

Descripción del problema

- ▶ Muestra de entrenamiento Tenemos $(X_1, Y_1), \dots, (X_n, Y_n) \in \mathbb{E} \times \{1, \dots, m\}$
 - ▶ X_i son características observadas y usualmente $\mathbb{E} = \mathbb{R}^d$
 - ▶ etiquetas indicando la naturaleza de las observaciones
- ▶ Muestra de Clasificación
 - ▶ características observadas, X .
 - ▶ etiquetas desconocidas.
- ▶ Problema: clasificar un nuevo dato del cual no se conoce la etiqueta.

CART

Los métodos de clasificación jerárquicos encuentran en forma iterativa grupos dividiendo o fusionando grupos hallados en una etapa previa.

- ▶ Pueden ser aglomerativos (o de “abajo a arriba”) o divisivos (de “arriba a abajo”) o combinar las dos alternativas.
- ▶ **Aglomerativo:** comienzan asumiendo que cada observación es un grupo y van fusionando siguiendo algún criterio.
- ▶ **Divisivo:** comienzan con un solo grupo (todos los elementos) y proceden a dividirlos en **dos** en cada etapa.
- ▶ **Mixtos:** puede ser comenzar dividiendo en grupos hasta cierta etapa en que aparezcan más de los necesarios (“splitting”) y luego aplicar un procedimiento aglomerativo (“pruning”) que junte nuevamente ciertos grupos.

Ejemplo Titanic

El objetivo es predecir el destino de los pasajeros del Titanic, que se hundió en su viaje inaugural entre UK y New York, luego de haber chocado contra un iceberg.



CART

Contamos con las siguientes variables.

Muestra de entrenamiento:

- ▶ 891 pasajeros.
- ▶ 62 % mueren.

Muestra de clasificación:

- ▶ 418 pasajeros.

CART

Regresoras:

- ▶ Passanger (int)
- ▶ Survived (int)
- ▶ Pclass (int)
- ▶ Name (chr)
- ▶ Sex (chr)
- ▶ Age (num)
- ▶ SibSp (int)
- ▶ Parch (int)
- ▶ Ticket (chr)
- ▶ Fair (num)
- ▶ Cabine (chr)
- ▶ Embarqued (chr)

CART

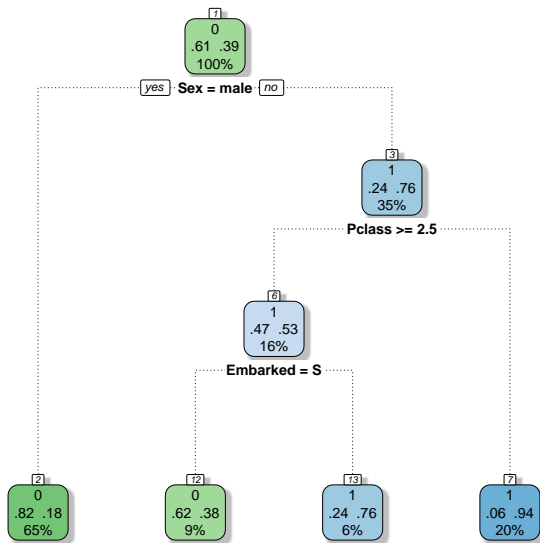
Para ilustrar voy a usar únicamente las variables:

- ▶ Sex
- ▶ Pclass
- ▶ Embarqued

```
titanic.tree.1= rpart(Survived ~ Pclass + Sex+  
Embarqued , data=train, method="class")  
fancyRpartPlot(titanic.tree.1)
```

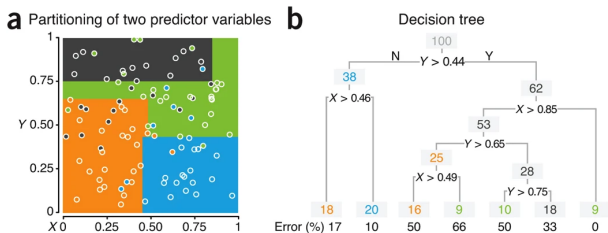
CART

Crean una jerarquía que se representa bien mediante un árbol.



CART

Los árboles determinan una partición del espacio con cortes paralelos a los ejes.



CART

El método divisivo más utilizado es **CART, Classification and Regression Trees**, propuesto por Breiman, Friedman, Olsen y Stone en 1984.

El árbol de particiones se construye a partir de unas pocas condiciones binarias respecto de las coordenadas originales de los datos, es decir, si $X = (X_1, \dots, X_p)$, las condiciones son del tipo

$$X_2 < a \text{ o bien } X_2 \geq a,$$

Si $X_2 < a$ luego,

$$X_4 < b \text{ o bien } X_4 \geq b$$

El resultado es muy sencillo de interpretar.

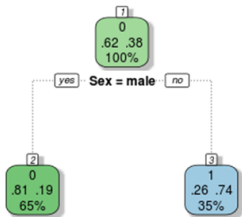
CART

- ▶ A partir del árbol es muy rápido clasificar nuevas observaciones.
- ▶ Se puede interpretar porque determinada observación pertenece a cierto grupo y caracterizar fácilmente a los grupos.
- ▶ Modelo de caja de vidrio: Una vez que el modelo haya encontrado los patrones en los datos, podrá ver exactamente qué decisiones se tomarán para los datos que desea predecir.
- ▶ Intuitivos y pueden ser leídos por personas con poca experiencia en aprendizaje automático después de una breve explicación.
- ▶ Son la base de algunos de los algoritmos de aprendizaje automático más potentes y populares.

Descripción del algoritmo

Paso 1

El algoritmo comienza con todos los datos en el nodo raíz (dibujado en la parte superior) y escanea todas las variables para encontrar la mejor. La mejor variable es aquella que da lugar a nodos más puros al dividirse, es decir, que las observaciones que pertenecen a cada nodo sean lo más homogéneas posible.



Descripción del algoritmo

Interpretación:

- ▶ Nodo raíz: 62% muere. **Voto: mueren.**

Ramas:

- ▶ Si el pasajero es hombre vamos a la izquierda, donde el 19% sobrevive. **Voto: mueren.**
- ▶ Si el pasajero es mujer vamos a la derecha, donde el 74% sobrevive. **Voto: sobreviven.**

Descripción del algoritmo

Riesgo

Sea (Y, X) un vector aleatorio, con $X \in \mathbb{R}^p$.

Buscamos una coordenada del vector X , $j \in 1, \dots, p$ y un valor real c de modo que la partición del espacio dada por

$$R_j^{left} = \{X \in \mathbb{R}^p : x_j < c\} \text{ y } R_j^{right} = \{X \in \mathbb{R}^p : x_j \geq c\}$$

El objetivo es obtener dos subconjuntos cuyas etiquetas sean lo más homogéneas posible en cada partición.

A cada nodo se le asigna la etiqueta mayoritaria.

Descripción del algoritmo

Paso 2

Aplicar el procedimiento explicado en el **Paso 1** en cada uno de los nodos terminales.

Iterar este proceso (en principio) hasta alcanzar la depuración total de los nodos.

En cada partición que se realice asignar la etiqueta mayoritaria a cada uno de los nodos terminales.

Cuándo parar?

- ▶ Todas las observaciones tienen la misma etiqueta.
- ▶ Si la rama tiene menos de un número preestablecido de observaciones.

De este modo se construye un árbol maximal T_{max} , que generalmente tiene muchas ramas y hojas y provocará sobreajuste.

Descripción del algoritmo

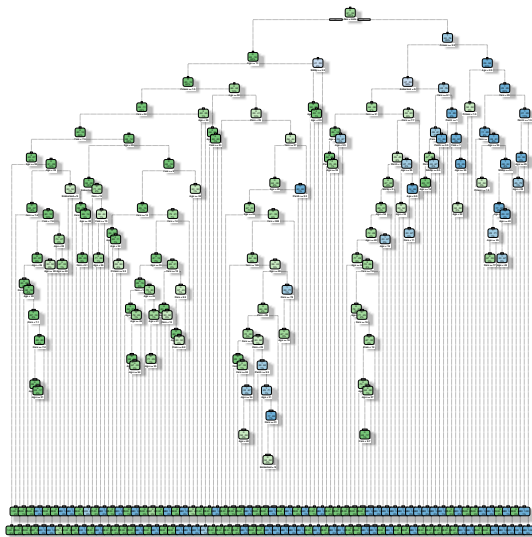
Volvamos al ejemplo de Titanic y consideremos las variables,

- ▶ Pclass
- ▶ Sex
- ▶ Age
- ▶ SibSp
- ▶ Parch
- ▶ Fare
- ▶ Embarked

Y construyo el árbol maximal.

```
tit.tree.max = rpart(Survived ~ Pclass + Sex + Age +  
SibSp + Parch + Fare + Embarked, data=train,  
method="class", control=rpart.control(minsplit=2,  
cp=0))  
fancyRpartPlot(tit.tree.max)
```

Descripción del algoritmo



Rattle 2020-ago.-06 12:58:50 Marcela Svarc

Descripción del algoritmo

Paso 3

Para evitar el sobreajuste se introduce un procedimiento backward, que poda el árbol, **pruning**.

Pruning:

- ▶ Eliminamos ramas y hojas para mejorar el desempeño del clasificador en un nuevo conjunto de datos para el cual no conocemos las etiquetas.
- ▶ La idea general es que al construir el árbol maximal, en determinado momento las ramas pasan a tener pocas observaciones que no son representativas de la población y se comienzan a hacer cortes espúreos que luego tienen que ser eliminados para que no afecten negativamente a las predicciones.

De este modo obtenemos un árbol más parsimonioso.

Descripción del algoritmo

Criterio de Pruning

$$R_{\alpha}(T) = \sum_{i=1}^{|T|} \text{criterio de impureza} + \alpha |T|,$$

donde $|T|$ es la cantidad de hojas que tiene el árbol T .

Este criterio depende de un parámetro α que se determina por **cross-validation**.

Descripción del algoritmo

Criterios de Impureza

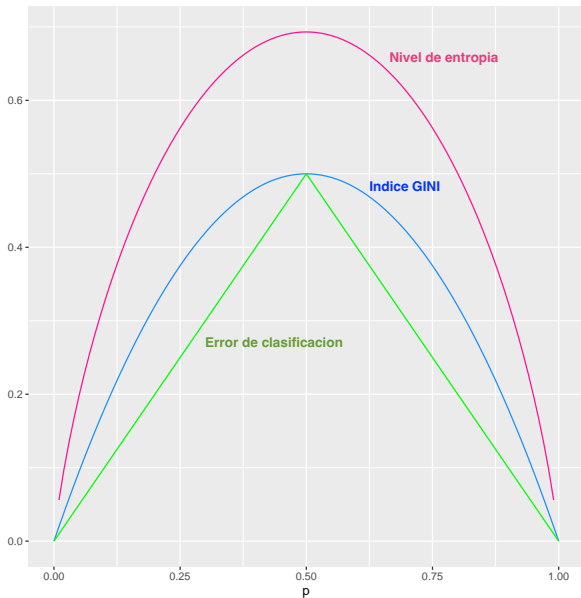
Necesitamos definir medidas de desajuste para parar de forma automática.

Sea \hat{p}_{mk} la proporción de observaciones del grupo k en el nodo m y

$$k(m) = \arg \max_k \hat{p}_{mk}.$$

- ▶ Error de clasificación: $1 - \hat{p}_{mk(m)}$.
- ▶ Índice de Gini: $\sum_k \hat{p}_{mk} (1 - \hat{p}_{mk})$. Suma de las varianzas de las clases, el peor error es 0.5
- ▶ Entropía: $-\sum_k \hat{p}_{mk} \log(\hat{p}_{mk})$.

Descripción del algoritmo



Descripción del algoritmo

- ▶ El error de clasificación no es derivable.
- ▶ El error de clasificación es menos sensible a cambios en las probabilidades de un nodo.
- ▶ Gini y entropía tienen a producir nodos más puros. Se pueden usar para hacer crecer el árbol. En la etapa de poda se recomienda el Error de clasificación que es más sencillo de interpretar.

Descripción del algoritmo

$$R_{\alpha} = \text{sumatoria}(\text{Impureza}) + \alpha \cdot |T|$$

Algunas propiedades, sea T el árbol maximal obtenido en el **Paso 1**,

- ▶ Consideremos dos subárboles de T , T_1 y T_2 . Si $R_{\alpha}(T_1) = R_{\alpha}(T_2)$, entonces T_1 es un subárbol de T_2 o viceversa, luego $|T_1| < |T_2|$ o $|T_2| < |T_1|$.
- ▶ Si $\alpha_1 < \alpha_2$ entonces $T_{\alpha_1} = T_{\alpha_2}$ o T_{α_2} es un subárbol estricto de T_{α_1} .
- ▶ Luego dada una sucesión $\alpha_1 < \dots < \alpha_m$ se pueden calcular eficientemente

$$T_{\alpha_1}, \dots, T_{\alpha_m}$$

y

$$R(T_{\alpha_1}), \dots, R(T_{\alpha_m})$$

Descripción del algoritmo

Luego, **para cada α podemos determinar T_α el menor árbol para el cual se minimiza $R_\alpha(T)$.**

Cualquier secuencia anidada de árboles tiene basada en T tiene a lo sumo $|T|$ árboles, luego, todos los posibles valores de α se pueden agrupar en m intervalos, $m < |T|$,

$$I_1 = [0, \alpha_1]$$

$$I_2 = (\alpha_1, \alpha_2]$$

$$\vdots = \vdots$$

$$I_m = (\alpha_{m-1}, \alpha_m]$$

para todo $\alpha \in I_i$ el subárbol que minimiza es el mismo

Cross Validación del parámetro α

1. Ajustar el modelo completo. Buscar I_1, \dots, I_m . Seterar,
 $\beta_1 = 0, \dots, \beta_i = \sqrt{\alpha_{i-1}\alpha_i}, \dots, \beta_m = \infty$ **Beta == alfa**
2. Dividir la muestra en K grupos de igual tamaño, G_1, \dots, G_K
3. Para cada subconjunto G_i
 - 3.1 Ajustar el modelo en $G \setminus G_i$ determinar $T_{\beta_1}, \dots, T_{\beta_m}$ para el conjunto de datos reducido.
 - 3.2 Predecir las etiquetas para el conjunto G_i , para cada modelo $T_{\beta_j}, j = 1, \dots, m$.
 - 3.3 Computar el riesgo para cada subconjunto.
4. Sumar sobre todos los G_i para estimar el riesgo para cada β_j .
Calcular el árbol T_β para el valor de β que minimice el riesgo,
este árbol construirlo con toda la muestra.

Resumimos el algoritmo

1. Considerando la muestra de entrenamiento, hacer crecer mediante las particiones recursivas un árbol grande, parando únicamente cuando un nodo tenga una cantidad preestablecida de observaciones o cuando todas tengan la misma etiqueta. Se obtiene el árbol maximal T_{max} .
2. Aplicar el *cost complexity* para podar el árbol maximal. Considerar varios valores $\alpha_1, \dots, \alpha_I$. De esta forma se obtiene una sucesión de subárboles del árbol original.
3. Usar K - fold cross validation para elegir α . Partir de la muestra de entrenamiento en K submuestras del mismo tamaño, $k = 1, \dots, K$.
 - 3.1 Repetir los pasos 1 y 2 para cada fracción de tamaño $\frac{K-1}{K}$ de la muestra de entrenamiento, excluyendo el k -ésimo fold.
 - 3.2 Considerando el mismo criterio de impureza, evaluar los datos correspondientes al k -ésimo fold en función de α .
4. Para cada valor de α promediar los criterios de impureza y elegir el valor de α correspondiente a la mínima impureza.
5. Retornar al paso 2 y elegir el árbol correspondiente al valor

CART

Veamos como construimos el árbol crosvalidando el parámetro α
Hacemos crecer el árbol arrancamos con un valor de α pequeño.

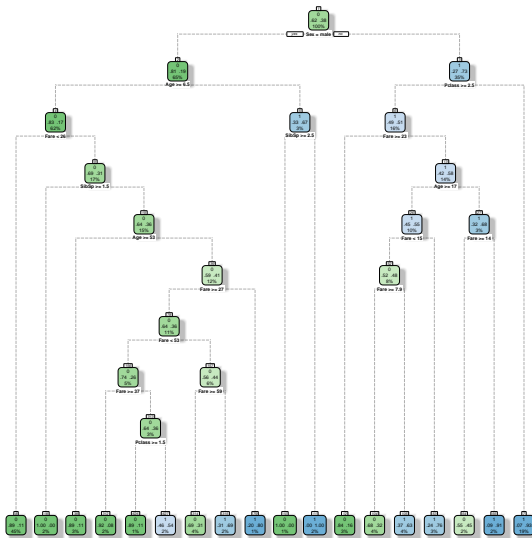
```
tit.tree.grow = rpart(Survived ~ Pclass + Sex + Age  
+ SibSp + Parch + Fare + Embarked, data=train,  
method="class", control = rpart.control(cp = 0.0001))  
fancyRpartPlot(tit.tree.grow)
```

Elegimos el árbol que minimice la tasa global de error de clasificación.

```
printcp(tit.tree.grow)
```

		Alfa	ITI, numero de particiones	Impureza	Riesgo	sdRiesgo
		CP	nsplit	rel error	xerror	xstd
Mejor alfa	1	0.48	0.00	1.00	1.00	0.05
	2	0.03	1.00	0.52	0.52	0.04
	3	0.03	3.00	0.46	0.49	0.04
	4	0.01	5.00	0.40	0.42	0.04
	5	0.01	8.00	0.37	0.45	0.04
	6	0.00	14.00	0.33	0.45	0.04

CART



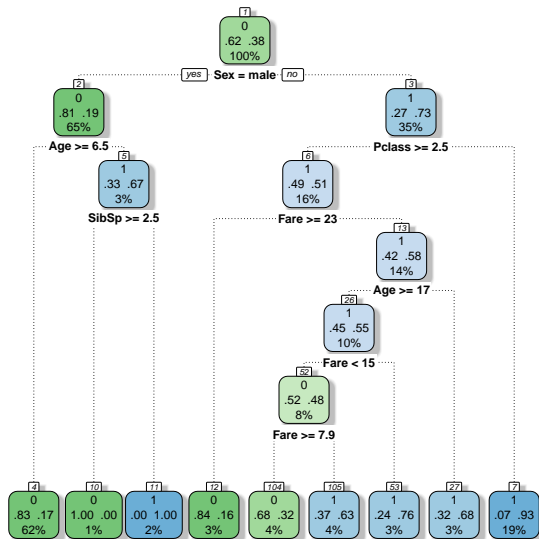
Rattle 2020-ago.-06 13:23:22 Marcela Svarc

CART

Tomamos el valor de penalidad óptimo y lo utilizamos para podar el árbol.

```
bestcp =  
tit.tree.grow$cptable[which.min(tit.tree.grow$cptable  
[, "xerror"]), "CP"]  
tit.pruned = prune(tit.tree.grow, cp = bestcp)
```

CART



CART

Ahora falta ver como se realizan las predicciones, las haremos en la muestra de testeo que separamos al principio.

```
pred.prune = predict(tit.pruned, test, type =  
"class")
```

Confeccionamos la matriz de confusión

```
conf.matrix = table(test$Survived, pred.prune)  
rownames(conf.matrix) = paste("Actual",  
rownames(conf.matrix), sep = ":")  
colnames(conf.matrix) = paste("Pred",  
colnames(conf.matrix), sep = ":")
```

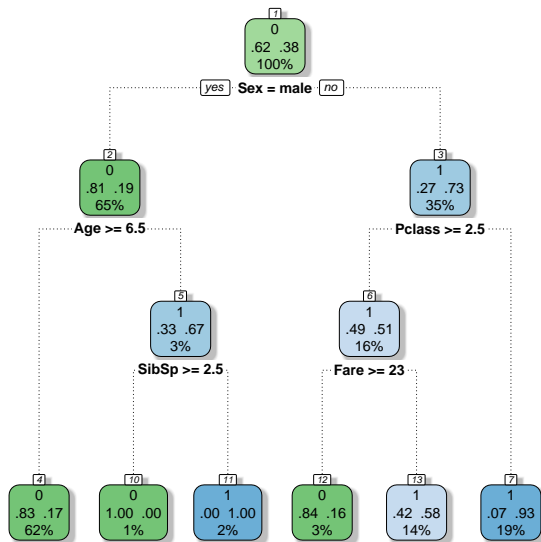
	Pred:0	Pred:1
Actual:0	127	4
Actual:1	31	55

CART

Podemos cambiar el Criterio de Impureza

```
tit.tree.infor = rpart(Survived ~ Pclass + Sex + Age  
+ SibSp + Parch + Fare + Embarked, data=train,  
method="class", control = rpart.control(cp =  
0.0001), parms = list(split= 'information'))  
bestcp = tit.tree.infor$cpstable  
[which.min(tit.tree.infor$cpstable[, "xerror"]), "CP"]  
tit.pruned.infor = prune(tit.tree.infor, cp = bestcp)
```

CART

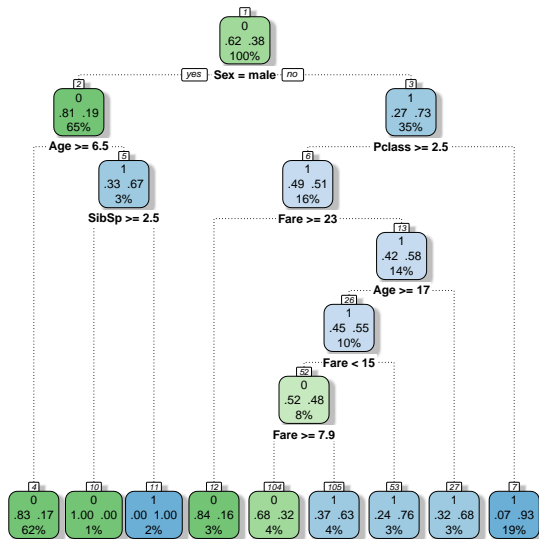


CART

Podemos cambiar el Criterio de Impureza

```
tit.tree.gini = rpart(Survived ~ Pclass + Sex + Age  
+ SibSp + Parch + Fare + Embarked, data=train,  
method="class", control = rpart.control(cp =  
0.0001), parms = list(split= 'gini'))  
bestcp = tit.tree.gini$cpstable  
[which.min(tit.tree.gini$cpstable[, "xerror"]), "CP"]  
tit.pruned.gini = prune(tit.tree.gini, cp = bestcp)
```

CART



CART

También podemos fijar

- ▶ Número mínimo que tiene que tener un nodo para ser partido
- ▶ Número mínimo que puede tener un nodo final

para evitar el sobreajuste.

Consideraciones vinculadas a regresores categóricos

- ▶ Si se tienen q categorías sin un orden, el problema puede volverse inmanejable computacionalmente. Conviene codificarlo con variables dummies.
- ▶ El algoritmo tiende a favorecer cortes a través de variables que tengan muchas categorías, esto puede provocar sobreajuste hay que tratar de evitarlas.

Datos Faltantes

- ▶ Se puede poner una categoría *extra* que indique que un dato es faltante.
- ▶ Construcción de variables surrogantes:
 - ▶ Se determina el mejor corte con las observaciones no faltantes.
 - ▶ Se busca otro corte (en otra variable que reproduzca del mejor modo posible al mejor corte)
 - ▶ Se sigue así determinando reglas que mejor "reproduzcan" a la mejor.
 - ▶ Cuando una observación tiene un dato faltante la variable del primer corte, se pasa a la primera surrogante y así.

CART

Cómo mejorar el modelo:

- ▶ Utilizar variables que no hayamos tenido en cuenta.
- ▶ Ingeniería de variables: definir variables nuevas a partir de las que ya tenemos:
 - ▶ interacciones.
 - ▶ transformaciones (lineales, cuadráticas, logaritmos).
 - ▶ definiendo nuevas variables, que tengan sentido en el contexto del problema

$$FamilySize = SibSp + Parch + 1$$

Al transformar variables, se puede ganar en poder predictivo, pero se pierde en interpretabilidad.

CART

Problemas de CART:

- ▶ Inestabilidad, alta varianza.
- ▶ Sobreajuste si se eligen mal los parámetros de pruning.

Bagging

CART aplicado al problema de regresión lineal

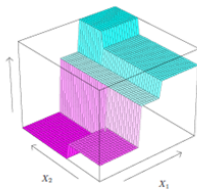
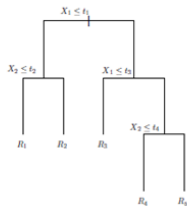
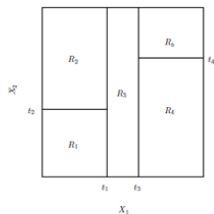
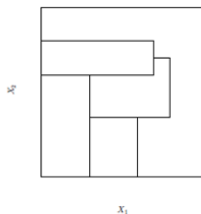
Los árboles de clasificación se pueden utilizar para tratar el problema de regresión, cuando la respuesta es lineal.

Se parte al espacio en rectángulos en forma recursiva, en cada rectángulo se asigna un valor constante.

Tenemos $(x_1, y_1), \dots, (x_n, y_n)$, partimos al espacio en R_1, \dots, R_M regiones, en cada región ajustamos la respuesta con una constante, c_m

$$f(x) = \sum_{i=1}^M c_m \mathcal{I}_{(x \in R_m)}$$

CART aplicado al problema de regresión lineal



CART aplicado al problema de regresión lineal

Si buscamos minimizar la suma del cuadrado de los residuos en cada region del espacio, entonces tenemos que el estimador está dado por el promedio en cada región.

$$\hat{c}_m = \frac{\sum_{x_i \in R_m} y_i}{\sum_{i=1}^n \mathcal{I}_{x_i \in R_m}}$$

Criterio para partir el espacio, para cada variable j y punto s , se definen los hiperplanos $R_1(j, s) = \{X | X_j \leq s\}$ y $R_2(j, s) = \{X | X_j > s\}$ Luego hay que hallar la **variable j** y el **punto s** tales que

$$\min_{s,j} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

Una vez determinada la partición $\hat{c}_l = \text{ave} \{y_i | x_i \in R_l(j, s)\}$ para $l = 1, 2$.

CART aplicado al problema de regresión lineal

El procedimiento se realiza secuencialmente.

Cost Complexity

Para un árbol T el cost complexity estará dado por

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

donde

$$N_m = \# \{x_i \in R_m\}$$

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

CART aplicado al problema de regresión lineal

- ▶ Objetivo: encontrar el subárbol T_α , que minimice $C_\alpha(T)$.
- ▶ α indica la relación entre el tamaño del árbol y el ajuste. A mayor α menor tamaño del árbol.
- ▶ α se estima por cross-validation.

CART aplicado al problema de regresión lineal

Consideremos el conjunto de datos **Boston** de la library MASS

► Variables regresoras:

crim tasa de crímenes per capita por ciudad.

zn prop. de zona residencial (lotes superiores a 25.000 sq ft).

indus proporción de negocios no minoristas por ciudad (acres).

chas variable dummy (=1 si limita el Río Charles; 0 caso contrario).

nox concentración de óxido de nitrógeno (partes por 10 millones).

rm nro. medio de ambientes por vivienda.

age prop. de viviendas ocupadas por sus dueños construidas antes de 1940.

dis media pesada de dist. a los 5 centros de empleos de Boston.

rad índice de accesibilidad a las autopistas radiales.

tax Tasa de impuesto a la propiedad de valor total por \$10.000.

ptratio relación de alumno-docente por ciudad.

black $1000(B_k - 0.63)^2$ donde B_k proporción de negros en la ciudad.

lstat porcentaje de individuos con nivel socioeconómico bajo.

► Variable de respuesta:

mdev media de propietarios que ocupan sus viviendas cada 10000 hab.

CART aplicado al problema de regresión lineal

```
library(tree)
library(MASS)
train = sample (1: nrow(Boston ), nrow(Boston )/2)
tree.boston =tree(medv ~ .,Boston ,subset =train)
```

CART aplicado al problema de regresión lineal

```
summary (tree.boston)
```

Regression tree:

```
tree(formula = medv ~ ., data = Boston, subset = train)
```

Variables actually used in tree construction:

```
[1] "rm" "lstat" "crim"
```

Number of terminal nodes: 7

Residual mean deviance: 12.83 = 3156 / 246

Distribution of residuals:

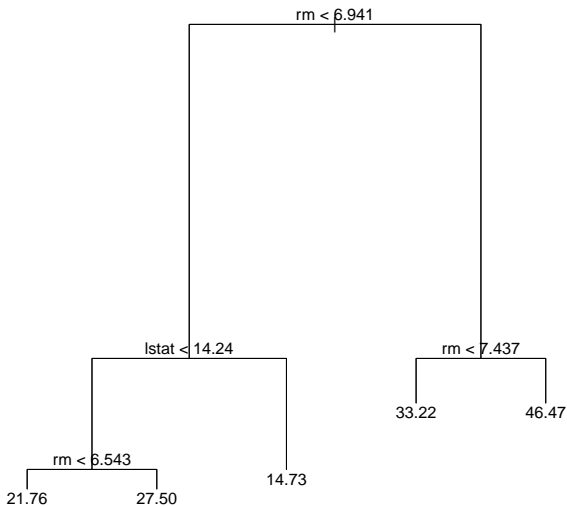
Min. 1st Qu. Median Mean 3rd Qu. Max.

-14.7500 -2.1300 0.2392 0.0000 2.0390 22.5000

CART aplicado al problema de regresión lineal

```
cv.boston=cv.tree(tree.boston)
plot(cv.boston$size,cv.boston$dev,type='b')
prune.boston=prune.tree(tree.boston ,best =5)
```


CART aplicado al problema de regresión lineal



CART aplicado al problema de regresión lineal

Inconvenientes:

- ▶ Falta de suavidad en la superficie de respuesta.
- ▶ Dificultad en el modelado de la estructura aditiva. Le puede demandar varios cortes capturar la estructura aditiva.

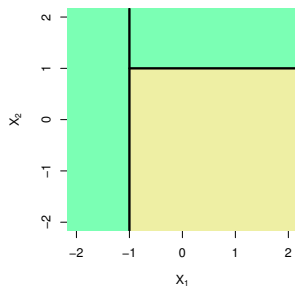
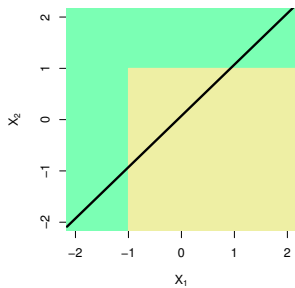
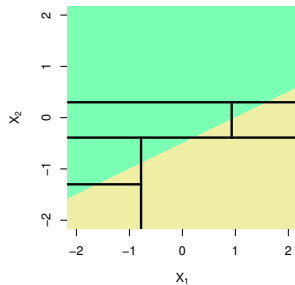
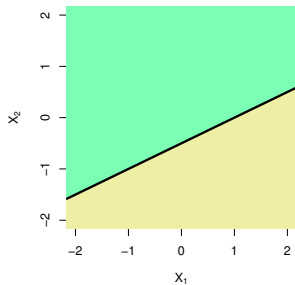
MARS

CART aplicado al problema de regresión lineal

Regresión Lineal o CART?

- ▶ Si hay estructura lineal \rightsquigarrow Regresión Lineal.
- ▶ Estructura altamente no lineal y relación compleja entre los regresores y la variable de respuesta \rightsquigarrow CART
- ▶ Interpretabilidad y visualización \rightsquigarrow CART.

CART



CART

Ventajas CART

- ▶ Fáciles de explicar e interpretar.
- ▶ A semejan el modo de razonar.
- ▶ Visualización agradable.
- ▶ Variables categóricas sin necesidad de definir variables dummies.

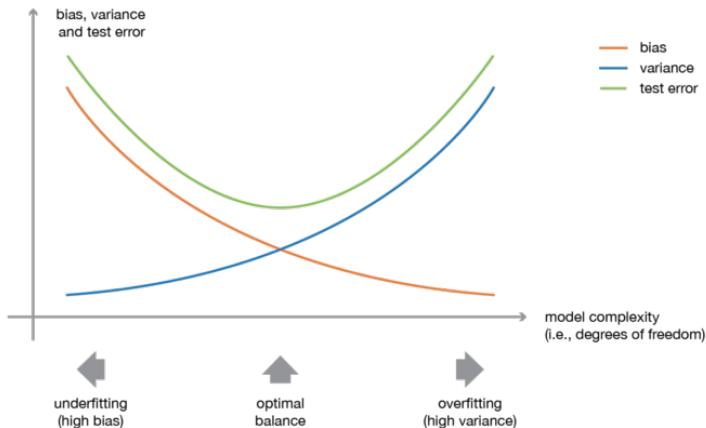
CART

Desventajas CART

- ▶ Mal poder predictivo.
- ▶ No son robustos, pequeños cambios en los datos afectan dramáticamente el árbol.

Clasificadores ensamblados

Usar en forma repetida un clasificador débil, agregar los resultados y tener uno fuerte



Bagging

La idea es utilizar bootstrap para crear un clasificador que tenga menor varianza que los árboles.

Sea $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$ una muestra de entrenamiento, la población tiene K clases.

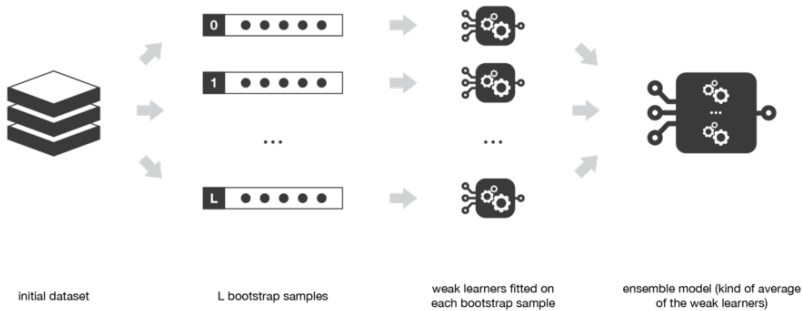
A partir de ella construimos un clasificador que *agregue* todos los clasificadores obtenidos en las muestras bootstrap \hat{g} , para todo x tenemos la clase predicha $\hat{g}(x)$.

Bagging

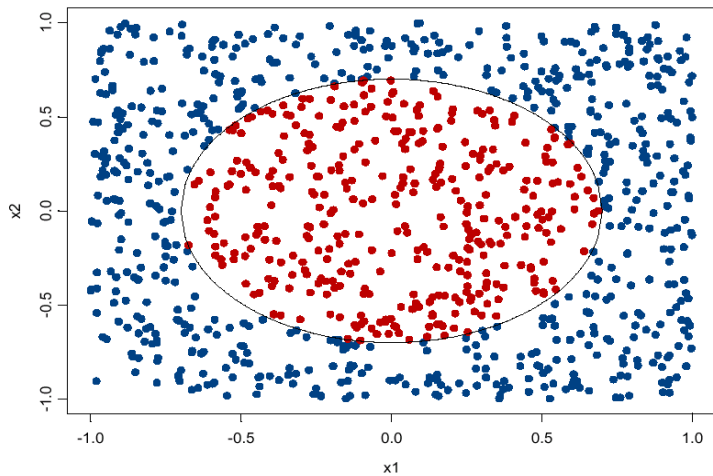
1. Construir B muestras bootstrap Z^{*1}, \dots, Z^{*B} .
2. Para cada muestra bootstrap Z^{*b} con $b = 1, \dots, B$. Construir un árbol de clasificación \hat{g}^{*b} .
3. Para todo x tenemos un vector que indica en cada coordenada el número de veces que x fue clasificado en el grupo k , $(N_1(x), \dots, N_K(x))$, donde $\sum_{k=1}^K N_k(x) = B$
4. Definir el clasificador **bagging** en la clase más votada

$$\hat{k}_{bag} = \arg \max_{k=1, \dots, K} (N_1(x), \dots, N_K(x))$$

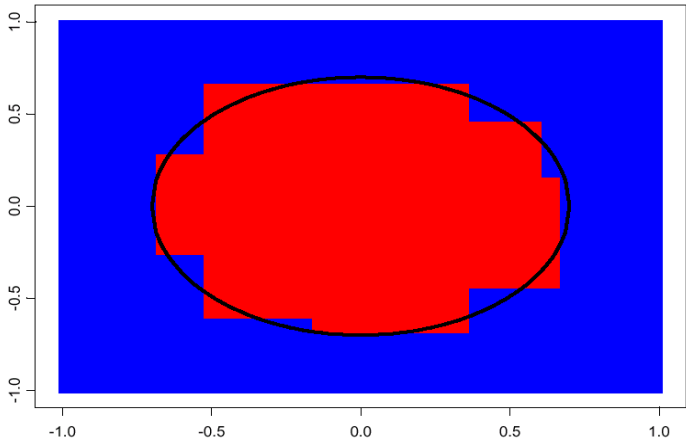
Bagging



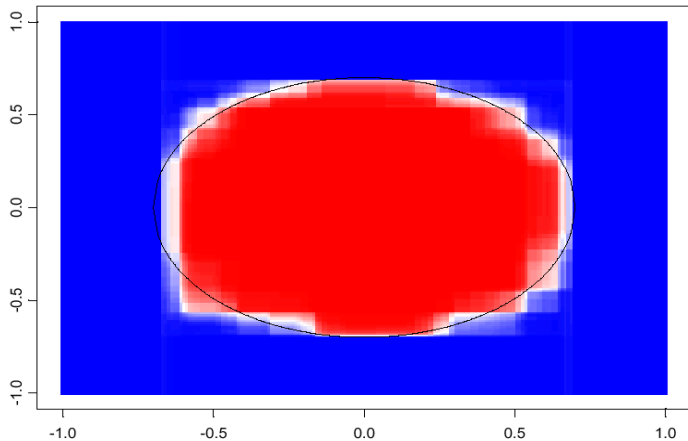
Bagging



Bagging



Bagging



Bagging

Sean $(x_1, y_1), \dots, (x_n, y_n)$ una muestra aleatoria, $(x_i, y_i) \sim \mathcal{P}$,
Supongamos que tenemos un *clasificador agregado* ideal,
 $g_{ag}(x) = E_{\mathcal{P}}(\hat{g}^*(x))$, donde g^* es el estimador basado en una
muestra bootstrap Z^* .

g_{ag} es el estimador bagging poblacional.

Bagging

Descomponemos el error cuadrático medio

$$\begin{aligned} E_{\mathcal{P}}(Y - g^*(x))^2 &= E_{\mathcal{P}}(Y - \textcolor{red}{g_{ag}(x)} + \textcolor{red}{g_{ag}(x)} - g^*(x))^2 \\ &= E_{\mathcal{P}}(Y - g_{ag}(x))^2 + \underbrace{E_{\mathcal{P}}(g_{ag}(x) - g^*(x))^2}_{(1)} + \\ &\quad + 2 \underbrace{E_{\mathcal{P}}(Y - g_{ag}(x)) E_{\mathcal{P}}(g_{ag}(x) - g^*(x))}_0 \\ &\stackrel{(1)}{\geq} E_{\mathcal{P}}(Y - g_{ag}(x))^2. \end{aligned}$$

Bagging

Wisdom of Crowds

- ▶ Buen clasificador \rightsquigarrow lo hace mejor.
- ▶ Mal clasificador \rightsquigarrow lo hace peor.

Supongamos $Y = 1 \forall x$, pero $P(\hat{g}(x) = 1) = 0.4 \Rightarrow$ error del clasificación, $e = 0.6$.

Si g_1^*, \dots, g_B^* fueran independientes el estimador de consenso

$$S = \sum_{b=1}^B \mathcal{I}(g_b^*(x) = 1) \sim \text{Bi}(B, e)$$

Luego

- ▶ $P(S > 0.5B) \rightarrow 0$ si $e > 0.5$
- ▶ $P(S > 0.5B) \rightarrow 1$ si $e < 0.5$

Bagging

Observación: En muchos casos es útil tener una probabilidad de pertenecer a cada clase y a partir de allí definir un clasificador.

Uno podría pensar que $\hat{p}_k(x) = \frac{N_{\hat{k}(x)}(c)}{B}$ es un buen estimador de $p(k|x)$ sin embargo esto es **falso**.

Por ejemplo, si $p(k|x) = 0.75$, todos los clasificadores bootstrap podrían clasificar a x en k , luego $\hat{p}_k(x) = 1$.

Bagging

Cómo solucionarlo?

En lugar de retener la etiqueta en cada muestra bootstrap, reterner el vector de la probabilidad de pertenecer a cada grupo, dada por el nodo terminal correspondiente a la observación x , i.e.,

$$\left(\widehat{p}_1^{*b}(x), \dots, \widehat{p}_K^{*b}(x) \right),$$

luego obtener

$$\left(\overline{p}_1^*(x), \dots, \overline{p}_K^*(x) \right),$$

donde $\overline{p}_k^*(x) = \frac{1}{B} \sum_{b=1}^B \widehat{p}_k^{*b}(x)$.

Bagging

Definir el estimador bagging

$$\hat{k}_{bag} = \arg \max_{k=1, \dots, K} (\bar{p}_1^*(x), \dots, \bar{p}_K^*(x))$$

- ▶ Este estimador suele ser más acertado.
- ▶ Tiene menor varianza.

Bagging

Ventajas Bagging

- ▶ Mejora el poder predictivo, en relación a CART.
- ▶ Tiene menor varianza.
- ▶ Su cálculo puede ser paralelizado, los modelos se ajustan independientes unos de otros en la B muestras bootstrap.

Bagging

Desventajas Bagging

- ▶ Se pierde la visualización del árbol.
- ▶ Ya no se puede interpretar fácilmente que variables son más relevantes para la clasificación (o regresión). Un atajo que se puede tomar para contrarrestar este punto es para cada variables mirar cómo varió el índice de Gini al realizar particiones con esas variables . Luego, promediarlo sobre todas las muestras bootstrap.
- ▶ Las muestras bootstrap no son independientes, los árboles pueden parecerse mucho, sobre todo en los primeros nodos, *tree correlation*, por lo tanto no puede bajar tanto la varianza.

Bagging

Retomamos el ejemplo de Titanic

```
library (randomForest)
```

```
train$Survived=as.factor(train$Survived)
```

```
bag.tit =randomForest(Survived  Pclass + Sex + Age +  
SibSp + Parch + Fare + Embarked,
```

```
data=train,ntree=500,mtry=7, importance =TRUE)
```

mtry=7 hay que poner el número total de variables.

```
bag.tit
```

Type of random forest:	<i>classification</i>
Number of trees:	500
No. of variables tried at each split:	7
OOB estimate of error rate:	23.08%

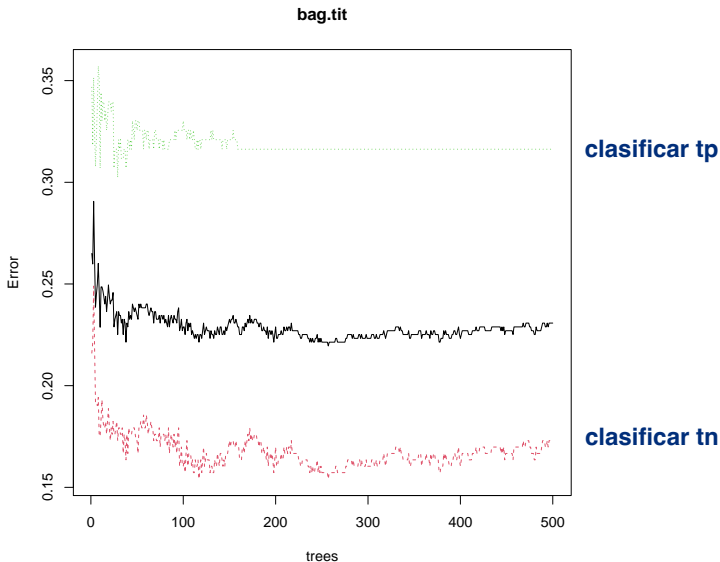
Bagging

Matriz de confusión calculada con los datos que no fueron incluidos en cada muestra bootstrap

	0	1	class.error
0	263.00	55.00	0.17
1	68.00	147.00	0.32

Bagging

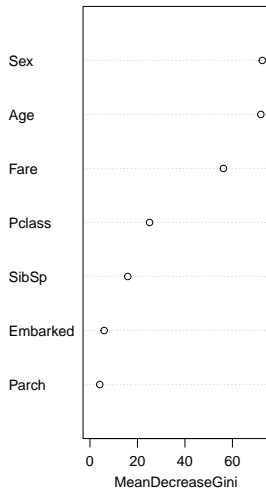
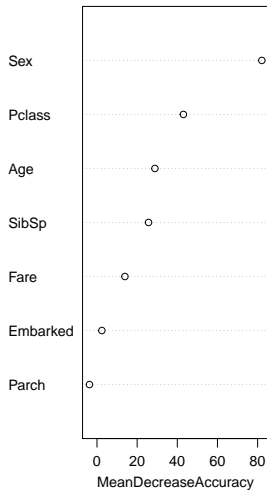
```
plot(bag.tit)
```



Bagging

```
varImpPlot(bag.tit, sort = TRUE)
```

bag.tit



Bagging

Predicción en un nuevo conjunto de datos

pred.tit.bag = predict (bag.tit ,newdata =test)

Mostramos la matriz de confusión

	Pred:0	Pred:1
Actual:0	97	9
Actual:1	18	57

Random Forest

- ▶ Es un modelo superador de *bagging*, busca corregir *tree correlation*.
- ▶ La performance en muchos ejemplos es similar a la de *boosting*, pero es más sencillo de entrenar y ajustar los parámetros.
- ▶ Mantiene la idea de promediar muchos clasificadores con alta varianza pero bajo sesgo (árboles siguen siendo buenos candidatos).

Random Forest

Sean T_1, \dots, T_B árboles *bagged*, y T el árbol de la muestra original,

- ▶ $E(T) = E(T_b)$ para $b = 1, \dots, B \rightsquigarrow$ mismo sesgo.
- ▶ Supongamos $\{T_b\}_1^B$ son i.d. con $\text{var}(T_{b_1}) = \sigma^2$ y $\text{cor}(T_{b_1}, T_{b_2}) = \rho > 0$.

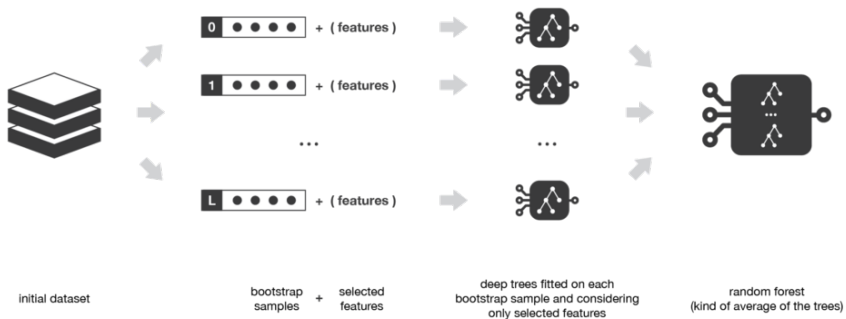
$$\begin{aligned}\text{var}(\bar{T}) &= \frac{\sigma^2}{B} + \frac{B(B-1)}{B^2} \rho \sigma^2 \\ &= \frac{\sigma^2}{B} + \rho \sigma^2 - \frac{\rho \sigma^2}{B} \\ &= \underbrace{\rho \sigma^2}_{\rightarrow 0} + \underbrace{\frac{(1-\rho)\sigma^2}{B}}_{\rightarrow 0}.\end{aligned}$$

Random Forest

Algoritmo: Random Forest

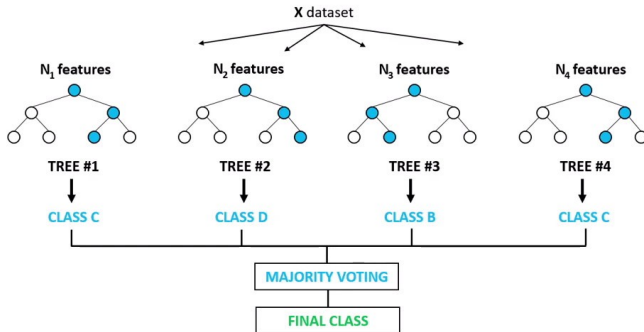
1. Para $b = 1, \dots, B$
 - 1.1 Extraer una muestra bootstrap \mathbf{Z}^* de tamaño n la muestra de entrenamiento.
 - 1.2 Hacer crecer una árbol T_b en base a \mathbf{Z}^* , en cada nodo terminal realizar:
 - 1.2.1 Elegir m variables entre las p posibles.
 - 1.2.2 Elegir el mejor corte entre las m variables.
 - 1.2.3 Hacer la división binaria del nodo.
2. Ensamblar los $\{T_1, \dots, T_B\}$ árboles obtenidos asignando la clase más votada entre los clasificadores de cada muestra bootstrap, $\{\hat{C}_1(x), \dots, \hat{C}_B(x)\}$.

Random Forest



Random Forest


Random Forest Classifier



Random Forest

[https://upload.wikimedia.org/wikipedia/commons/c/c7/
Randomforests_ensemble.gif](https://upload.wikimedia.org/wikipedia/commons/c/c7/Randomforests_ensemble.gif)

Random Forest

En cada paso del *splitting* se eligen m variables, típicamente $m \approx \sqrt{p}$  **para problemas de clasificacion**

Beneficio, reducción de la varianza sobre todo en regresores altamente no lineales \rightsquigarrow árboles.

para problemas de regresion, la cantidad de variables “m” es mas pequeña siendo $m = \sqrt{p/3}$ aproximadamente

Random Forest

Out-of-Bag (oob) error estimate

- ▶ No es necesario separar un conjunto de datos para validar el error.
- ▶ En cada muestra bootstrap, aprox. $1/3$ de los datos son dejados afuera de $Z^* \rightsquigarrow$ usarla para calcular un **estimador insesgado del error de clasificación**
- ▶ Al finalizar, para cada muestra asignar el grupo j como el más votado cada vez que x oob, luego si promediar el nro de veces que $j \neq$ a la verdadera etiqueta.

Random Forest

Importancia de las variables

- ▶ En cada corte de un árbol la disminución del *criterio de corte* da información sobre la disminución de las variables.
- ▶ Para cada variable agregar estos valores a lo largo de todos los árboles del bosque.

Random Forest

Proximidad entre las observaciones

- ▶ Poner en una matriz de $n \times n$ todas las observaciones, inicializarlas en 0
- ▶ Para cada árbol, poner un i en la posición (i, j) si las observaciones x_i y j pertenecen al mismo nodo terminal.
- ▶ Dividir por la cantidad de árboles.

OBSERVACION: Esta es una matriz de similaridad van a ver en Aprendizaje no supervisado una tecnica llamada ESCALAMIENTO MULTIDIMENSIONAL para poder dar una representacion grafica

Random Forest

Variables No Informativa

- ▶ Si la proporción de variables informativas es baja, la probabilidad de que sean elegidas en cada corte de los árboles es baja \Rightarrow muchos árboles van a clasificar mal.
- ▶ Si son muy pocas las variables informativas \rightsquigarrow boosting lo soluciona.
- ▶ Si son bastantes (aunque pocas en comparación) las variables informativas \rightsquigarrow RF es robusto.

Random Forest

Nuevamente Titanic... el análisis es idéntico, aunque ahora conviene $m = \sqrt{7} \approx 2.65 \rightsquigarrow 2$

```
rf.tit = randomForest(Survived ~ Pclass + Sex + Age +  
SibSp + Parch + Fare + Embarked,  
data=train, ntree=500, mtry=2, importance = TRUE)  
rf.tit
```

**toma grupos de 2 features del total
para cada nodo**

Type of random forest:	<i>classification</i>
Number of trees:	500
No. of variables tried at each split:	2
OOB estimate of error rate:	20.08%

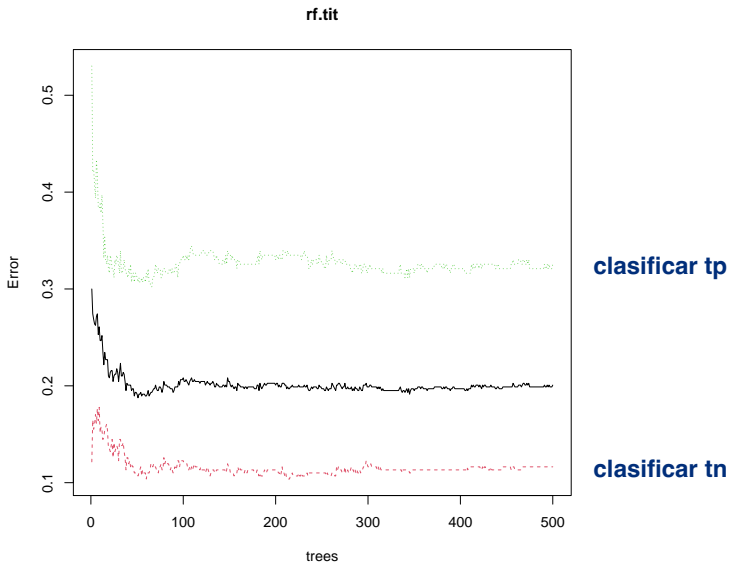
Random Forest

```
rf.tit$confusion
```

	0	1	class.error
0	281.00	37.00	0.12
1	70.00	145.00	0.33

Random Forest

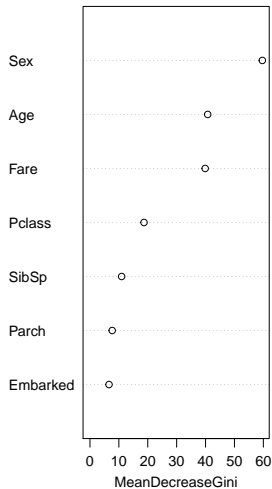
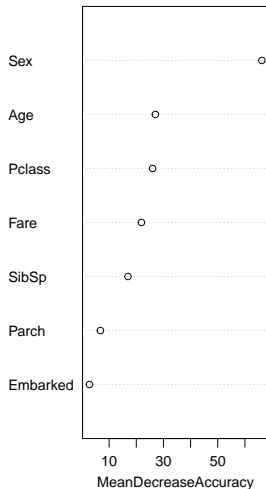
```
plot(rf.tit)
```



Random Forest

```
varImpPlot(rf.tit, sort = TRUE)
```

rf.tit



Random Forest

Predicción en un nuevo conjunto de datos

pred.tit.rf = predict (rf.tit ,newdata =test)

Mostramos la matriz de confusión

	Pred:0	Pred:1
Actual:0	99	7
Actual:1	21	54


Boosting

- ▶ Se busca combinar varios clasificadores sencillos para obtener un nuevo clasificador que tenga en cuenta todos los outputs simultáneamente.
- ▶ Todo estimador debería ser mejorado por boosting.
- ▶ El algoritmo más utilizado es **AdaBoostM1**.
- ▶ problema de clasificación con dos clases $Y \in \{-1, 1\}$.
- ▶ clasificador débil \rightsquigarrow su performance es apenas mejor que asignar al azar.

algoritmo sensillo: Árboles cortos, con profundidad de 1, maximo 3

Boosting

El error de predicción en la muestra de entrenamiento está dado por,

 **error de clasificacion del clasificador debil**

$$\overline{err} = \frac{1}{n} \sum_{i=1}^n \mathcal{I}(y_i \neq g(x_i)),$$

y el error de clasificación para una nueva observación está dado por,

$$E_{XY}(\mathcal{I}(Y \neq g(X)))$$

Boosting

Idea es aplicar secuencialmente el algoritmo débil de clasificación a versiones modificadas de la muestra de entrenamiento, produciendo de este modo una **sucesión de clasificadores débiles**, $g_m(x), m = 1, \dots, M$,
La predicción final está dada por

$$g(x) = \text{signo} \left\{ \sum_{m=1}^M \overset{\text{peso del clasificador}}{\alpha_m} g_m(x) \right\},$$

donde $\alpha_1, \dots, \alpha_M$ son los pesos que calcula el algoritmo boosting, le da mayor importancia a los clasificadores más precisos.

Boosting

El algoritmo: AdaBoost.M1

1. Inicializar los pesos para cada observación como $w_i = \frac{1}{n}$, $i = 1, \dots, n$.
2. Para $m = 1, \dots, M$ *arbol simple*
 - 2.1 Hallar el clasificador $g_m(x)$ en la muestra de entrenamiento, considerando los pesos w_i

2.2 Calcular

$$err_m = \frac{\sum_{i=1}^n w_i \mathcal{I}(y_i \neq g_m(x_i))}{\sum_{i=1}^n w_i}$$

pone un 1, cuando clasifica mal

2.3 Calcular $\alpha_m = \frac{1}{2} \log((1 - err_m)/err_m)$

2.4 Asignar

$$w_i \leftarrow w_i \exp(\alpha_m \mathcal{I}(y_i \neq g_m(x_i)))$$

para $i = 1, \dots, n$.

3. Asignar

$$g(x) = \text{signo} \left\{ \sum_{m=1}^M \alpha_m g_m(x) \right\},$$

Observaciones mas difíciles, tendran al final un mayor peso, siempre que el clasificador tenga un error menor a 0.5.. Inicialmente todos tienen el mismo peso

Boosting



train a weak model
and aggregate it to
the ensemble model



update the training dataset
(values or weights) based on the
current ensemble model results

*initial
dataset*

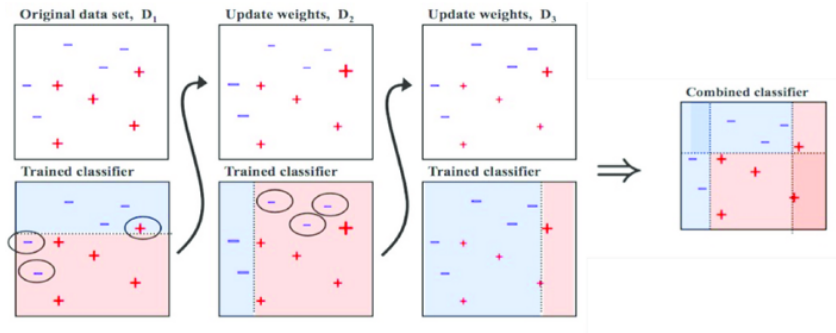


...



*final
model*

Boosting



Boosting

- ▶ En cada paso se actualizan los pesos de las observaciones.
 - ▶ Si $y_i = g(x_i) \Rightarrow w_i$ queda igual que en el paso anterior.
 - ▶ Si $y_i \neq g(x_i) \Rightarrow w_i = w_i \exp(\alpha_m)$, si $err_m < 0.5$ el peso de la observación aumenta.
- ▶ En los pasos sucesivos las observaciones difíciles de clasificar correctamente reciben más peso.
- ▶ El estimador boost tiene menor error de clasificación en los sucesivos pasos, supongamos que el error de clasificación del estimador débil está dado por err_m

$$g_M(x) \leq \prod_{m=1}^M \sqrt{err_m(1 - err_m)}$$

Boosting

Ventajas Boosting

- ▶ **Rápido.**
- ▶ **Sin parámetros** para ajustar, salvo M .
- ▶ **Flexible**, se puede combinar con cualquier algoritmo de aprendizaje.
- ▶ **Sin conocimiento previo** del algoritmo de aprendizaje.
- ▶ **probablemente efectivo**, si se pueden encontrar clasificadores débiles.
- ▶ **Versatil**, se puede aplicar con datos numéricos, categóricos, textuales, etc.

Boosting

Desventajas Boosting

- ▶ el desempeño de AdaBoost depende de los **datos** y de **reglas débiles de clasificación**.
- ▶ AdaBoost puede **fallar** si:
 - ▶ el clasificador débil es muy **complejo**.
 - ▶ el clasificador débil es muy **débil**, i.e $err_m \rightarrow 0.5$
- ▶ AdaBoost es susceptible a variables **ruido**.

Boosting

Extensión a K clases, **todos contra uno**.

- ▶ Partir el problema en K problemas binarios y resolverlos en forma separada.
- ▶ Clasificar una nueva observación en la categoría **más** votada.

Boosting

Retomamos el ejemplo de Titanic...

```
library("C50")  
train$Survived=as.factor(train$Survived)  
boost.tit = C5.0(Survived ~ Pclass + Sex + Age +  
SibSp + Parch + Fare , data=train, trials=500)
```

Classification Tree

Number of samples: 674

Number of predictors: 6

Number of boosting iterations: 50 requested; 18 used due to early stopping





Average tree size: 5

Non-standard options: attempt to group attributes

Boosting

```
xtest=test[,-c(1,2)]  
pred= predict(boost.tit, newdata = xtest, type =  
"class")  
table(predicho = pred, real = test$Survived)
```

	0	1
pred.0	125	33
pred.1	6	53

-  L. Breiman, J. Friedman, R. Olsen, C. Stone
Classification and Regression Trees
Chapman and Hall, 1984.
-  T. Hastie, R. Tibshirani, J. Friedman.
Elements of Statistical Learning, 2nd Ed.
Springer-Verlag, 2009.
Capítulo 9, 12 y 15.
-  J. Garret, D. Witten, T. Hastie, R. Tibshirani.
An Introduction to Statistical Learning.
Springer-Verlag, 2013.
Capítulo .
-  R.E. Shapiro
The Boosting Approach to Machine Learning.An Overview
MSRI Workshop on Nonlinear Estimation and Classification,
2002. Las cuentas detras de boosting