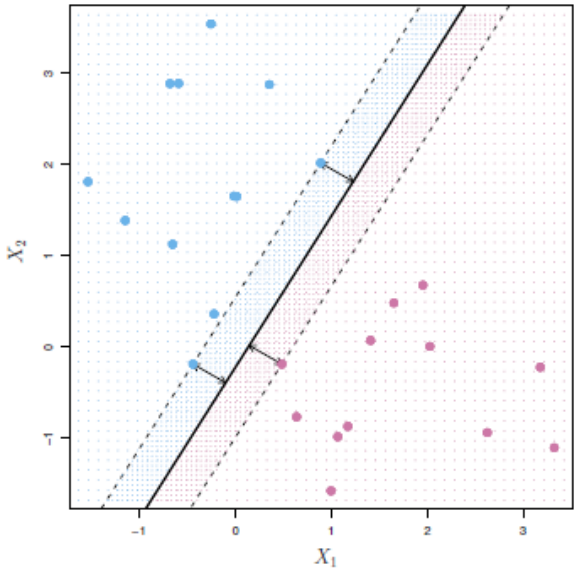
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{for all } i = 1, \ldots, N.$$

Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

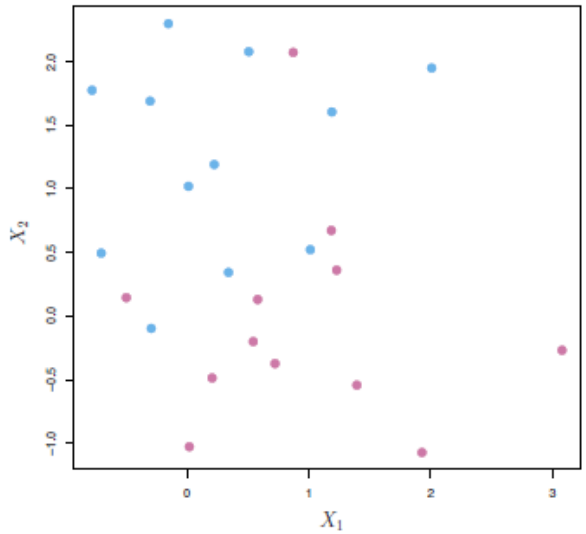
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{for all } i = 1, \ldots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package `e1071` solves this problem efficiently

Non-separable Data



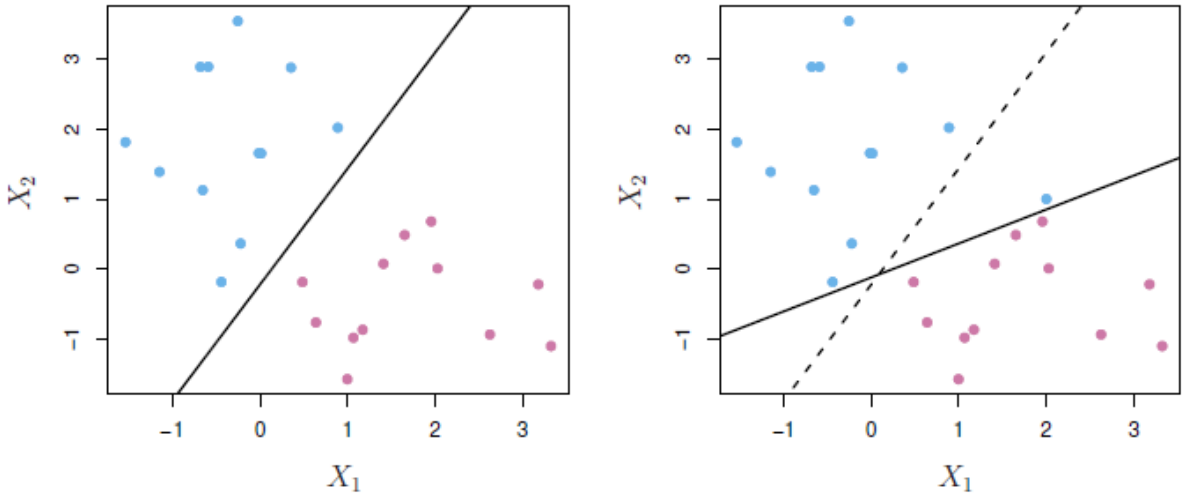
The data on the left are not separable by a linear boundary.

This is often the case, unless $N < p$.

Sometimes it is possible that the data is non separable by a hyperplane.

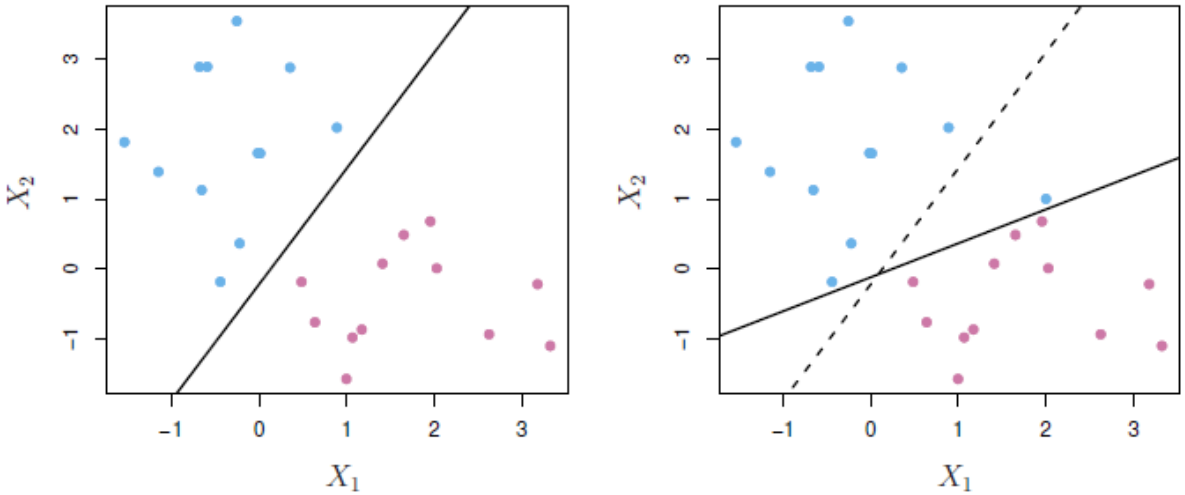
In such a case a kernel function is introduced that uses some transformation to transform the data points to higher dimension.

Noisy Data



Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

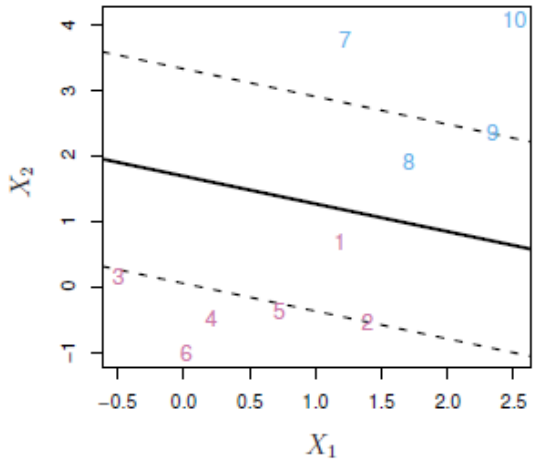
Noisy Data

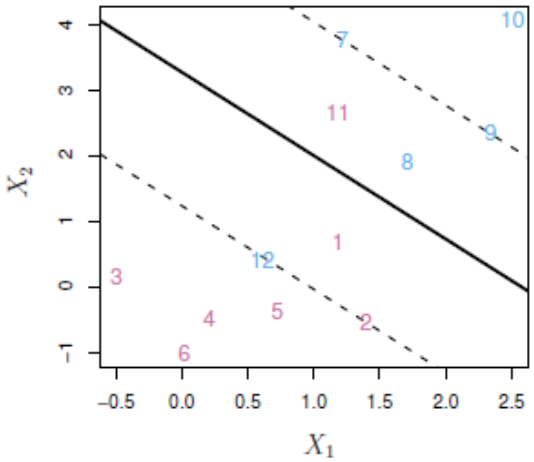


Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

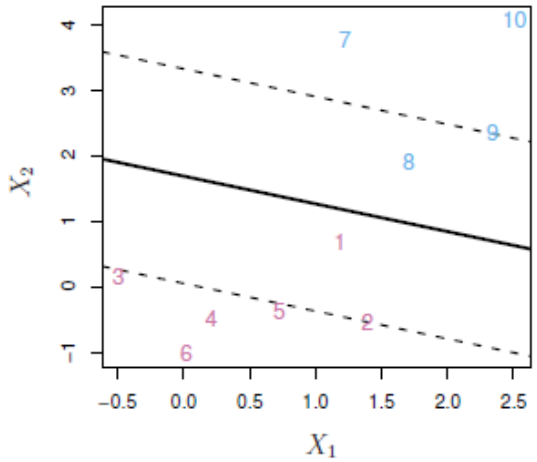
The *support vector classifier* maximizes a *soft* margin.

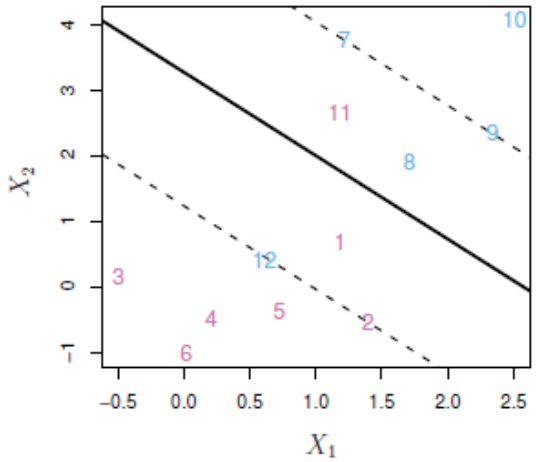
Support Vector Classifier





Support Vector Classifier



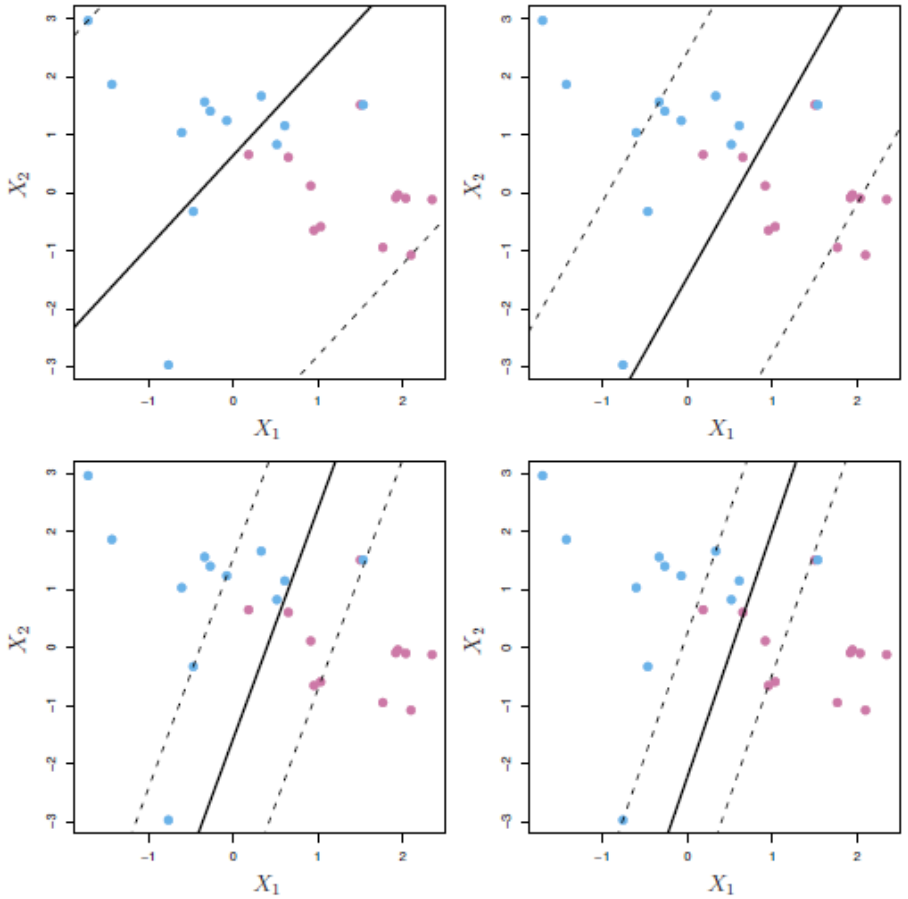


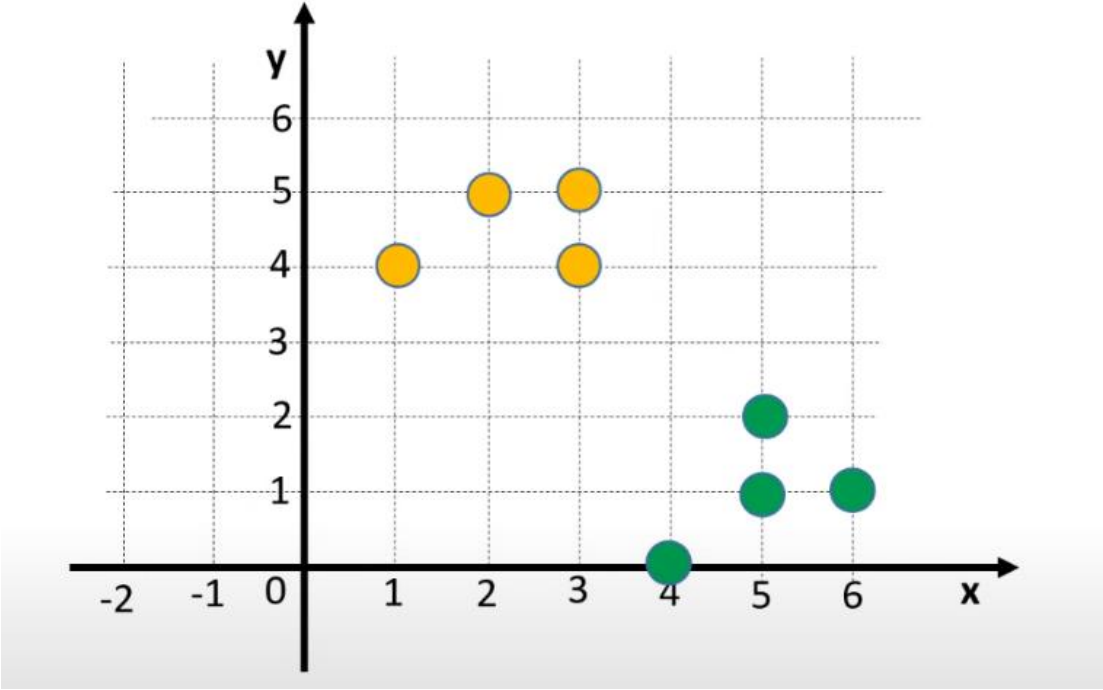
$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \; M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

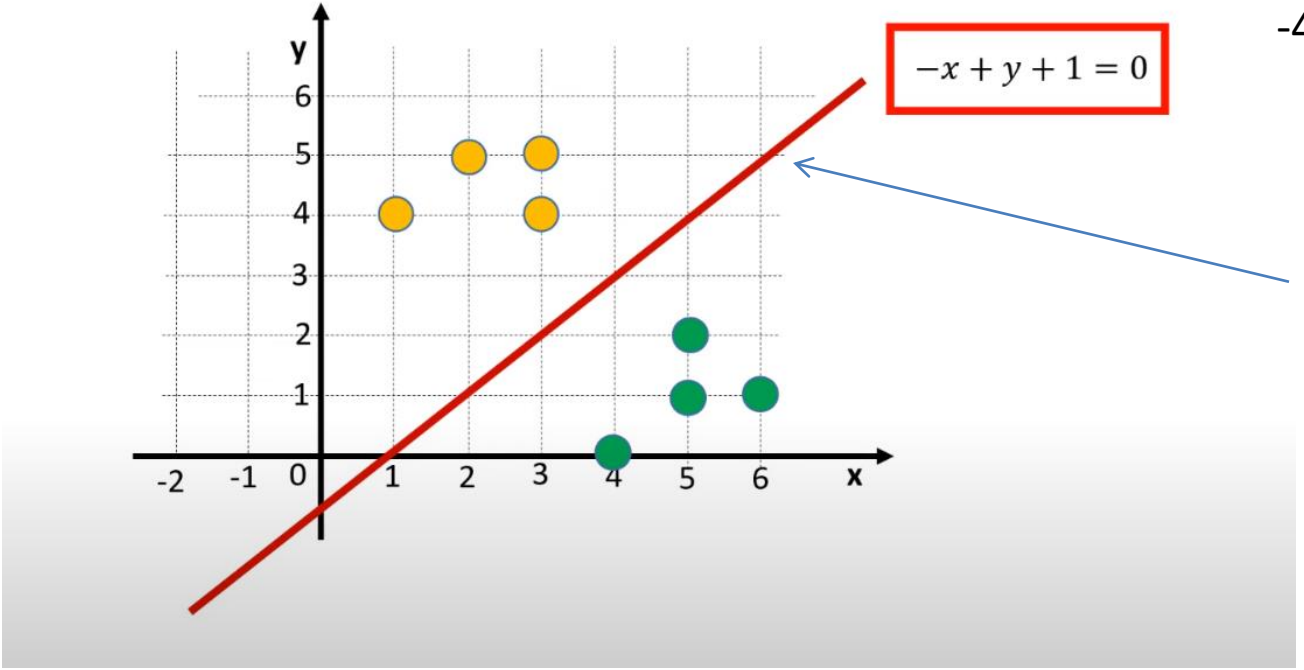
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i \leq C,$$

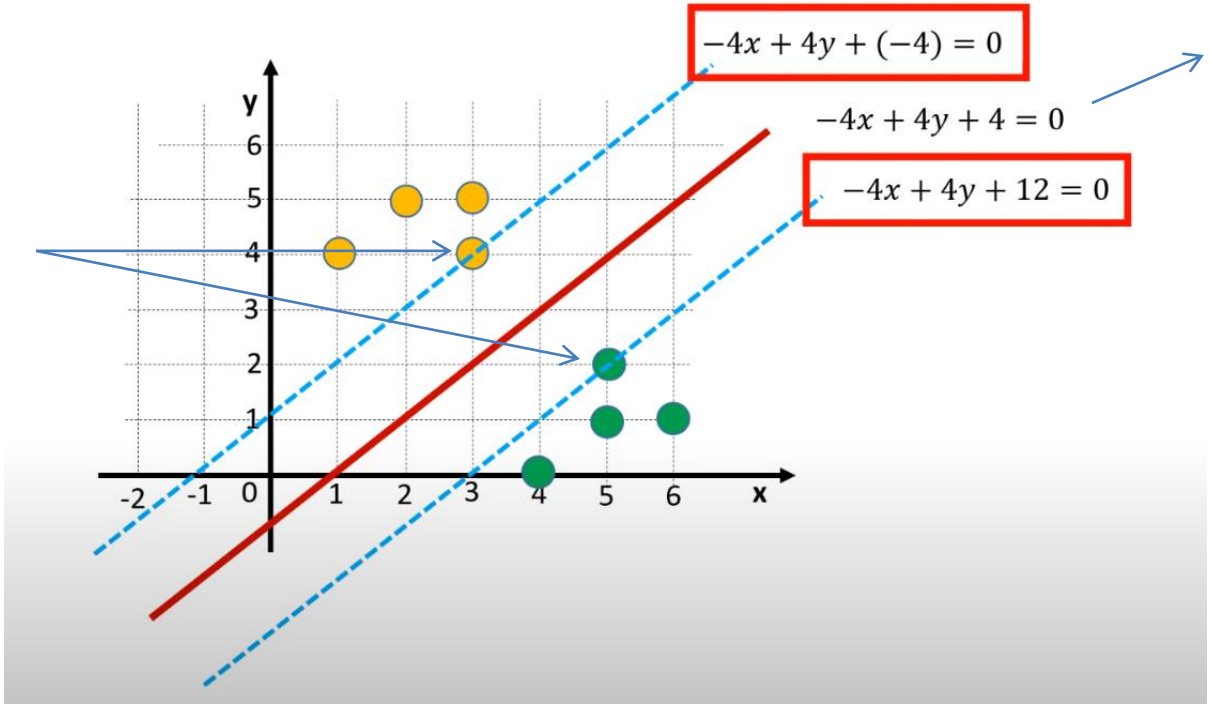
C is a regularization parameter




y
6
5
4
3
2
1
-2
-1
0
1
2
3
4
5
6
x


y
6
5
4
3
2
1
-2
-1
0
1
2
3
4
5
6
x
$-x + y + 1 = 0$
-4x+4y+4=0
Hyperplane

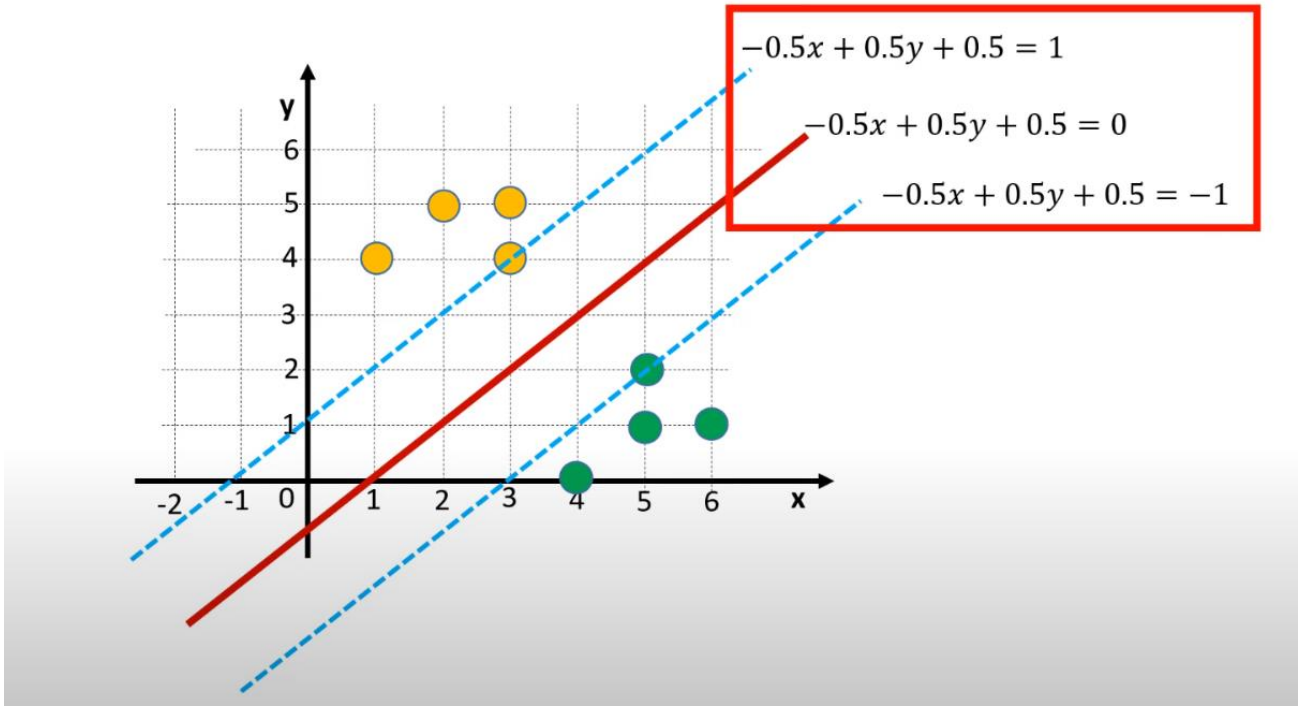
Support vectors

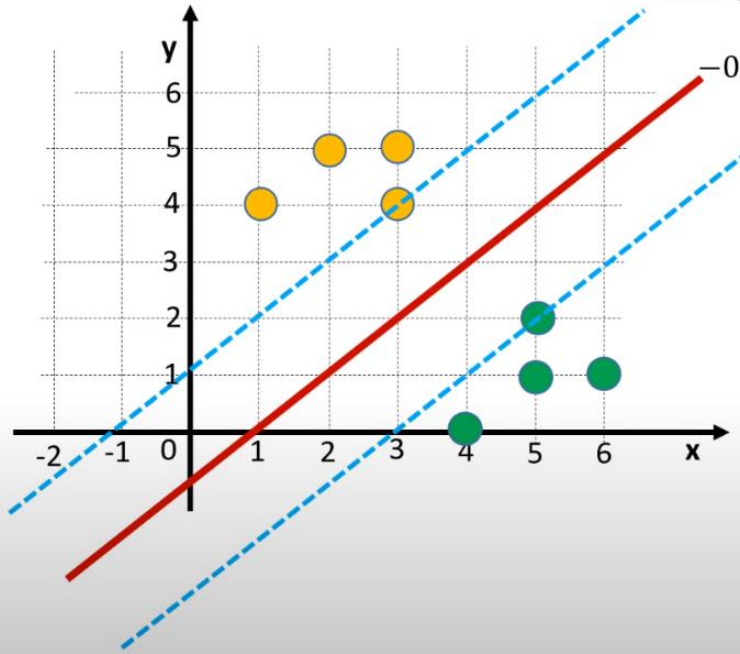


We need to normalize the hyperplane equation such that the left and right parallel lines are equal to +1 and -1 respectively

Need to find the value of k such that k(-4x+4y+4)=1

Plug in the values of support vector (3,4), k=1/8


$-0.5x + 0.5y + 0.5 = 1$
$-0.5x + 0.5y + 0.5 = 0$
$-0.5x + 0.5y + 0.5 = -1$
y
x
-2
-1
0
1
2
3
4
5
6



$-0.5x + 0.5y + b = 1$

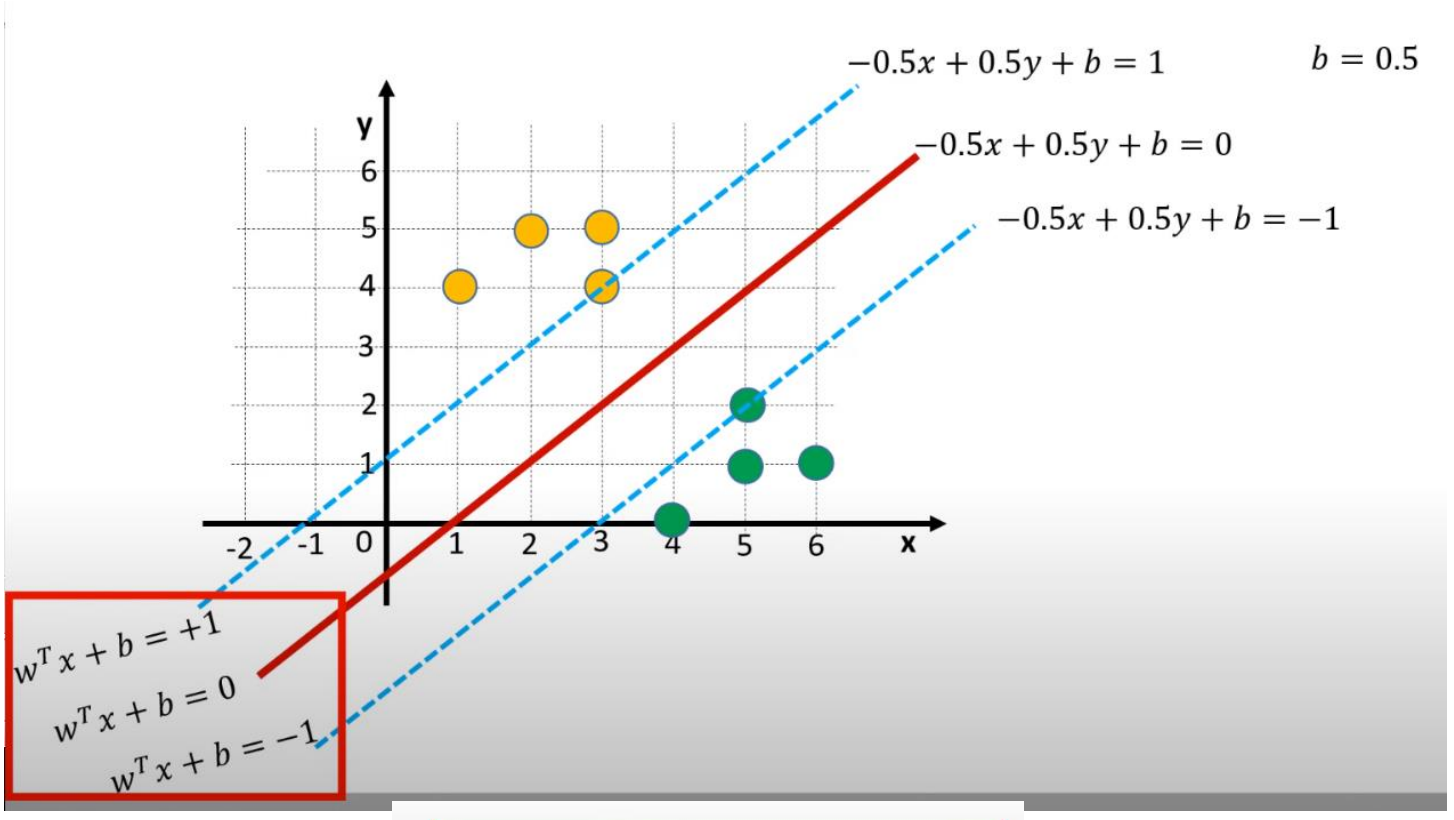
$b = 0.5$

$-0.5x + 0.5y + b = 0$

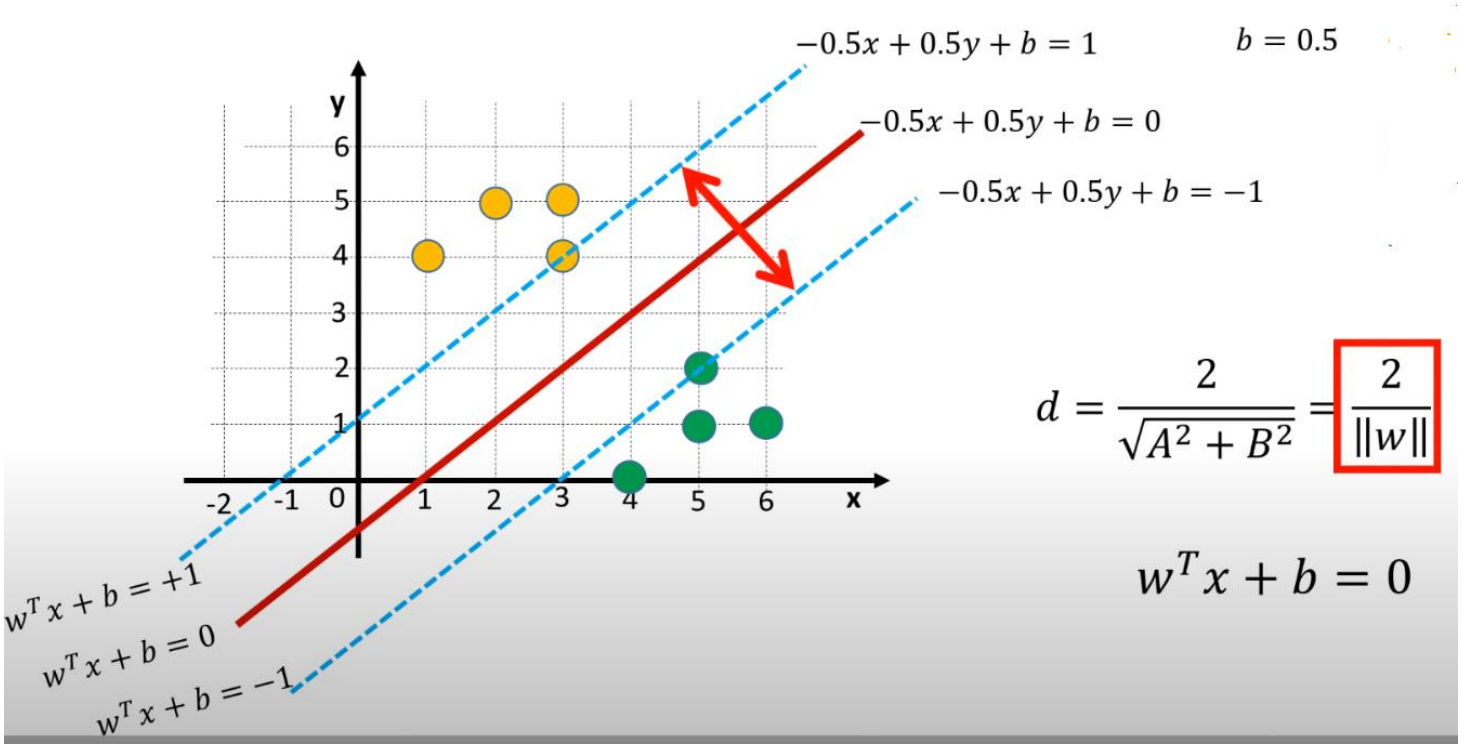
$-0.5x + 0.5y + b = -1$

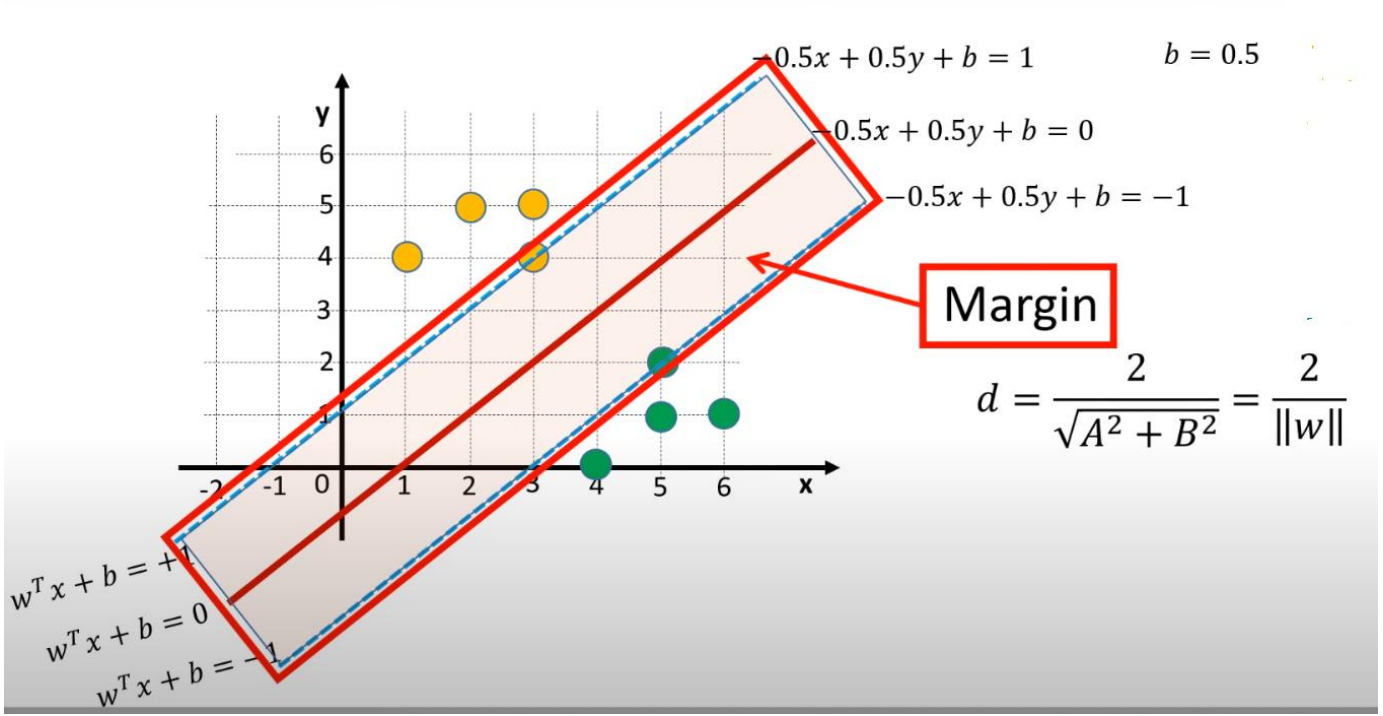
$$w^T x + b = 0$$

$$[-0.5 \quad 0.5] \cdot \begin{bmatrix} x \\ y \end{bmatrix} + b = 0$$

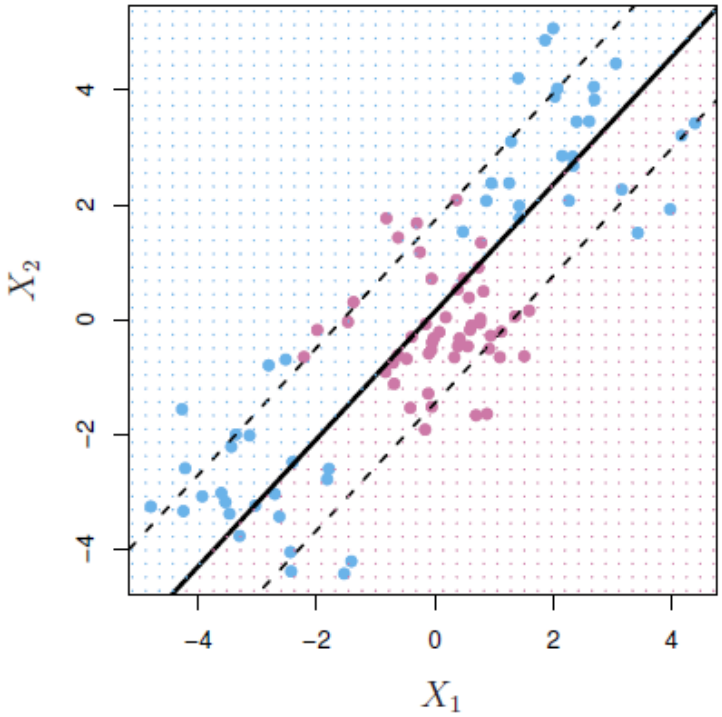


$$Y \begin{cases} +1 \text{ if } w^T x + b \geq 0 \\ -1 \text{ if } w^T x + b < 0 \end{cases}$$


$-0.5x + 0.5y + b = 1$
$b = 0.5$
$-0.5x + 0.5y + b = 0$
$-0.5x + 0.5y + b = -1$
y
x
-2
-1
0
1
2
3
4
5
6
$w^T x + b = +1$
$w^T x + b = 0$
$w^T x + b = -1$
$d = \frac{2}{\sqrt{A^2 + B^2}} = \frac{2}{\|w\|}$
$w^T x + b = 0$


$-0.5x + 0.5y + b = 1$
$b = 0.5$
$-0.5x + 0.5y + b = 0$
$-0.5x + 0.5y + b = -1$
Margin
$d = \frac{2}{\sqrt{A^2 + B^2}} = \frac{2}{\|w\|}$
y
x
$w^T x + b = +1$
$w^T x + b = 0$
$w^T x + b = -1$

Linear boundary can fail



Sometime a linear boundary simply won't work, no matter what value of C.

The example on the left is such a case.

What to do?

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

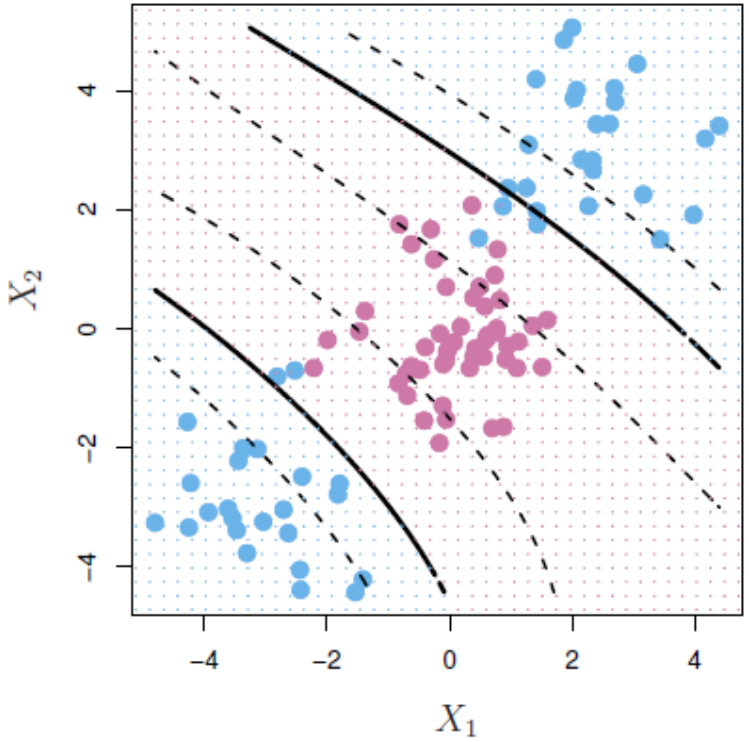
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Cubic Polynomials

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space



$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2=0$$

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

$$\text{subject to } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i),$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1.$$

Nonlinearities and Kernels

- Polynomials (especially high-dimensional ones) get wild rather fast.
- There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers — through the use of *kernels*.
- Before we discuss these, we must understand the role of *inner products* in support-vector classifiers.

Inner products and support vectors

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \text{— } \textit{inner product between vectors}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \quad \text{— } n \textit{ parameters}$$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

$\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$.

- If we can compute inner-products between observations, we can fit a SV classifier. Can be quite abstract!
- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

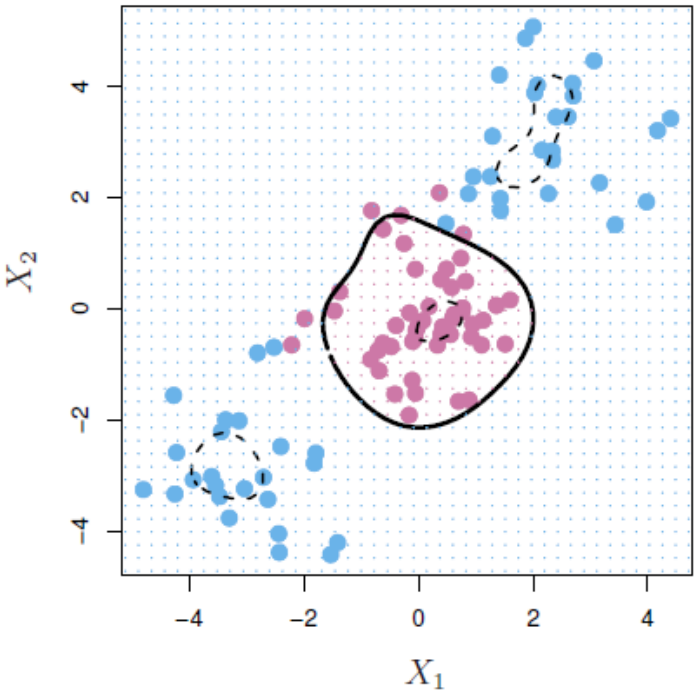
Try it for $p = 2$ and $d = 2$.

- The solution has the form

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

Radial Kernel

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

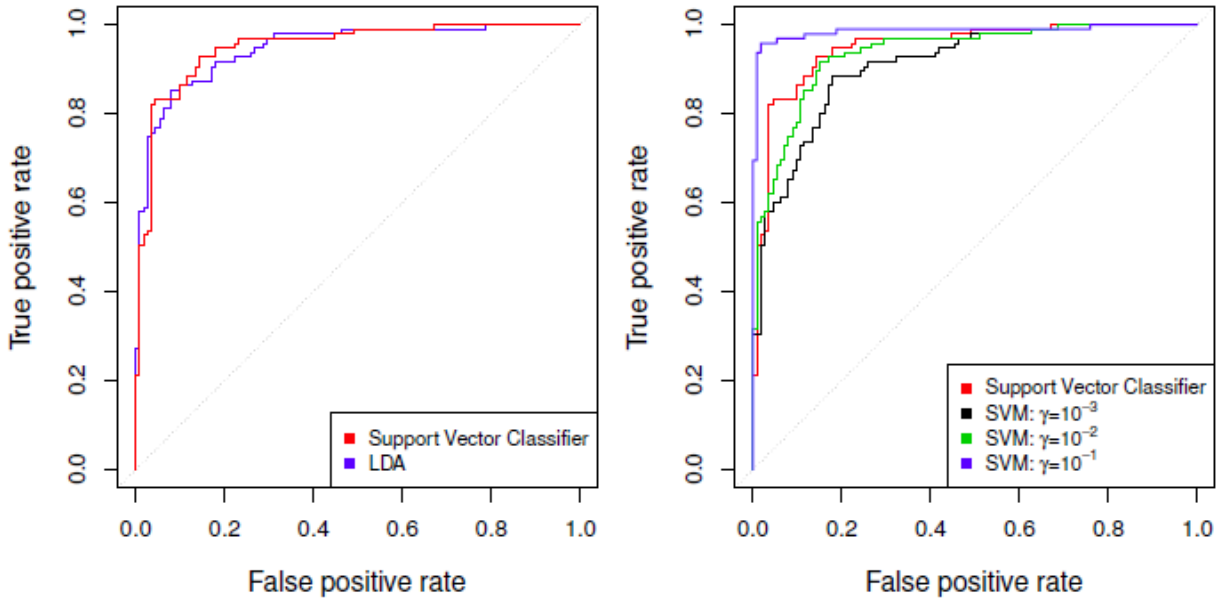


$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

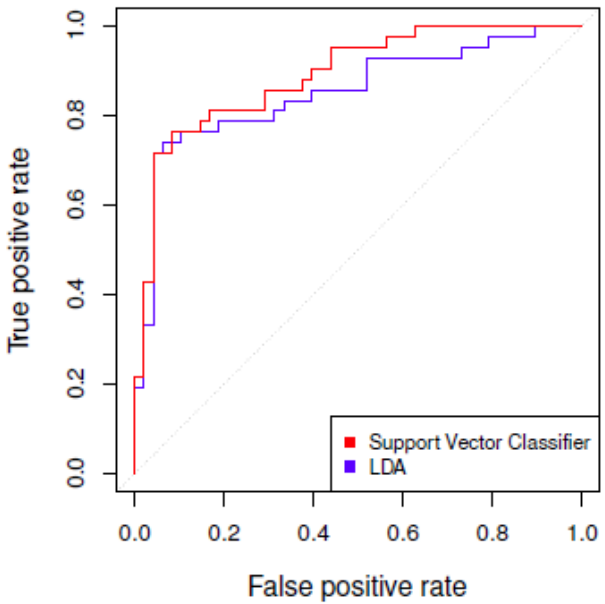
Controls variance by squashing down most dimensions severely

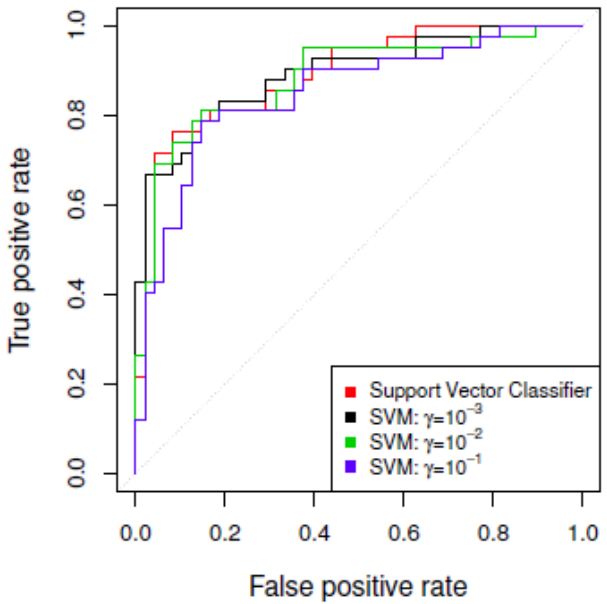
Example: Heart Data



ROC curve is obtained by changing the threshold 0 to threshold t in $\hat{f}(X) > t$, and recording *false positive* and *true positive* rates as t varies. Here we see ROC curves on training data.

Example continued: Heart Test Data





SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.