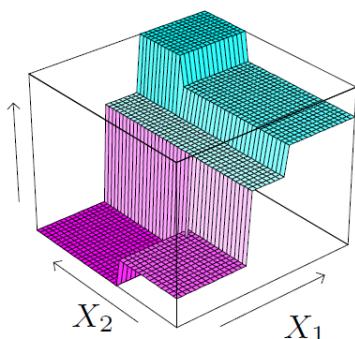
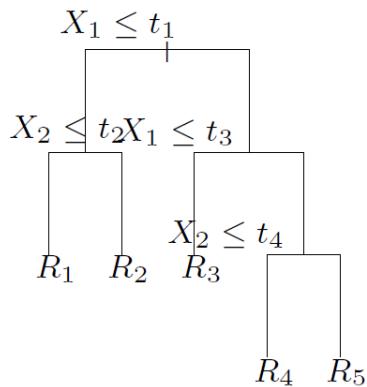
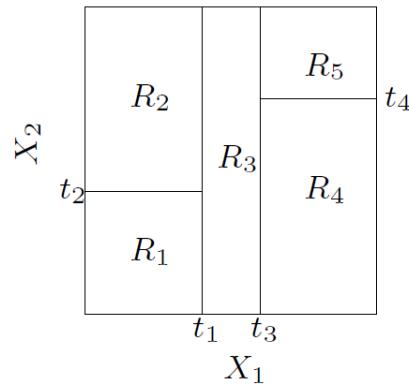
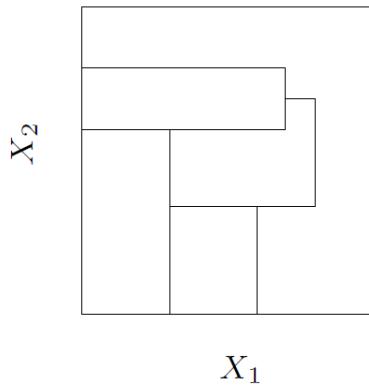


# Decision Trees

Nicolás Pérez de la Blanca

DECSAI

# Model: Partitioning the input space



This plot represents  
the node means

There exists different ways of partitioning the input space using hyperplanes parallel to the axis.

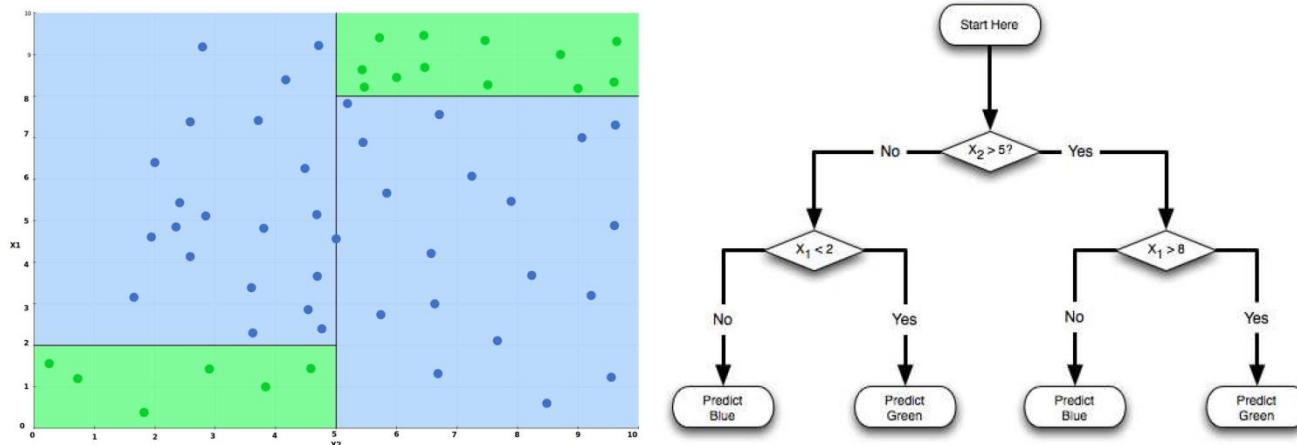
By simplicity we restrict our attention to recursive partitions

Binary partitions represent the most popular models

$$f(\mathbf{X}) = \sum_{i=1}^5 c_i \mathbb{I}[(X_1, X_2) \in R_m]$$

# Decision Tree Learning: Example 1

- Identifying the region (blue or green) a point lies in
  - A classification problem (blue vs green)
  - Each input has 2 features: co-ordinates  $\{x_1, x_2\}$  in the 2D plane
  - Left: Training data, Right: A decision tree constructed using this data

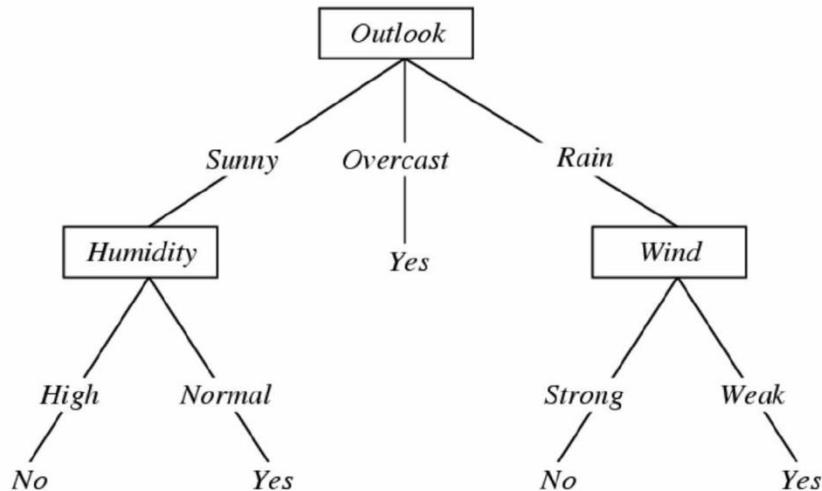


- The DT can be used to predict the region (blue/green) of a new test point
  - By testing the features of the test point
  - In the order defined by the DT (first  $x_2$  and then  $x_1$ )

# Decision Tree Learning: Example 2

- Deciding whether to play or not to play Tennis on a Saturday
  - A classification problem (play vs no-play)
  - Each input has 4 features: Outlook, Temperature, Humidity, Wind
  - Left: Training data, Right: A decision tree constructed using this data

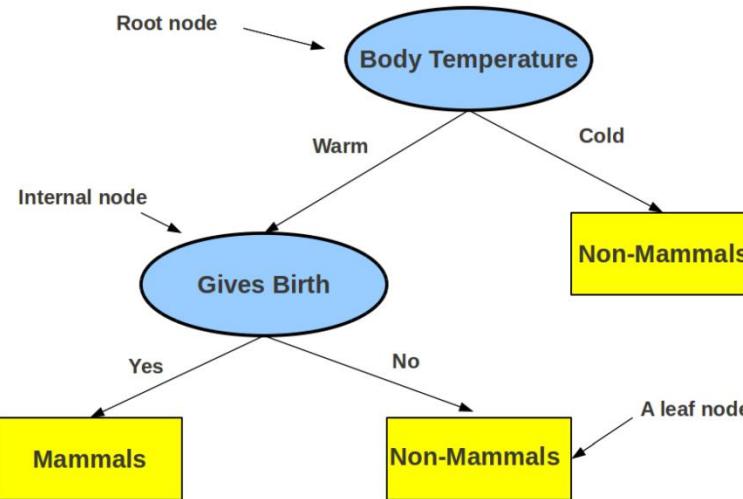
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- The DT can be used to predict play vs no-play for a new Saturday
  - By testing the features of that day
  - In the order defined by the DT

# Decision Tree

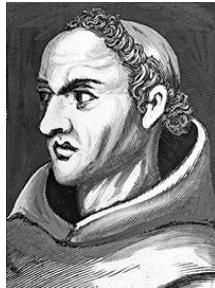
- Defined by a **hierarchy** of rules (in form of a tree)



- Rules form the **internal nodes** of the tree (topmost internal node = **root**)
- Each rule (internal node) tests the value of some property the data
- Decision Tree Learning
  - Training data is used to construct the Decision Tree (DT)
  - The DT is used to predict label **y** for test input **x**

# How large to grow a tree ?

- The standard splitting rule is :  $\mathbf{1}_{[x_i < \theta]}$  where  $i \in [d]$  is the index of the relevant feature and  $\theta \in \mathbb{R}$  is the threshold.
- A tree with  $k$  leaves can shatter a set of  $k$  instances. Hence trees of arbitrary size can provide hypothesis set of infinity VC dimension.
- In order to overcome this obstacle a selection criteria is applied ( Occam Razor)



Entities should not be multiplied beyond necessity, “Occam’s razor” principle attributed to William of Occam c. 1280–1349

We should seek simpler models over complex ones and optimize the tradeoff between model complexity and the accuracy of model’s description of the training data

# How to bound the generalization error?

## Minimum Description Length (MDL)

### prior knowledge

$\mathcal{H}$  is a countable hypothesis class

$\mathcal{H}$  is described by a prefix-free language over {0,1}

For every  $h \in \mathcal{H}$ ,  $|h|$  is the length of the representation of  $h$

**input:** A training set  $S \sim \mathbb{P}^N$ , confidence  $\delta$

$$\text{output: } h \in \operatorname{argmin}_{h \in \mathcal{H}} \left[ E_{in}(h) + \sqrt{\frac{|h| + \ln(\frac{2}{\delta})}{2N}} \right]$$

Ejemplo: let  $\mathcal{H}$  be the class of all predictors that can be implemented using a programming language, say C++. Let us represent each program using the binary string obtained running the gzip command on the program. Then,  $|h|$  is simply the length (in bits) of the output of gzip when running on the C++ program corresponding to  $h$ .

Assuming  $\mathcal{X} = \{0,1\}^d$ , then with probability of at least  $1-\delta$ , on a sample of size  $N$ , for every  $n$  and every  $h \in \mathcal{H}$  with  $n$  nodes it holds that

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{(n+1) \log_2(d+3) + \log(\frac{2}{\delta})}{2N}}$$

# Regression Trees

- To grow a regression tree, the algorithm needs to automatically decide the splitting variables, the split values and what topology (shape) the tree should have.
- Suppose first that we have a partition into  $M$  regions  $R_1, R_2, \dots, R_M$  and we model the response as a constant  $c_i$  in each region

$$f(\mathbf{x}) = \sum_{i=1}^M c_i \mathbb{I}[\mathbf{x} \in R_i]$$

- If we estimate the constants  $c_i$  minimizing  $\sum(y_i - f(x_i))^2$  is easy see that the best  $\hat{c}_i$  is just the average of  $y_j$  in region  $R_i$
- BUT finding the best partitioning in terms of minimum sum of squares is computationally infeasible. Hence a greedy algorithm is applied:
  - For each variable fix a set of splitting points
  - Look for the variable and splitting point that minimize some splitting criterion on the node data
  - Repeat the last step recursively on each node

# Cost-Complexity Criterion

- The best strategy is to grow the largest possible (stop when the number of items in a node is below a threshold) and to prune the results using a cost-complexity criteria.
- Let's denote by  $T$  a tree and let  $|T|$  be the number of terminal nodes in  $T$ . Letting

$$\begin{aligned}N_m &= \#\{\mathbf{x}_i \in R_m\} \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2\end{aligned}$$

We define the cost complexity criterion as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

# Classification Trees

- This is a multivalued model that allows classification in K classes,  $K > 2$
- The splitting criteria on each node has to be different than in regression
- In a node  $m$  representing a region  $R_m$  with  $N_m$  samples, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}[y_i = k]$$

the proportion of class  $k$  samples in node  $m$ .

- We classify the samples in node  $m$  to class  $k(m) = \arg \max_k \hat{p}_{mk}$ , the majority class in node  $m$ .
- Different measures  $Q_m(T)$  of node impurity are:
  - Misclassification error:  $1 - \hat{p}_{m k(m)}$ ,  $m$  is the majority class
  - Gini index:  $\sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$  (training error of classify item to a class with its probability)
  - Entropy (cross-entropy):  $-\sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$
- In binary classification the three criteria are equivalents.

# Decision Tree Construction

Summarizing:

- The training data is used to construct the DT
- Each internal node is a rule (testing the value of some feature)
- Highly informative features are placed higher up in the tree
- We need a way to rank features according to their information content
- We will use **Entropy** and **Information Gain** as the criteria
- Note: There are several specific versions of the Decision Tree
  - ID3, C4.5, Classification and Regression Trees (CART), etc.
  - We will be looking at the ID3 algorithm

# Entropy

- Entropy measures the **randomness/uncertainty** in the data
- Let's consider a set  $S$  of examples with  $C$  many classes. Entropy of this set:

$$H(S) = - \sum_{c \in C} p_c \log_2 p_c$$

- $p_c$  is the probability that an element of  $S$  belongs to class  $c$ 
  - .. basically, the fraction of elements of  $S$  belonging to class  $c$
- Intuition: Entropy is a measure of the “degree of surprise”
  - Some dominant classes  $\implies$  small entropy (less uncertainty)
  - Equiprobable classes  $\implies$  high entropy (more uncertainty)
- Entropy denotes the average number of bits needed to encode  $S$

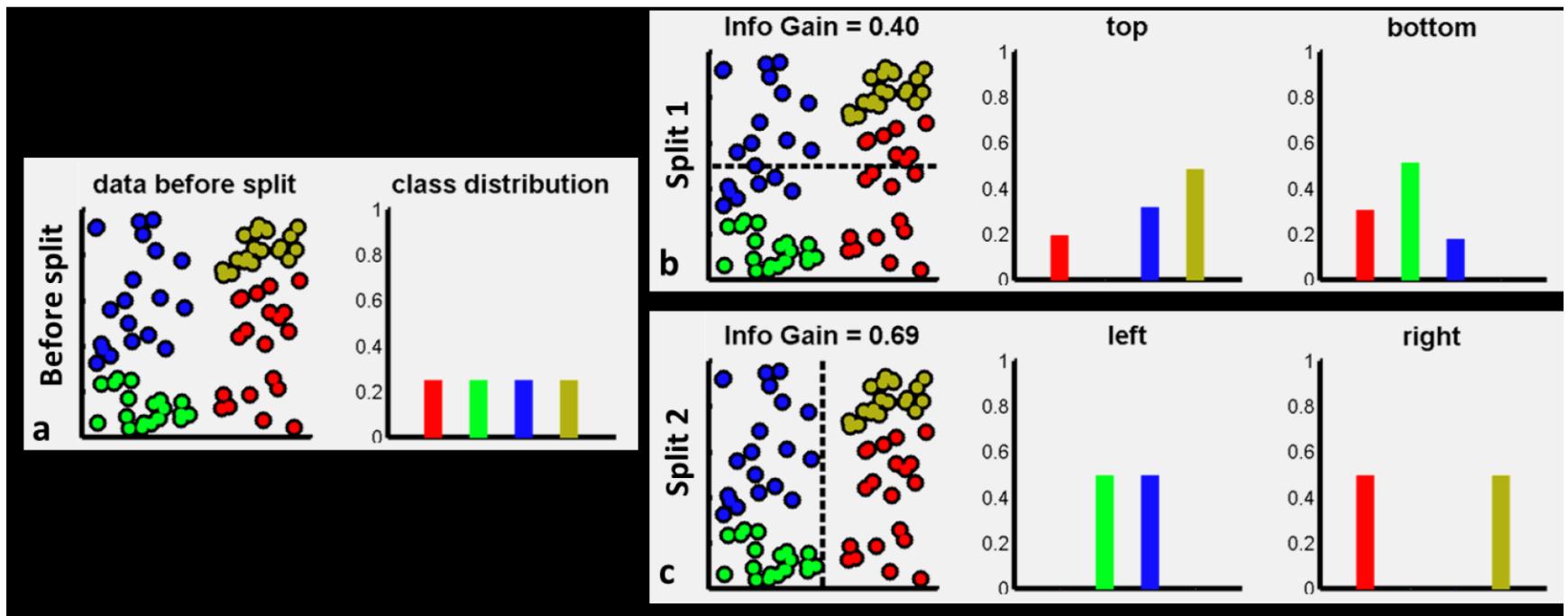
# Information Gain

- Let's assume each element of  $S$  consists of a set of features
- Information Gain (IG) on a feature  $F$

$$IG(S, F) = H(S) - \sum_{f \in F} \frac{|S_f|}{|S|} H(S_f)$$

- $S_f$  number of elements of  $S$  with feature  $F$  having value  $f$
- $IG(S, F)$  measures the increase in our certainty about  $S$  once we know the value of  $F$
- $IG(S, F)$  denotes the number of bits saved while encoding  $S$  once we know the value of the feature  $F$

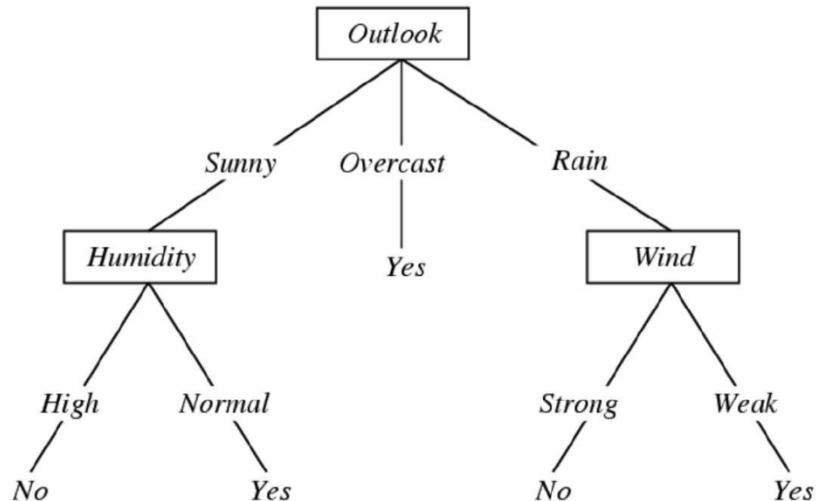
# Information Gain



# Decision Tree Construction

- Now let's look at the Tennis playing example

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question:** Why does it make more sense to test the feature "outlook" first?
- Answer:** Of all the 4 features, it's most informative
- We will see shortly how to quantify the informativeness
- Information content** of a feature decides its position in the DT

# Computing Information Gain

- Let's begin with the root node of the DT and compute  $IG$  of each feature
- Consider feature "wind"  $\in \{\text{weak, strong}\}$  and its  $IG$  w.r.t. the root node

1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

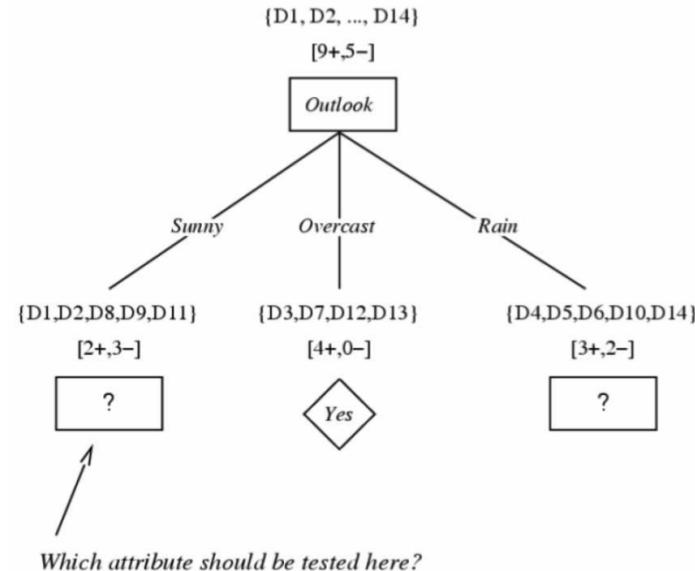
- Root node:  $S = [9+, 5-]$  (all training data: 9 play, 5 no-play)
- Entropy:  $H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$
- $S_{\text{weak}} = [6+, 2-] \implies H(S_{\text{weak}}) = 0.811$
- $S_{\text{strong}} = [3+, 3-] \implies H(S_{\text{strong}}) = 1$

$$\begin{aligned} IG(S, \text{wind}) &= H(S) - \frac{|S_{\text{weak}}|}{|S|} H(S_{\text{weak}}) - \frac{|S_{\text{strong}}|}{|S|} H(S_{\text{strong}}) \\ &= 0.94 - 8/14 * 0.811 - 6/14 * 1 \\ &= 0.048 \end{aligned}$$

# Choosing the most informative feature

- At the root node, the information gains are:
  - $IG(S, \text{wind}) = 0.048$  (we already saw)
  - $IG(S, \text{outlook}) = 0.246$
  - $IG(S, \text{humidity}) = 0.151$
  - $IG(S, \text{temperature}) = 0.029$
- “outlook” has the maximum  $IG \implies$  chosen as the root node

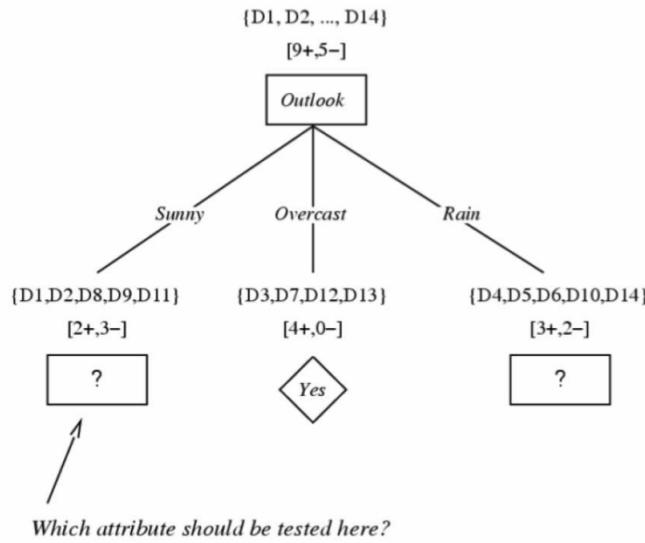
- Growing the tree:
  - Iteratively select the feature with the highest information gain for each child of the previous node



# Growing The Tree

- How to decide which feature to choose as we descend the tree?
- Rule:** Iterate - for each child node, select the feature with the highest IG

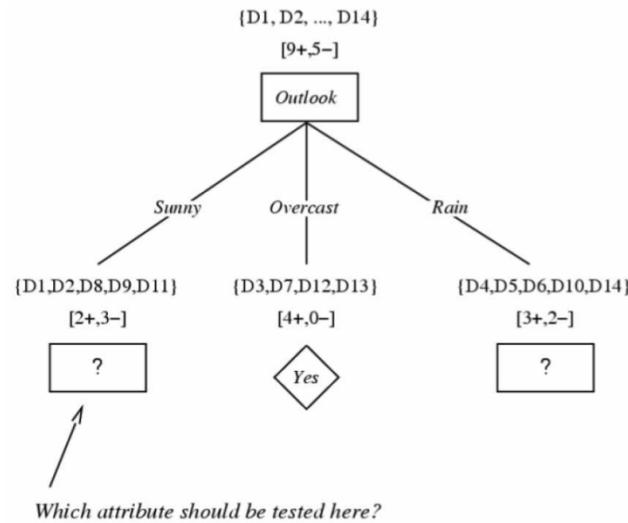
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- For level-2, left node:  $S = [2+, 3-]$  (days 1,2,8,9,11)
- Let's compute the Information Gain for each feature (except **outlook**)
- The feature with the highest Information Gain should be chosen for this node

# Growing The Tree

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



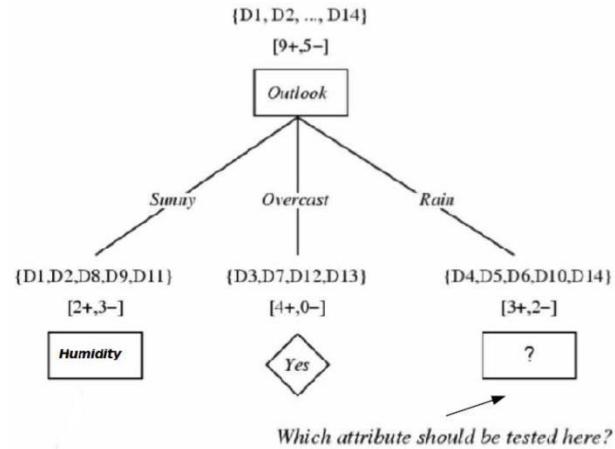
- For this node ( $S = [2+, 3-]$ ), the *IG* for the feature **temperature**:

$$IG(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)$$

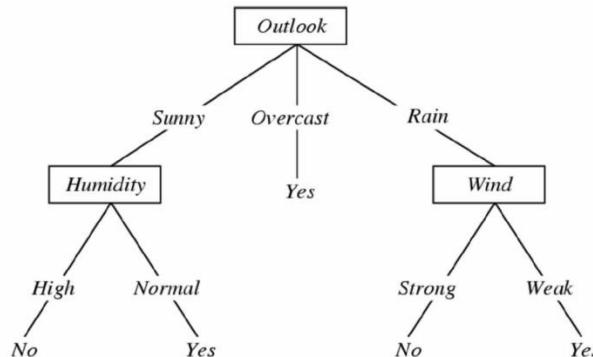
- $S = [2+, 3-] \implies H(S) = -(2/5) * \log_2(2/5) - (3/5) * \log_2(3/5) = 0.971$
- $S_{\text{hot}} = [0+, 2-] \implies H(S_{\text{hot}}) = -0 * \log_2(0) - (2/2) * \log_2(2/2) = 0$
- $S_{\text{mild}} = [1+, 1-] \implies H(S_{\text{mild}}) = -(1/2) * \log_2(1/2) - (1/2) * \log_2(1/2) = 1$
- $S_{\text{cool}} = [1+, 0-] \implies H(S_{\text{cool}}) = -(1/1) * \log_2(1/1) - (0/1) * \log_2(0/1) = 0$
- $IG(S, \text{temperature}) = 0.971 - 2/5 * 0 - 2/5 * 1 - 1/5 * 0 = 0.570$
- Likewise we can compute:  $IG(S, \text{humidity}) = 0.970$ ,  $IG(S, \text{wind}) = 0.019$
- Therefore, we choose “humidity” (with highest  $IG = 0.970$ ) for the level-2 left node

# Growing The Tree

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

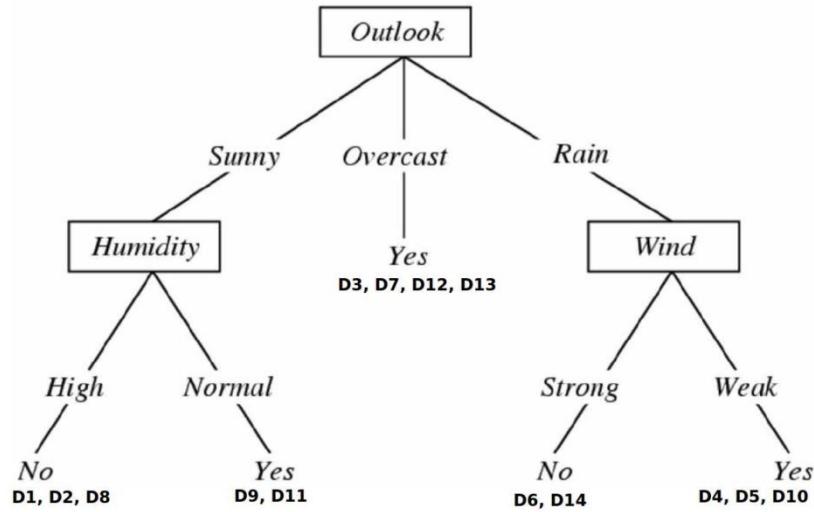


- Level-2, middle node: no need to grow (already a leaf)
- Level-2, right node: repeat the same exercise!
  - Compute  $IG$  for each feature (except outlook)
  - **Exercise:** Verify that wind has the highest  $IG$  for this node
- Level-2 expansion gives us the following tree:



# Growing The Tree: Stopping Criteria

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Stop expanding a node further when:
  - It consists of examples all having the same label
  - Or we run out of features to test!

# Decision Tree Algorithm (ID3)

A recursive algorithm:

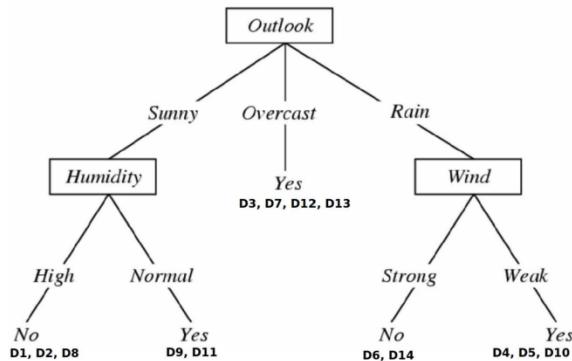
$\text{DT}(\text{Examples}, \text{Labels}, \text{Features})$ :

- If all examples are positive, return a single node tree  $\text{Root}$  with label = +
- If all examples are negative, return a single node tree  $\text{Root}$  with label = -
- If **all features exhausted**, return a single node tree  $\text{Root}$  with **majority label**
- Otherwise, let  $F$  be the feature having the highest information gain
- $\text{Root} \leftarrow F$
- For each possible value  $f$  of  $F$ 
  - Add a tree branch below  $\text{Root}$  corresponding to the test  $F = f$
  - Let  $\text{Examples}_f$  be the set of examples with feature  $F$  having value  $f$
  - Let  $\text{Labels}_f$  be the corresponding labels
  - If  $\text{Examples}_f$  is empty, add a leaf node below this branch with label = most common label in  $\text{Examples}$
  - Otherwise, add the following subtree below this branch:  
$$\text{DT}(\text{Examples}_f, \text{Labels}_f, \text{Features} - \{F\})$$

- Note:  $\text{Features} - \{F\}$  removes feature  $F$  from the feature set  $\text{Features}$

# Overfitting in Decision Trees

- What if we added a noisy example in our Tennis Playing dataset?
- Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, Play=No
- This Play=No example would be grouped with the node D9, D11 (both Play=Yes)

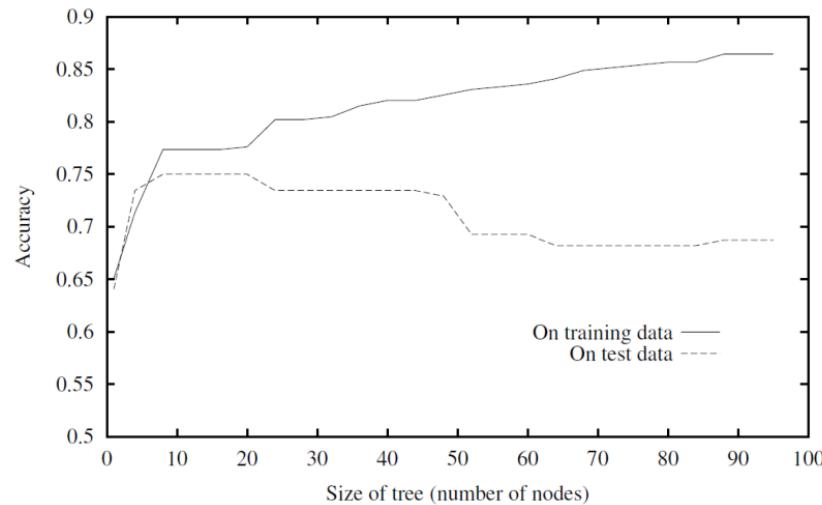


- This node will need to be expanded by testing some other feature
- The new tree would be **more complex** than the earlier one (trying to fit noise)
- **The extra complexity may not be worth it**  $\Rightarrow$  may lead to overfitting if the test data follows the same pattern as our **normal** training data
- **Note:** Overfitting may also occur if the training data is not sufficient

# Overfitting in Decision Trees

Minimum Description Length Criteria:  $E_{out}(h) \leq E_{in}(h) + \mathcal{O}\left(\frac{\#\text{nodes}}{N}\right)^{1/2}$

- Overfitting Illustration



Clearly there is a tradeoff between the size of the tree (#nodes) and  $E_{in}(h)$

# Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Mainly two approaches
  - Prune while building the tree (**stopping early**)
  - Prune after building the tree (**post-pruning**)
- Criteria for judging which nodes could potentially be pruned
  - Use a **validation set** (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - **Greedily remove** the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
  - Statistical tests such as the  $\chi^2$  **test** (Quinlan, 1986)

# Dealing with Missing Features

- Want to compute  $IG(S, F)$  for feature  $F$  on a (sub)set of training data  $S$
- What if a training example in  $\mathbf{x}$  in  $S$  has feature  $F$  missing?
- We will need some way to *approximate* the value of this feature for  $\mathbf{x}$
- **One way:** Assign the value of  $F$  which a majority of elements in  $S$  have
- **Another (maybe better?) way:** Assign the value of  $F$  which a majority of elements in  $S$  **with the same label as  $\mathbf{x}$**  have

# Decision Tree Extension

- Real-valued features can be dealt with using thresholding
- Real-valued labels (Regression Trees) by re-defining entropy or using other criteria (how similar to each other are the  $y$ 's at any node)
- Other criteria for judging feature informativeness
  - Gini-index, misclassification rate

# BAGGING & RANDOM FORESTS

---

# Index

## ➤ Bagging

- Bootstrapping
- Regression Trees Bagging
- Classification Trees Bagging
- “Out-of-Bag” error estimation
- Importance of the fatures: Dibujos de influencia relativa

## ➤ Random Forests

# BAGGING

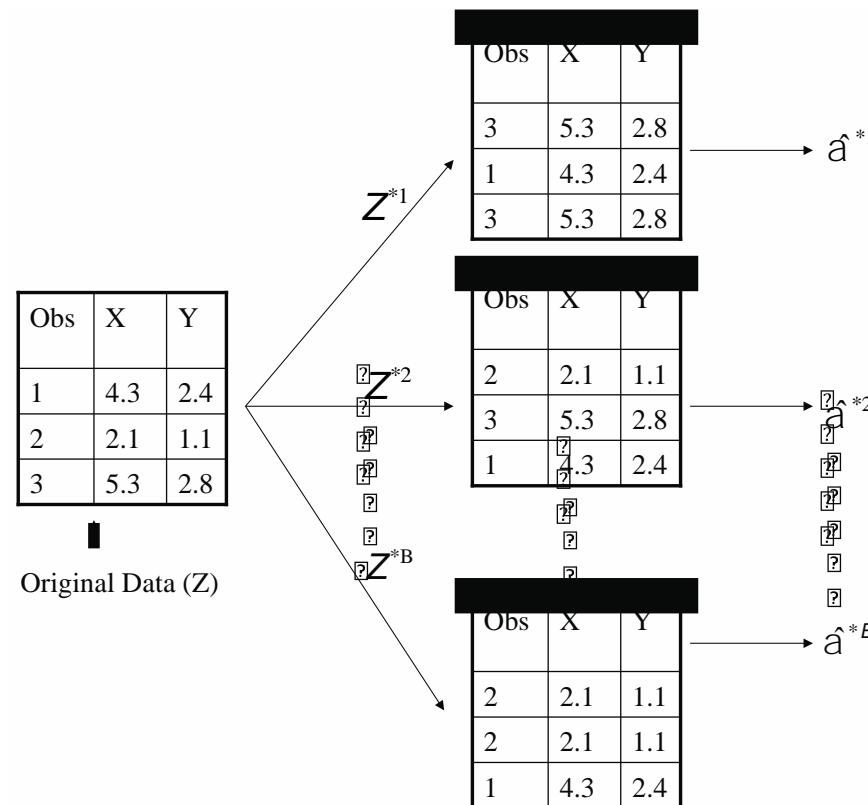
---

# Bootstrap

- Bootstrap is a statistic approach to compute the uncertainty of an estimate or a learning method.
  - For instance regression coefficients uncertainty
- The Bootstrap usefulness is based on its general applicability including those cases in which do not exist enough data to compute parameters
- EJEMPLO: Sean X e Y v.a y deseamos conocer la variabilidad del valor de  $\alpha$  que minimiza  $\text{Var}(\alpha X + (1 - \alpha)Y)$
- Podemos ver que el valor de  $\alpha$  que minimiza dicha expresión es
$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_Y^2 + \sigma_X^2 - 2\sigma_{XY}}$$
- Las varianzas y covarianzas pueden ser estimadas desde datos.
- ¿Cómo calcular la incertidumbre de  $\alpha$  con una sola muestra ?

# Bootstrapping is a simple idea!

- Random subsampling with replacement of the original dataset to create new datasets of the same size.



# In detail

- Sample with replacement from the original dataset :
    - Get B-samples datasets
    - Get an estimate of the parameter from each B-samples dataset
    - Compute the variability of the parameter estimation
    -
- The obtained value can be used as an estimate of the variability of the population parameter.

# Problem!

- Decision trees show a high variance!
  - Different dataset from the same population can give very different results. Let consider to split the original training dataset and fitting a tree to each part.
- Stable model are the goal
- Bagging (bootstrap aggregating) is a cure for this illness

# ¿What's Bagging and how works?

- We combine two ideas:
  - Bootstrapping: many different training dataset!
  - Average: implies variance reduction!
  - Let  $Z_1, \dots, Z_n$ , be i.i.d random variable with variance  $\sigma^2$  , then the variance of the average  $\bar{Z}$  is  $\sigma^2/n$
- How works
  - Sampling B bootstrapping training datasets
  - Trainig a model on each dataset
  - Ensamble:
    - Regression: average of the prediction of the models
    - Classification: majority vote

# Bagging con Árboles de regresión

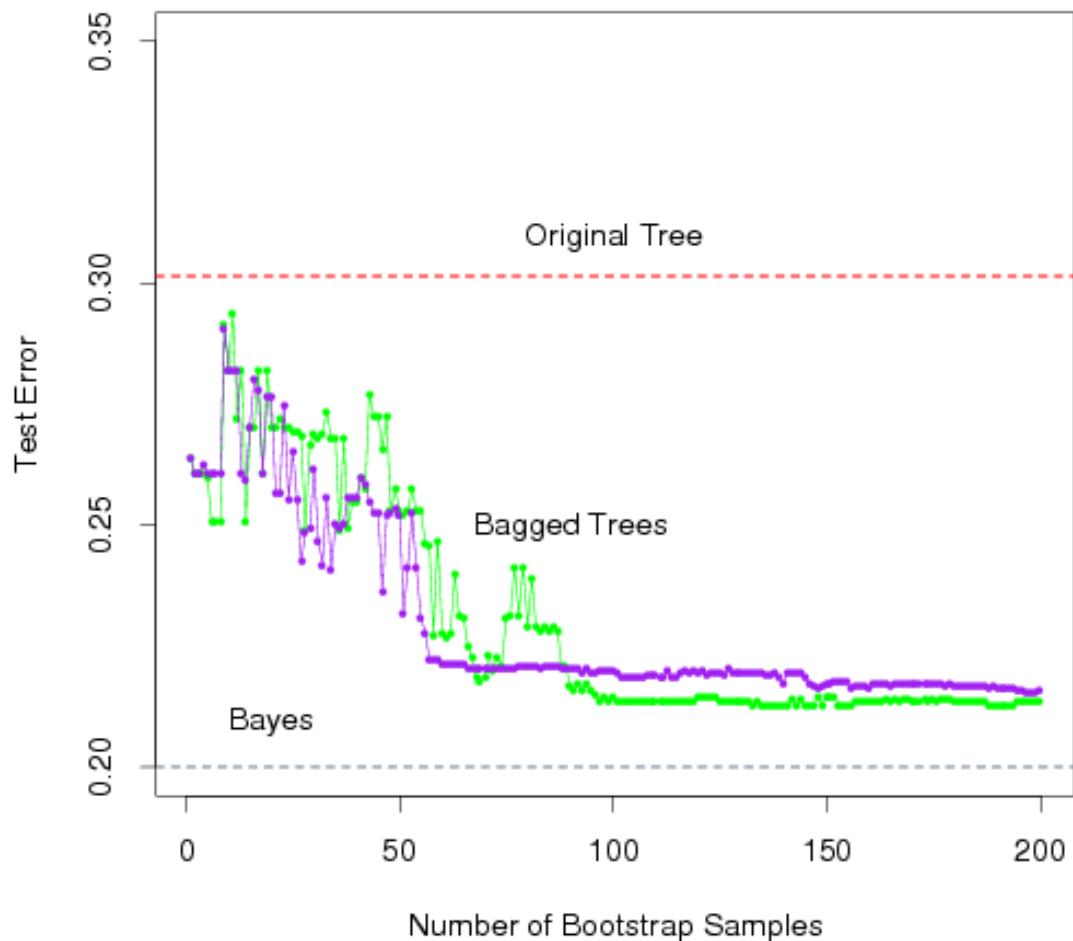
- Construye  $B$  árboles de regresión usando  $B$  conjuntos de entrenamiento extraídos con bootstrapping. Promedia las predicciones resultantes.
- Nota: Los árboles no están podados, por tanto cada árbol tiene una alta varianza pero bajo sesgo. El promedio de los árboles reduce la varianza y finalmente tendremos bajo sesgo y varianza ☺

# Bagging con Árboles de Clasificación

- Construye  $B$  árboles usando  $B$  conjuntos de entrenamiento extraídos con bootstrapping.
- Para predicción existen dos aproximaciones:
  1. Anotar la clase predicha por cada árbol (conjunto de datos) y asignar a la clase mayoritaria (mayoría de votos).
  2. Si el clasificador produce estimación de probabilidades, podemos promediar las probabilidades y entonces predecir la clase con mayor probabilidad.
- Ambos métodos funcionan bien.

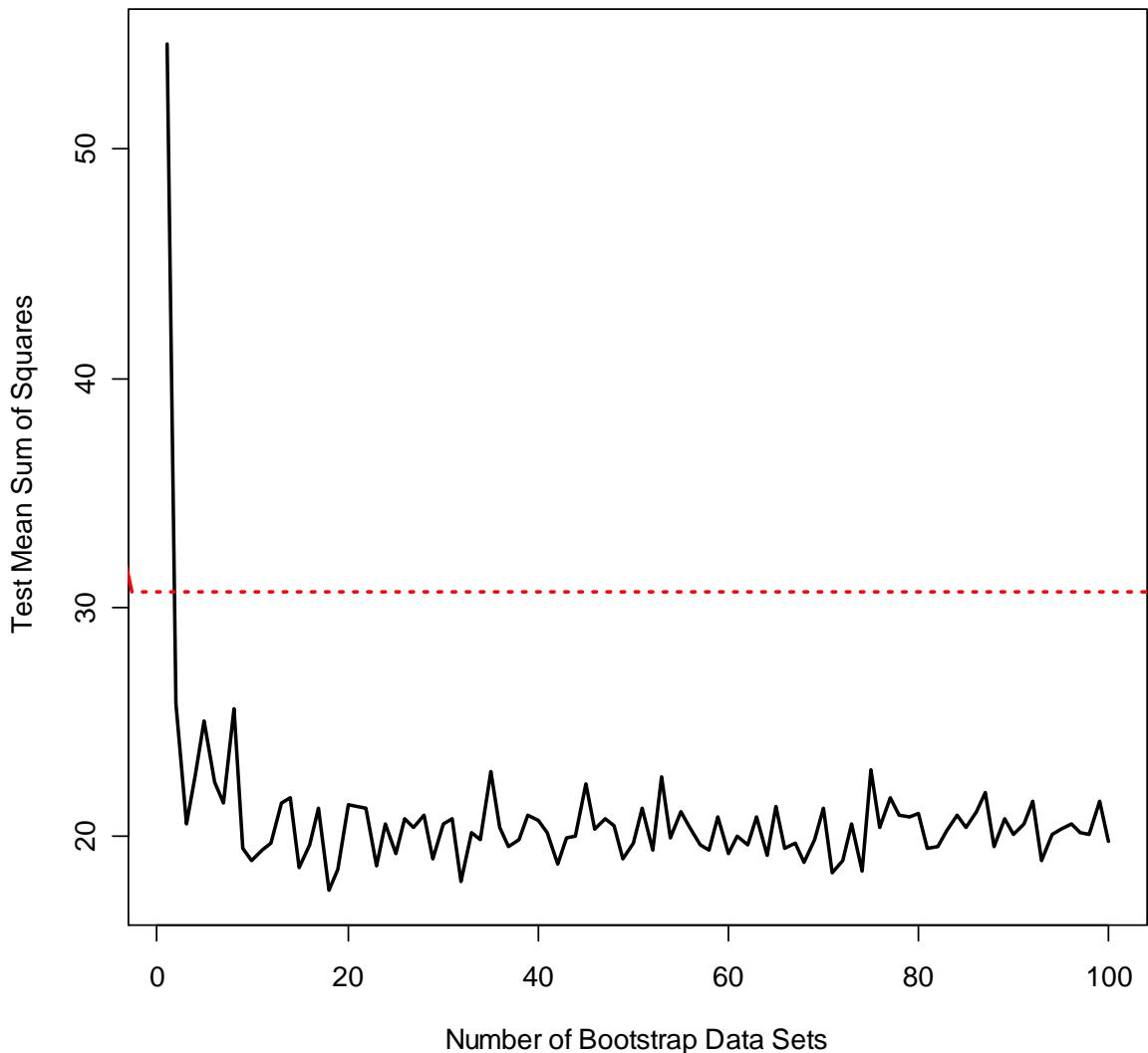
# Una comparación de error de test

- Aquí la línea verde representa el voto mayoritario
- La linea púrpura representa la media de las probabilidades.
- Ambas se comportan mucho mejor que un único árbol (roja a trozos) y están muy cerca del error de Bayes (gris a trozos) para un número de árboles suficiente



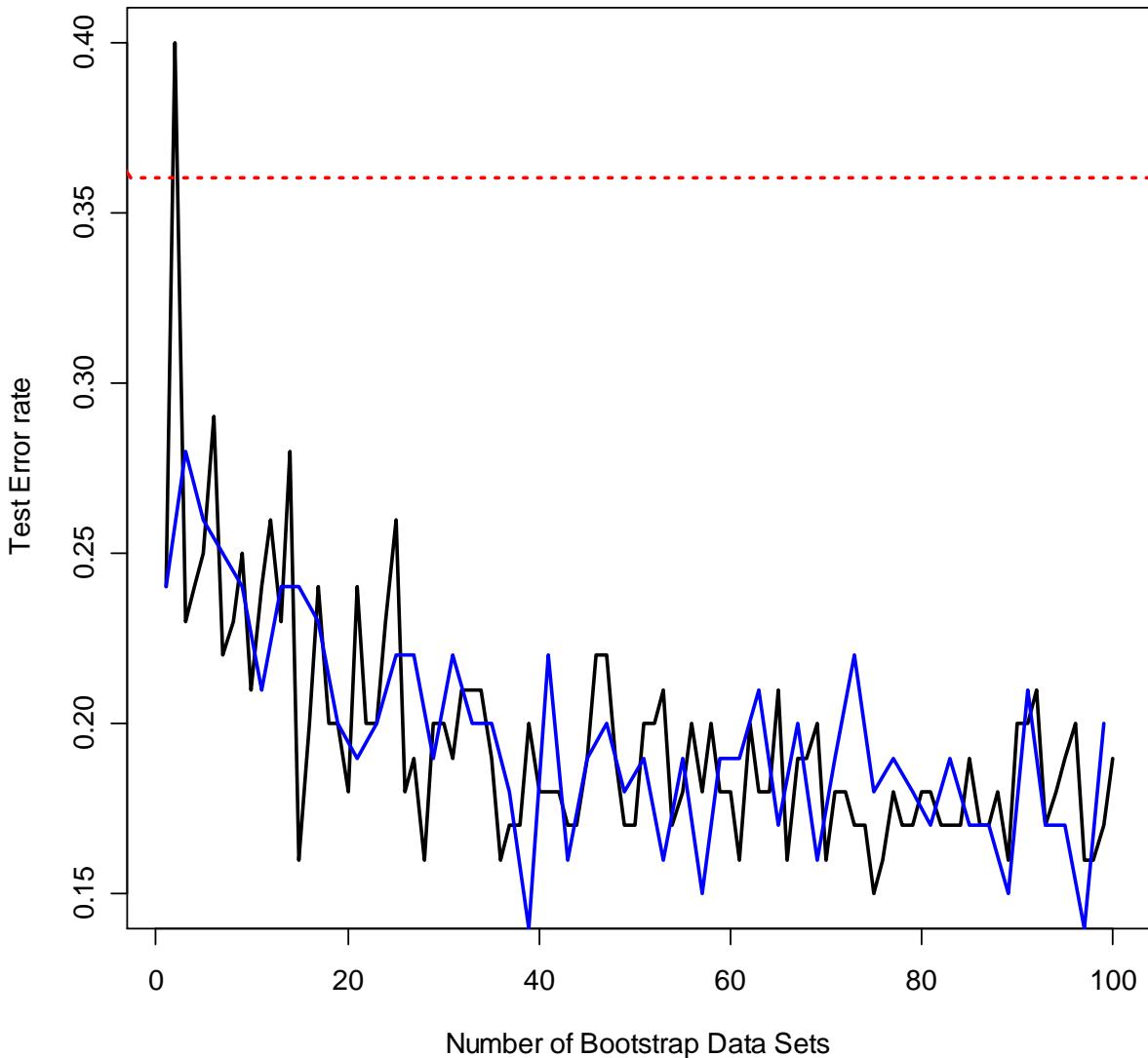
# Ejemplo 1: Housing Data

- La línea roja representa la media de la suma de cuadrados del test usando un único árbol
- La línea negra corresponde a la tasa de error de bagging
- 



# Ejemplo 2: Car Seat Data

- La **línea roja** representa la tasa de error del test con un **único árbol**.
- La **línea negra** corresponde a la tasa de error de bagging usando voto mayoritario mientras que la **línea azul** es con el **promedio de las probabilidades**



# Estimación del Error Out-of-Bag

- Ya que bootstrap significa remuestrear los datos originales, aquellos datos que no participan de la muestra de entrenamiento pueden ser usados para test.
- En promedio, cada árbol de bootstrap hace uso de  $2/3$  of las observaciones, por tanto tendremos  $1/3$  de las observaciones para verificación

# Medida de la Importancia de las Variables

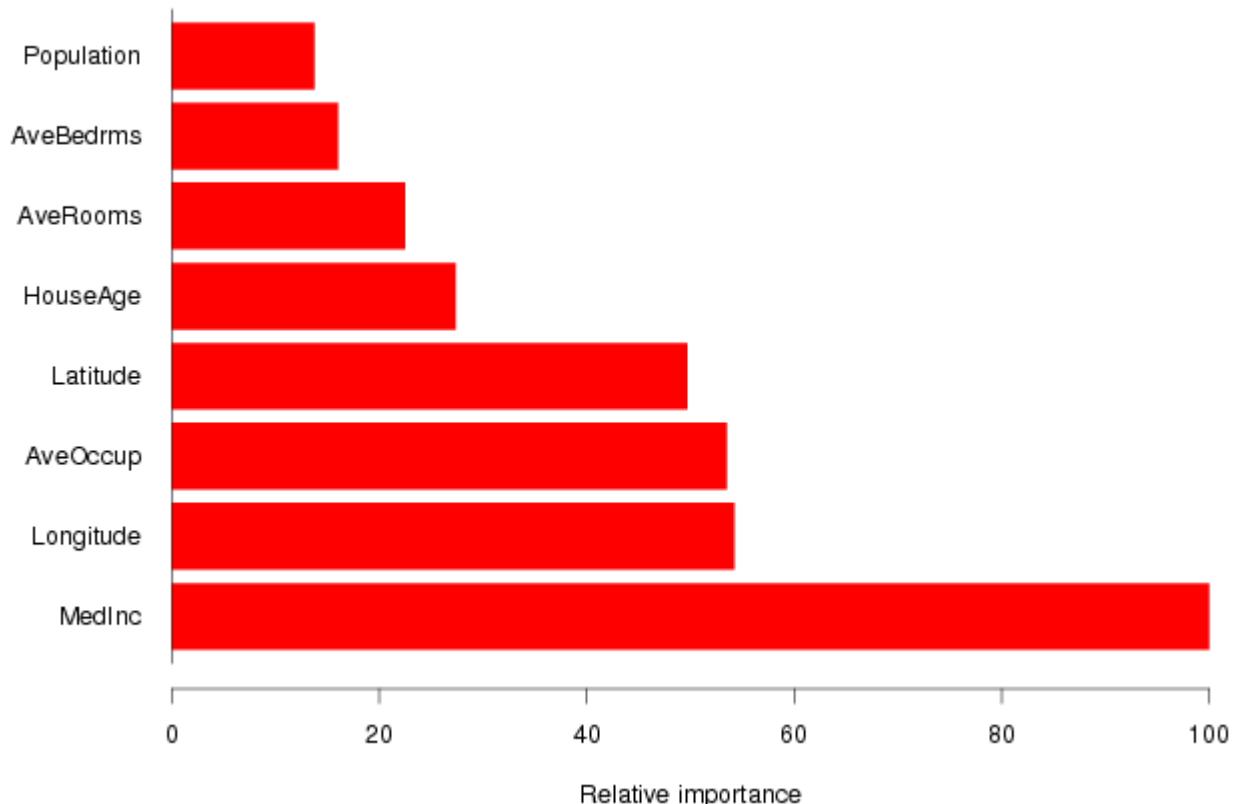
- Bagging normalmente mejora la ratio de predicción en comparación con un único árbol, pero es un modelo difícil de interpretar!
- Tenemos cientos de árboles, y no está claro que variables son las más importantes
- Por tanto bagging mejora la predicción a costa de interpretabilidad
- Pero, podemos hacer un resumen global de la importancia de cada predictor usando Gráficos de Importancia Relativa

# Gráficos de Importancia Relativa

- ¿Como decidir que variables son más útiles para predecir una respuesta ?
- Podemos calcular Gráficos de Importancia Relativa.
  - Estos gráficos nos dan un valor para cada variable.
  - Estos valores representan el decrecimiento en MSE cuando se divide por una variable particular
  - Un número cercano a cero indica que la variable no es importante y podría eliminarse.
  - A mayor valor mayor influencia en el modelo.

# Ejemplo: Housing Data

- Median Income (MedInc) es la variable más importante.
- Longitude, Latitude y Average occupancy (AveOccup) son las siguientes más importantes



# RANDOM FORESTS

---

ISLR, capítulo-8

# Random Forests

- Es un predictor muy eficiente
- Esta construido sobre la idea de bagging, pero aporta una mejora ya que construye árboles no correlados
- ¿Como funciona?
  - Construye un número de árboles de decisión sobre muestras de bootstrap,
  - Estima los árboles, ( las variables para las ramificaciones, m, como los valores para las mismas ) de forma aleatoria de entre los p predictores .
  - En general se eligen  $m \gg \sqrt{p}$  para clasificación y  $p/3$  para regresión

## ¿Porque considerar una muestra aleatoria de m predictores en lugar de todos los predictores para la partición ?

- Si existe un predictor muy fuerte en el conjunto de datos entonces, en la colección de árboles del bagging, muchos de ellos usarán dicho predictor para la primera partición
  - Todos los árboles del bagging se parecerán. Por tanto todas las predicciones estarán altamente correladas
- El promedio de cantidades altamente correladas no reduce mucho su varianza, por tanto “random forest” decorrela los árboles de bagging para obtener una mayor reducción en varianza

$$\sigma^2 \left( \rho + \frac{(1 - \rho)}{B} \right)$$

---

**Algorithm 15.1** Random Forest for Regression or Classification.

1. For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$ .

To make a prediction at a new point  $x$ :

$$\text{Regression: } \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

# Random Forest con diferentes valores de “m”

- Ejemplo de tasa de error para distintos criterios de selección de predictores
- Notar que cuando  $m=p$  simplemente obtenemos bagging
- El error de un árbol único es de 45.7%
- El error de la clase más probable 75.4%

