TREE-BASED METHODS

Minería de Datos: Preprocesamiento y clasificación

Descripción

- **Temario**:
  - Modelos no lineales.
  - Árboles de Decisión. Multiclasificadores.
  - Descomposición de problemas multiclase.
  - Aprendizaje de Reglas.
  - Máquinas soporte vectorial (SVM).
  - Preprocesamiento de Datos.

- **Bibliografía**:
  - "An Introduction to Statistical Learning with Applications in R", Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, Springer, 2013
  - "Introduction to Data Mining", Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Pearson, 2013 .
  - "Foundations of Rule Learning", Johannes Fürnkranz, Dragan Gambergerm Nada Lavrac, Springer, 2012.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera, Springer, 2015.

- **Relacionado con**: Introducción a la Programación para Ciencia de Datos e Introducción a la Ciencia de Datos

Descripción

- **Temario**:
  - Modelos no lineales.
  - Árboles de Decisión. Multiclasificadores.
  - Descomposición de problemas multiclase.
  - Aprendizaje de Reglas.
  - Máquinas soporte vectorial (SVM).
  - Preprocesamiento de Datos.

- **Bibliografía**:
  - "An Introduction to Statistical Learning with Applications in R", Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, Springer, 2013.
  - "Introduction to Data Mining", Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Pearson, 2013.
  - "Foundations of Rule Learning", Johannes Fürnkranz, Dragan Gambergerm Nada Lavrac, Springer, 2012.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera, Springer, 2015.

- **Relacionado con**: Introducción a la Programación para Ciencia de Datos e Introducción a la Ciencia de Datos

Tree-based Methods

- Here we describe tree-based methods for classification (and regression).

- These involve stratifying or segmenting the predictor space into a number of simple regions

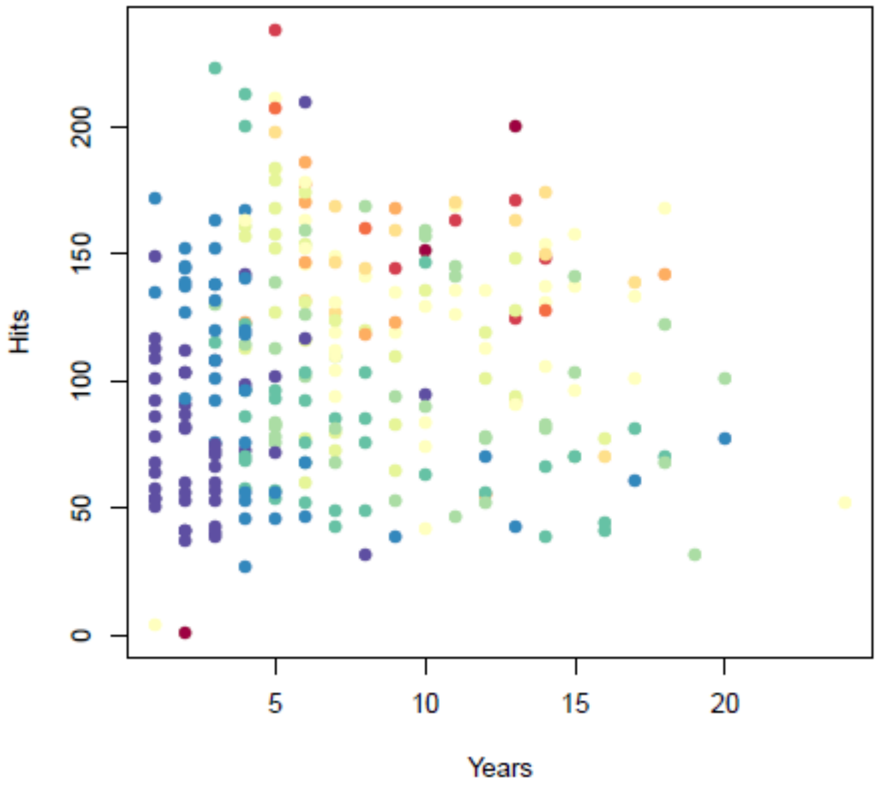
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods

Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss bagging, random forests, and boosting. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

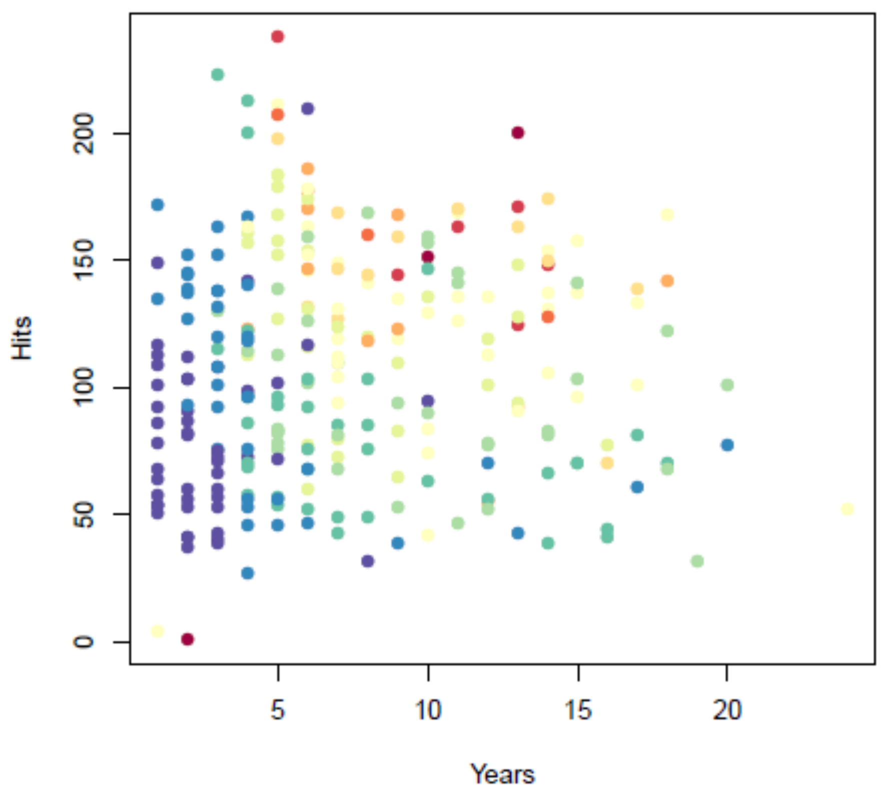
Decision tree for these data

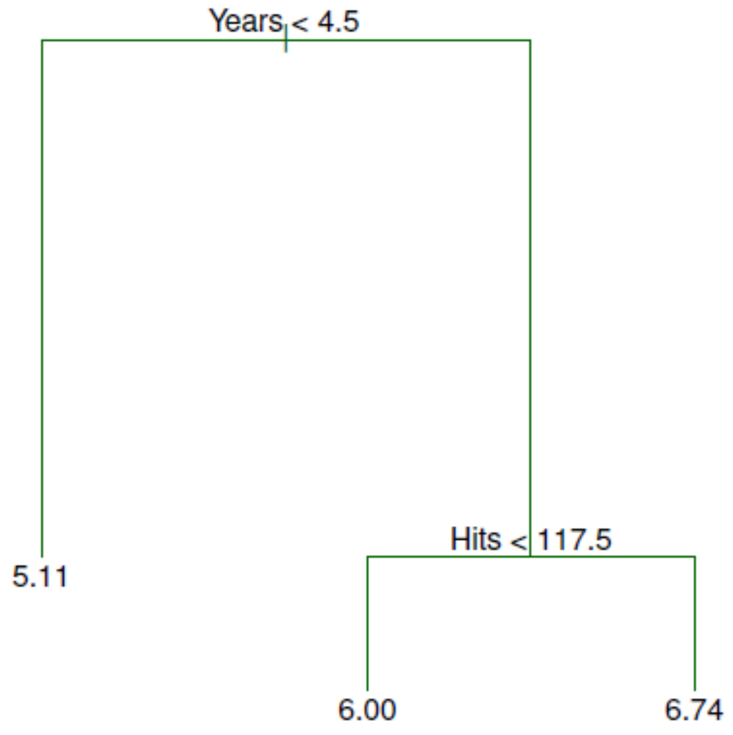
Baseball salary data: how would you stratify it?
Salary is color-coded from low (blue, green) to high (yellow, red)



Decision tree for these data

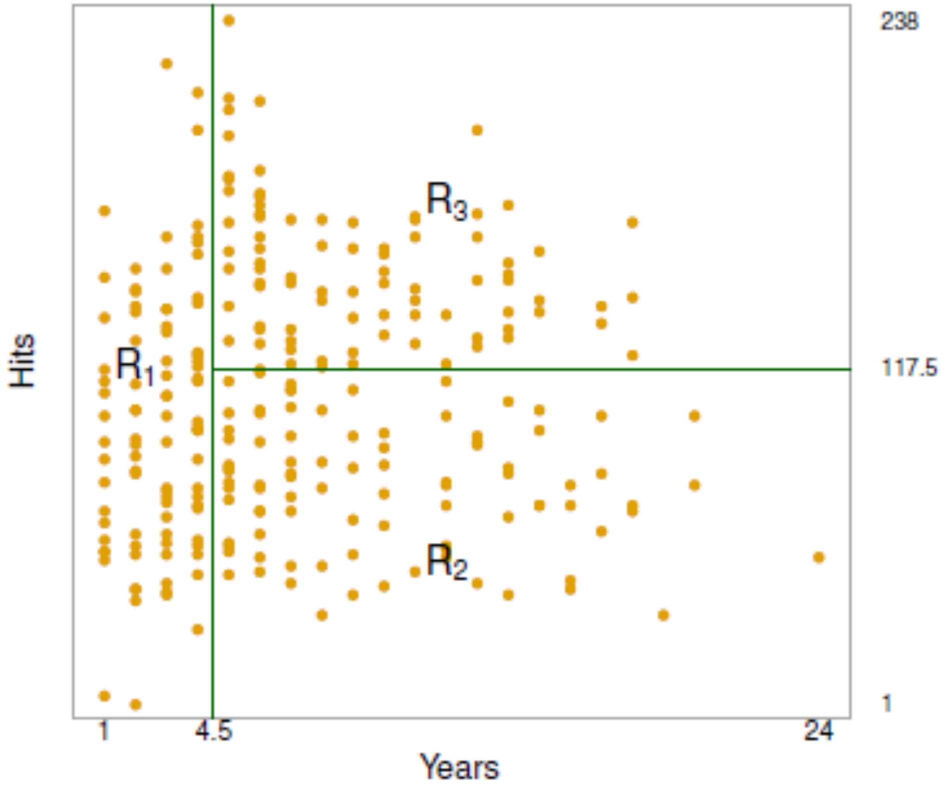
Baseball salary data: how would you stratify it?
Salary is color-coded from low (blue, green) to high (yellow, red)





Results

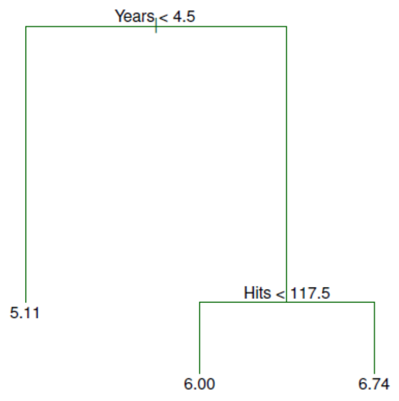
- Overall, the tree stratifies or segments the players into three regions of predictor space: R_1 ={X | Years < 4.5}, R_2 ={X | Years>=4.5, Hits<117.5}, and R_3 ={X | Years>=4.5, Hits>=117.5}.



Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R3 are known as terminal nodes
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes
- In the hitters tree, the two internal nodes are indicated by the text Years<4.5 and Hits<117.5.

Interpretation of Results



- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year does affect Salary, and players who made more Hits last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain.

Details of the tree-building process

- We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
- For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values (alternatively, a linear function has been used) for the training observations in R_j.

Details of the tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.

- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Details of the tree-building process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Details of the tree-building process

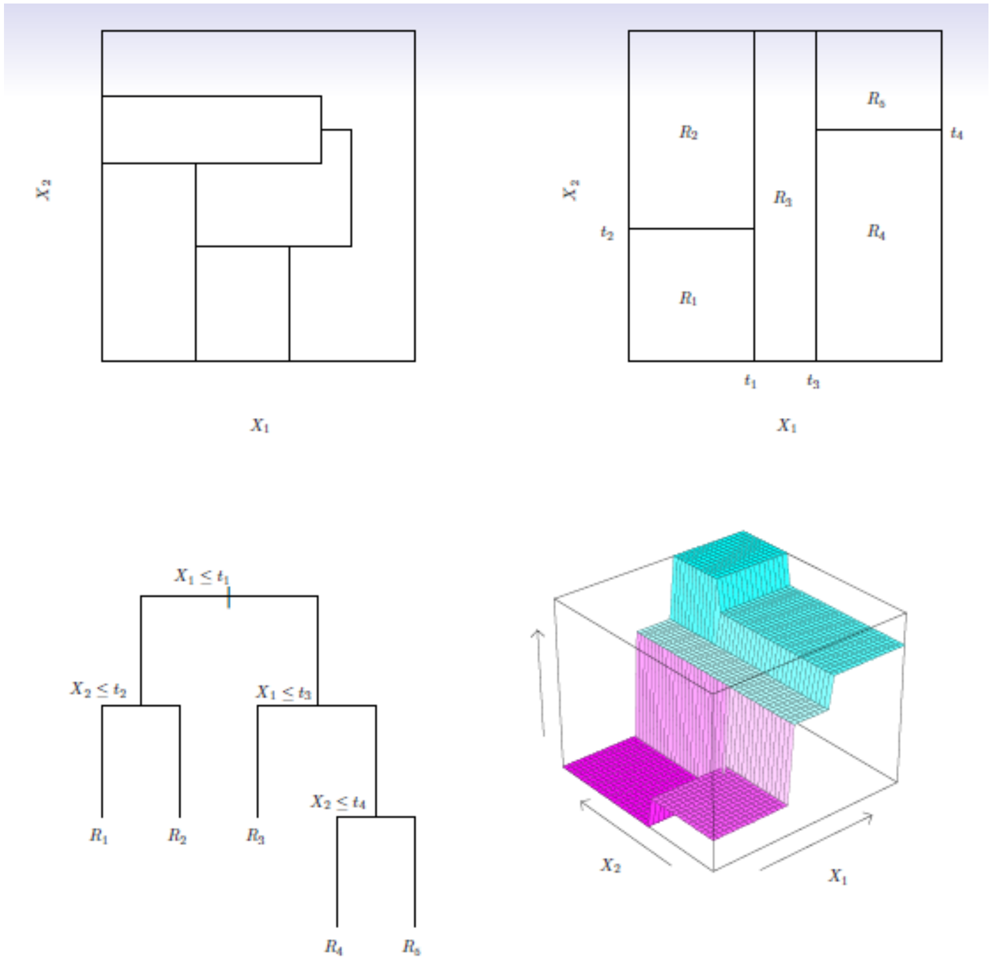
- We first select de predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X | X_j < s\}$ and $\{X | X_j \geq s\}$ leads to the greatest possible reduction in RSS.

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.
- Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Predictions

We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

Details of classification trees

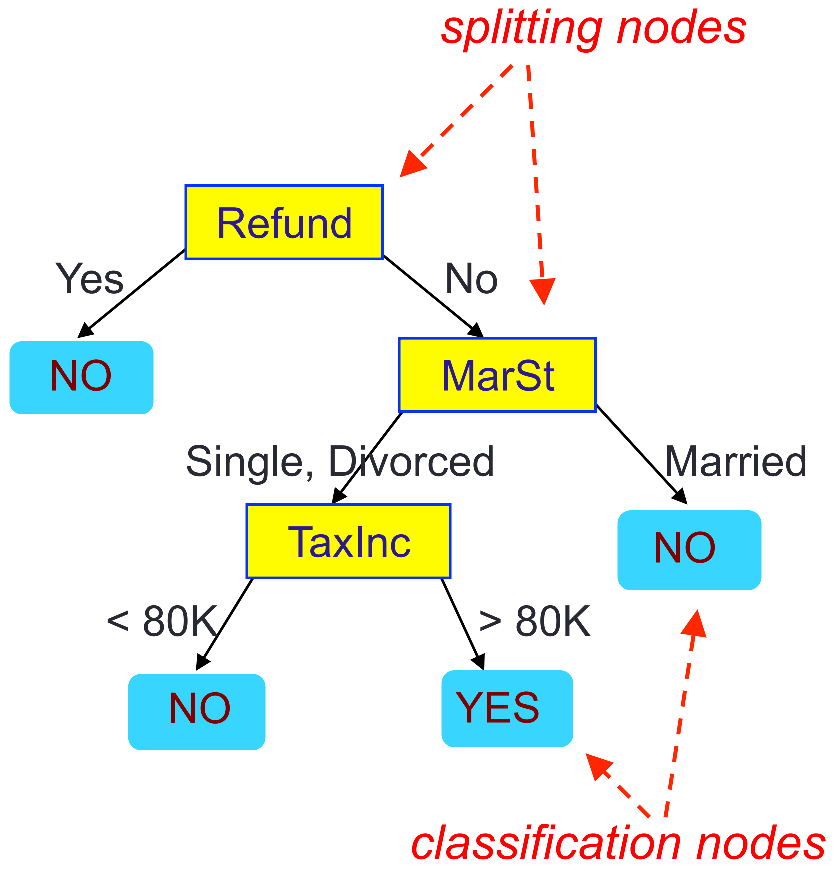
- Just as in the regression setting, we can use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the classification error rate. this is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

- Here $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable (Gini index and Entropy).

Example of a decision tree

	nominal	nominal	ratio	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

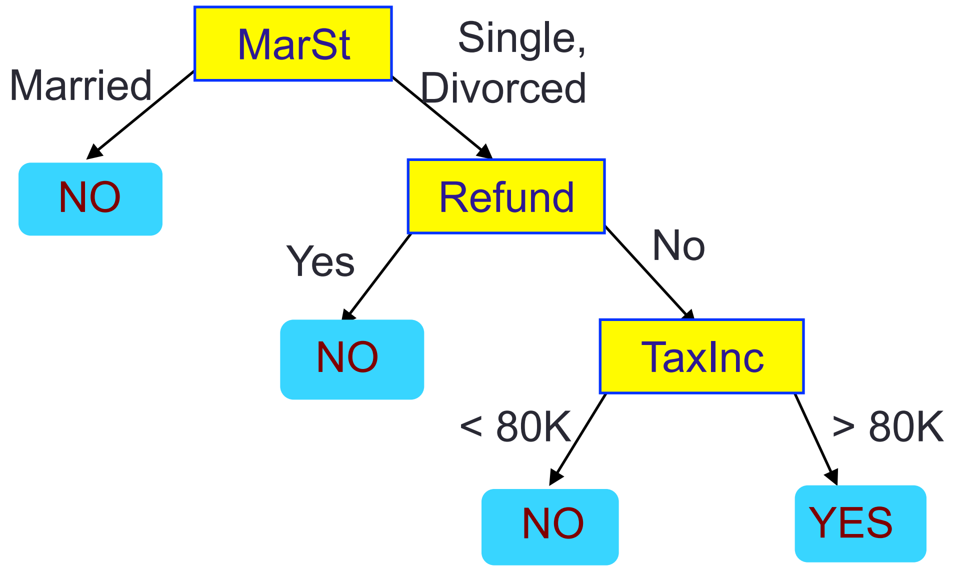


training data

model: decision tree

Example of decision tree

	nominal	nominal	ratio	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**



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

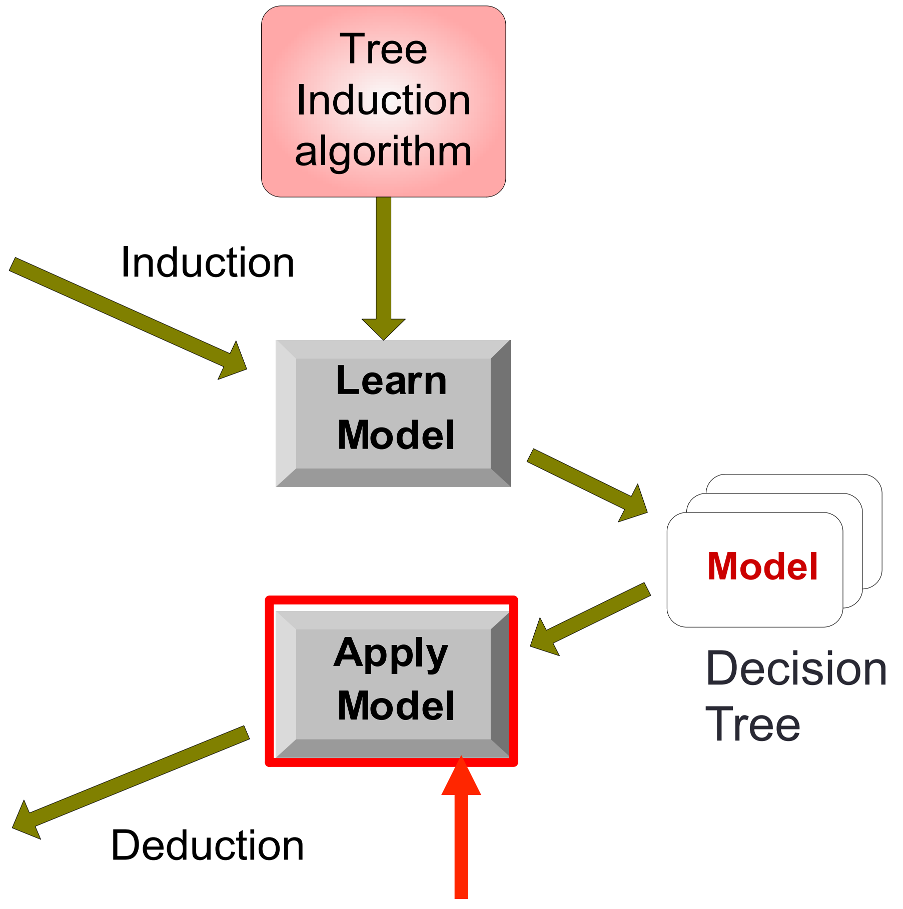
Decision tree classification task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

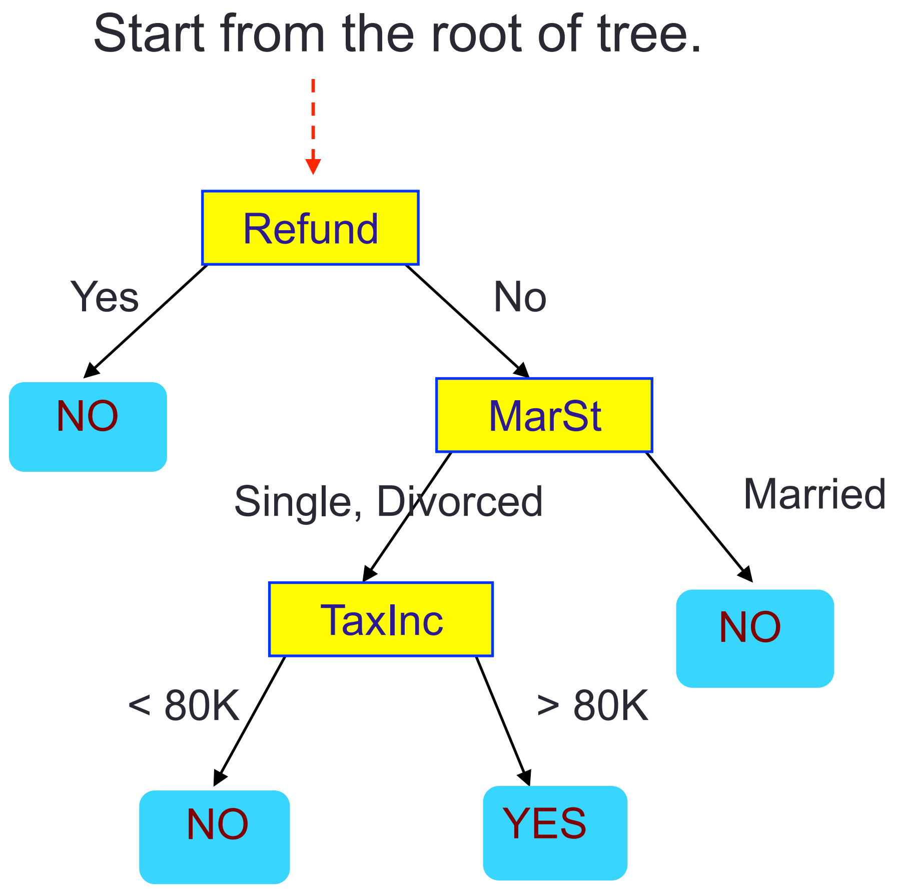
Training Set

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

Test Set



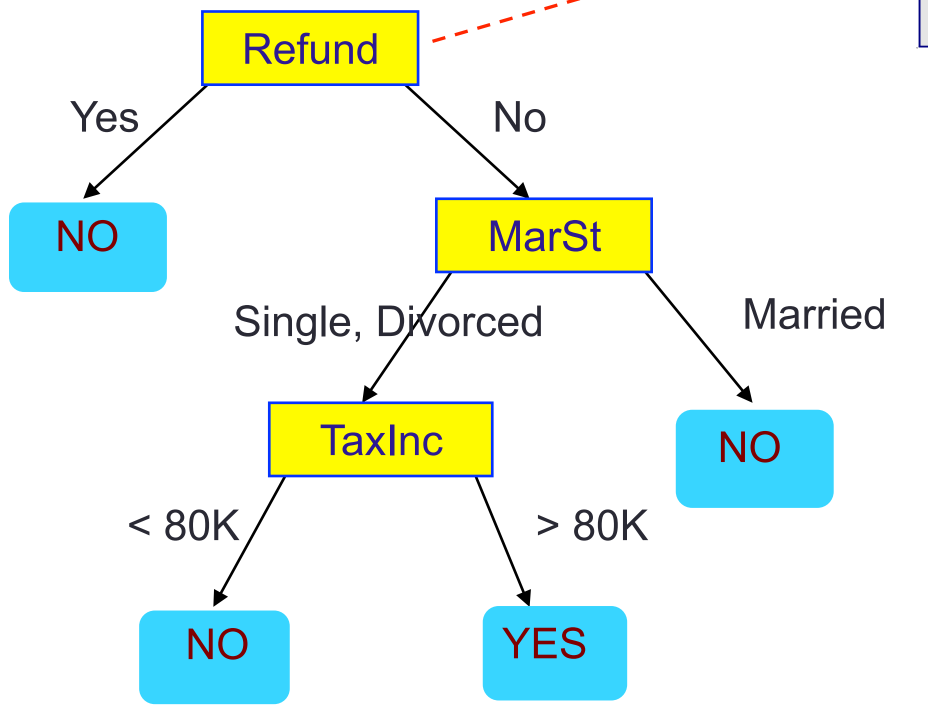
Apply model to test data



Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply model to test data



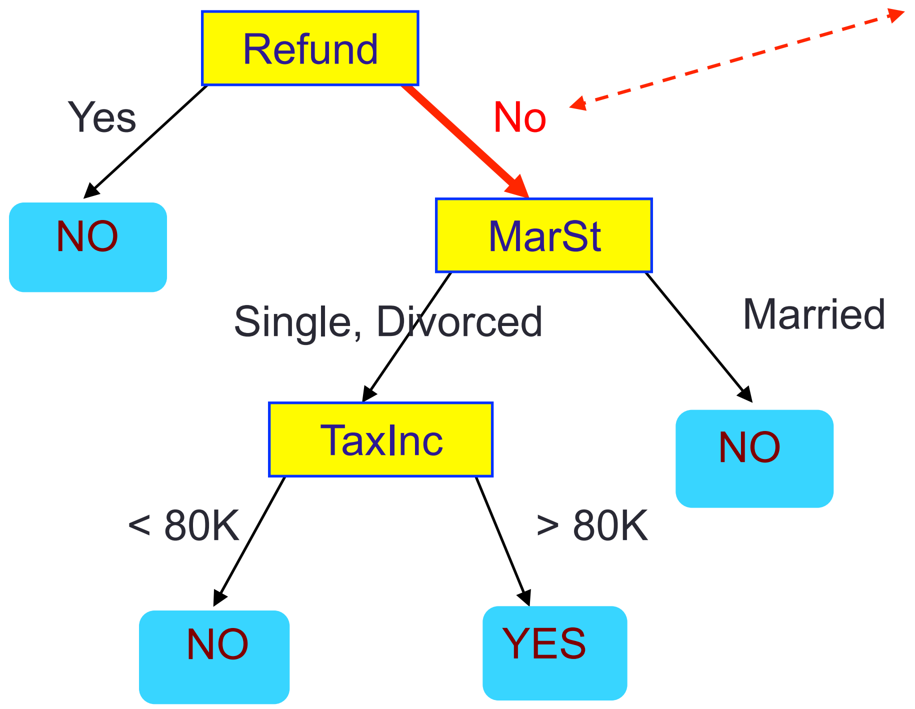
Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply model to test data

Test data

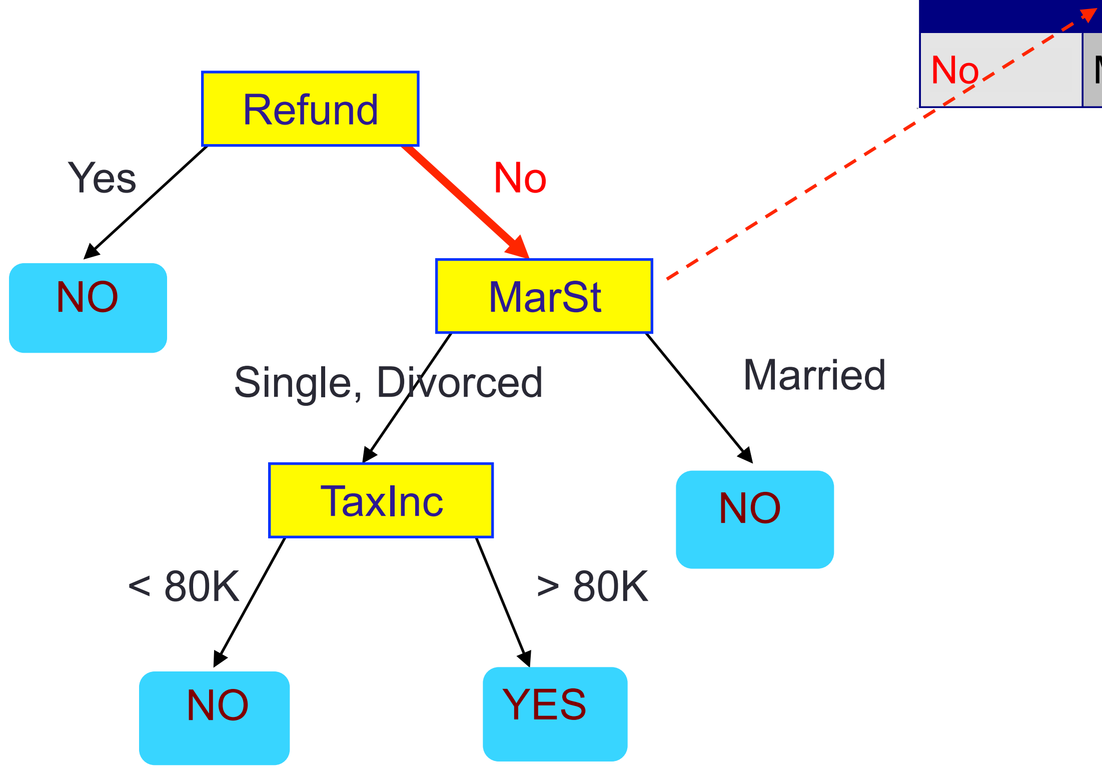
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

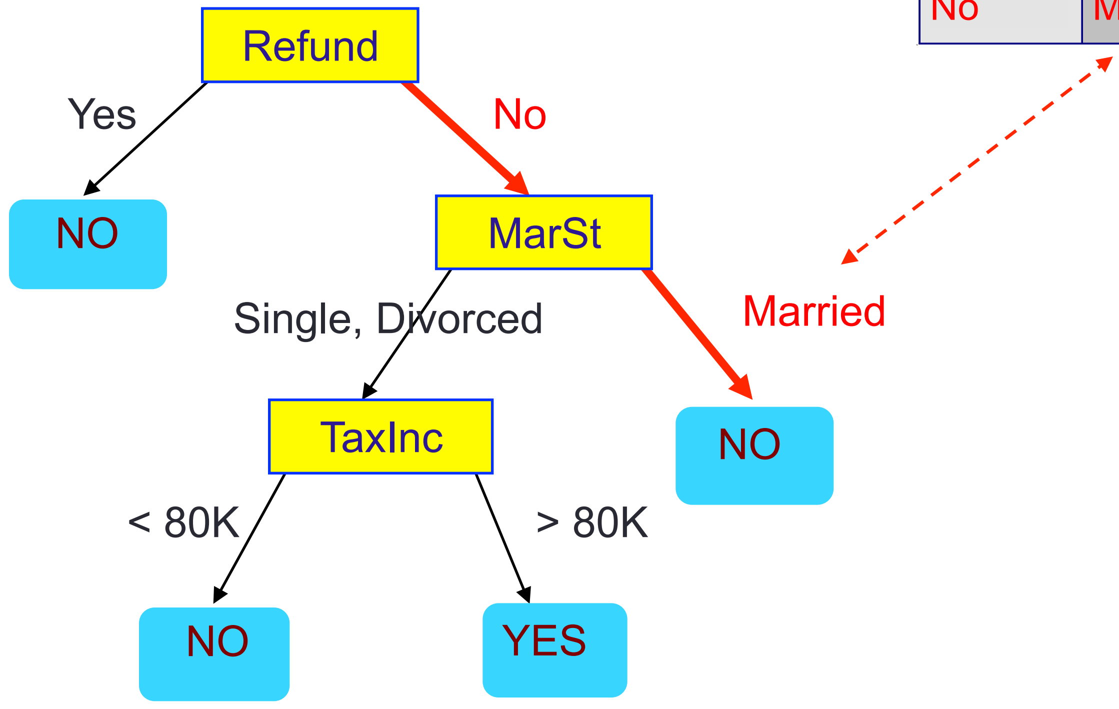
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

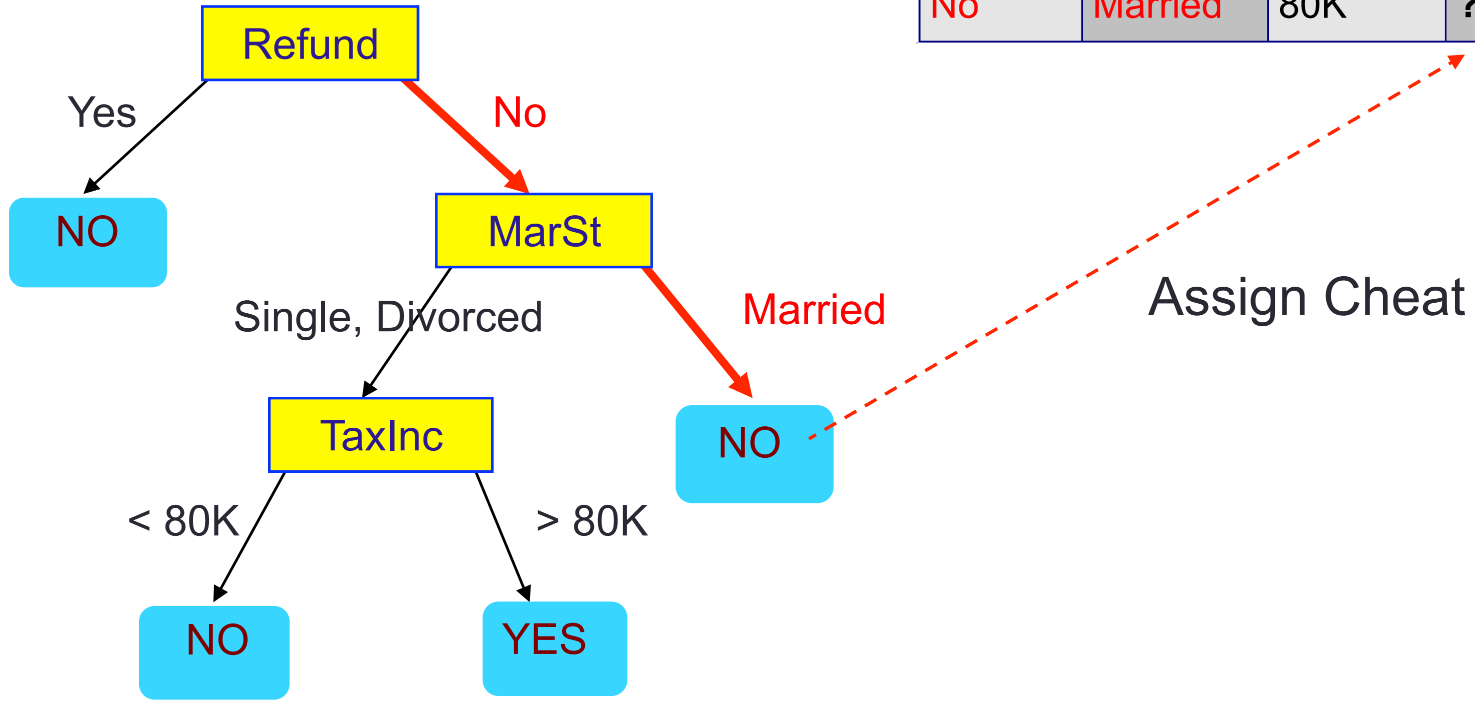
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

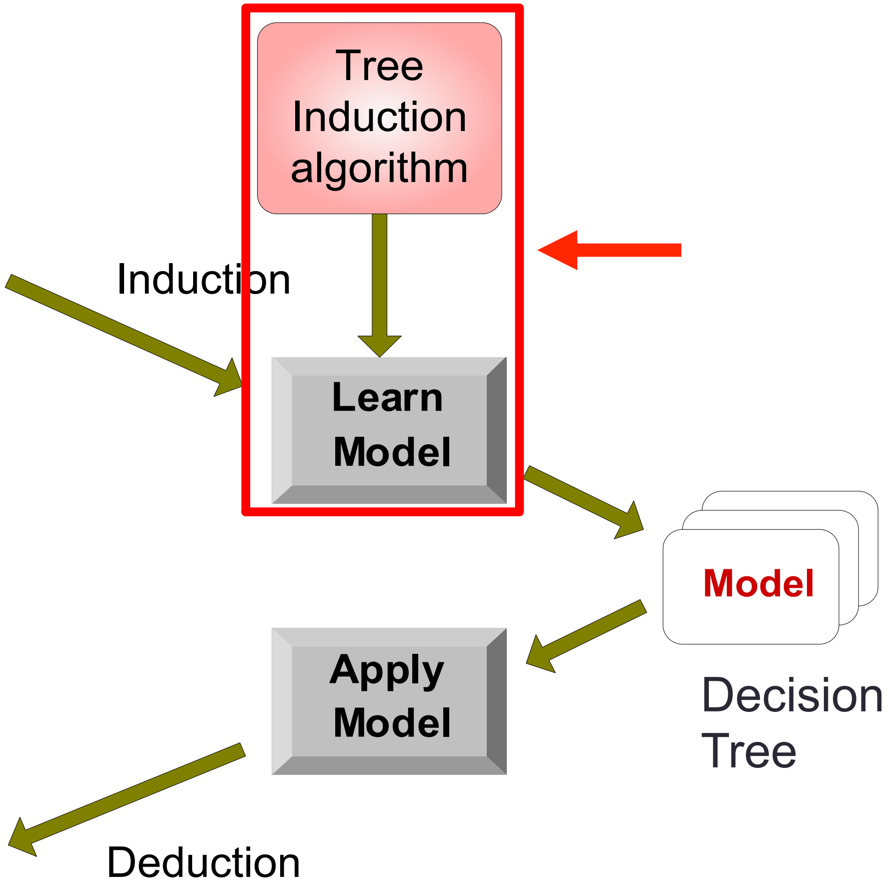


Assign Cheat to “No”

Decision tree classification task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



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

Test Set

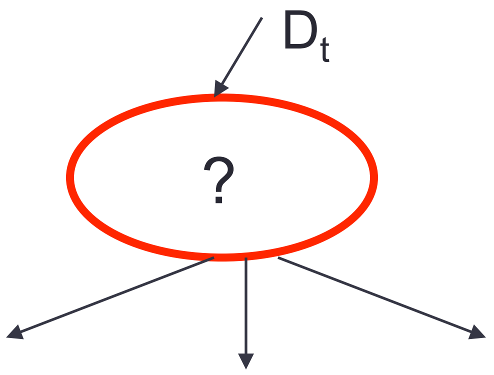
Decision Tree Induction

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

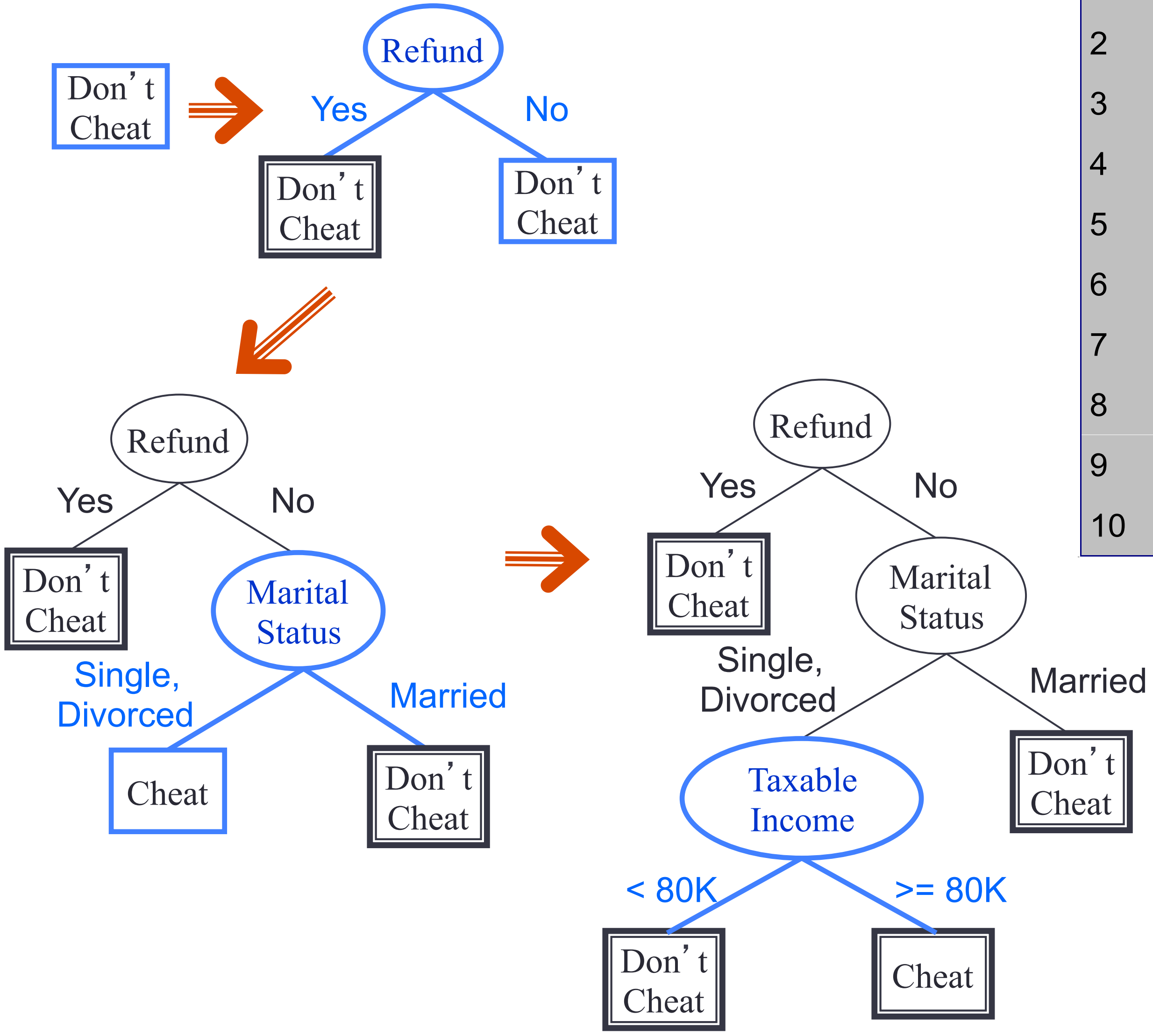
General Structure of Hunt's Algorithm

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

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**



Hunt’s Algorithm



Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.
- Issues
  - Determine how to split the records
    - How to specify structure of split?
    - What is best attribute / attribute value for splitting?
  - Determine when to stop splitting

Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.
- Issues
  - Determine how to split the records
    - How to specify structure of split?
    - What is best attribute / attribute value for splitting?
  - Determine when to stop splitting

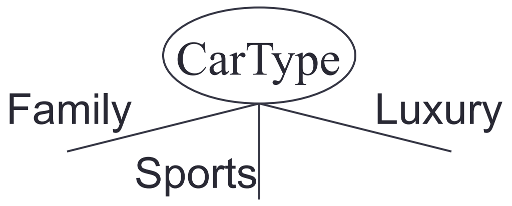
Specifying structure of split

- Depends on attribute type
  - Nominal
  - Ordinal
  - Continuous

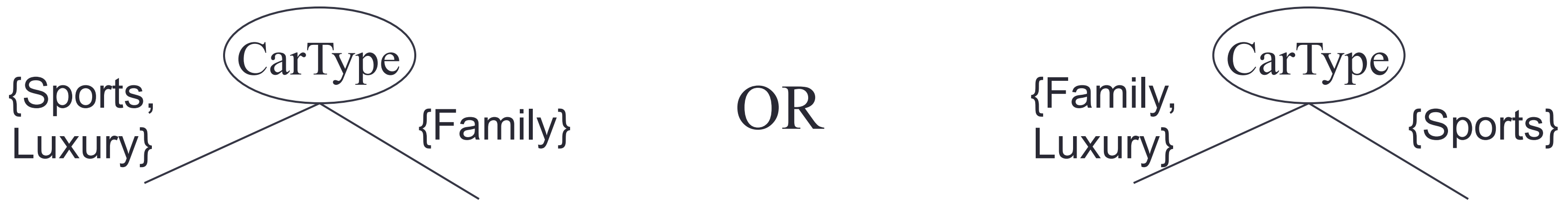
- Depends on number of ways to split
  - Binary (two-way) split
  - Multi-way split

Splitting based on nominal attributes

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

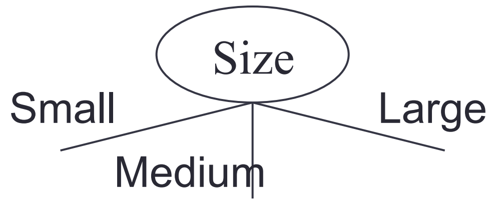


- Binary split: Divides values into two subsets.
  Need to find optimal partitioning.

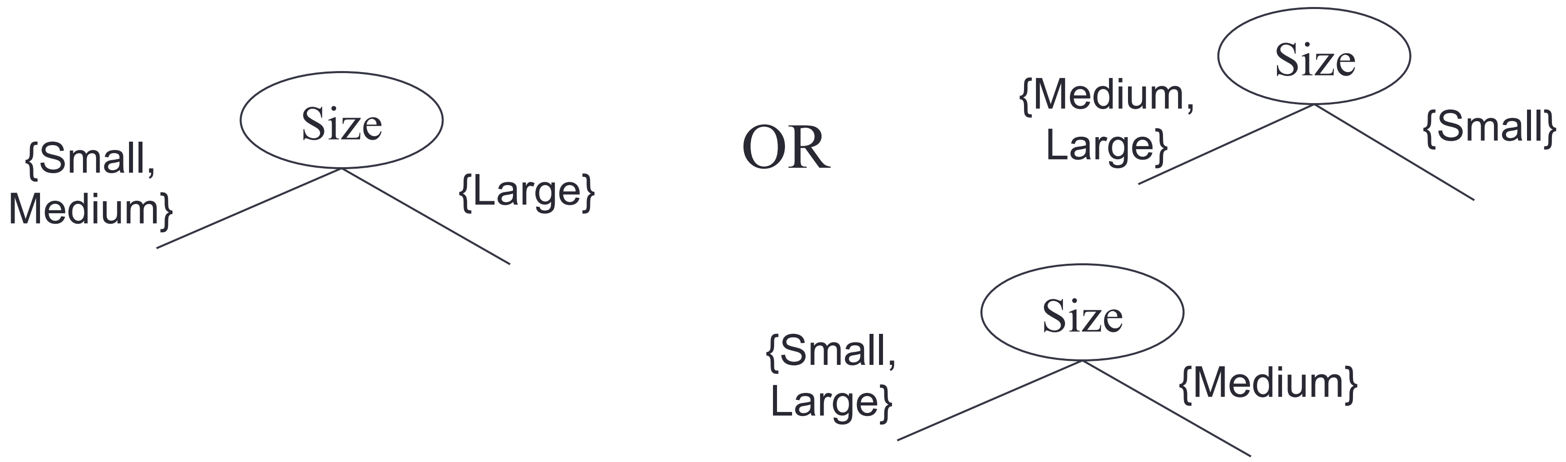


Splitting based on ordinal attributes

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



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

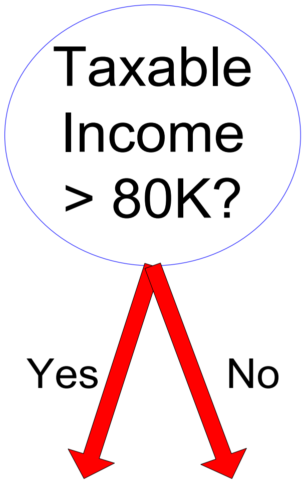


Splitting based on continuous attributes

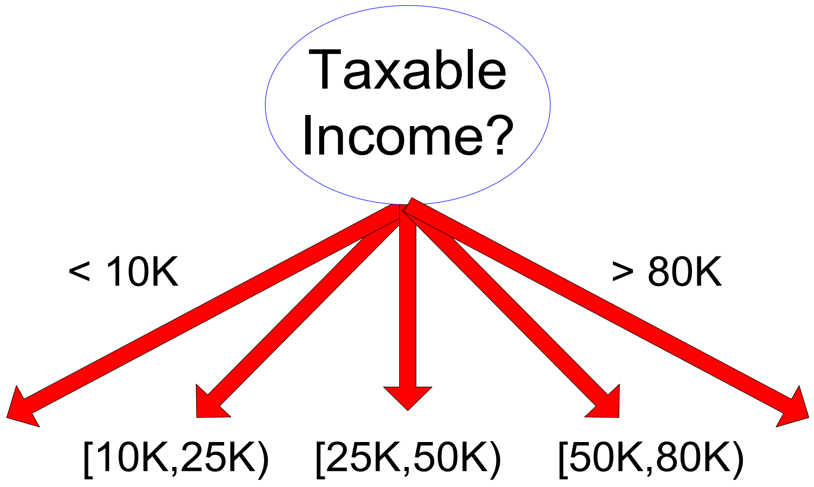
- Different ways of handling
  - Discretization to form an ordinal attribute
    - static – discretize once at the beginning
    - dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Threshold decision: $(A < v)$ or $(A \geq v)$
    - consider all possible split points v and find the one that gives the best split
    - can be more compute intensive

Splitting based on continuous attributes

- Splitting based on threshold decision



(i) Binary split



(ii) Multi-way split

Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.

- Issues
  - Determine how to split the records
    - How to specify structure of split?
    - What is best attribute / attribute value for splitting?
  - Determine when to stop splitting

Ejemplo: Problema de asignación de crédito

crédito	ingresos	propietario	Gastos-mensuales
N	Bajos	N	Altos
N	Bajos	S	Altos
N	Medios	S	Altos
N	Medios	N	Altos
N	Altos	N	Altos
S	Altos	S	Altos
N	Bajos	N	Bajos
N	Medios	N	Bajos
N	Altos	N	Bajos
S	Medios	S	Bajos

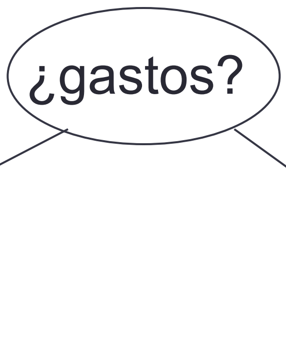
Ejemplo: Problema de asignación de crédito

Se llama al algoritmo sobre el nodo raíz

crédito	ingresos	propietario	Gastos-mensuales
N	Bajos	N	Altos
N	Bajos	S	Altos
N	Medios	S	Altos
N	Medios	N	Altos
N	Altos	N	Altos
S	Altos	S	Altos
N	Bajos	N	Bajos
N	Medios	N	Bajos
N	Altos	N	Bajos
S	Medios	S	Bajos

Ejemplo: Problema de asignación de crédito

Seleccionamos gastos como test

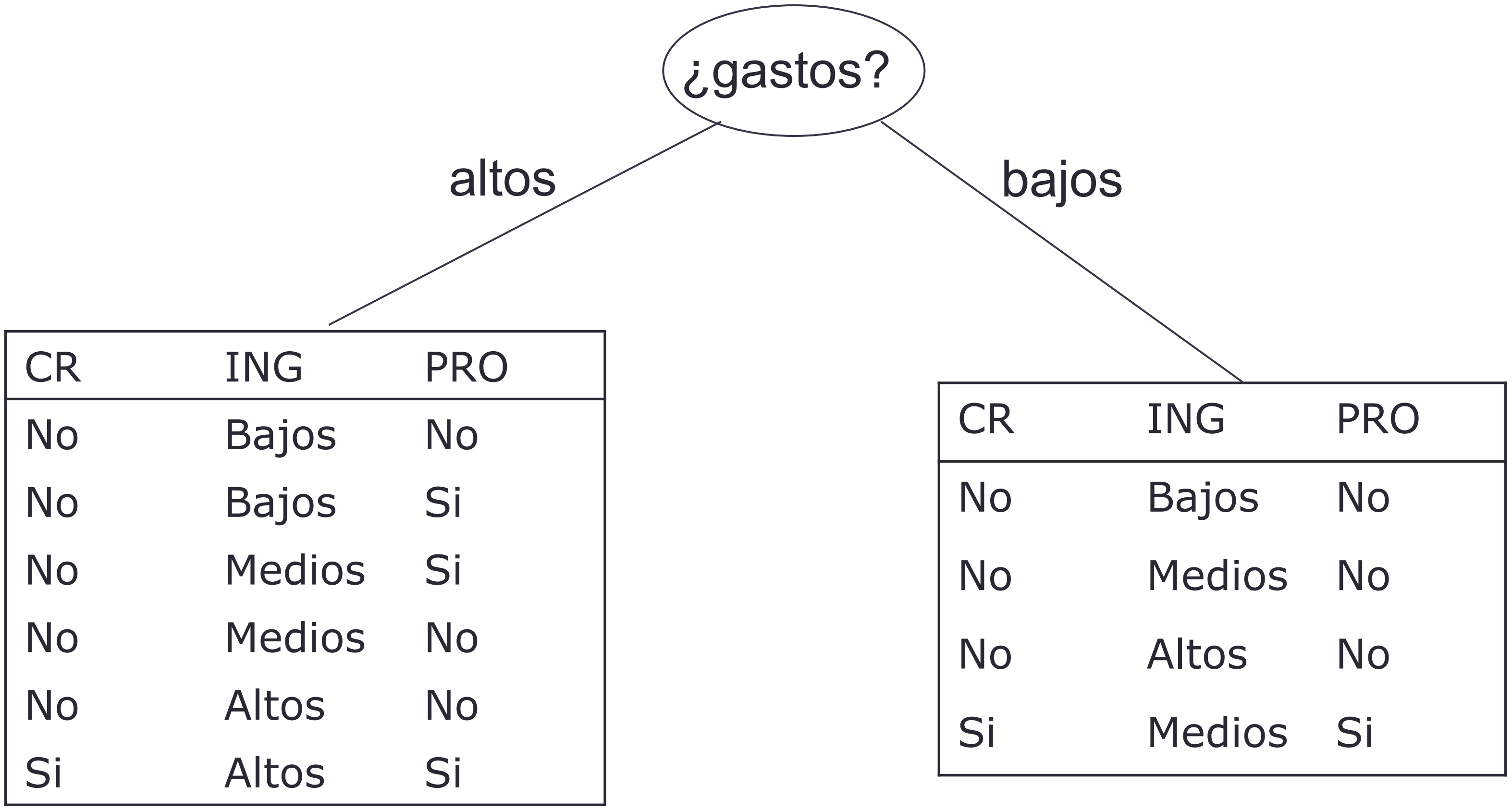


CR	ING	PRO	gastos
No	Bajos	No	Altos
No	Bajos	Si	Altos
No	Medios	Si	Altos
No	Medios	No	Altos
No	Altos	No	Altos
Si	Altos	Si	Altos

CR	ING	PRO	gastos
No	Bajos	No	Bajos
No	Medios	No	Bajos
No	Altos	No	Bajos
Si	Medios	Si	Bajos

Ejemplo: Problema de asignación de crédito

Preparamos los nodos para las llamadas recursivas

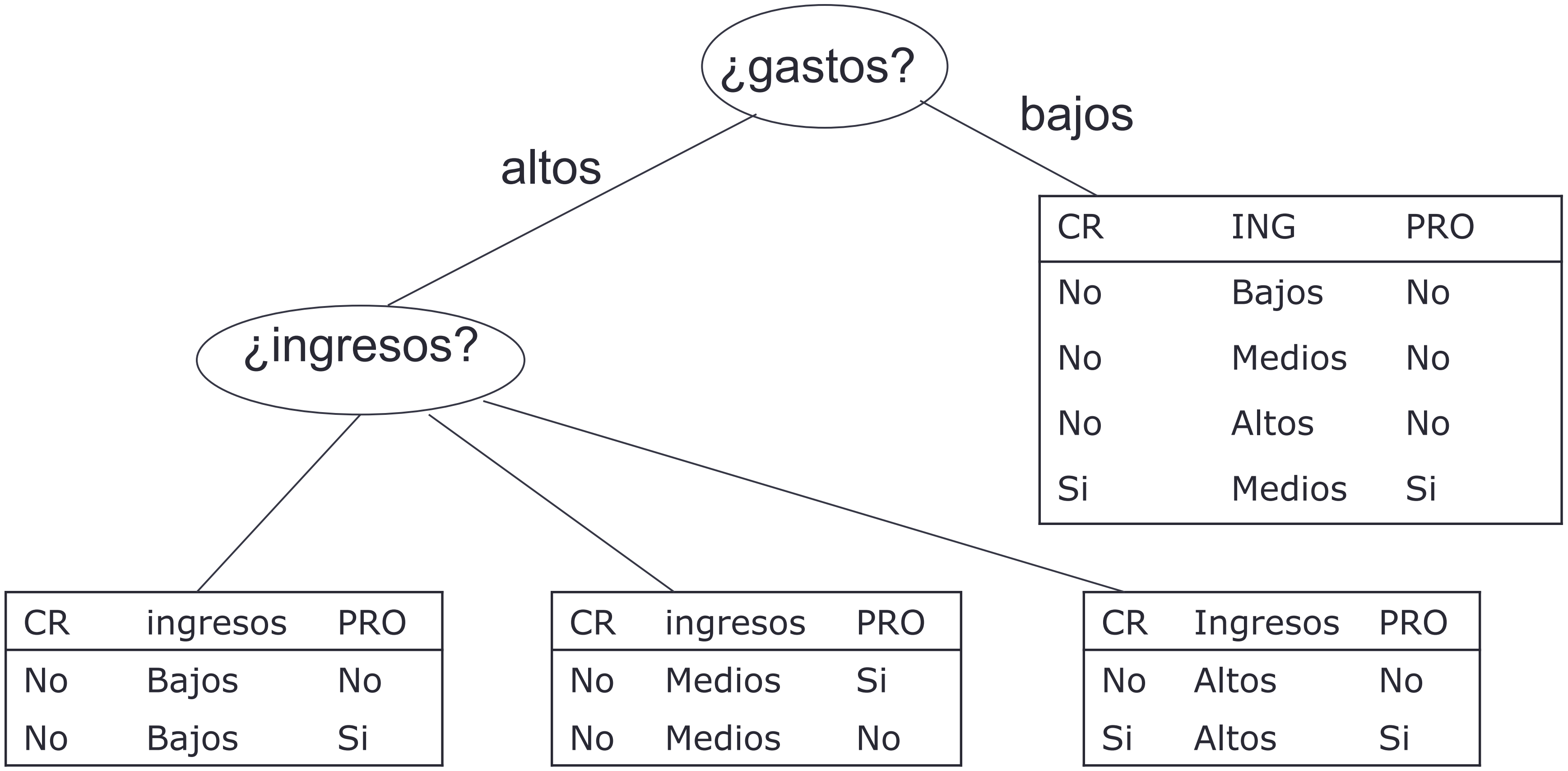


CR	ING	PRO
No	Bajos	No
No	Bajos	Si
No	Medios	Si
No	Medios	No
No	Altos	No
Si	Altos	Si

CR	ING	PRO
No	Bajos	No
No	Medios	No
No	Altos	No
Si	Medios	Si

Ejemplo: Problema de asignación de crédito

Seleccionamos ingresos como test en gastos = altos



CR	ING	PRO
No	Bajos	No
No	Medios	No
No	Altos	No
Si	Medios	Si

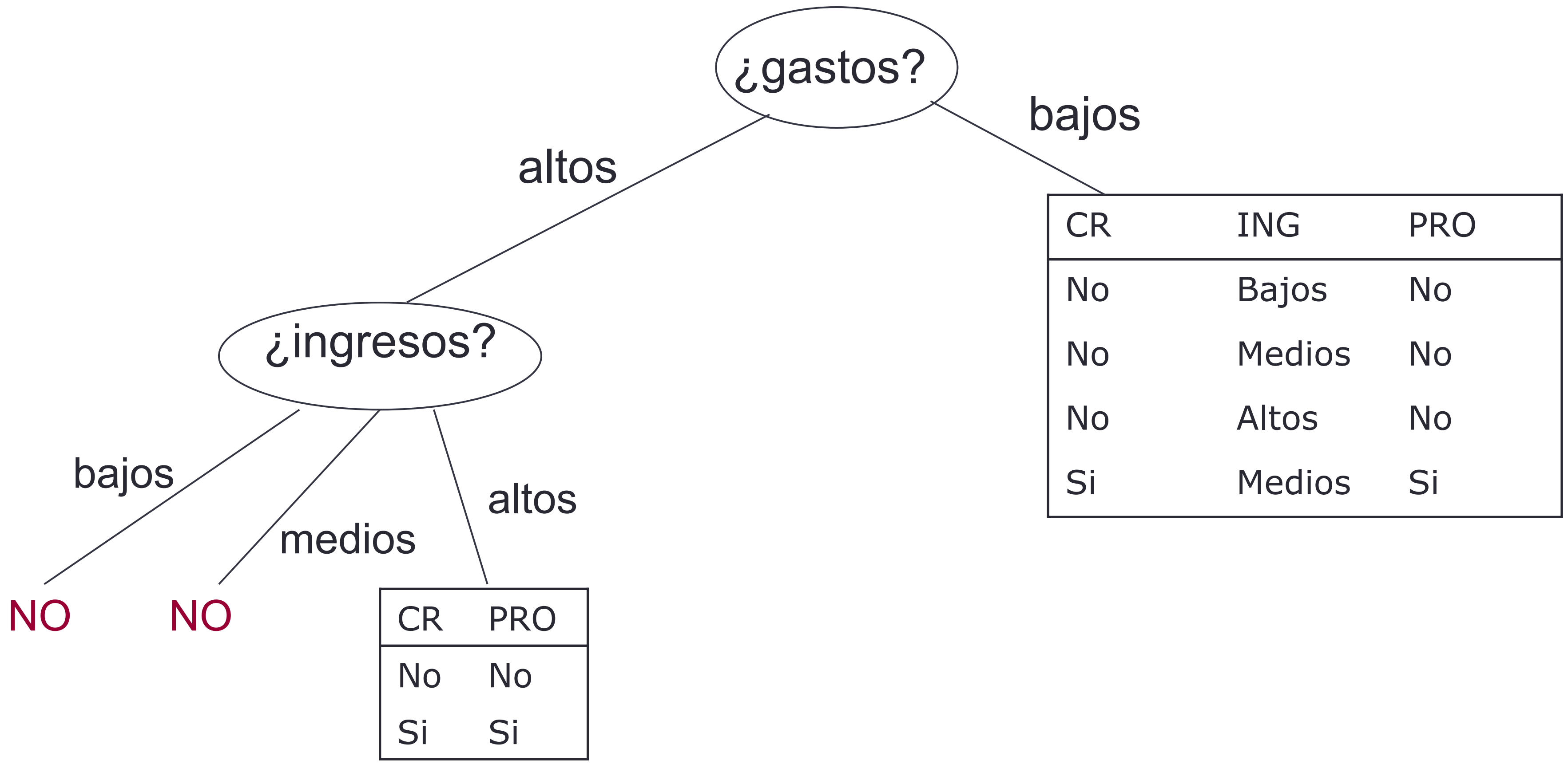
CR	ingresos	PRO
No	Bajos	No
No	Bajos	Si

CR	ingresos	PRO
No	Medios	Si
No	Medios	No

CR	Ingresos	PRO
No	Altos	No
Si	Altos	Si

Ejemplo: Problema de asignación de crédito

Creamos nodos hoja

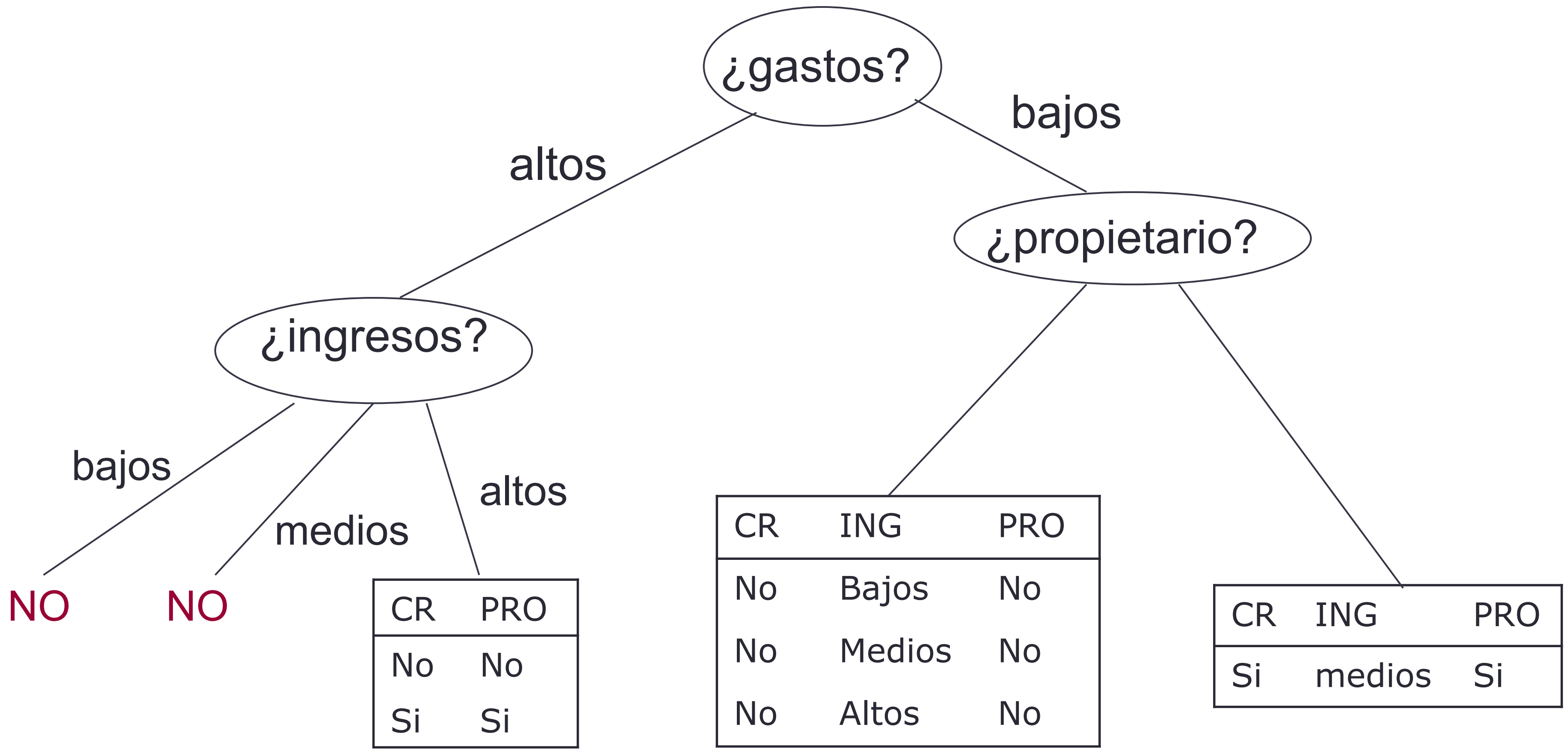


CR	PRO
No	No
Si	Si

CR	ING	PRO
No	Bajos	No
No	Medios	No
No	Altos	No
Si	Medios	Si

Ejemplo: Problema de asignación de crédito

Seleccionamos propietario como test en gastos = bajos



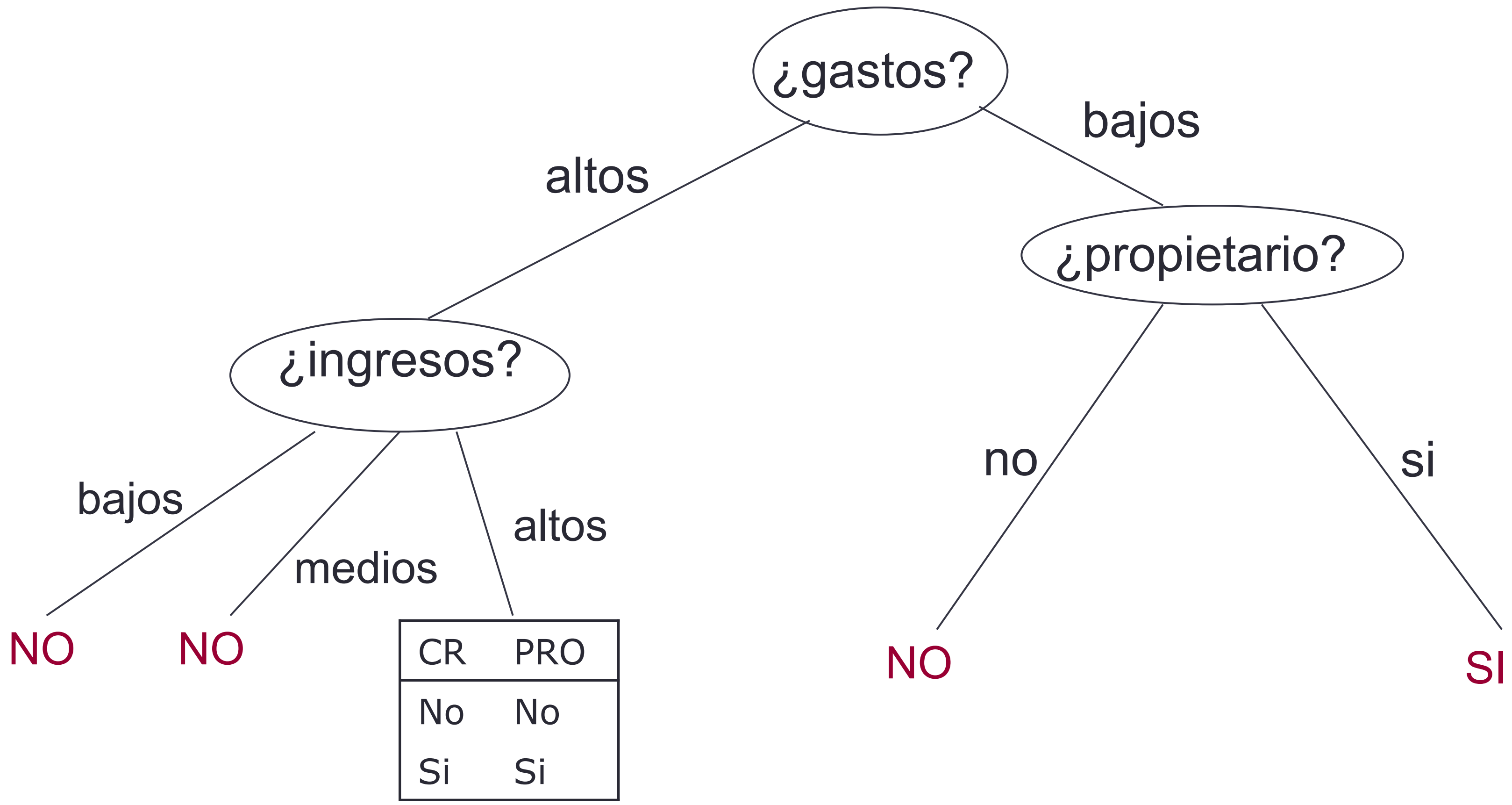
CR	PRO
No	No
Si	Si

CR	ING	PRO
No	Bajos	No
No	Medios	No
No	Altos	No

CR	ING	PRO
Si	medios	Si

Ejemplo: Problema de asignación de crédito

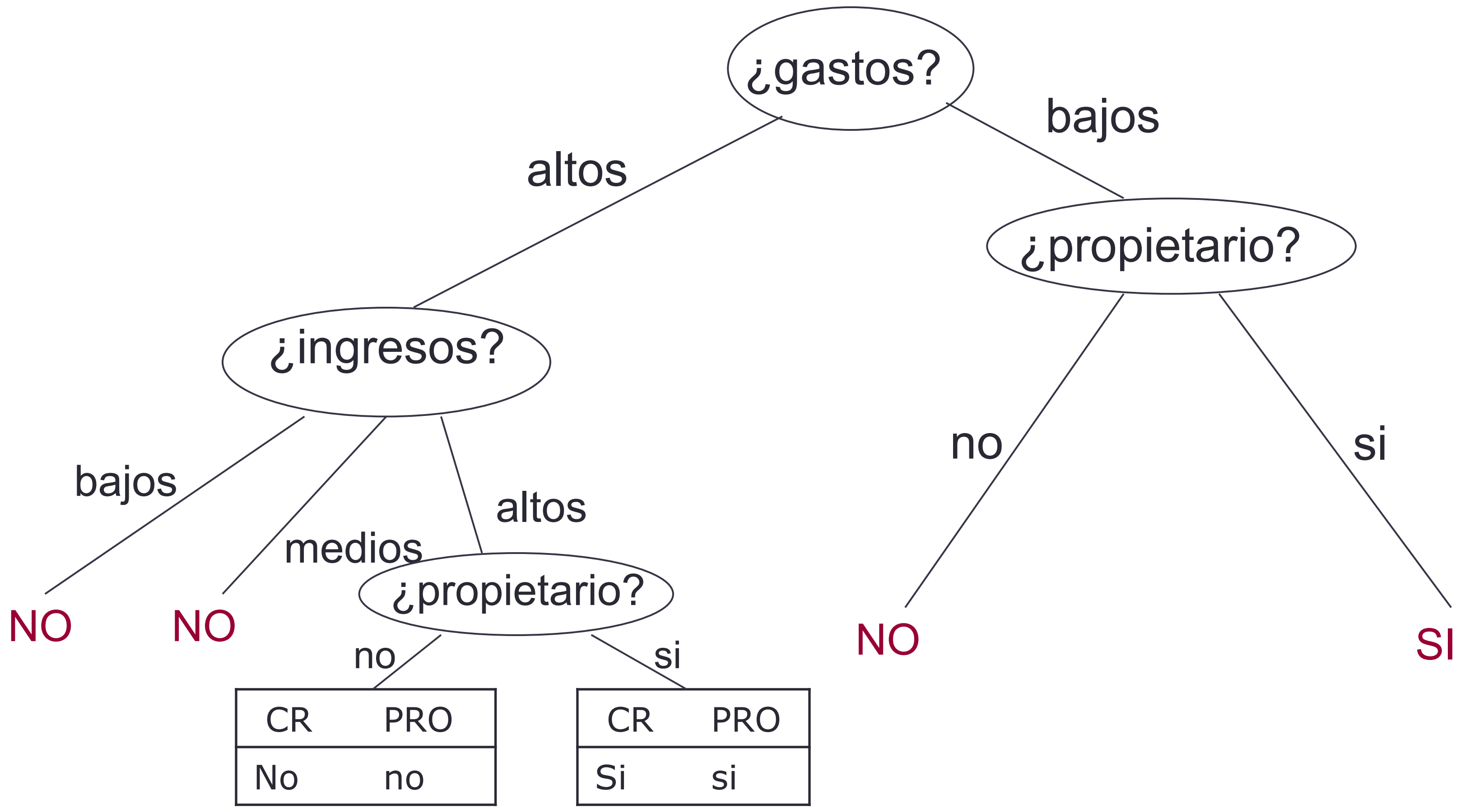
Creamos nodos hoja



CR	PRO
No	No
Si	Si

Ejemplo: Problema de asignación de crédito

Seleccionamos propietario como test en gastos=altos, ingresos=altos

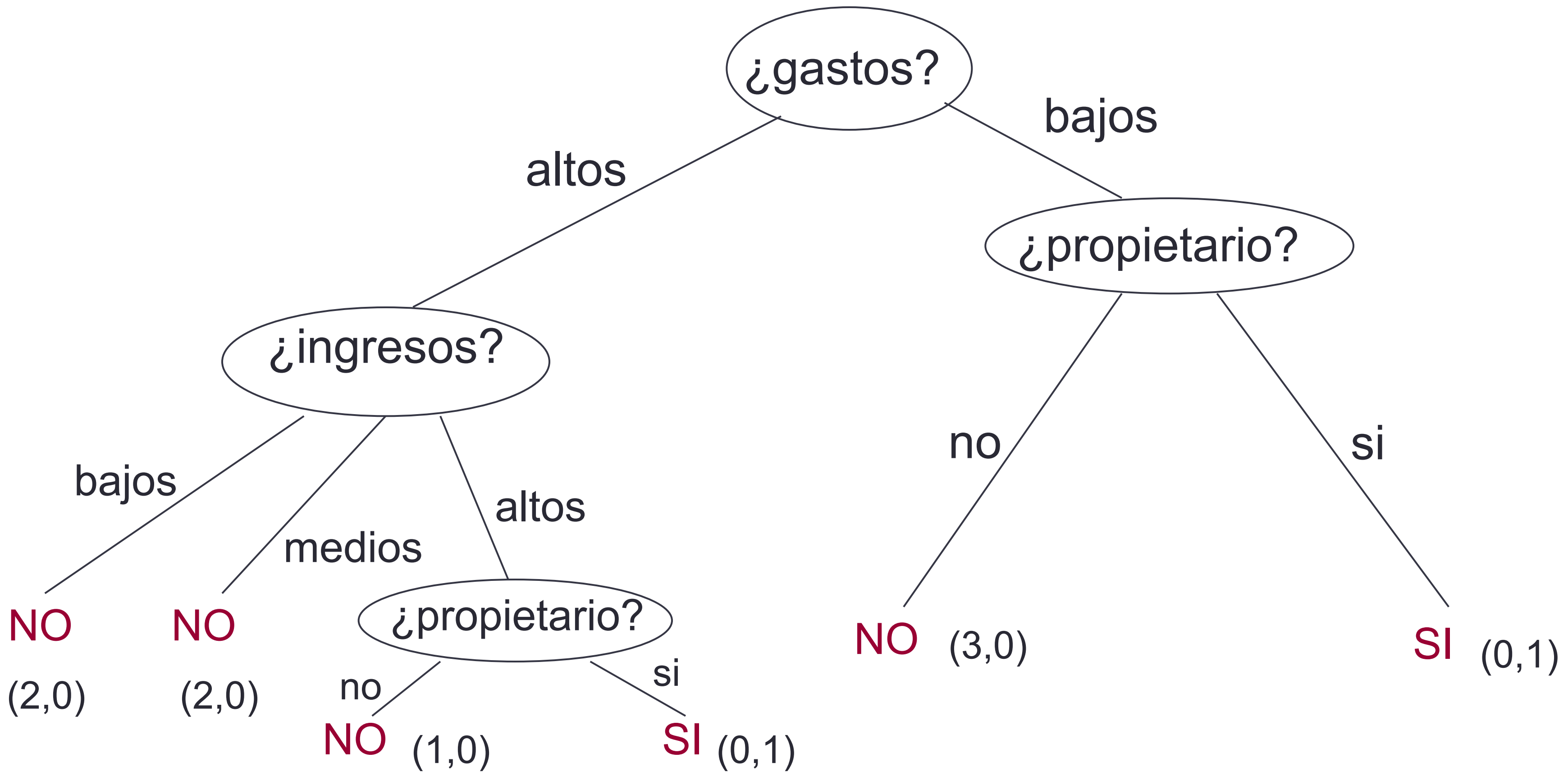


CR	PRO
No	no

CR	PRO
Si	si

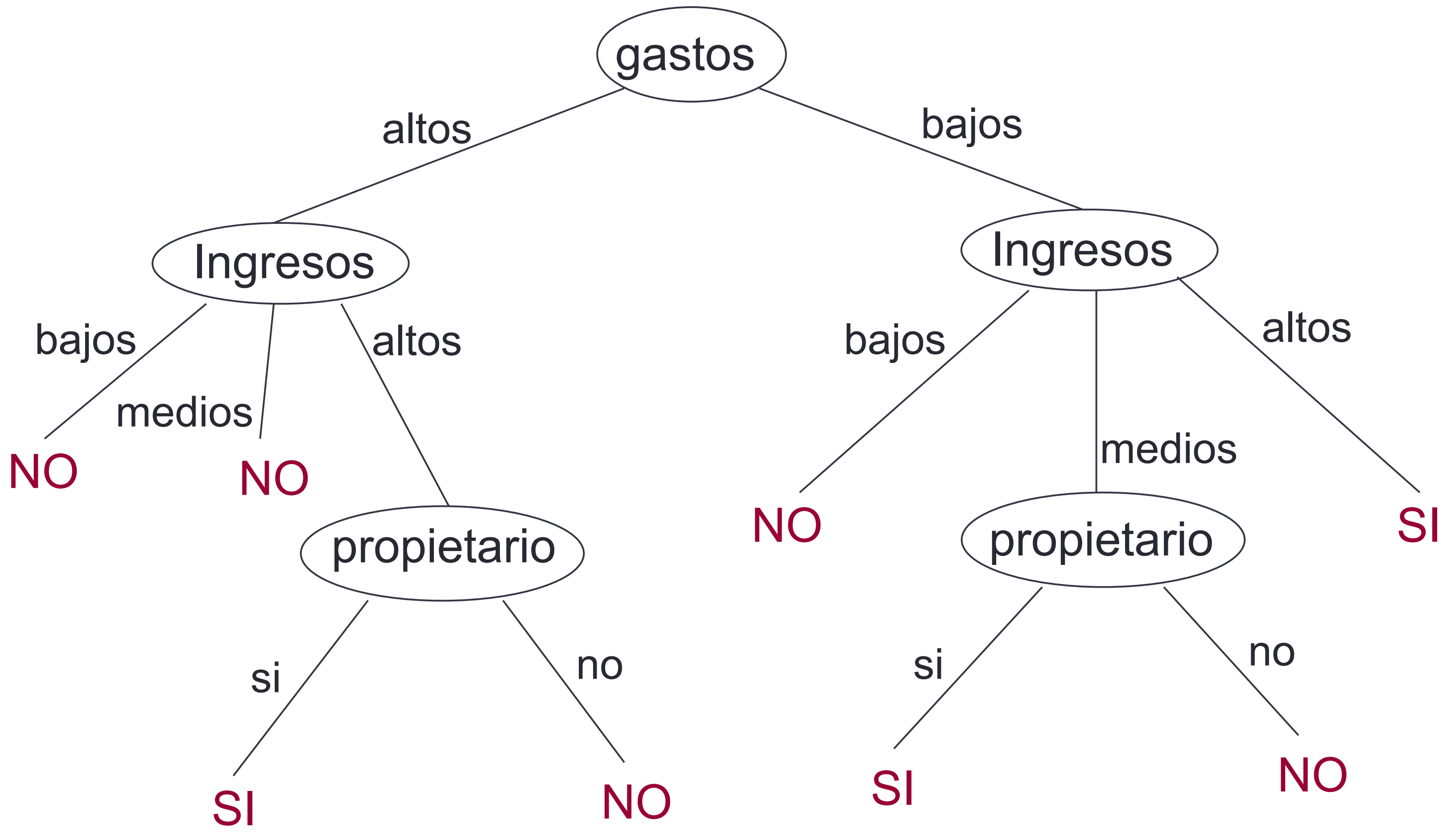
Ejemplo: Problema de asignación de crédito

Árbol de decisión (#no, #si)



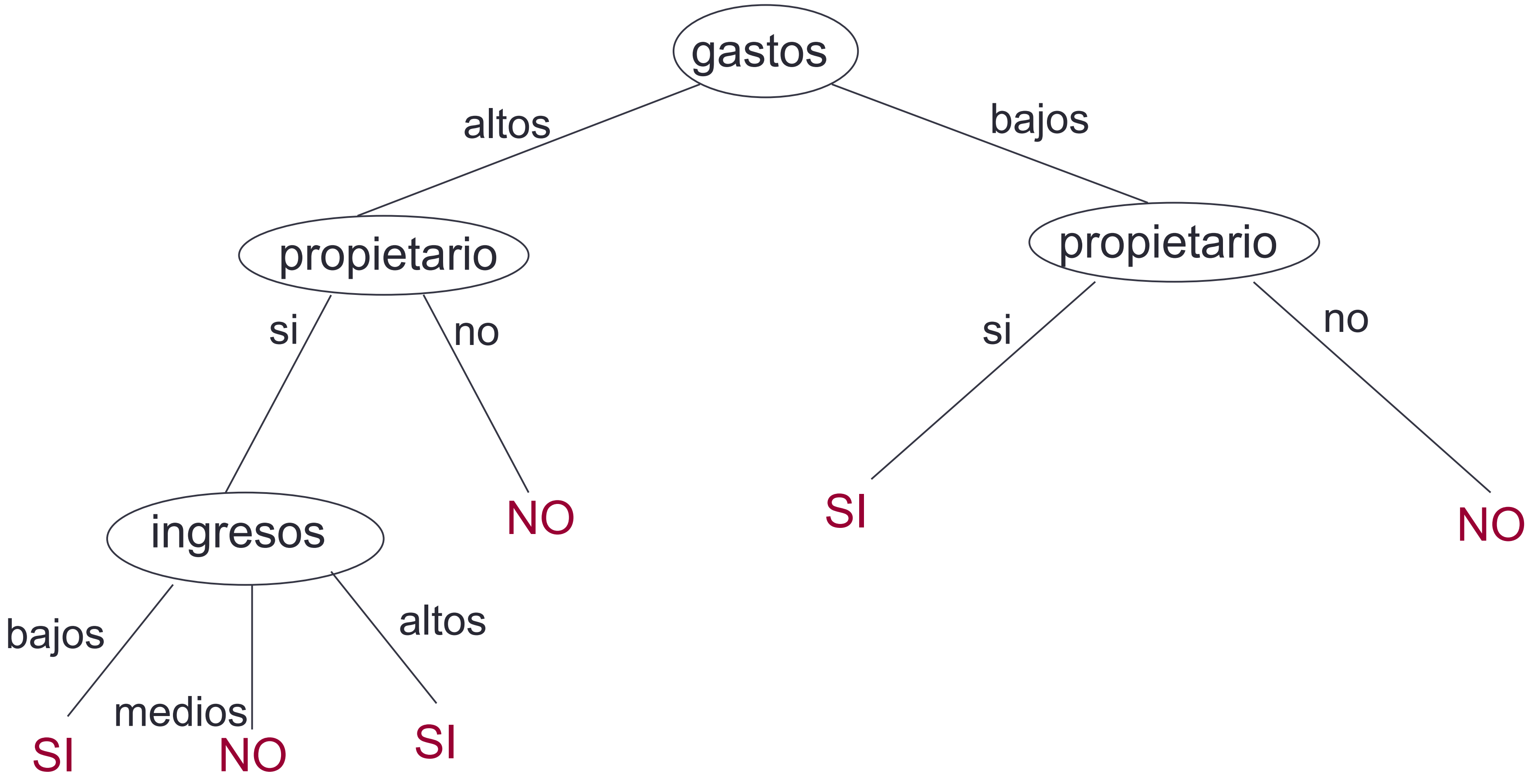
Ejemplo: Problema de asignación de crédito

Otro árbol de decisión para crédito (árbol 2)



Ejemplo: Problema de asignación de crédito

Otro árbol de decisión para crédito (árbol 3)



Ejemplo: Problema de asignación de crédito

- 1er. árbol: 6 reglas (2.33 premisas por regla)
- 2do. árbol: 8 reglas (2.5 premisas por regla)
- 3er. árbol: 6 reglas (2.5 premisas por regla)

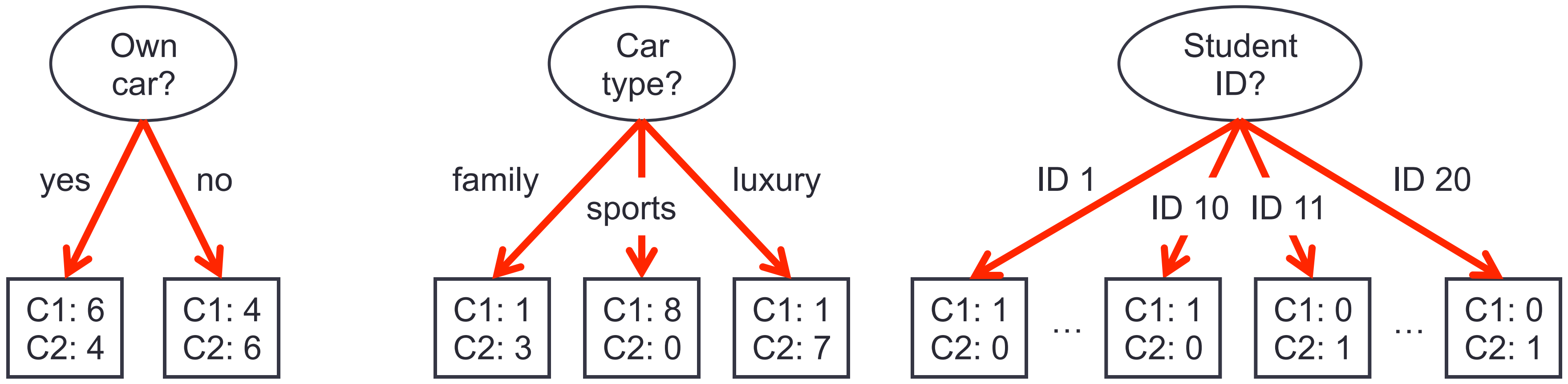
- Dependiendo del orden en el que se van tomando los atributos obtenemos clasificadores de distinta complejidad

- Lo ideal sería tomar en todo momento el atributo que mejor clasifica

¿Cómo decidir qué atributo es el mejor?

Determining the best split

Before splitting: 10 records of class 1 (C1)
10 records of class 2 (C2)



Which attribute gives the best split?

Occam’s Razor

- Given two models with similar generalization errors, one should prefer the simpler model over the more complex model.
- For complex models, there is a greater chance it was fitted accidentally by errors in data.
- Model complexity should therefore be considered when evaluating a model.

Determining the best split

- Greedy approach:

  Nodes with homogeneous class distribution are preferred.
- Need a measure of node impurity:

class 1: 5
class 2: 5
	class 1: 9
	class 2: 1

Non-homogeneous, high degree of impurity	Homogeneous, low degree of impurity

Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error

Measure of Impurity: GINI

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Maximum (1 - 1/n_c) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

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

C1	**1**
C2	**5**

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

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

C1	**2**
C2	**4**

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

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

Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

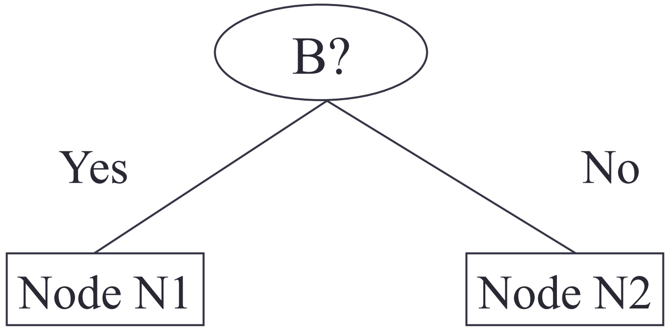
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

Binary Attributes: Computing GINI Index

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



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

Gini(N1)
$= 1 – (4/7)^2 – (3/7)^2$
$= 0.489796$

Gini(N2)
$= 1 – (2/5)^2 – (3/5)^2$
$= 0.48$

	N1	**N2**
C1	**4**	**2**
C2	**3**	**3**
Gini=0.486		

Gini(Children)
$= 7/12 * 0.489796 +$
$5/12 * 0.48$
$= 0.486$

Categorical Attributes: Computing Gini Index

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

Multi-way split

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

Two-way split
(find best partition of values)

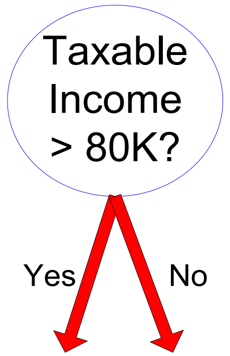
	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

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

Continuous Attributes: Computing Gini Index

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

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**



Continuous Attributes: Computing Gini Index

- 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
		Taxable Income									
Sorted Values →		60	70	75	85	90	95	100	120	125	220

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

Alternative Splitting Criteria based on Entropy

- Entropy at a given node t:

$$\boxed{Entropy(t) = -\sum_j p(j\,|\,t)\log p(j\,|\,t)}$$

(NOTE: $p(j\,|\,t)$ is the relative frequency of class j at node t).

- Measures homogeneity of a node.
  - Maximum ($\log n_c$) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
- Entropy based computations are similar to the GINI index computations

Examples for computing Entropy

$$\boxed{Entropy(t) = -\sum_j p(j|t)\log_2 p(j|t)}$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

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

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

C1	**2**
C2	**4**

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

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

Splitting Based on INFO...

- Information Gain:

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

Parent Node, p is split into k partitions;

n_i is number of records in partition i

- Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
- Used in ID3
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure

Splitting Based on INFO...

- Gain Ratio:

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

Parent Node, p is split into k partitions

n_i is the number of records in partition i

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

Splitting Criteria based on Classification Error

- Classification error at a node t :

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

- Measures misclassification error made by a node.
    - Maximum (1 - 1/n_c) when records are equally distributed among all classes, implying least interesting information
    - Minimum (0.0) when all records belong to one class, implying most interesting information

Examples for Computing Error

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

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

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

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

C1	**2**
C2	**4**

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

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

Comparison among Splitting Criteria

For a 2-class problem:

Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$
$$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$$
$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

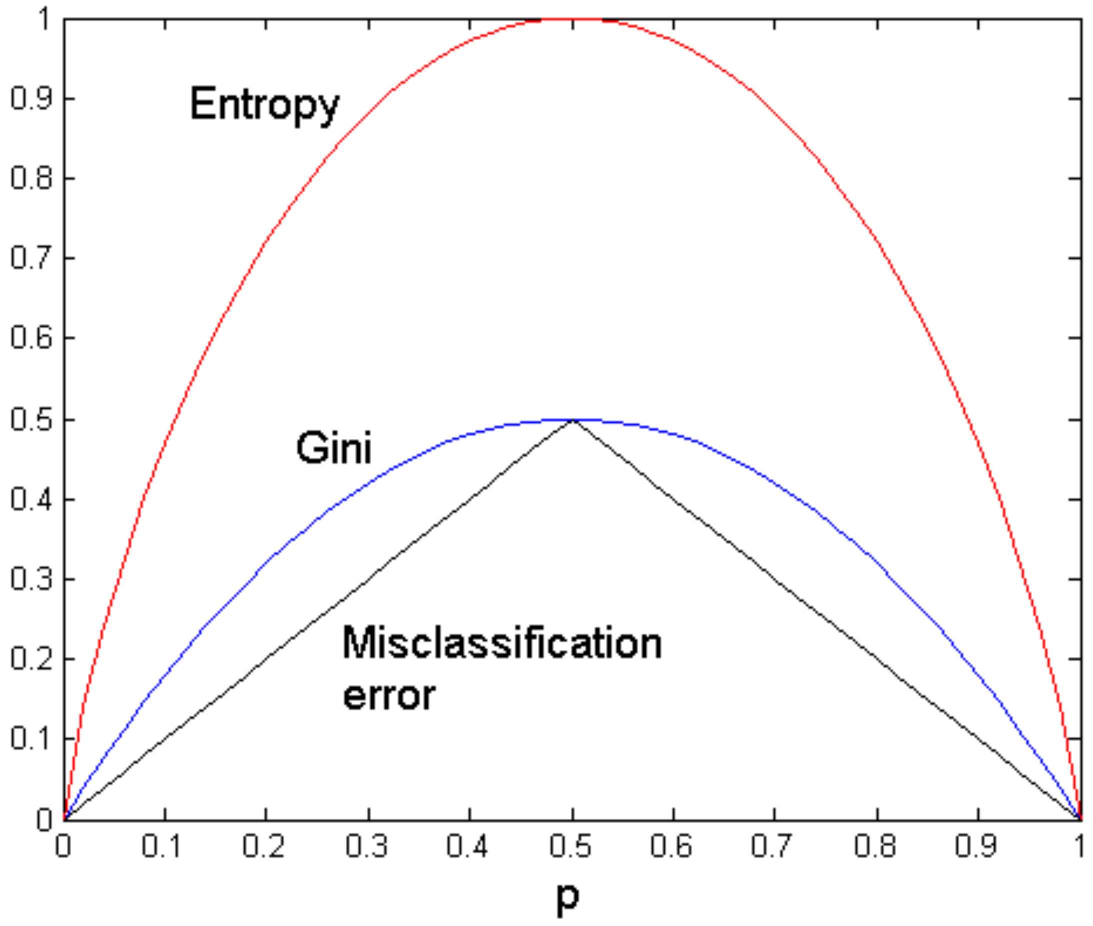
$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$
$$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$$
$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$
$$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$$
$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

Comparison among Splitting Criteria

For a 2-class problem:



Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.

- Issues
  - Determine how to split the records
    - How to specify structure of split?
    - What is best attribute / attribute value for splitting?
  - Determine when to stop splitting

Stopping criteria for tree induction

- Stop expanding a node when all the records belong to the same class.
- Early termination: can also prune the tree.

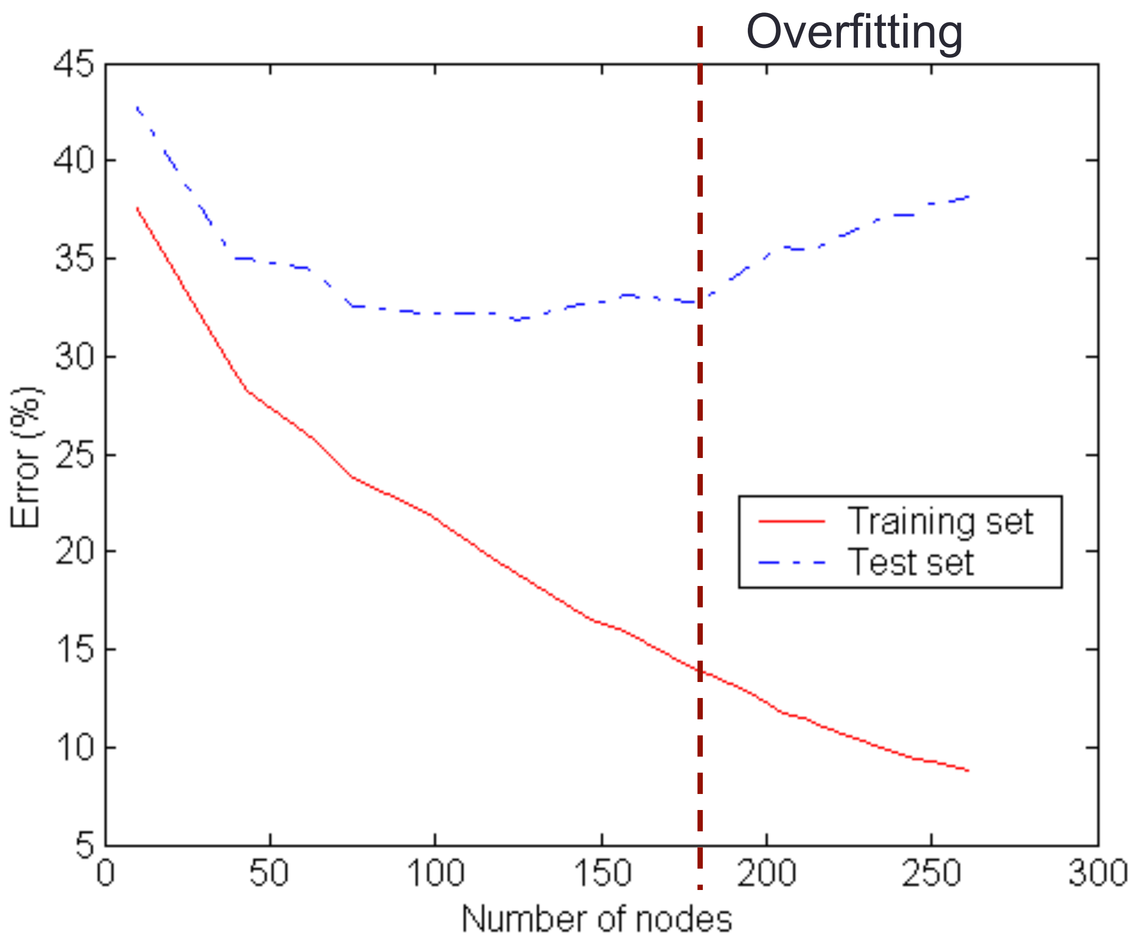
Decision Tree Based Classification

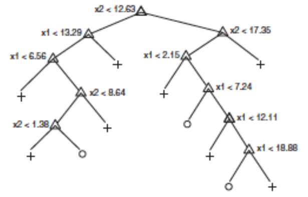
- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets

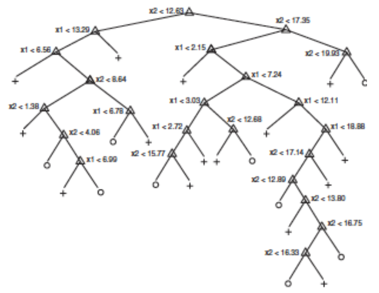
Practical Issues of Classification

- Underfitting and Overfitting
- Missing Values

Underfitting and Overfitting

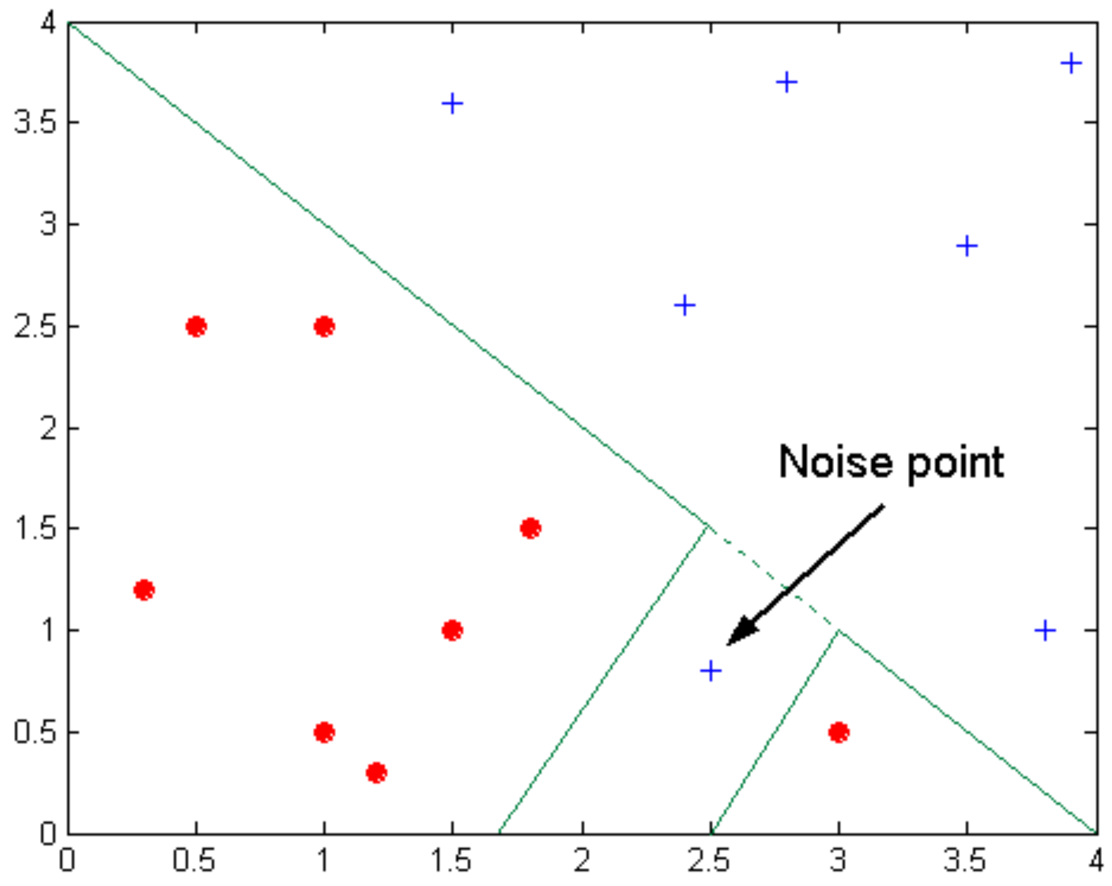



(a) Decision tree with 11 leaf nodes.


(b) Decision tree with 24 leaf nodes.

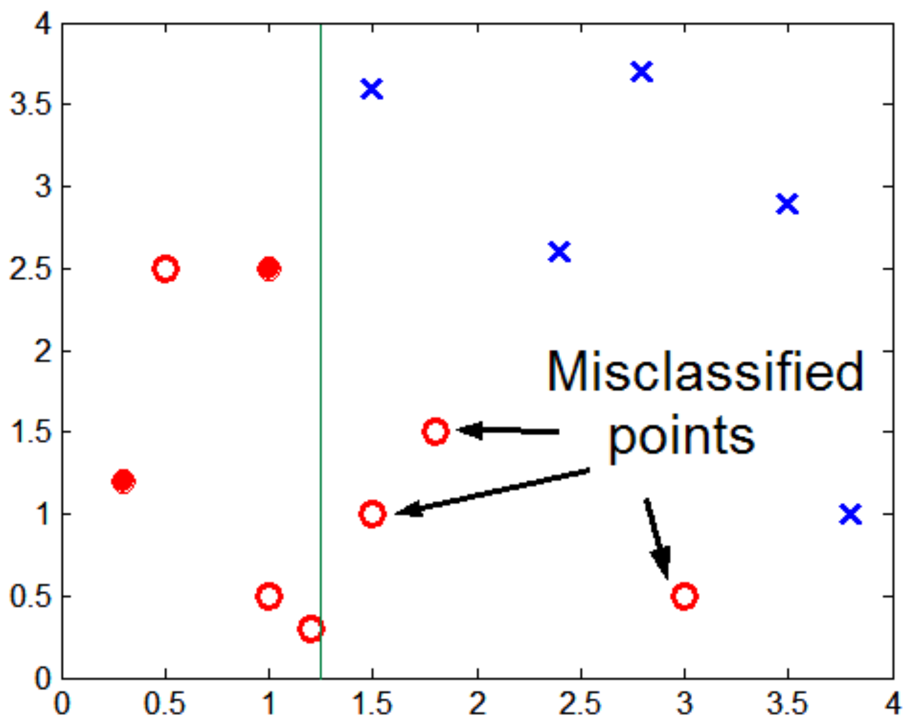
Underfitting: when model is too simple, both training and test errors are large

Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples



- Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region
- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

- Need new ways for estimating errors

How to Address Overfitting

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping condition for a node:
    - Stop if all instances belong to the same class
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

How to Address Overfitting

- Post-pruning
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - If generalization error improves after trimming, replace sub-tree by a leaf node.
  - Class label of leaf node is determined from majority class of instances in the sub-tree

Example of Post-Pruning

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

Training Error (Before splitting) = 10/30

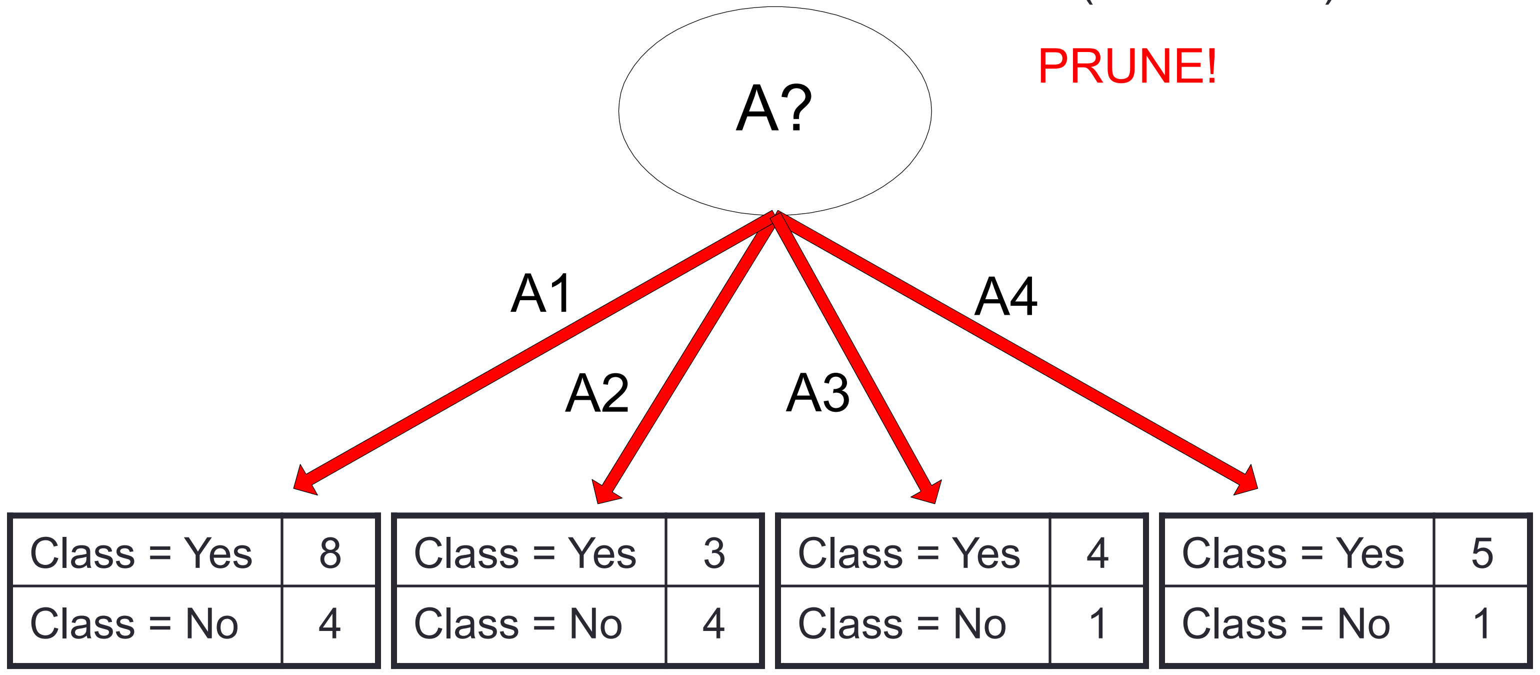
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 × 0.5)/30 = 11/30

PRUNE!



A1		A2		A3		A4	
Class = Yes	8	Class = Yes	3	Class = Yes	4	Class = Yes	5
Class = No	4	Class = No	4	Class = No	1	Class = No	1

Handling Missing Attribute Values

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

Computing Impurity Measure

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	?	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)

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

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

Split on Refund:

Entropy(Refund=Yes) = 0

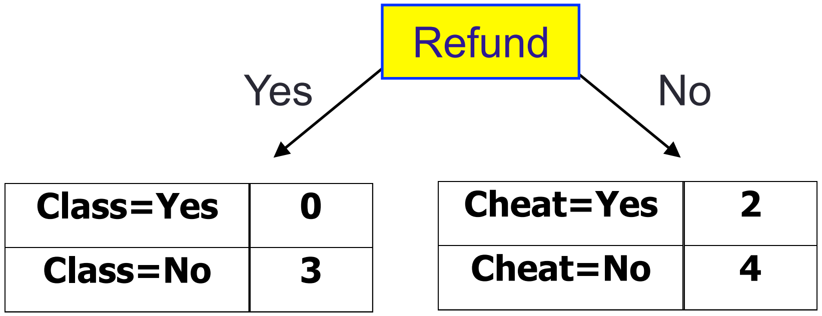
Entropy(Refund=No)
= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

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

Gain = 0.9 × (0.8813 – 0.551) = 0.3303

Distribute Instances

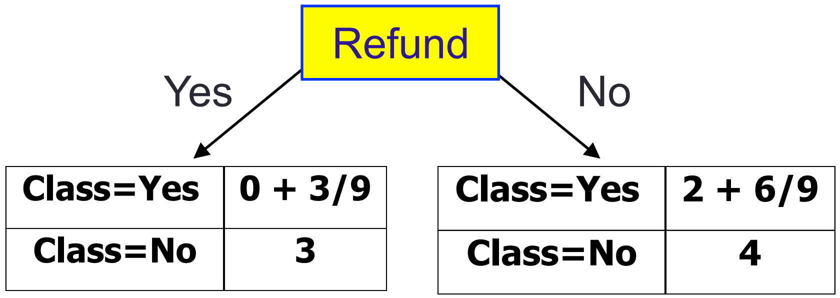
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Class**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**



Class=Yes	**0**
Class=No	**3**

Cheat=Yes	**2**
Cheat=No	**4**

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



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

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

Probability that Refund=Yes is 3/9

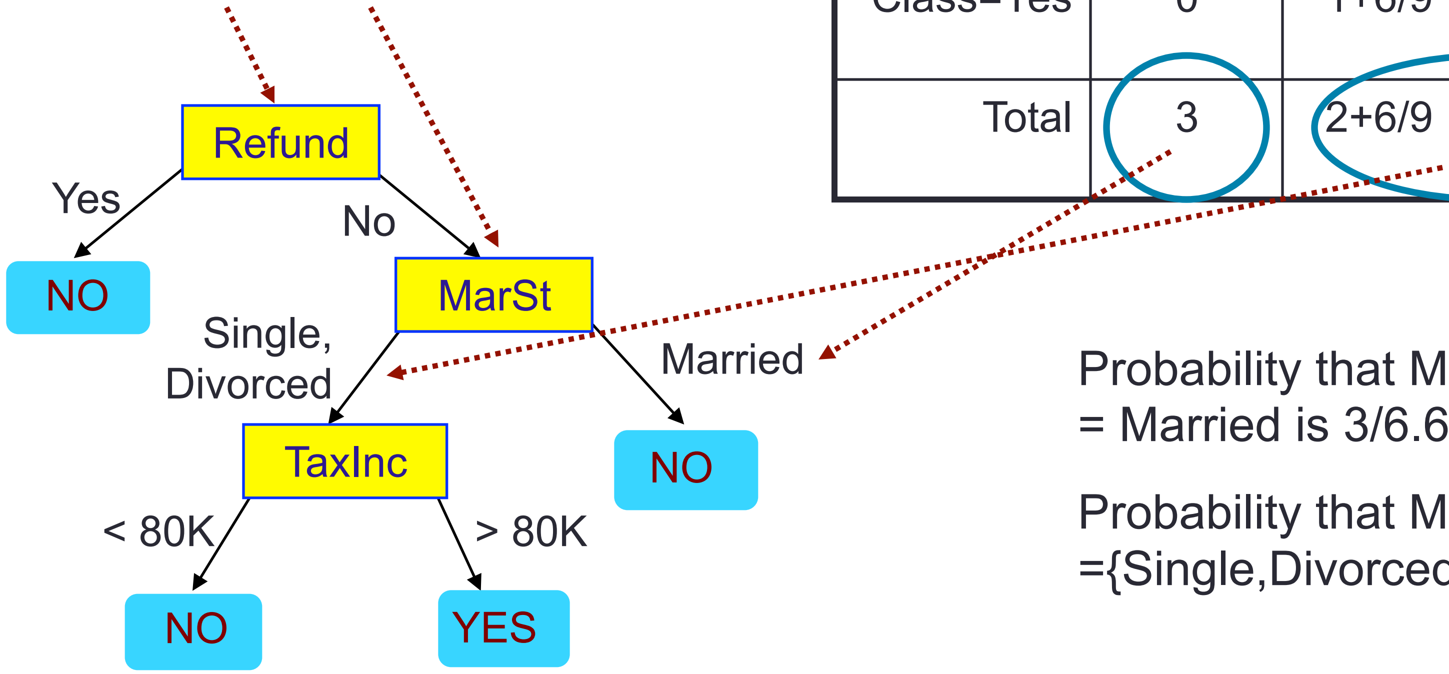
Probability that Refund=No is 6/9

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

Classify Instances

New record:

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



	Married	Single	Divorced	Total
Class=No	3	1	0	4
Class=Yes	0	1+6/9	1	2’67
Total	3	2+6/9	1	6’67

Probability that Marital Status = Married is 3/6.67

Probability that Marital Status ={Single,Divorced} is 3’67/6.67

Algunos algoritmos de minería de datos basados en árboles de decisión

ID3

J.R. Quinlan. Induction of Decision Trees. Machine Learning, vol. 1, pp 81-106, 1986

- Crea el árbol utilizando conceptos de teoría de información
- Intenta reducir el número de comparaciones
- ID3 elige el atributo test con máxima ganancia de información
    - Basada en la entropía que se utiliza como una medida de la cantidad de incertidumbre o sorpresa en un conjunto de datos

Algunos algoritmos de minería de datos basados en árboles de decisión: ID3

Esquema algoritmo ID3

1. Seleccionar el atributo A que maximice la ganancia G(S,A)
2. Crear un nodo para ese atributo con tantos sucesores como valores tenga
3. Introducir los ejemplos en los sucesores según el valor que tenga el atributo A
4. Por cada sucesor:
   Si sólo hay ejemplos de una clase, C_k
   Entonces etiquetarlo con C_k
   Si no, llamar a ID3 con un conjunto de ejemplos formado por los ejemplos de ese nodo, eliminando la columna del atributo A

- Termina cuando todos los datos del nodo son de la misma clase y la entropía es cero

Algunos algoritmos de minería de datos basados en árboles de decisión: C4.5

C4.5

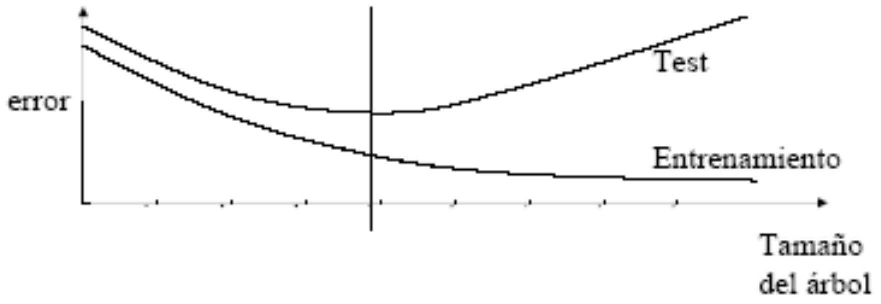
J.R. Quinlan. C4.5: Programs for Machine Learning. San Francisco: Morgan Kaufmann, 1993

Mejora a ID3 en los siguientes aspectos:

- Datos perdidos:
  - Cuando se construye el árbol, los datos perdidos se ignoran (el árbol de decisión se construye mirando sólo los registros que tienen valor para ese atributo)
  - Para clasificar un ejemplo con valor perdido, éste se predice en base a lo que se sabe sobre los valores del atributo para otros registros
- Datos continuos: Se divide en rangos en base a los valores encontrados en el conjunto de entrenamiento
- Propone soluciones para el sobreaprendizaje. Posibilidades
  - pre-poda: se decide cuándo dejar de subdividir el árbol
  - post-poda: se construye el árbol y después se poda

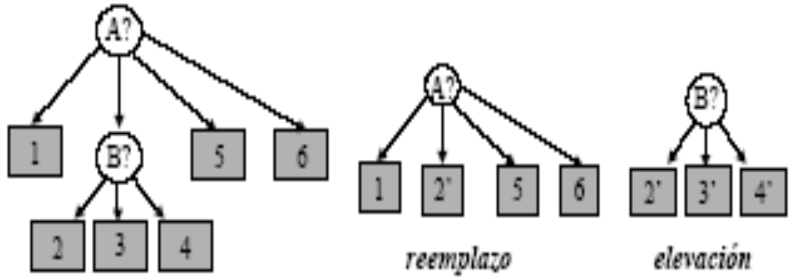
Algunos algoritmos de minería de datos basados en árboles de decisión: C4.5

Pre-poda:
• no se divide un nodo si se tiene poca confianza en él (no es significativa la diferencia de clases), o
• se valida con un conjunto de test independiente y se para cuando la curva del conjunto de test empieza a subir



Hay dos estrategias de post-poda en C4.5

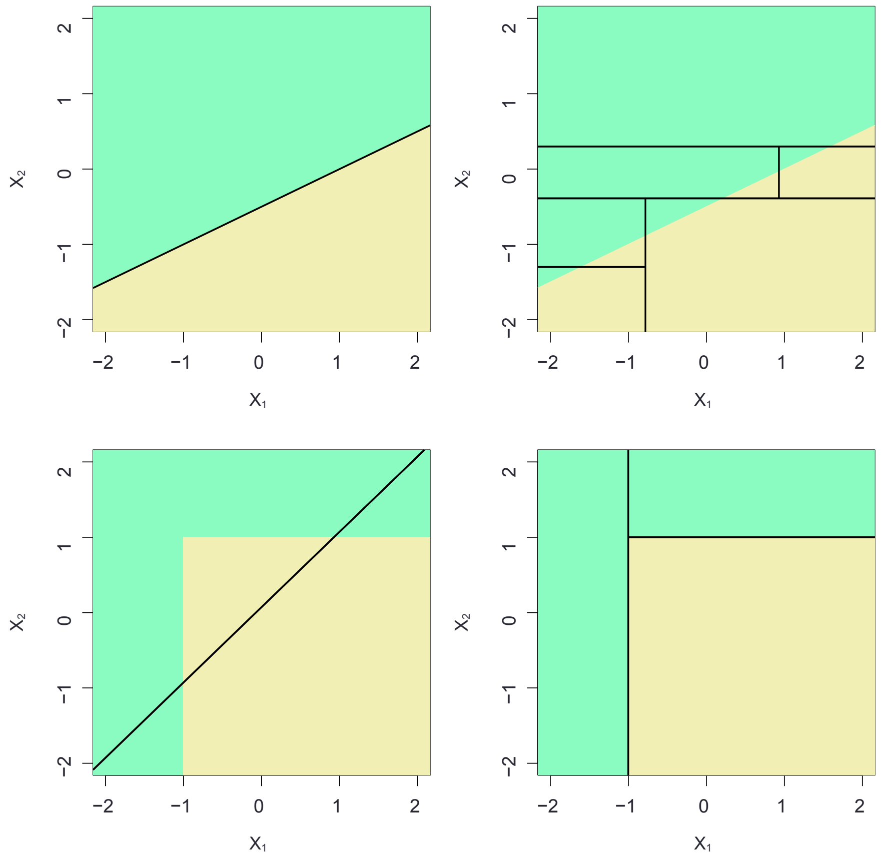
- • Reemplazamiento de subárboles: Se reemplaza un subárbol por una hoja si al hacerlo el error es similar al original
- • Elevación de subárbol: Reemplaza un subárbol por su subárbol más utilizado (un subárbol se mueve de su localización a un nodo superior en el árbol)



Algunos algoritmos de minería de datos basados en árboles de decisión: C4.5

- Reglas: C4.5 permite clasificación mediante el árbol o a través de las reglas que se generan a partir de él
  - Además, incluye técnicas para simplificar las reglas.
- Selección de atributos:
  - ID3 favorece atributos con muchas divisiones.
  - En C4.5 se utiliza como criterio de selección el ratio de ganancia de información que tiene en cuenta la cardinalidad de cada división.

Trees Versus Linear Models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model

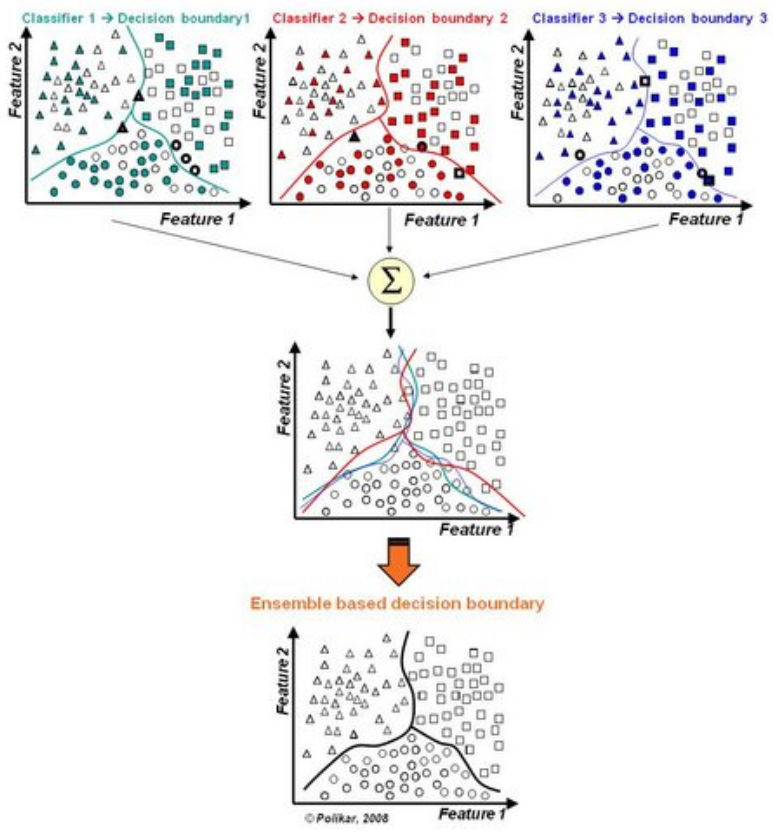
Advantages and Disadvantages of Trees

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and other classification approaches.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

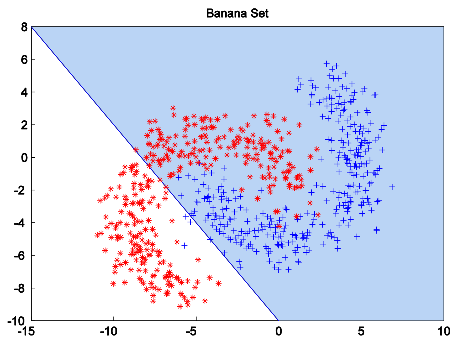
However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

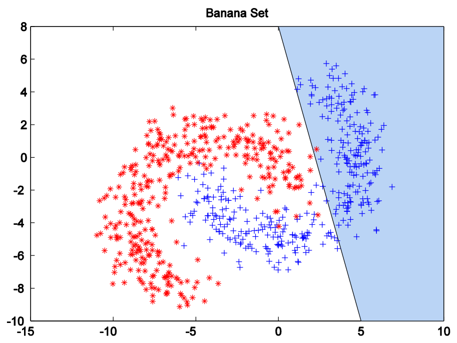
Ensembles, multiclasificadores,…

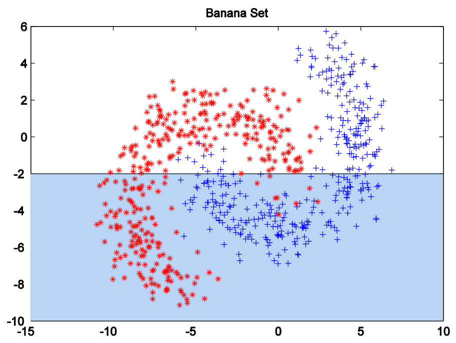
- La idea es inducir *n* clasificadores en lugar de uno solo.
- Para clasificar se utilizará una combinación de la salida que proporciona cada clasificador.
- Los clasificadores pueden estar basados en distintas técnicas (p.e. árboles, reglas, instancias,…).
- Se puede aplicar sobre el mismo clasificador o con diferentes.

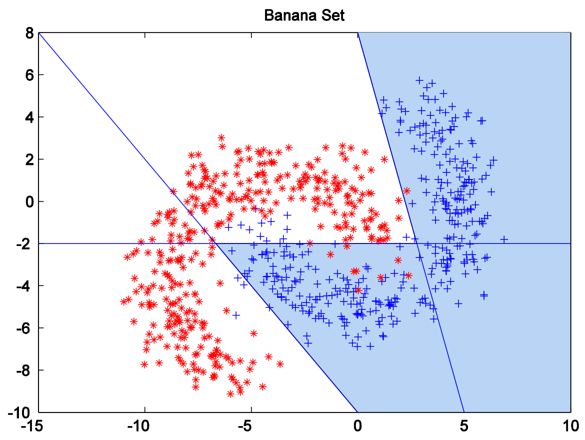

Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 1
Feature 2
Σ
Ensemble based decision boundary
©Polikar, 2008

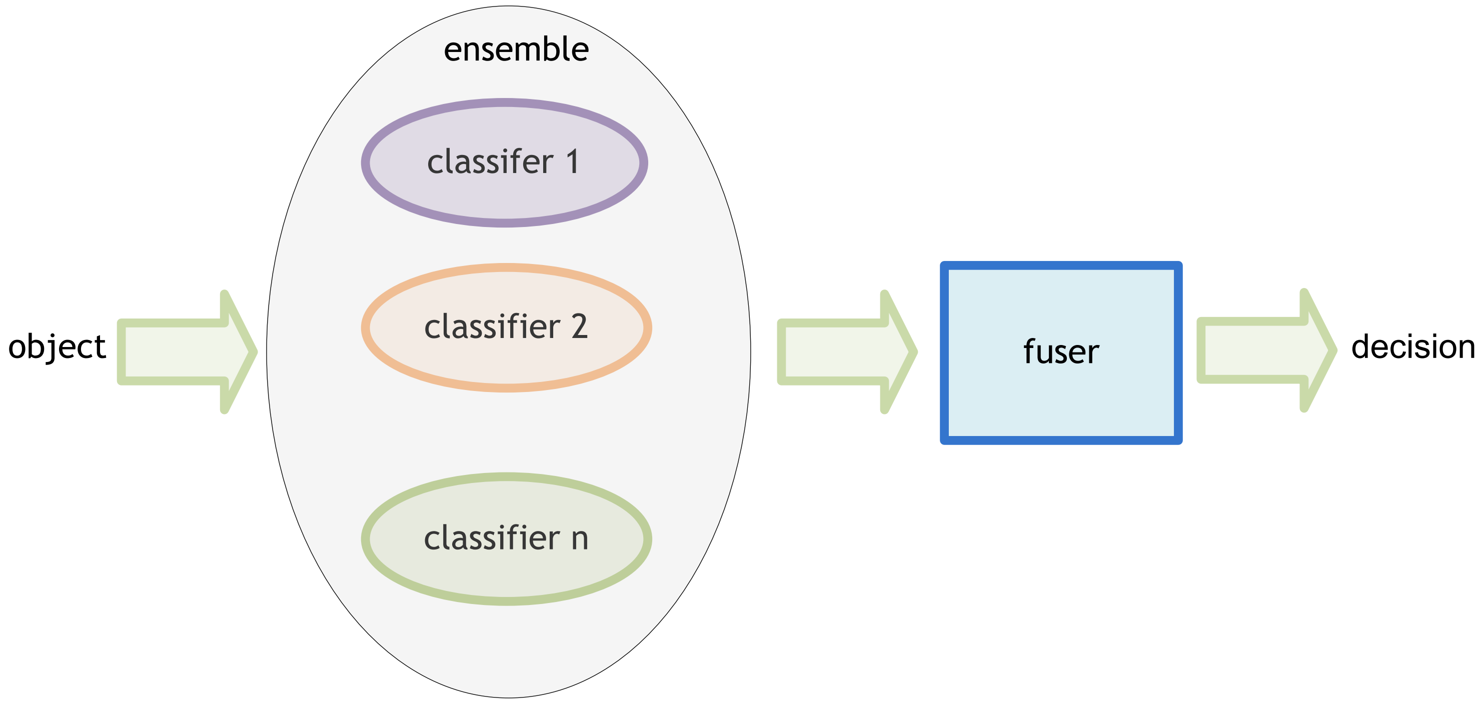
Bagging
Random Forest
Boosting


Banana Set


Banana Set


Banana Set


Banana Set


object
ensemble
classifer 1
classifier 2
classifier n
fuser
decision

Bagging

- Bootstrap aggregation, or bagging [Breiman, 94], is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging — continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get f^^{*b} (x), the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

This is called bagging.

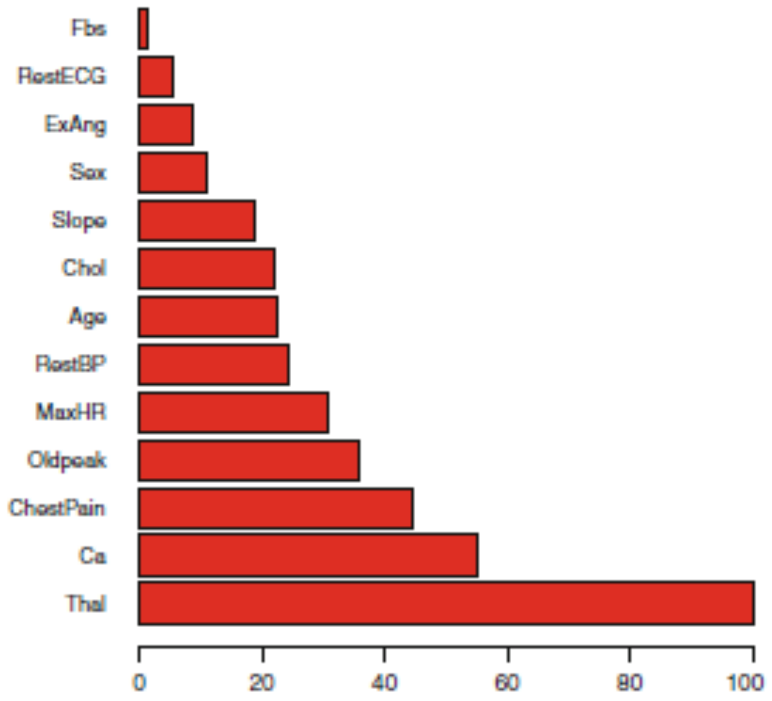
Bagging classification trees

- The above prescription applied to regression trees.
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation.
- The resulting OOB error is a valid estimate of the test error for the bagged model.

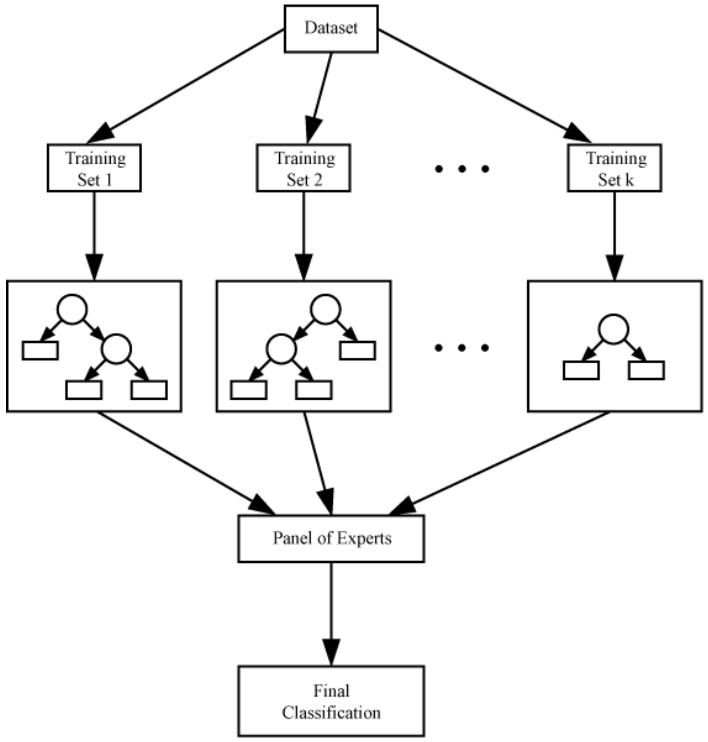
When we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure.



We can record the total amount that the RSS (or gain information) is decreased due to splits over a given predictor, averaged over all B trees.

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.

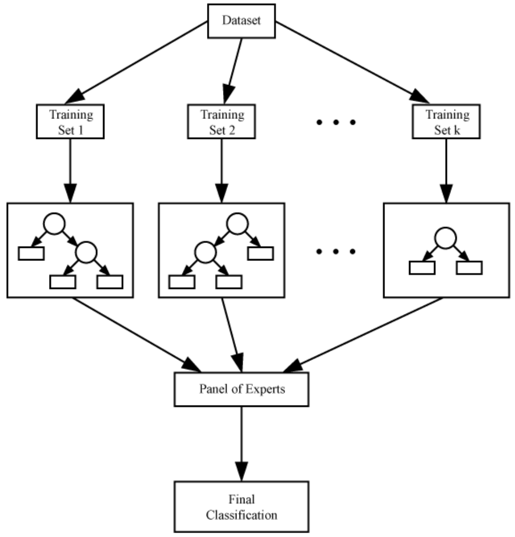


Breiman, Leo (2001). “Random Forests”. *Machine Learning* **45** (1): pp. 5–32. doi:10.1023/A:1010933404324

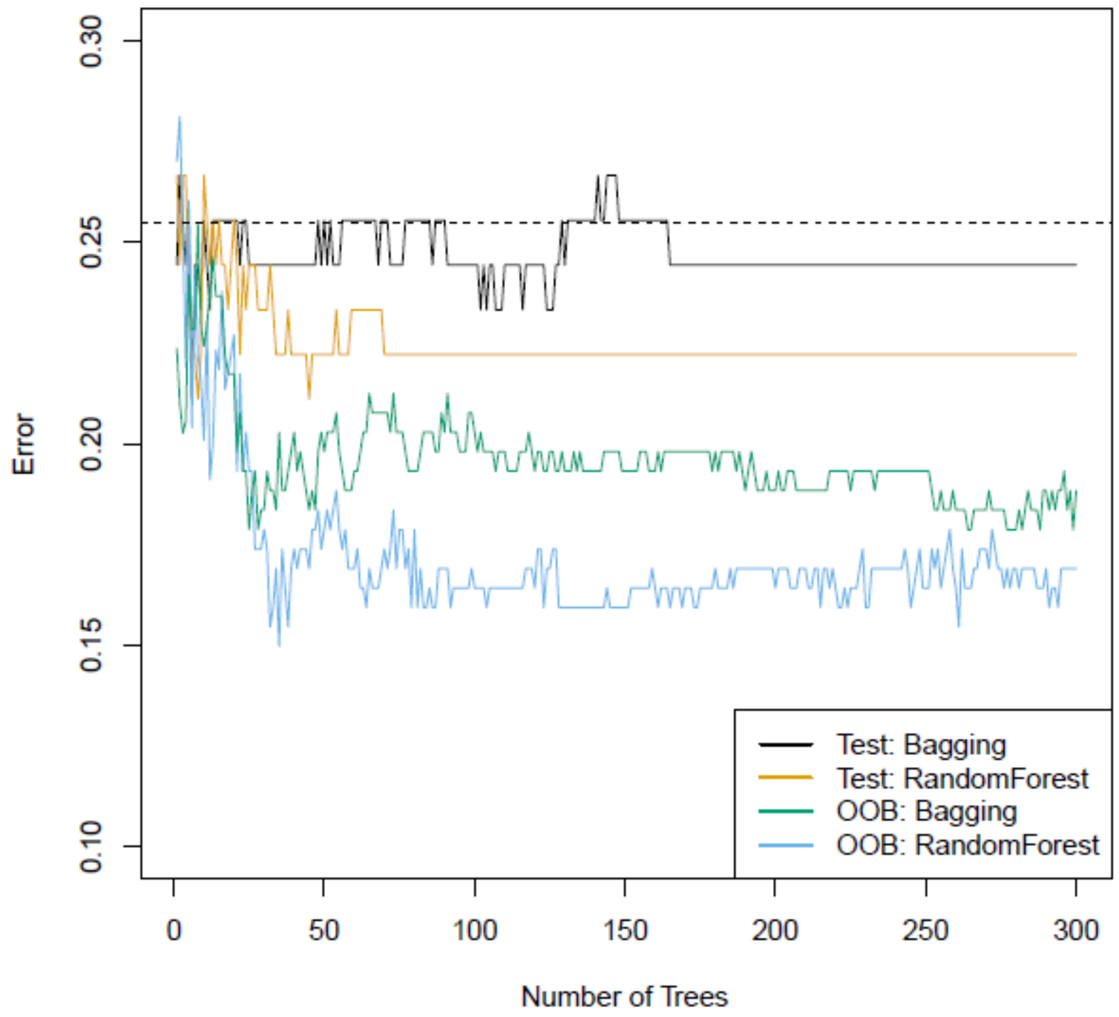
Random Forests

- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

- A selection of m predictors is taken at each split, and typically we choose m $\approx\sqrt{p}$ - that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).



Random Forests: the heart data

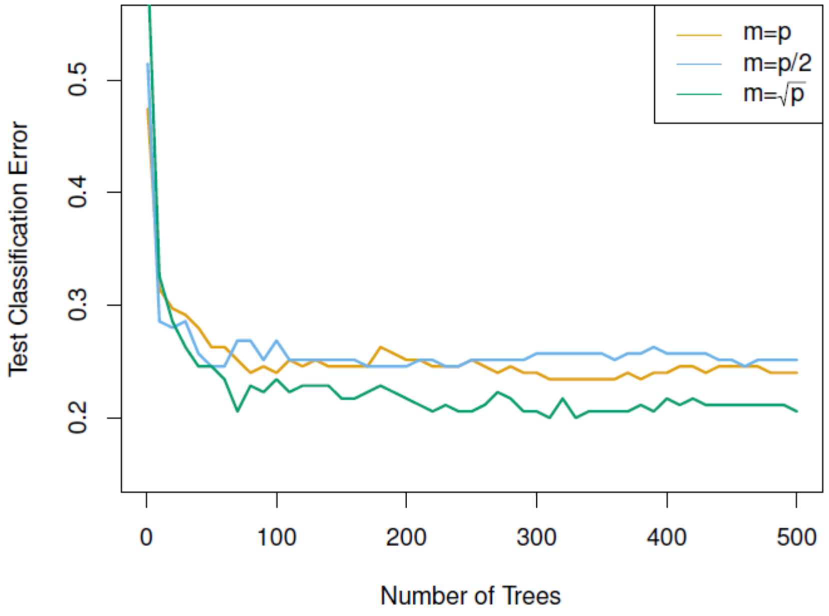


Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set three different values of the number of splitting variables m.

Results: gene expression data

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.



Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting algorithm for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - 2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve f in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

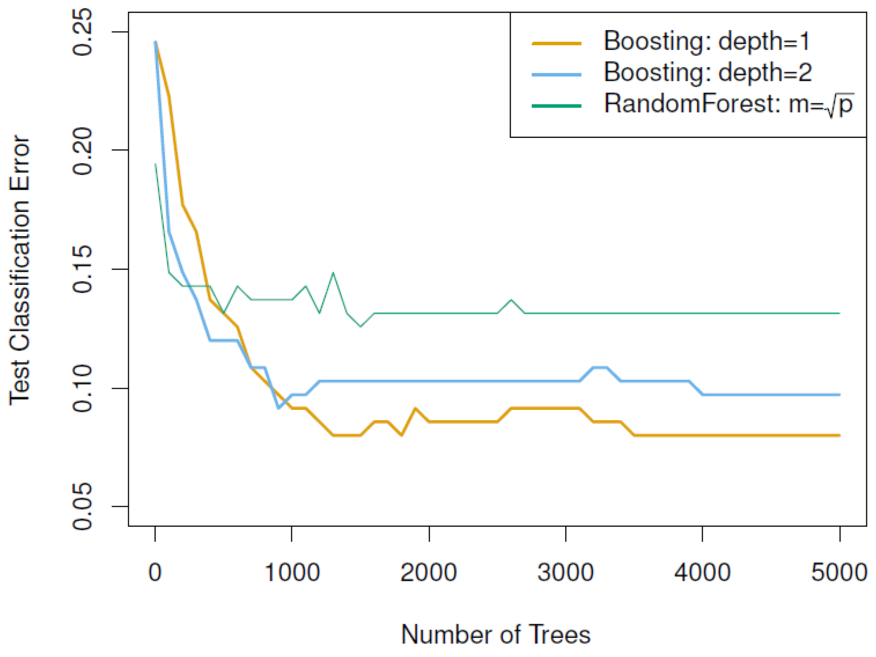
Tuning parameters for boosting

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Gene expression data

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict cancer versus normal.

- The test error is displayed as a function of the number of trees. For the two boosted models, λ = 0.01. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.

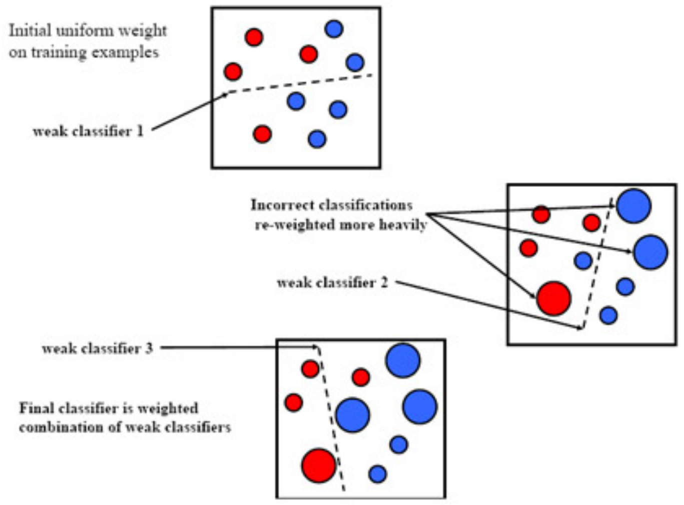


Boosting for classification

- **Muestreo ponderado (ejemplos):**
  - En lugar de hacer un muestreo aleatorio de los datos de entrenamiento, se ponderan las muestras para concentrar el aprendizaje en los ejemplos más difíciles.
- **Votos ponderados (clasificadores):**
  - En lugar de combinar los clasificadores con el mismo peso en el voto, se usa un voto ponderado.

Boosting for classification

AdaBoost, abreviatura de "Adaptive Boosting", es un algoritmo de aprendizaje formulado por Yoav Freund y Robert Schapire que ganó el prestigioso "Premio Gödel" en 2003 por su trabajo. Se puede utilizar en conjunción con muchos otros tipos de algoritmos de aprendizaje para mejorar su rendimiento.


Initial uniform weight on training examples
weak classifier 1
Incorrect classifications re-weighted more heavily
weak classifier 2
weak classifier 3
Final classifier is weighted combination of weak classifiers

$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

Boosting for classification

AdaBoost. Adaptive Boosting [Freund,Schapire,96]

- Initialize distribution over training set $D_1(i) = 1/N$.
- For $t = 1, \ldots, T$

1. Train weak learner using distribution D_t and obtain h_t.
2. Choose a weight (confidence value) $\alpha_t \in R$.
3. Update distribution over training set:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

- Set $H(x) = sign(f(x)) = sign\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right)$

1. Entrenar un clasificador débil usando D_t y obtener h_t

- Normalmente se muestrean los ejemplos de entrenamiento usando D_t (muestreo por importancia)
- Inicialmente, cuando T=1 todos los ejemplos son igualmente probables.
- En las siguientes iteraciones, es más probable seleccionar los ejemplos más difíciles (los que hacen fallar al clasificador).

2. Escoger un valor de confianza α_t

Sea ε_t el error asociado a h_t

$$\epsilon_t = Pr_{D_t}[h_t(\mathbf{x}_i) \neq y_i]$$

El valor de α_t surge de intentar optimizar dicho error y es:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$$

Actualizar la distribución D:

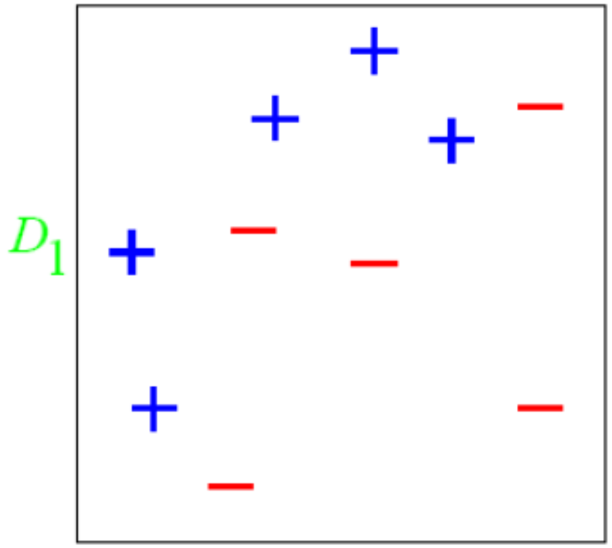
- Inicialmente, cuando T=1 todos los ejemplos son igualmente probables.
- En las siguientes iteraciones, es más probable seleccionar los ejemplos más difíciles (los que hacen fallar al clasificador).

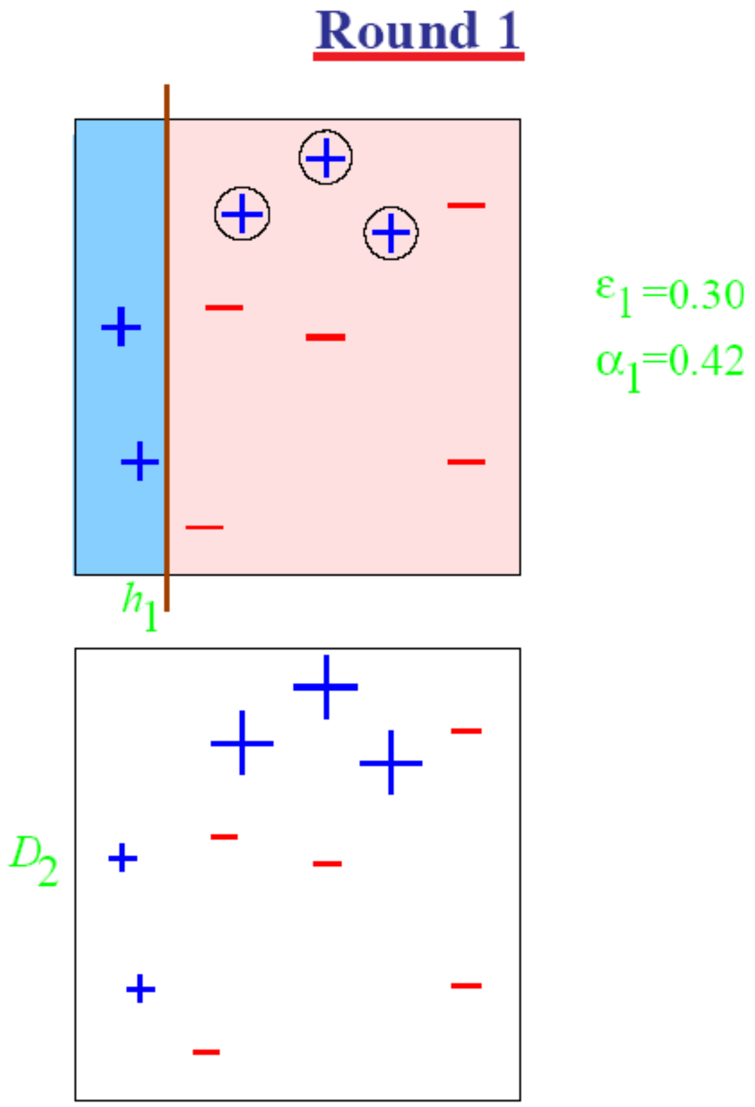
$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot A$$

$$\Big\Downarrow$$

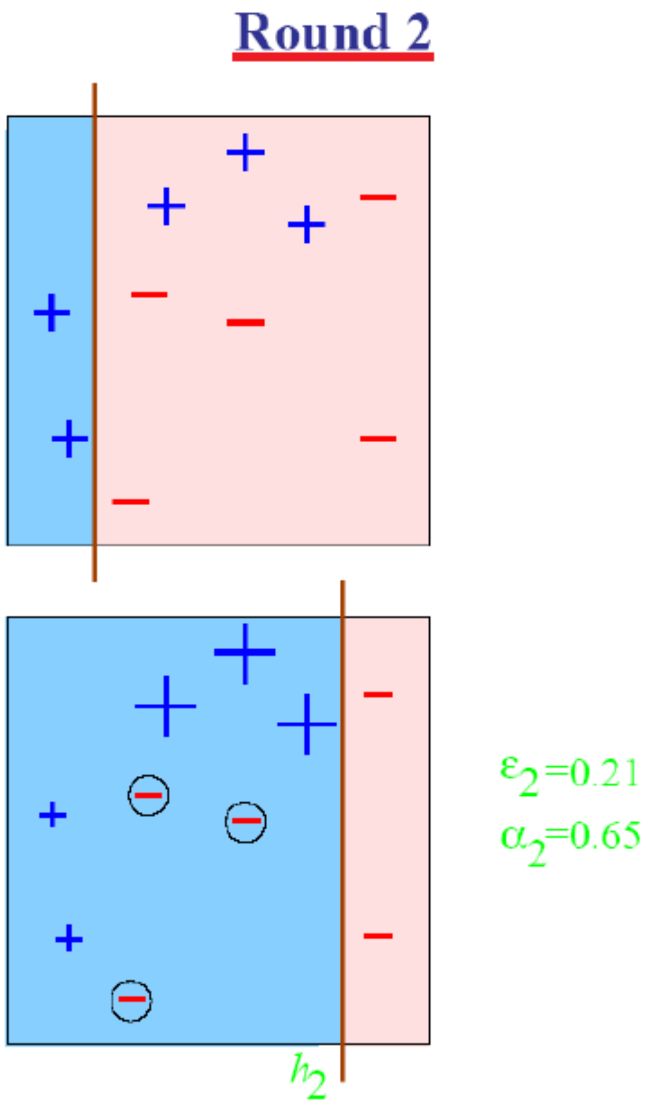
$$\begin{aligned} &if \quad h_t(\mathbf{x}_i) = y_i \Longrightarrow A = e^{-\alpha_t} \\ &if \quad h_t(\mathbf{x}_i) \neq y_i \Longrightarrow A = e^{\alpha_t} \end{aligned}$$

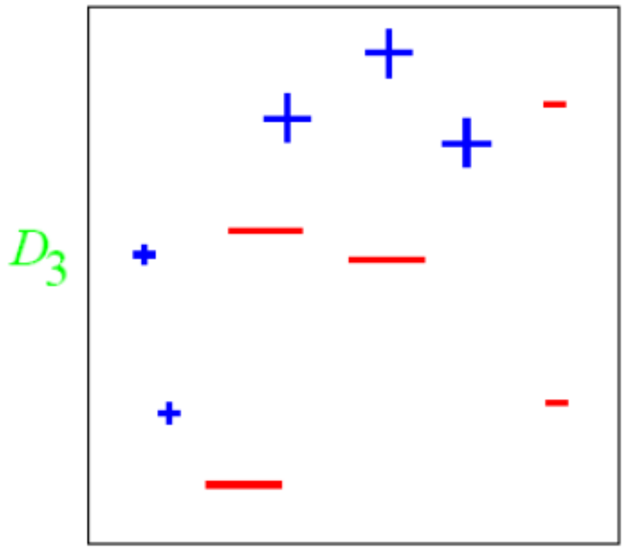
Boosting for classification



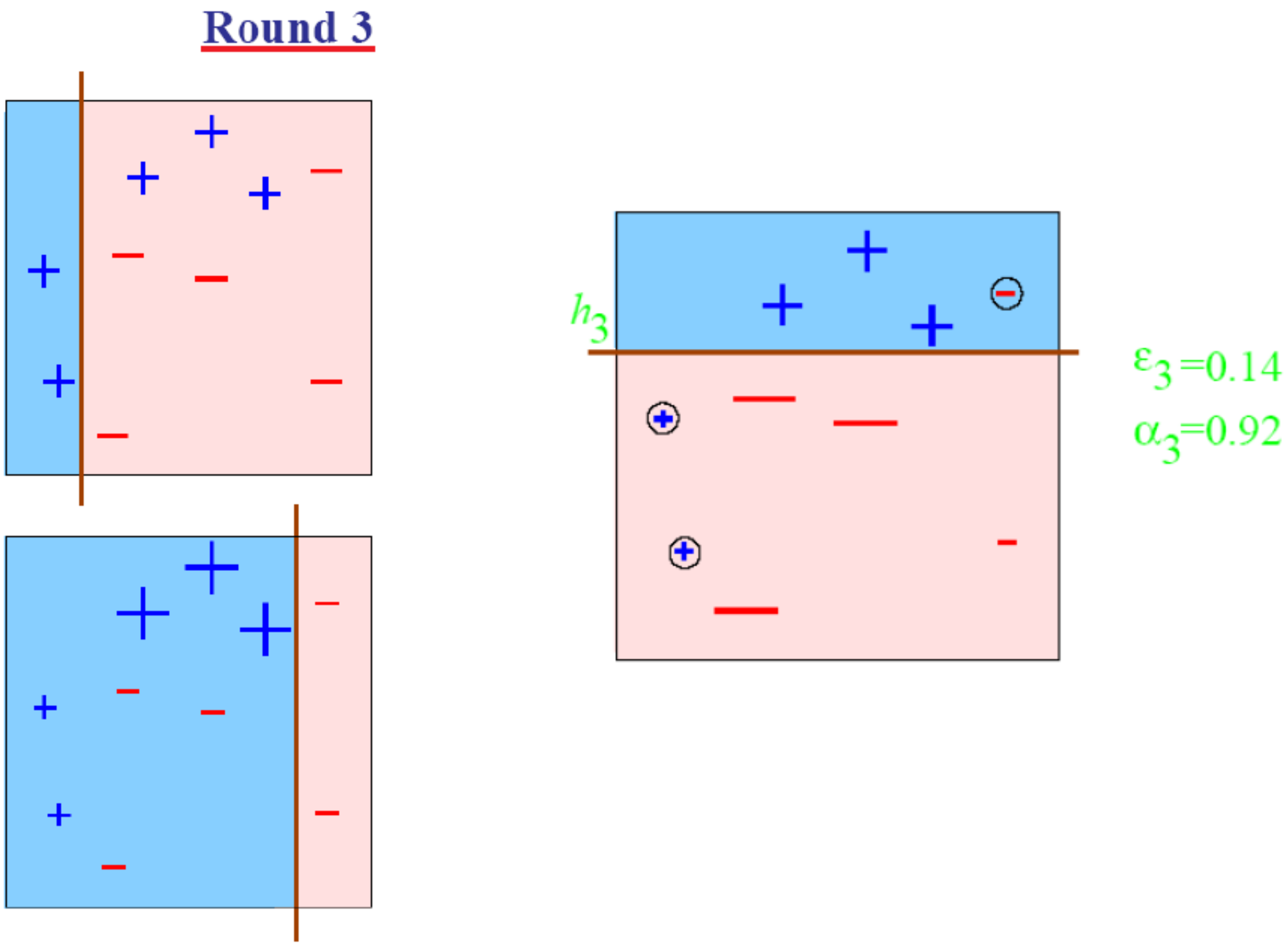


Boosting for classification

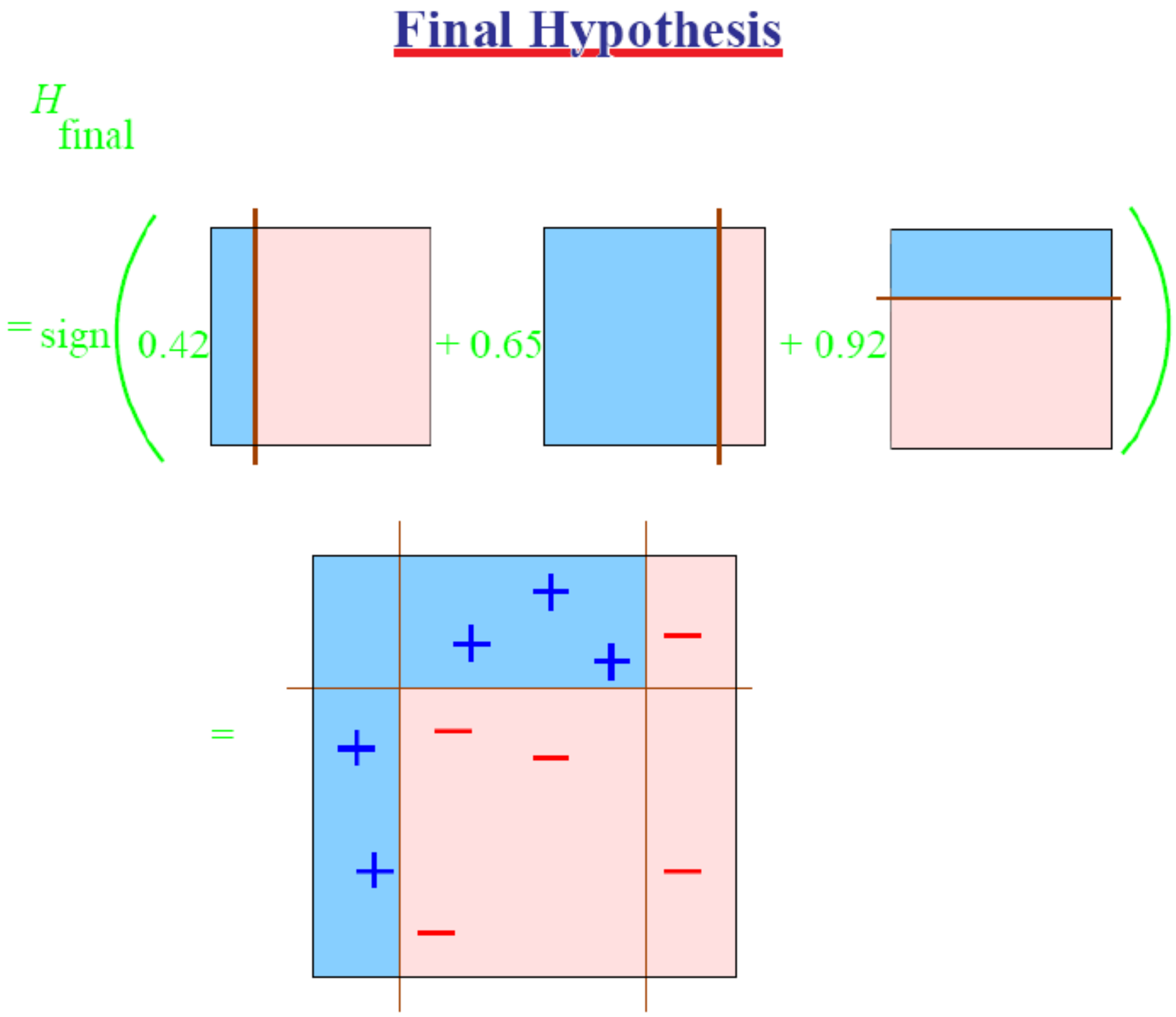




Boosting for classification



Boosting for classification



Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are often not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Summary

Manuel Fernandez-Delgado, Eva Cernadas, Senen Barro, Dinani Amorim, Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?, Journal of Machine Learning Research 15 (2014) 3133-3181

“We evaluate 179 classifiers arising from 17 families (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), …

We use 121 data sets, which represent the whole UCI data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classier behavior, not dependent on the data set collection.

The classifiers most likely to be the bests are the random forest (RF) versions, the best of which achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel, which achieves 92.3% of the maximum accuracy.