CLASIFICACIÓN: ÁRBOLES Y REGLAS

TRABAJO EN EL LABORATORIO (SESIÓN 3)

BOOK: Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

An Introduction to Statistical Learning with Applications in R

Springer, 2013

Chapter 08

Sumario

1. Árboles de decisión para clasificación (Script4)
   1. Construcción de un árbol de decisión tree()
   2. Capacidad de predicción
   3. Random Forest randomForest()
   4. Ejercicio 3.1.
   5. Boosting gbm()
   6. Otras posibilidades de árboles: RWeka y C4.5 J48()
2. Algoritmos de reglas para clasificación (Script 5)
   1. Ripper JRip()
   2. PART PART()
   3. Ejercicios del 3.2. al 3.5.

Árboles de decisión para clasificación

1. Construcción de un árbol de decisión tree()
2. Random Forest randomForest()
3. Ejercicio 3.1.
4. Boosting gbm()
5. Otras posibilidades de árboles: RWeka y C4.5 J48()

Árboles de decisión para clasificación

Construcción de un árbol de decisión

- Haremos uso del *script4_Arboles.R*

- Los árboles de decisión son un método tradicional muy conocido para abordar los problemas de clasificación supervisada.

- Tienen una gran ventaja, requieren muy poco tiempo para aprender, lo que los hace muy eficiente en este sentido frente a otras aproximación que usan otras formas de representar el conocimiento.

- También esta representación es su principal deficiencia. El uso de árboles sesga el tipo de conocimiento que se puede aprender y hace que pierdan capacidad de expresividad frente a otros modelos.

- Otro inconveniente es que suelen ser muy vulnerables al ruido.

Árboles de decisión para clasificación

Construcción de un árbol de decisión

- Nos situamos en la primera línea del script.
- Esta primera línea carga la biblioteca para usar árboles de decisión.

Comando en R

```
library(tree)
```

- Las siguientes sentencias construyen un árbol de clasificación para el problema del iris, fijando como variable independiente “Species” y fijando el resto de variables como dependientes.

Comando en R

```
# Construir un árbol que clasifica la “species” en base al resto de variables
tree.iris = tree(Species~.,iris)
```

Árboles de decisión para clasificación

Construcción de un árbol de decisión

- El siguiente comando nos da información sobre el árbol que se ha aprendido.

Comando en R

```
summary(tree.iris)
```

Resultado

```
Classification tree:
tree(formula = Species ~ ., data = iris)
Variables actually used in tree construction:
[1] "Petal.Length" "Petal.Width" "Sepal.Length"
Number of terminal nodes: 6
Residual mean deviance: 0.1253 = 18.05 / 144
Misclassification error rate: 0.02667 = 4 / 150
```

Variables involucradas

Número de hojas

Error

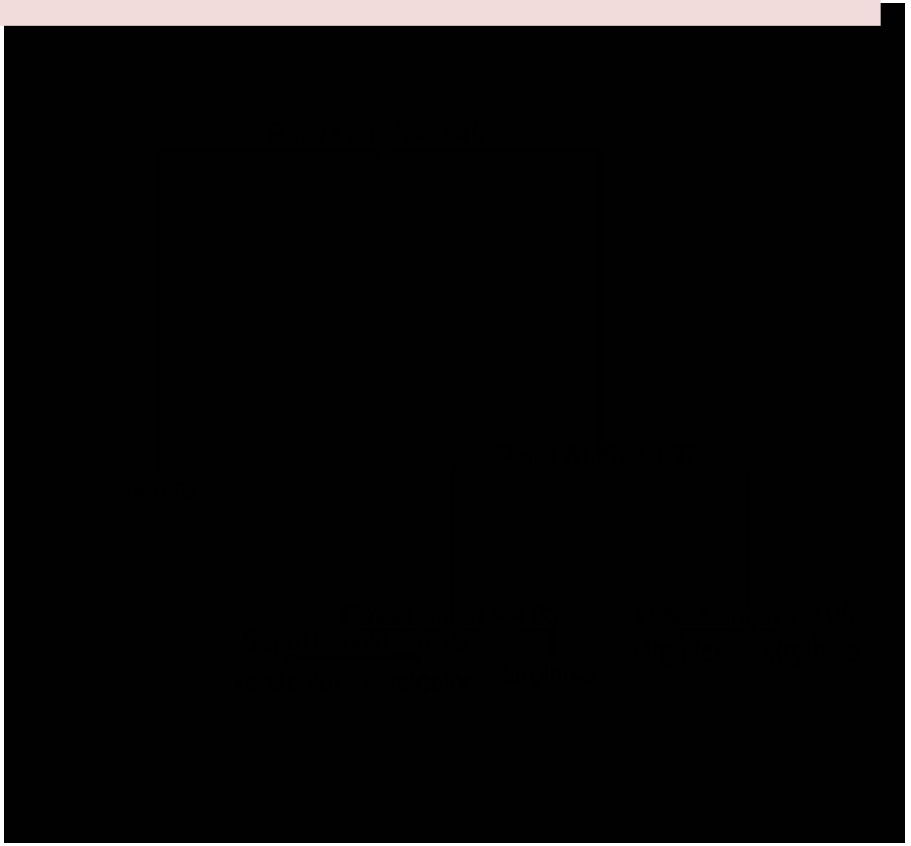
Árboles de decisión para clasificación

Construcción de un árbol de decisión

- Las dos siguientes instrucciones permiten dibujar el árbol.

Comando en R

```
plot(tree.iris)
text(tree.iris, pretty=0)
```



- Como se puede observar, el dibujo podría mejorarse, pero a pesar de todo, expresa de forma relativamente simple el comportamiento de los datos.

Árboles de decisión para clasificación

Construcción de un árbol de decisión

- Podemos ver la descripción del mismo árbol en modo texto con más información.

Comando en R

```
tree.iris
```

Resultado

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
  2) Petal.Length < 2.45 50 0.000 setosa ( 1.00000 0.00000 0.00000 ) *
  3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
    6) Petal.Width < 1.75 54 33.320 versicolor ( 0.00000 0.90741 0.09259 )
     12) Petal.Length < 4.95 48 9.721 versicolor ( 0.00000 0.97917 0.02083 )
       24) Sepal.Length < 5.15 5 5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
       25) Sepal.Length > 5.15 43 0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
     13) Petal.Length > 4.95 6 7.638 virginica ( 0.00000 0.33333 0.66667 ) *
    7) Petal.Width > 1.75 46 9.635 virginica ( 0.00000 0.02174 0.97826 )
     14) Petal.Length < 4.95 6 5.407 virginica ( 0.00000 0.16667 0.83333 ) *
     15) Petal.Length > 4.95 40 0.000 virginica ( 0.00000 0.00000 1.00000 ) *
```

Árboles de decisión para clasificación

Capacidad de predicción

- TEST/TRAINING

Comando en R

```
# Dividir en training y test
set.seed (2)
train=sample (1:nrow(iris), 100)
iris.test=iris [-train ,]

# Construyo el arbol sobre el conjunto de entrenamiento
tree.iris =tree(Species~. ,iris ,subset =train )

# Aplico el arbol sobre el conjunto de test
tree.pred =predict (tree.iris ,iris.test ,type ="class")

# Visualizo la matriz de confusion
table(tree.pred , iris.test[,5])
```

Árboles de decisión para clasificación

Capacidad de predicción

- TEST/TRAINING
- La matriz de confusión sobre el conjunto de test muestra en la diagonal principal los aciertos del modelo. Los valores fuera de esa diagonal son los errores. En este caso, un ejemplo de virginica ha sido clasificado como versicolor.

	Resultado		
tree.pred	setosa	versicolor	virginica
setosa	15	0	0
versicolor	0	16	0
virginica	0	1	18

- El error que se comete es por tanto de 1/50 = 2% y por consiguiente, un acierto del 98%

Árboles de decisión para clasificación

Validación cruzada / Poda

- En la biblioteca de <tree> hay definido un proceso de validación cruzada para trabajar con los árboles, diseñado para encontrar una poda apropiada del árbol.

- Los mecanismos de poda en los árboles son habituales y permiten reducir la complejidad del árbol sin perder excesiva capacidad de predicción.

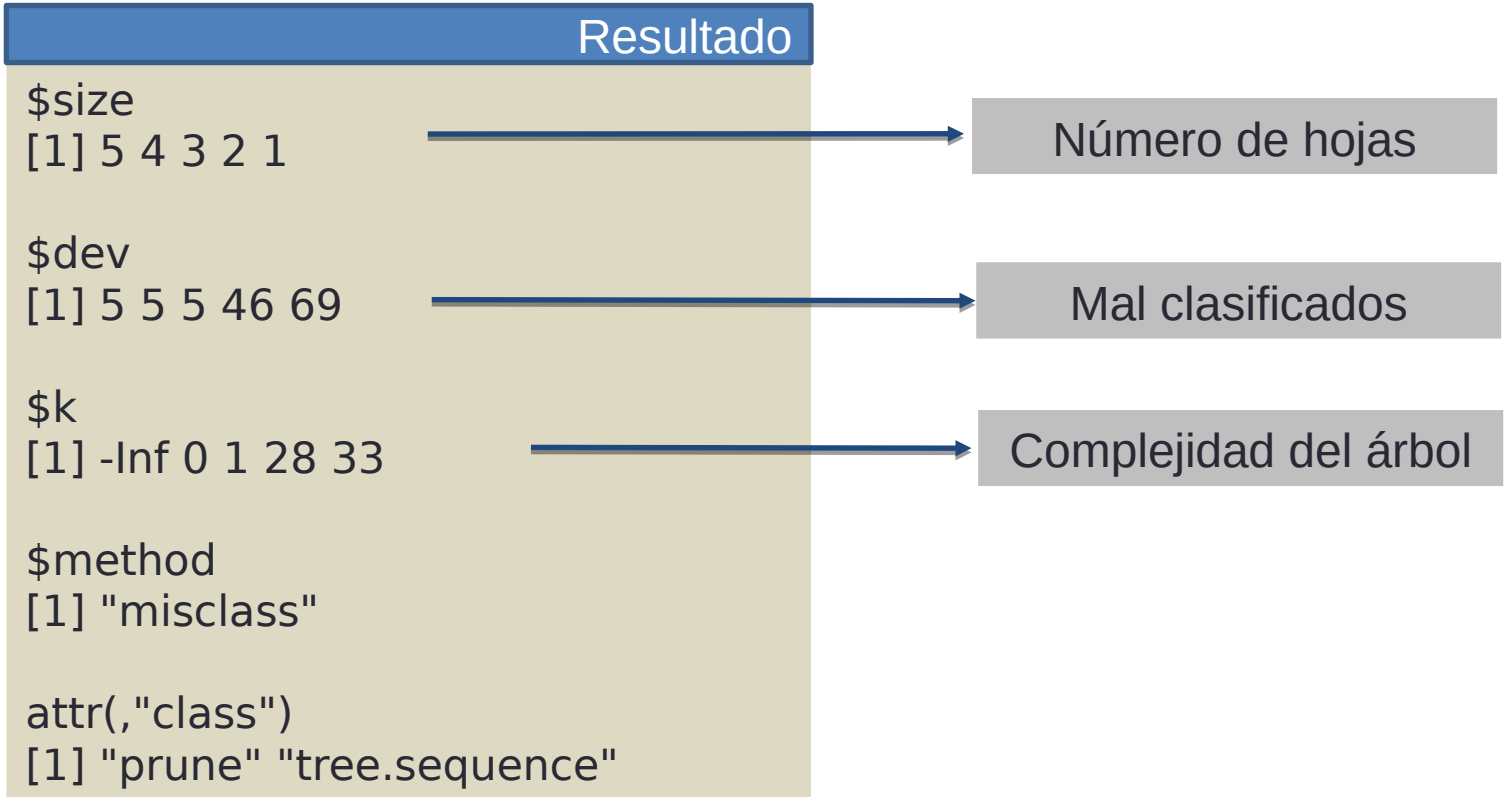
Comando en R

```
# Podar el arbol usando cv
set.seed (3)
cv.iris = cv.tree(tree.iris ,FUN=prune.misclass )
names(cv.iris )
cv.iris
```

Árboles de decisión para clasificación

Validación cruzada / Poda

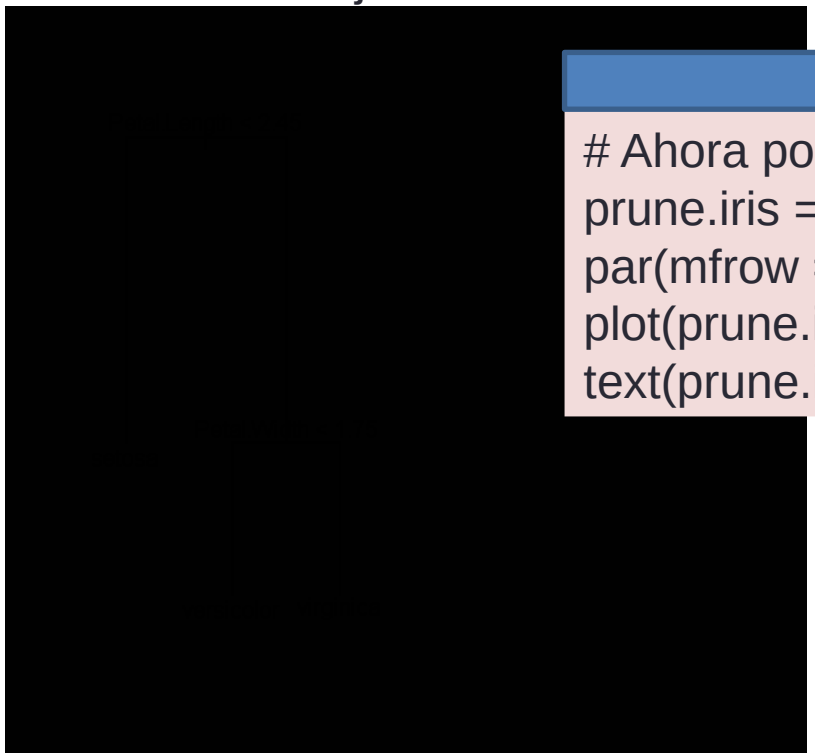
- El resultado que ilustra el comando cv.iris es el siguiente:



Árboles de decisión para clasificación

Validación cruzada / Poda

- Aquí usamos la función prune.misclass() que dado un árbol, te devuelve la mejor poda de dicho árbol reduciendo los errores de clasificación. Se le añade un parámetro indicándolo el número de nodos hojas.
- Así, en el ejemplo siguiente, buscamos la mejor poda del árbol tree.iris con tres nodos hoja.



Comando en R

```
# Ahora podamos el arbol con prune.misclass
prune.iris =prune.misclass (tree.iris ,best =3)
par(mfrow =c(1,1))
plot(prune.iris)
text(prune.iris ,pretty =0)
```

Árboles de decisión para clasificación

Validación cruzada / Poda

- Comprobamos la capacidad de predicción del árbol podado.

Comando en R

```
tree.pred=predict (prune.iris , iris.test ,type="class")
table(tree.pred ,iris.test[,5])
```

Resultado

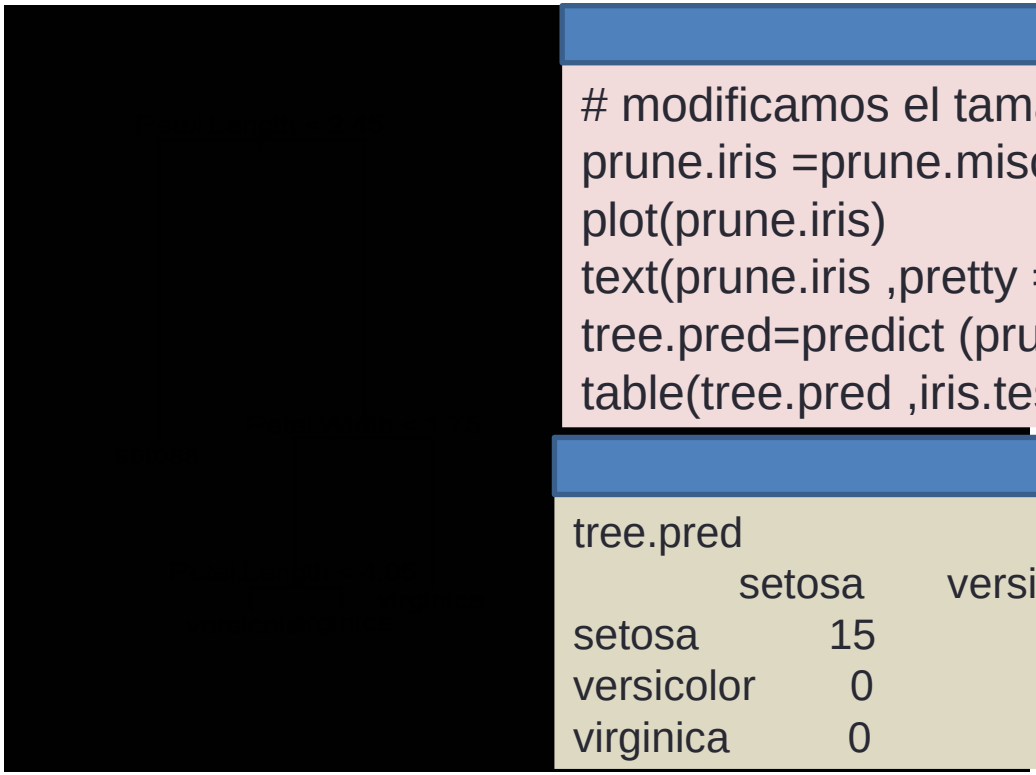
```
tree.pred
             setosa versicolor virginica
setosa           15          0         0
versicolor        0         16         1
virginica         0          1        17
```

- El error es de 2/50=4%, por consiguiente el acierto es del 96%

Árboles de decisión para clasificación

Validación cruzada / Poda

- ¿Qué ocurre si buscamos el mejor árbol con 4 hojas?



Comando en R

```
# modificamos el tamanio del arbol modificando best
prune.iris =prune.misclass (tree.iris ,best =4)
plot(prune.iris)
text(prune.iris ,pretty =0)
tree.pred=predict (prune.iris , iris.test ,type="class")
table(tree.pred ,iris.test[,5])
```

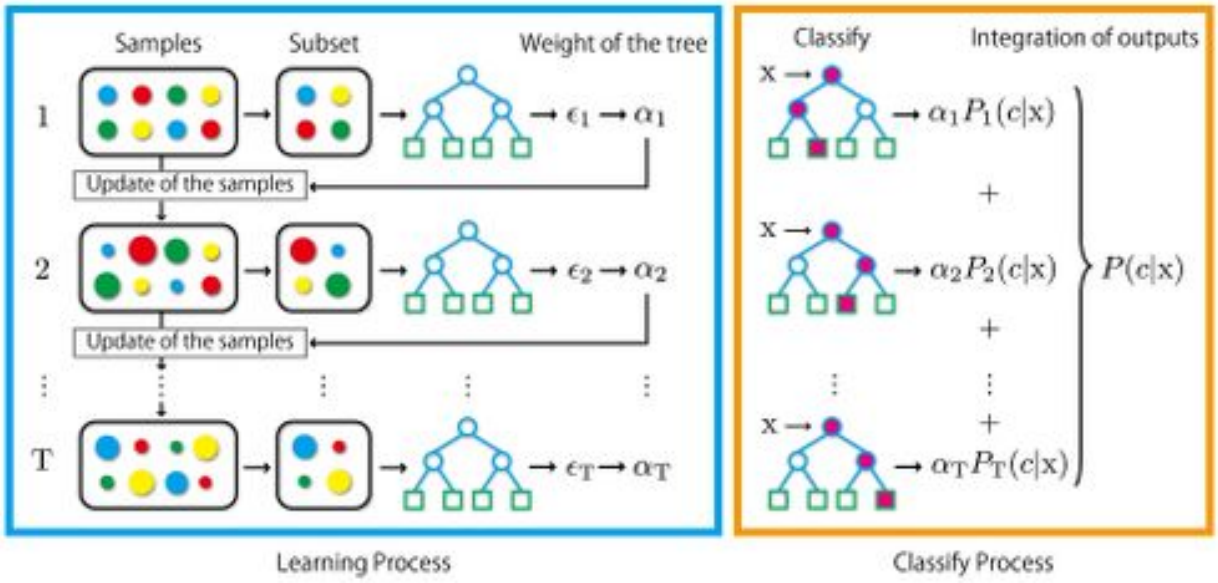
Resultado

```
tree.pred
            setosa versicolor virginica
setosa          15          0         0
versicolor       0         16         0
virginica        0          1        18
```

- El acierto es del 98%

Árboles de decisión para clasificación
Random Forest

- Random Forest es quizás el método más conocido de “ensemble” y está considerado como “el mejor” método de clasificación.



- Random Forest hace uso de “Bagging”.

Árboles de decisión para clasificación

Random Forest

- En R, es bastante fácil usar “Random Forest”.

Comando en R

```
# Random Forest
library (randomForest)
set.seed (1)
bag.iris = randomForest(Species~., data=iris, subset=train)
bag.iris
```

Resultado

```
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of error rate: 6%

Confusion matrix:
           setosa versicolor virginica class.error
setosa         35          0         0  0.00000000
versicolor      0         30         3  0.09090909
virginica       0          3        29  0.09375000
```

Árboles de decisión para clasificación
Random Forest

- Ahora predecimos sobre los ejemplos de test.

Comando en R

```
yhat.bag = predict (bag.iris ,newdata =iris.test)
yhat.bag
```

- Nos da para cada uno de los ejemplos la clase a la que es asociada.
- Para obtener el porcentaje de acierto definimos la siguiente función:

Comando en R

```
acierto <- function(bag.datos){
  return (sum (sapply(1:length(bag.datos$y), function(x){
    if (is.na(bag.datos$predicted[x])){
      0
    }
    else if (as.numeric(bag.datos$y[x])==as.numeric(bag.datos$predicted[x])){
      1
    }
    else{
      0
    }
  }))/length(bag.datos$y))
}
```

Árboles de decisión para clasificación
Random Forest

- Para usar la función anterior, tenemos que hacer lo siguiente:

Comando en R

```
resul = as.data.frame(cbind(predicted = yhat.bag, y=iris.test[,5]))
acierto(resul)
```

- El resultado de acierto que se obtiene es del 96%.

- Podemos cambiar el número de árboles de la siguiente forma:

Comando en R

```
bag.iris = randomForest(Species~.,data=iris ,subset =train , ntree=25)
bag.iris
```

Ejercicio 3.1.

1. Construye una función en R que permita expresar gráficamente el comportamiento de un Random Forest en función del número de árboles para el problema del IRIS.

   A. ¿Cuál es el número de árboles a partir del cual no se produce una mejora sustancial?

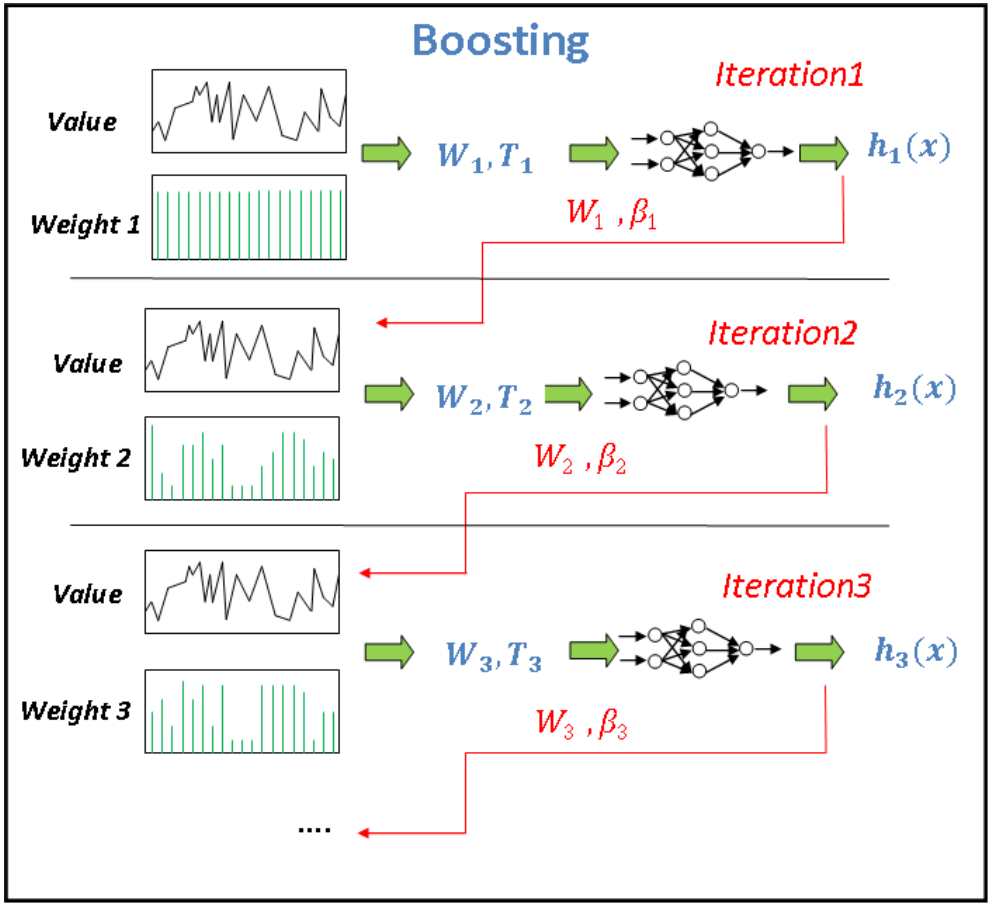
2. Aplica la función anterior a la base de datos “Auto” que puedes encontrar en <library(ISLR)>, tomando como factor para la clasificación el atributo de la columna 8 y no teniendo en cuenta la columna 9 (columna de nombre “name”)

   A. ¿Se produce un comportamiento semejante al observado en la base de datos IRIS?
   B. En este caso, ¿cuál es el valor del número de árboles apropiado?

Árboles de decisión para clasificación

Boosting

- El “boosting” es un proceso iterativo que consiste en modificar el peso de los ejemplos de la muestra original con el fin de determinar los ejemplos relevantes y mejorar así el comportamiento del clasificador.



Árboles de decisión para clasificación
Boosting

- Hacemos uso de la biblioteca gdm y construimos el modelo.

Comando en R

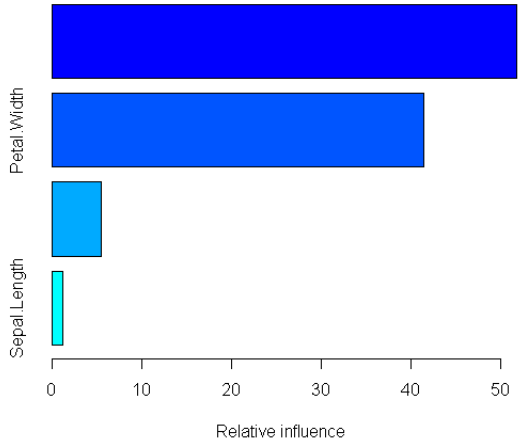
```
library (gbm)
set.seed (1)

boost.iris =gbm(Species~.,data=iris[train,],
                distribution="multinomial",n.trees =5000,
                interaction.depth =4)

summary(boost.iris)
```

- gdm se usa tanto para regresión como para clasificación. En el modelo de clasificación es necesario fijar distribution=“multinomial”. El parámetro interaction.depth fija la profundidad máxima del árbol, y obviamente n.trees fija el número de iteraciones o árboles a construir.

Árboles de decisión para clasificación

Boosting

- El resultado que se tiene es la relevancia de cada uno de los atributos y se muestra en una gráfica como la siguiente:



Comando en R

```
yhat.boost=predict (boost.iris ,newdata =iris[-train,], n.trees =5000)

yhat.boost
```

- Esta secuencia de instrucciones establece la capacidad de predicción del modelo creado sobre el conjunto de entrenamiento.

Árboles de decisión para clasificación
Boosting

- Cuando miramos yhat.boost, la variable que almacena el resultado de la predicción, observamos que para cada posible salida nos da un valor real.
- La clase asociada a cada ejemplo es aquella que alcanza el máximo valor. Por consiguiente para conocer el porcentaje de acierto hacemos:

Comando en R

```
# Obtenemos el porcentaje de acierto
yhat.boost = matrix(yhat.boost,ncol=(length(yhat.boost)/nrow(iris.test)))

yhat.boost.y = sapply(1:nrow(iris.test), function(x){
  which.max(yhat.boost[x,])
})

(sum(yhat.boost.y == as.numeric(iris.test[,5]))/nrow(iris.test))*100
```

- El porcentaje de acierto es del 96%.

Árboles de decisión para clasificación
Otras posibilidades de árboles: Rweka y C4.5

- La biblioteca Rweka de R implementa algunos modelos de árboles de decisión de la plataforma Weka.
- En concreto, estamos interesado en usar el árbol de decisión más conocido que es C4.5.
- Los arboles vistos hasta ahora se basan en CART, que son árboles binarios cuyo criterio de decisión se basa en la medida de GINI.
- A diferencia de CART, C4.5 implementa árboles de decisión usando como factor de división una medida de entropía.
- En Weka, C4.5 recibe se denomina J48

Árboles de decisión para clasificación

Otras posibilidades de árboles: Rweka y C4.5

- Vamos a ver un ejemplo de uso de esta biblioteca y este algoritmo.

Comando en R

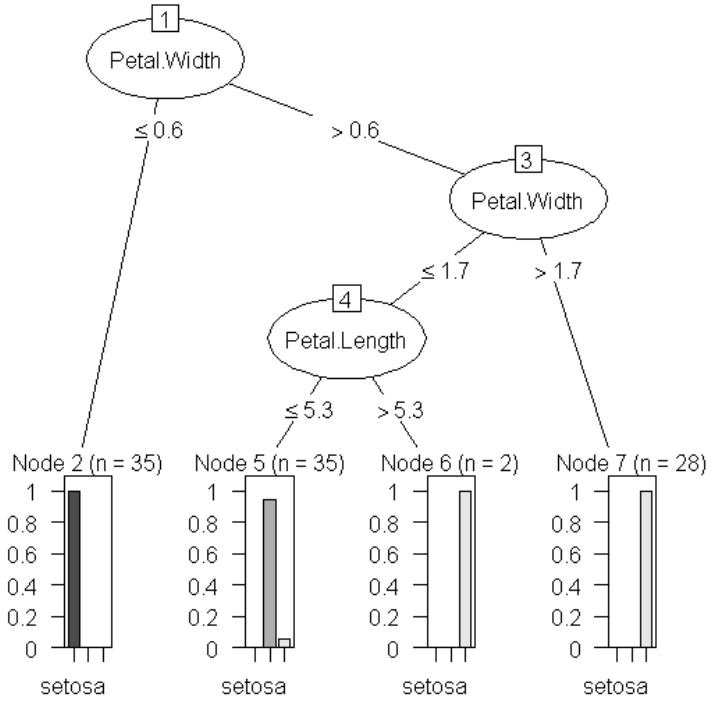
```
library(RWeka)

modelC4.5 = J48(Species~., data=iris, subset=train)
```

- Su uso es semejante al comando tree.

Comando en R

```
library(partykit)
plot(modelC4.5)
```

- La biblioteca partykit permite mejorar la representación gráfica del árbol de decisión.



Árboles de decisión para clasificación

Otras posibilidades de árboles: Rweka y C4.5

- Evaluamos la capacidad de predicción del modelo de la siguiente forma:

Comando en R

```
resul = as.data.frame(cbind(predicted = modelC4.5.pred, y=iris.test[,5]))
acierto(resul)
```

- El resultado obtenido es del 96%.

- Una de las ventajas de usar Rweka es que tiene un comando simple para realizar la validación cruzada de los modelos que se obtienen con sus algoritmos.

Comando en R

```
modelC4.5 = J48(Species~., data=iris)
cv_resul = evaluate_Weka_classifier(modelC4.5, numFolds=10)
cv_resul
```

- donde numFolds indica el número de subconjuntos de la muestra.
- cv_resul muestra detalla información sobre el proceso de validación.

Algoritmos de reglas para clasificación

- Haremos uso del *script5_Rules.R*
- Las reglas de clasificación son elementos de representación de conocimiento que ofrecen una mayor capacidad de expresividad frente a los árboles de decisión.
- A diferencia de estos, normalmente los algoritmos de aprendizaje que los extraen suelen ser más lentos y por consiguiente, requieren más tiempo para sintetizar el modelo.
- Aquí veremos dos ejemplos de algoritmos de aprendizaje basados en reglas.

Algoritmos de reglas para clasificación

Ripper JRip()

- Haremos uso directamente de la biblioteca Rweka de R para usar Ripper.
- Ejecutar las primeras 6 líneas del script, para cargar Rweka y construir los datos de training y test.

Comando en R

```
# Aplico el algoritmo Ripper
model.Ripper = JRip(Species~., iris, subset=train)
summary(model.Ripper)

model.Ripper.pred = predict(model.Ripper, newdata = iris.test)

# Acierto
(sum(model.Ripper.pred == iris.test[,5])/nrow(iris.test)*100)
```

- Su uso es análogo al ya visto para árboles
- Su capacidad de predicción sobre el conjunto de test es del 98%.

Algoritmos de reglas para clasificación

Ripper JRip()

- Las reglas que extrae Ripper para este problema se ven al ejecutar:

Comando en R	Resultado
model.Ripper	JRIP rules: ========== (Petal.Width >= 1.7) => Species=virginica (30.0/1.0) (Petal.Length >= 5) => Species=virginica (4.0/1.0) (Petal.Length >= 3) => Species=versicolor (31.0/0.0) => Species=setosa (35.0/0.0) Number of Rules : 4

- Podemos observar que obtiene 4 reglas, la última es una regla por defecto, es decir, si ninguna de las anteriores se verifica, entonces la clase devuelta es setosa.

- Junto a cada regla aparece el número de aciertos y fallos en el formato (aciertos/fallos).

Algoritmos de reglas para clasificación

Part Part()

- Otro algoritmo bastante conocido de extracción de reglas es Part, y también viene incluido en el paquete de Rweka. Un ejemplo de uso es el siguiente:

Comando en R

```
model.Part = PART(Species~., iris, subset=train)
summary(model.Part)

model.Part.pred = predict(model.Part, newdata = iris.test)

# Acierto
(sum(model.Part.pred == iris.test[,5])/nrow(iris.test)*100)
```

- Su capacidad de predicción sobre el conjunto de test es del 94%.

Algoritmos de reglas para clasificación

Part Part()

- Las reglas que extrae Part son:

Comando en R	Resultado
model.Part	PART decision list ---------- Petal.Width <= 0.6: setosa (35.0) Petal.Width > 1.7: virginica (28.0) Petal.Length <= 5.3 AND Petal.Width <= 1.4: versicolor (23.0) Petal.Length <= 5.3 AND Sepal.Width > 2.6: versicolor (9.0) : virginica (5.0/1.0) Number of Rules : 5

- En este caso se obtienen 5 reglas, y en este caso la regla por defecto involucra a la clase virginica.
- En este caso aparecen reglas que involucran varios atributos mediante el operador lógico AND.

Algoritmos de reglas para clasificación
Validación Cruzada

- Como ya comentamos, una ventaja de Rweka es que permite realizar de forma muy simple el proceso de validación cruzada.
- Vamos a aplicárselo a los modelos de Ripper y Part que hemos definido antes.

Comando en R

```
model.Ripper = JRip(Species~., iris)
cv_JRip = evaluate_Weka_classifier(model.Ripper,numFolds=nrow(iris))
# Acierto
cv_JRip$details[1]
```

- El resultado es 95.33%.
- Si lo hacemos de igual forma con Part, observamos que en su caso el resultado obtenido es de 94.66%

Ejercicio 3.2.

Los modelos basados en GAM sólo pueden abordar problemas de clasificación binaria. Una idea para adaptarlos a la clasificación de n clases, con n>2 consiste en construir n-1 clasificaciones binarios y combinar sus salidas para obtener la verdadera clasificación.

A) Implementa en R esta idea de combinar clasificadores binarios para resolver el problema de clasificación de IRIS (tiene 3 clases) con una aproximación OVA.

B) Haz lo mismo del apartado anterior, pero con una aproximación OVO.

Ejercicio 3.3.

1. Hemos visto que *“bagging”*, *“randomForest”* y *“boosting”* son técnicas que pueden ser útiles para mejorar la capacidad de predicción de los algoritmos de clasificación. En RWeka no está disponible esta posibilidad para algoritmos basados en reglas. Se pide:

   A. Define una función de R que permita realizar *“bagging”* con RIPPER.

   B. Define una función de R que permita realizar algo similar al *“randomForest”* pero con RIPPER, haciendo uso de la función *“bagging”* definida en la sección anterior.

   C. Realiza un estudio experimental usando al menos 10 bases de datos “arff” de WEKA para clasificación y los algoritmos RIPPER, RIPPER con *“bagging”* (apartado A) y RIPPER con *“randomForest”* (apartado B). ¿Se verifica que se produce una mejora en la capacidad de predicción?

Ejercicio 3.4.

1. Usa la base de datos *CoordenadasMunicipios.csv*, que contiene las coordenadas GPS de los municipios de Andalucía y la provincia a la que pertenecen. Se trata de determinar la provincia de un municipio a partir de su posición GPS. Visualizar los espacios de decisión de los siguientes clasificadores:
   - (a) un multiclasificador basado en regresión logística usando un modelo no lineal con spline naturales de hasta grado 4. (glm)
   - (b) un árbol de decisión tipo CART (tree)
   - (c) RandomForest con 500 árboles (randomForest)
   - (d) Boosting con 500 árboles (gbm)
   - (e) C4.5 (J48)
   - (f) Ripper (JRip)
2. ¿Qué porcentaje de los espacios de decisión comparten todos los clasificadores anteriores para este problema? Visualiza estas zonas de decisión común.

Toma el Script5b_EspaciosDeDecision.R como base para realizar este ejercicio.

Ejercicio 3.5.

1. Usa la base de datos *DatosSocialesAndalucia.csv*, que contiene información sobre los municipios andaluces. En concreto, las siguientes variables son : Provincia, Población Total, Porcentaje de hombres y de mujeres, Porcentaje de la población entre 20 y 65 años, Incremento de la población en los últimos 10 años, Porcentaje de extranjeros, Porcentaje de Nacimientos y Defunciones, Número de matrimonios de personas de distinto sexo y Número de vehículos particulares. Responde a las siguientes preguntas:

2. Usando las variables anteriores, ¿qué otra provincia andaluza se parece más a la provincia de Granada?

3. ¿Cuál es la variable de entre las involucradas en los datos que permite discriminar más entre los municipios de Sevilla y de Almería?

4. ¿Alguna provincia se puede discriminar claramente del resto?