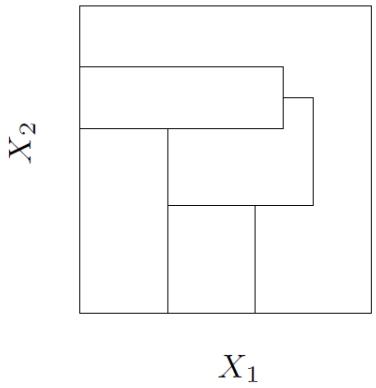
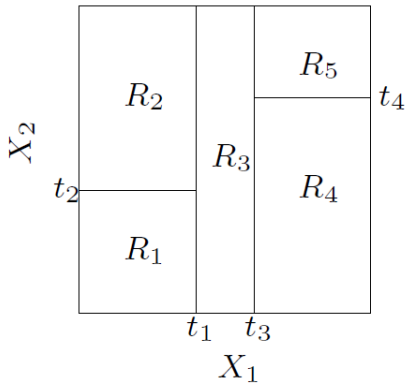
Decision Trees

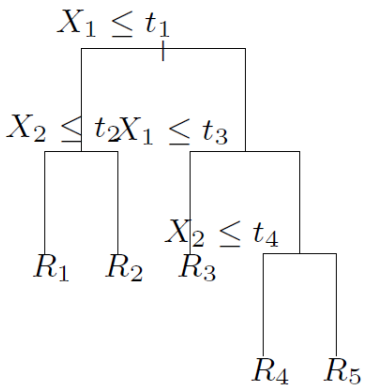
Nicolás Pérez de la Blanca

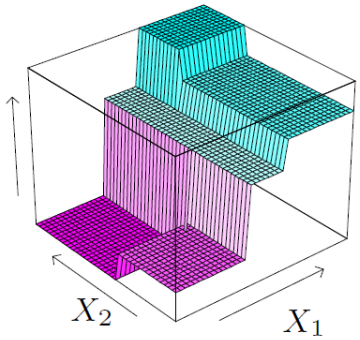
DECSAI

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.

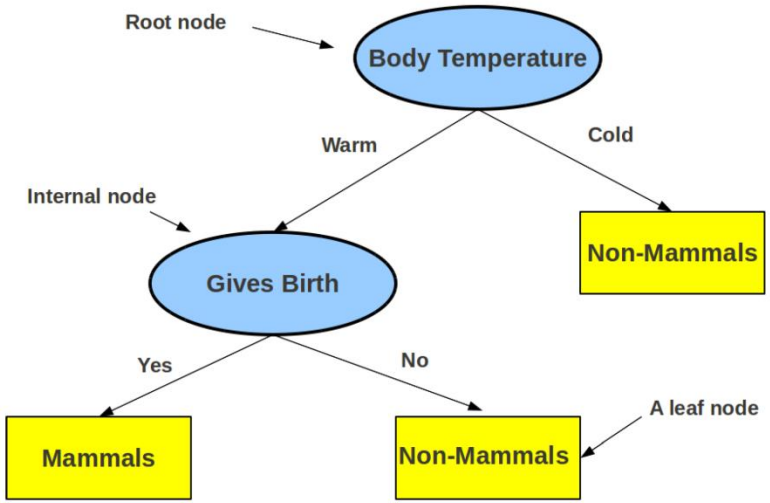
By simplicity we restrict our attention to recursive partitions

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

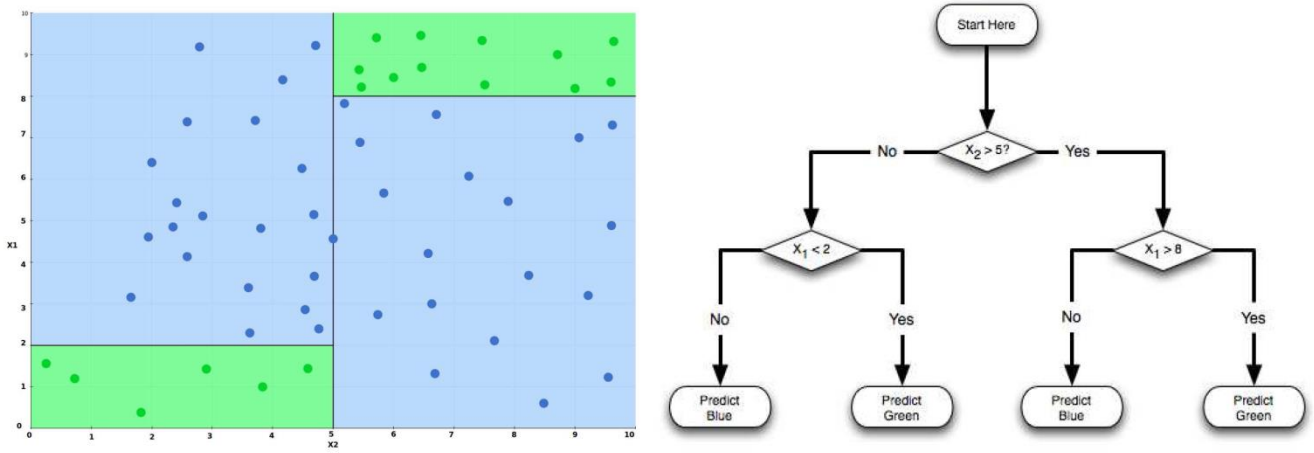
- Defined by a hierarchy of rules (in form of a tree)



- 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**

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

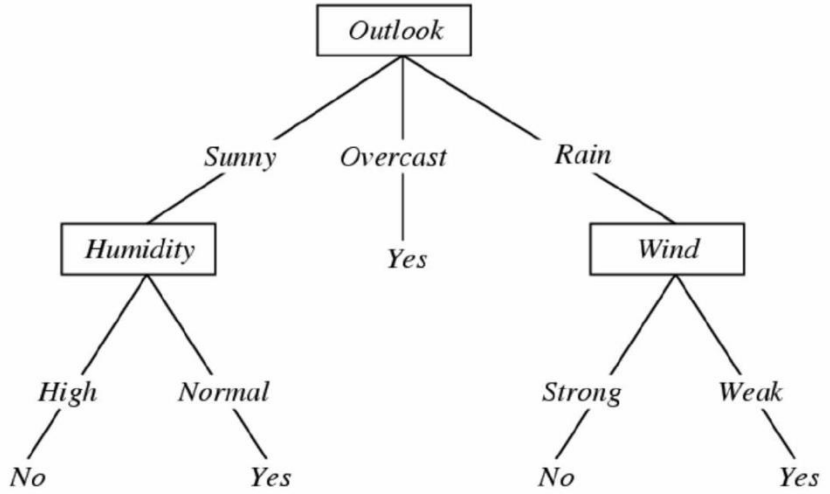


- The DT can be used to predict the region (blue/green) of a new test point
  - By testing the features of the test point
  - In the order defined by the DT (first x_2 and then x_1)

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

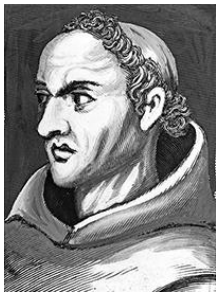


- The DT can be used to predict play vs no-play for a *new* Saturday
  - By testing the features of that day
  - In the order defined by the DT

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 δ

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-δ, 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 tha best partitioning in terms of minimum sum of squares is computationally infeasible. Hence a greedy algorithm in 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 threshol) and to prune the results using a cost-complexity criteria.

- Let's denote by T a tree and be |T| the number of terminal nodes in T. Letting

$$N_m = \#[\mathbf{x}_i \in R_m]$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

We define the cost complexity criterion as

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

Classification Trees

- This a multivalued model that allows classification in K clases, K > 2
- The splitting criteria on each node has to be different that in regression

- In a node m representing a region R_m with N_m samples, let

$$\widehat{p_{mk}} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}[y_i = k]$$

the proportion of class k samples in node m.

- We classify the samples in node m to class $k(m) = \arg\max_k \widehat{p_{mk}}$, the majority class in node m.

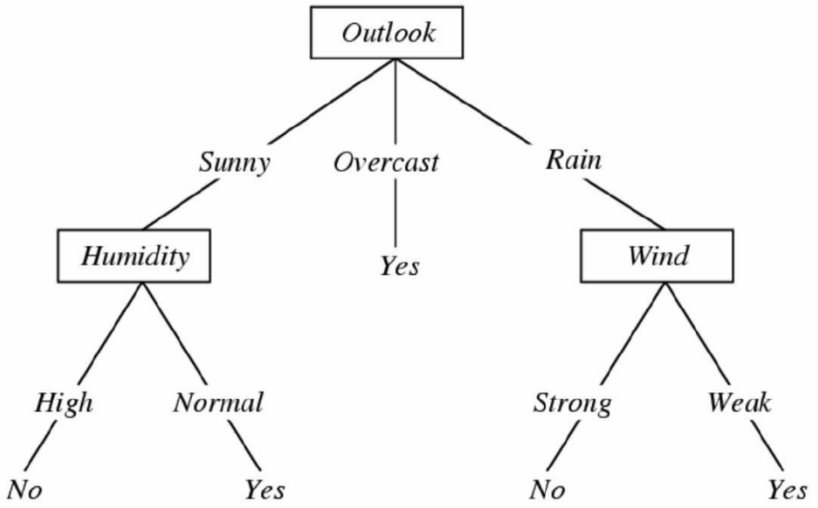
- Different measures $Q_m(T)$ of node impurity are:
  - Misclassification error: $1 - \hat{p}_{mk(m)}$, m is the majority class
  - Gini index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$ (training error of classify item to a class with its probability)
  - Entropy (cross-entropy): $-\sum_{k=1}^{K}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

- In binary classification the three criteria are equivalents.

Decision Tree Construction

- 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 quantity the informativeness
- **Information content** of a feature decides its position in the DT

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 $\Longrightarrow$ small entropy (less uncertainty)
  - Equiprobable classes $\Longrightarrow$ high entropy (more uncertainty)
- Entropy denotes the average number of bits needed to encode S

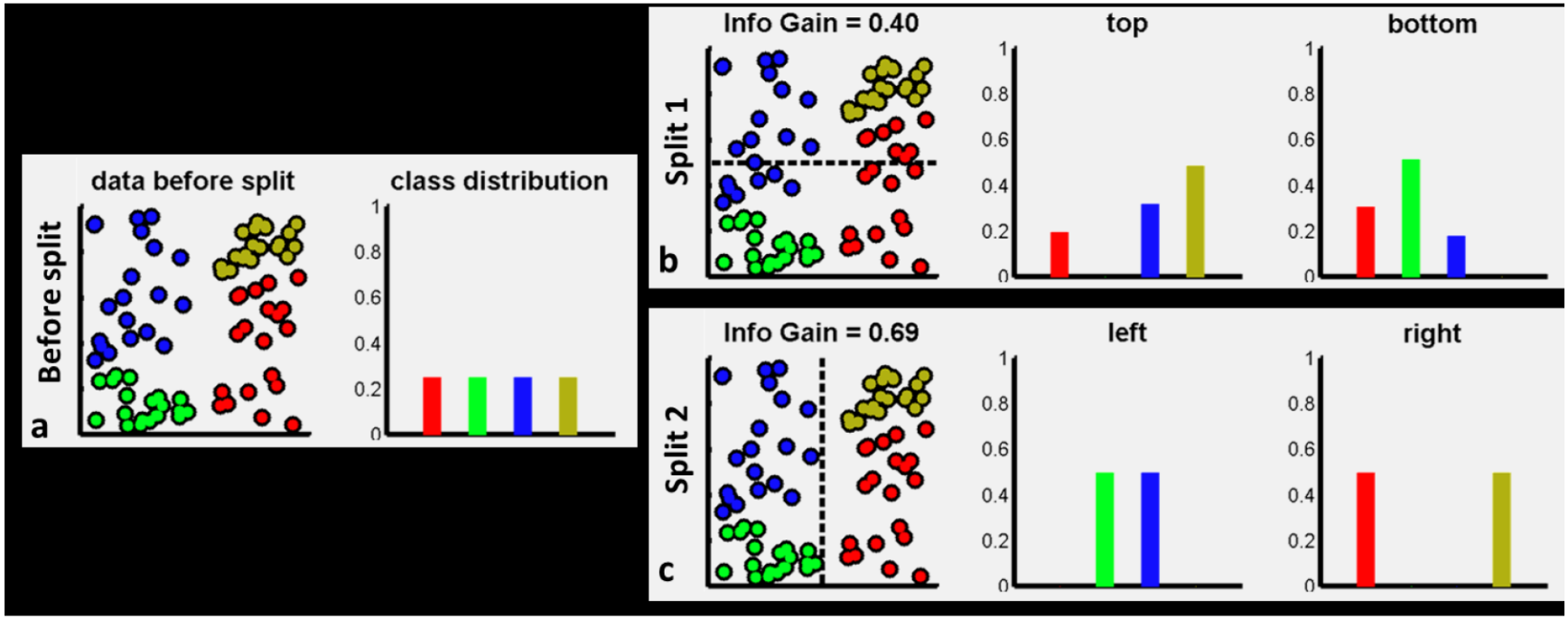
Information Gain

- 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 Algorithm (ID3)

A recursive algorithm:

DT(*Examples*, *Labels*, *Features*):

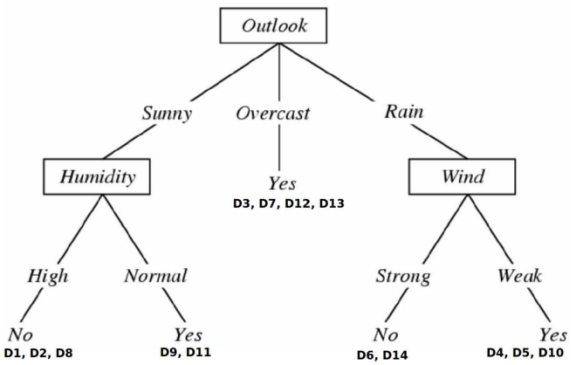
- If all examples are positive, return a single node tree *Root* with label $= +$
- If all examples are negative, return a single node tree *Root* with label $=$ -
- If all features exhausted, return a single node tree *Root* with majority label
- Otherwise, let F be the feature having the highest information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

    $$\mathrm{DT}(Examples_f,\ Labels_f,\ Features \text{ - } \{F\})$$

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

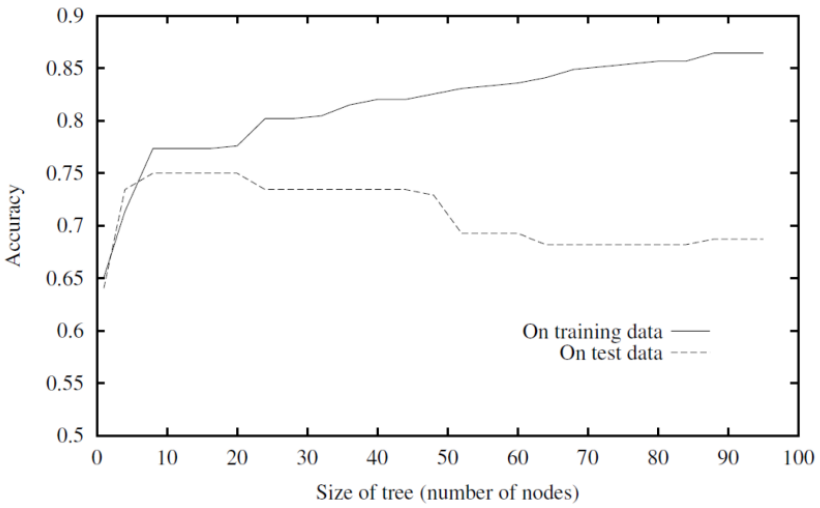


- This node will need to be expanded by testing some other feature
- The new tree would be more complex than the earlier one (trying to fit noise)
- The extra complexity may not be worth it $\Rightarrow$ may lead to overfitting if the test data follows the same pattern as our normal training data
- **Note:**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

Overfitting in Decision Trees

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

- 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 χ^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 $\mathbf{y}$'s at any node)
- Other criteria for judging feature informativeness
  - Gini-index, misclassification rate

BAGGING & RANDOM FORESTS

Índice

- Bagging
  - Bootstrapping
  - Bagging de árboles de regresión
  - Bagging de árboles de clasificación
  - Estimación del error “Out-of-Bag”
  - Importancia de las variables: Dibujos de influencia relativa
- Random Forests

BAGGING

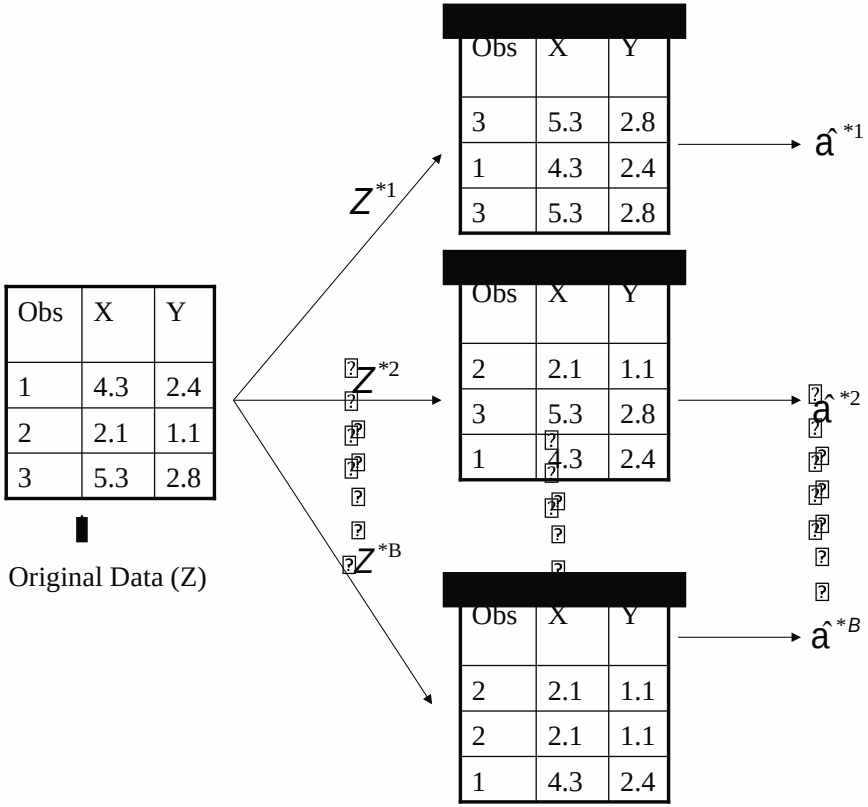
Bootstrap

- El bootstrap es una técnica estadística que puede usarse para cuantificar la incertidumbre asociada con un determinado estimador o un método de aprendizaje.
  - Por ejemplo podría usarse para estimar el error estándar de los coeficientes de regresión.

- Sin embargo la potencia del bootstrap está en el hecho que puede ser facilmente aplicable a un amplio rango de métodos de aprendizaje estadístico, incluyendo algunos para los cuales es difícil de obtener una medida de su variabilidad y tampoco hay software estadístico que podamos usar.

Bootstrapping es simple!

- Remuestreamos de forma aleatoria y con reemplazamiento el conjunto de datos para obtener nuevas muestras (de igual tamaño al conjunto de datos).



En concreto

- Extraer B-muestras a partir de los datos originales:
  - Muestrear con reemplazamiento los datos originales para obtener cada muestra-bootstrap.
  - A partir de cada B-muestra obtener un estimador del parámetro que nos interesa
  - A partir del conjunto de parámetros estimados obtener una estimación de la variabilidad de la estimación
  - El valor obtenido se usa como un estimador de la variabilidad del parámetro de la población

Problema!

- Los árboles de decisión discutidos presentan una alta varianza!
  - Si dividimos de forma aleatoria los datos de entrenamiento en dos partes y ajustamos árboles de decisión a ambas partes, los resultados podrían ser bastante diferentes
- Nos gustaría tener modelos con varianza baja

- Para resolver este problema, podemos usar bagging (***b***ootstrap ***agg***regat***ing***).

¿Que es bagging?

- Bagging es una idea muy poderosa basada en dos actuaciones:
  - Bootstrapping: muchos conjunto de entrenamiento distintos!
  - Promedios: reducción de la varianza!

- ¿Porque el promedio reduce la varianza?
  - Promediar reduce la varianza. (recordar que dado un conjunto de n observaciones independientes Z_1, …, Z_n, cada una con varianza s^2, la varianza de la media de las observaciones $\bar{Z}$ esta dad por s^2/n

¿Como funciona bagging?

- Genera B conjuntos entrenamiento distintos usando bootstraping
- Entrena el método de aprendizaje con cada uno de los B conjuntos y obtiene B modelos de predicción
- En predicción:
  - Regresión: promedia todas las predicciones de los B modelos
  - Clasificación: voto mayoritario de los B-modelos

Bagging con Árboles de regresión

- Construye B árboles de regresión usando B conjuntos de entrenamiento extraidos 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 ☺

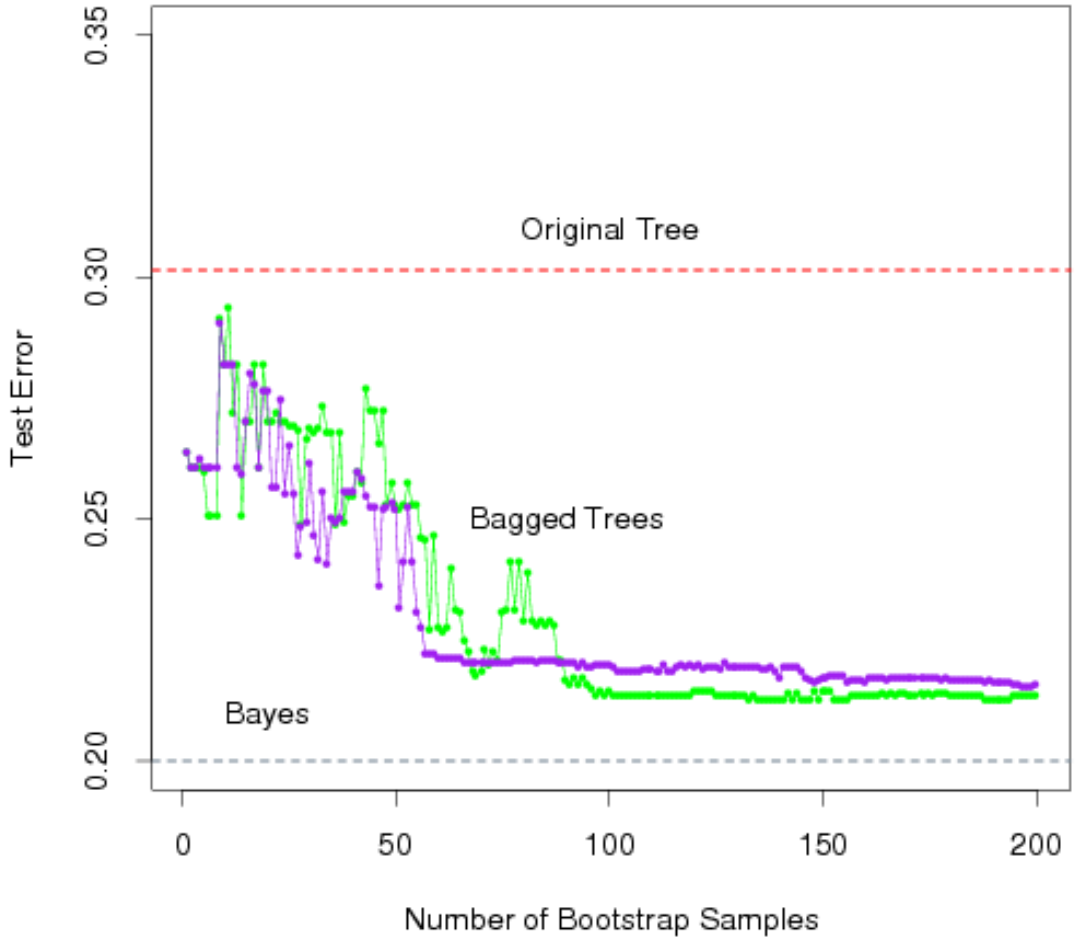
Bagging con Árboles de Clasificación

- Construye B árboles usando B conjuntos de entrenamiento extraidos con bootstrapping.

- Para predicción existen dos aproximaciones:
  1. Anotar la clase predicha por cada árbol (conjunto de datos) y asignar a la clase mayoritaria (mayoria 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.

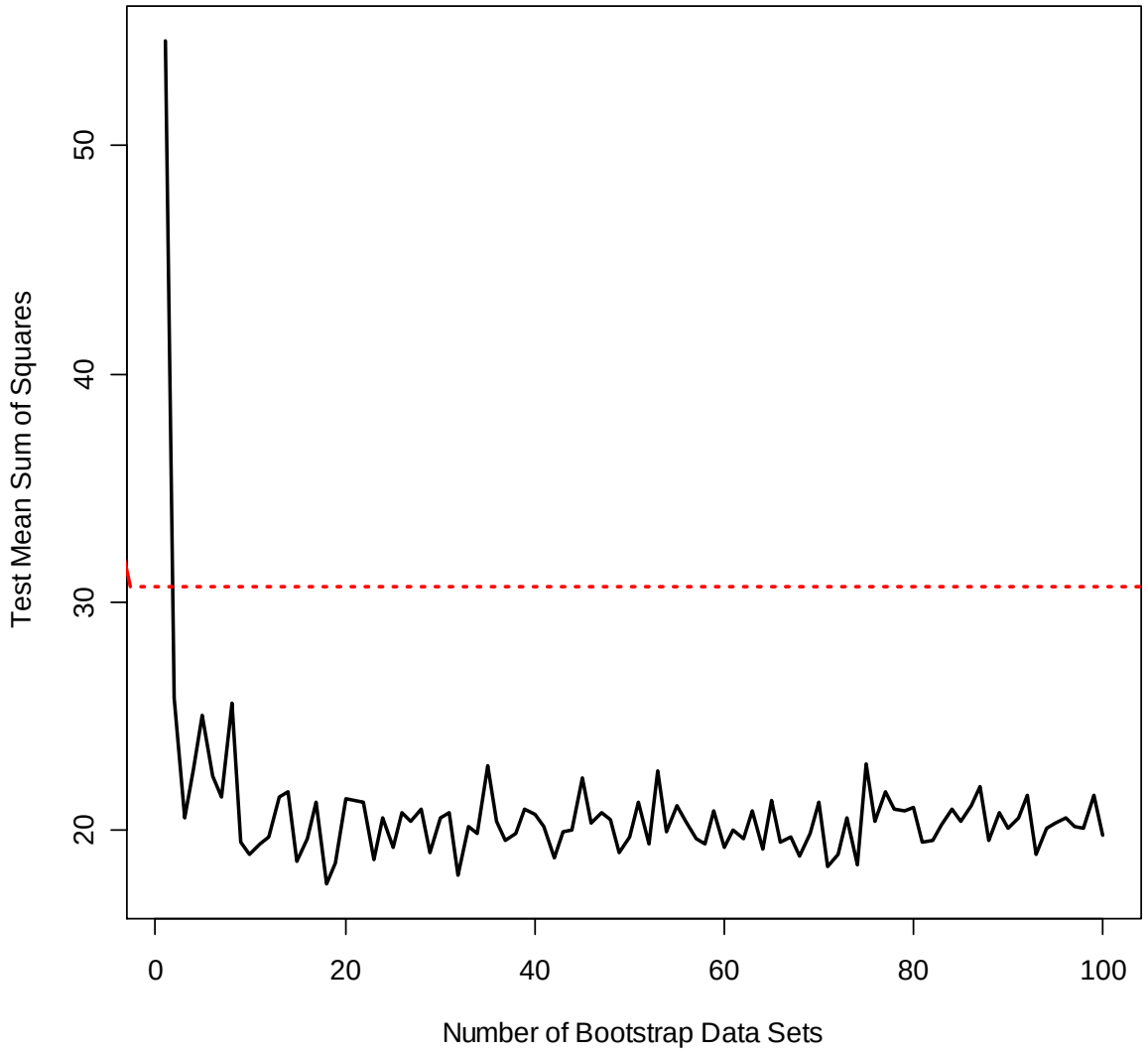
Una comparación de error de test

- Aqui 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



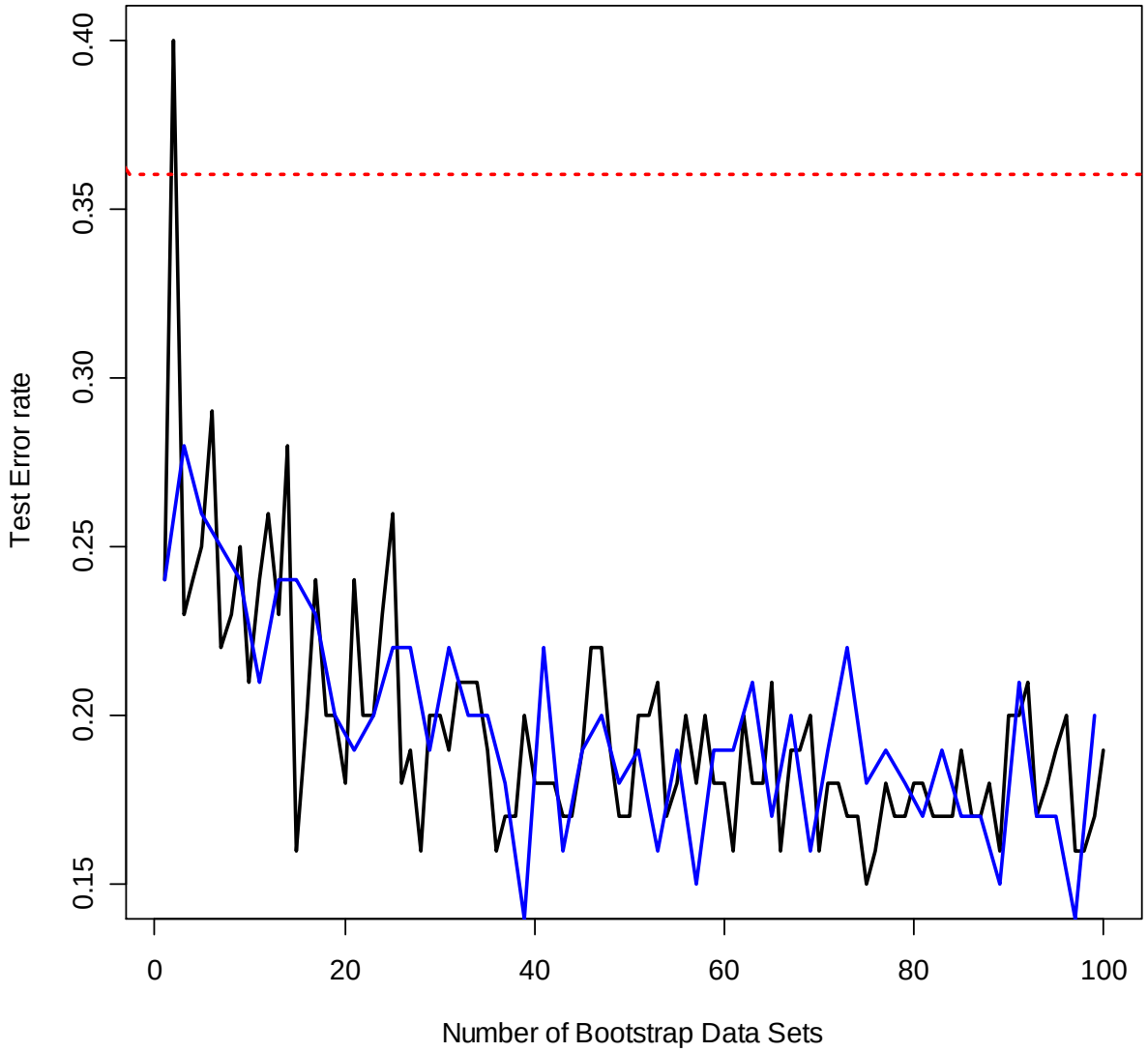
Ejemplo 1: Housing Data

- 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

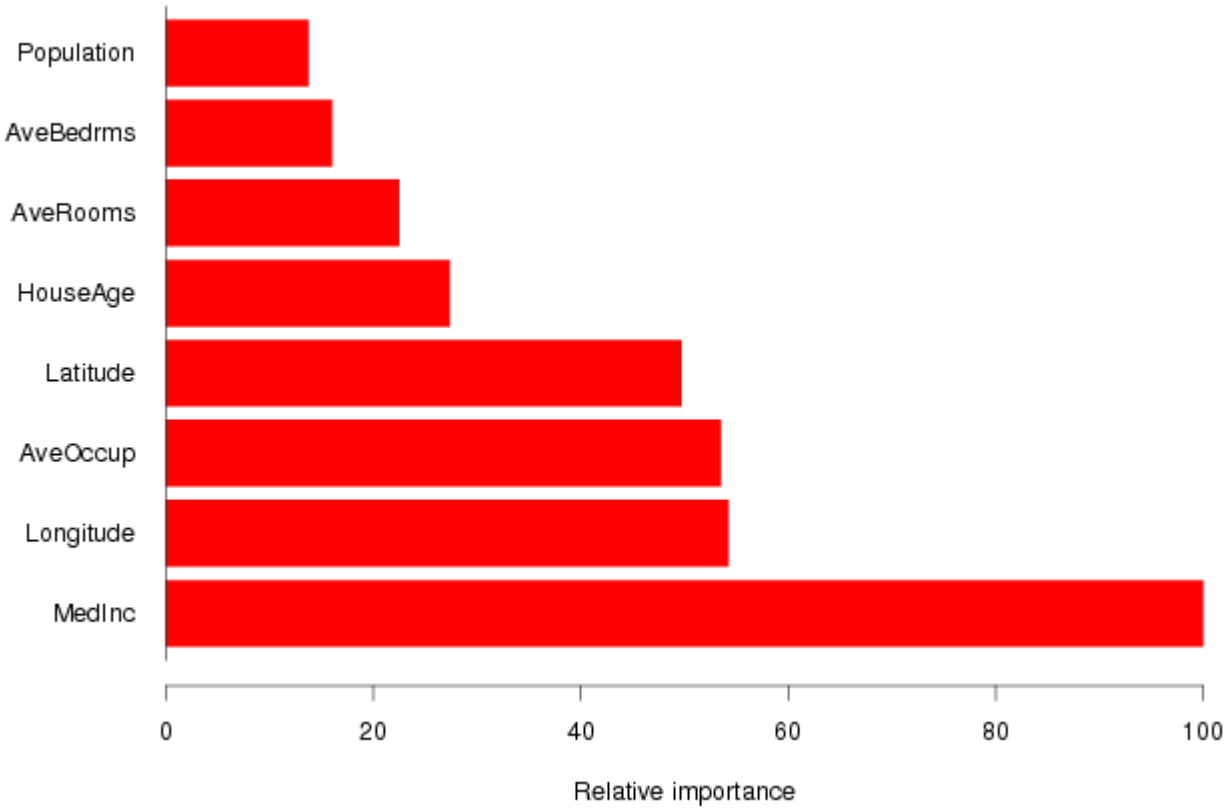
Gráficos de Importancia Relativa

- ¿Como decidir que variables son más útiles para preedecir 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 ídea de bagging, pero aporta una mejora ya que construye árboles no correlados
- ¿Como funciona?
  - Construye un número de árboles de decisión sobre muestras de boostrap,
  - 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 $m \approx \sqrt{p}$

¿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

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%