semana 3

luis

11/7/2021

Contents

prediciendo con arboles **1**
Notas y recursos adicionales 7
Notas y otros recursos 11

Random forests **11**
Notas y otros recursos 15

boosting **15**
Notas y lectura adicional 20

prediccion basada en modelos **20**
notas y futuras lecturas 23

prediciendo con arboles

Ideas claves

- Dividir iterativamente las variables en grupos
- Evaluar la "homogeneidad" dentro de cada grupo.
- Dividir de nuevo si es necesario

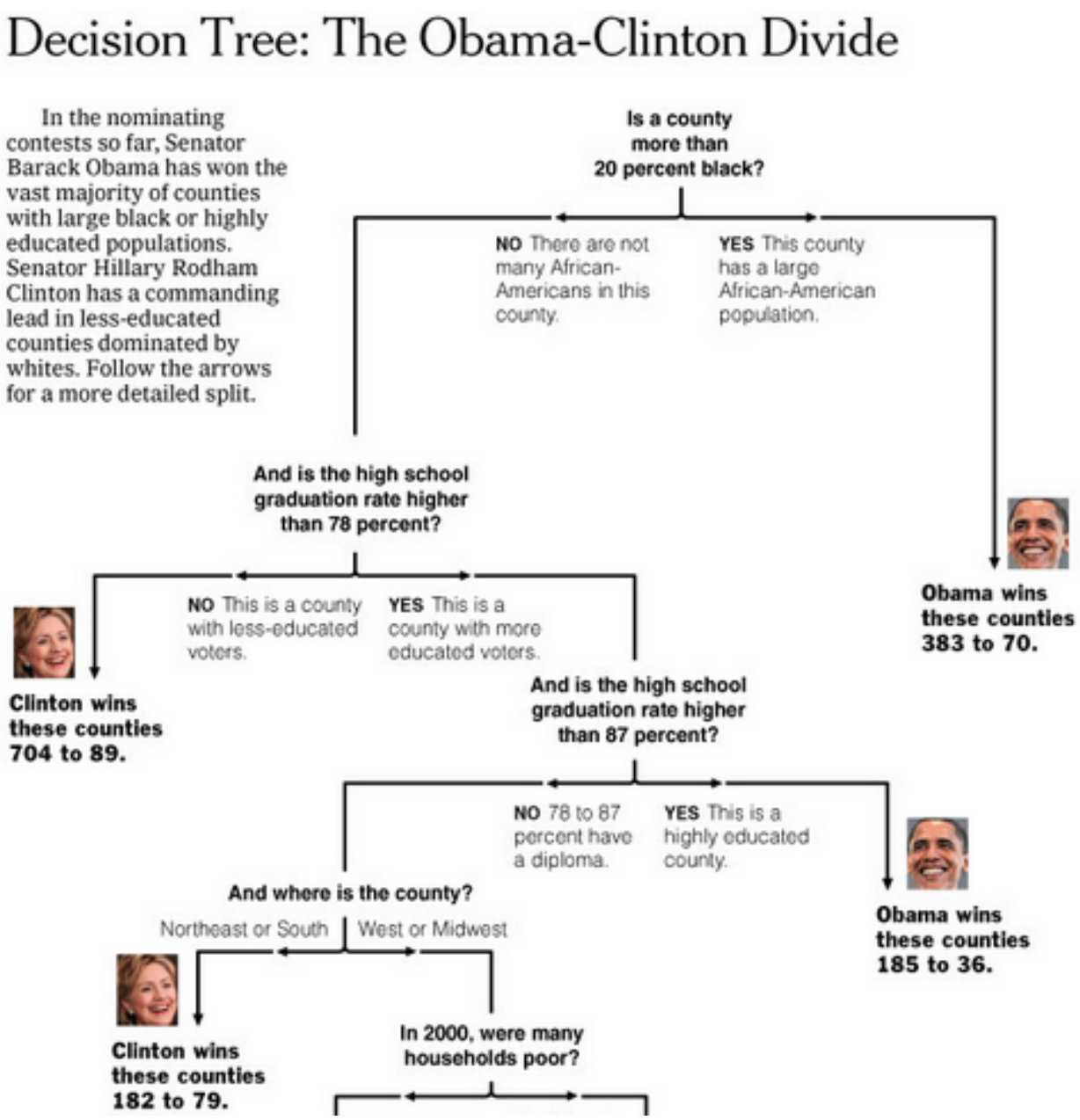
Pros:

- Fácil de interpretar
- Mejor rendimiento en entornos no lineales

Contras:

- Sin poda / validación cruzada puede provocar un sobreajuste
- Más difícil de estimar la incertidumbre
- Los resultados pueden variar

ejemplo de un arbol



Algoritmo basico

1. Comience con todas las variables en un grupo
2. Encuentre la variable / división que mejor separe los resultados
3. Divida los datos en dos grupos ("hojas") en esa división ("nodo")
4. Dentro de cada división, encuentre la mejor variable / división que separe los resultados
5. Continúe hasta que los grupos sean demasiado pequeños o suficientemente "puros"

medidas de impureza

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \; in \; Leaf \; m} \mathbb{1}(y_i = k)$$

Error de clasificación errónea:

$$1 - \hat{p}_{mk(m)}; k(m) = \text{most; common; k}$$

- 0 = pureza perfecta
- 0.5 = sin pureza

Índice de Gini:

$$\sum_{k \neq k'} \hat{p}_{mk} \times \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} p_{mk}^2$$

- 0 = pureza perfecta
- 0.5 = sin pureza

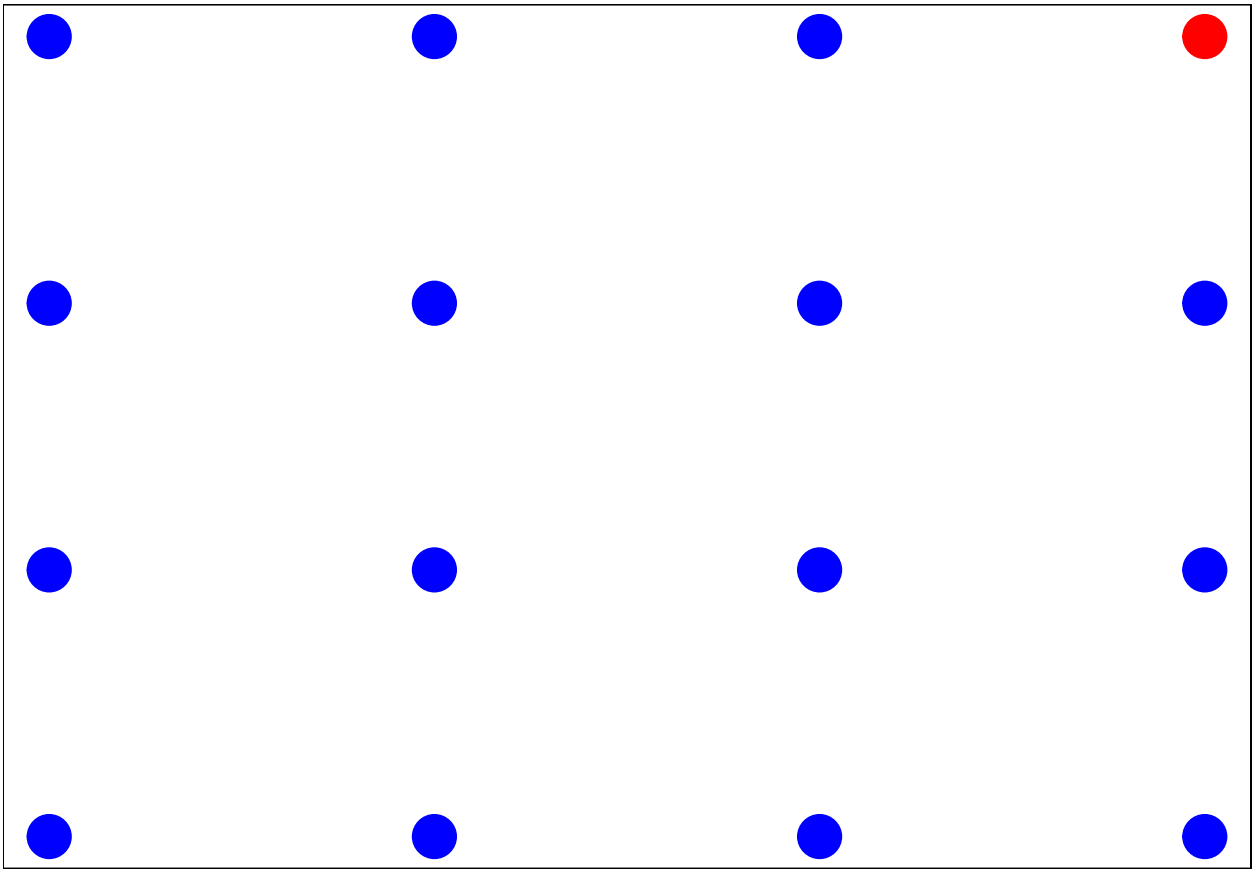
Desviación / ganancia de información:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

* 0 = pureza perfecta * 1 = sin pureza

http://en.wikipedia.org/wiki/Decision_tree_learning

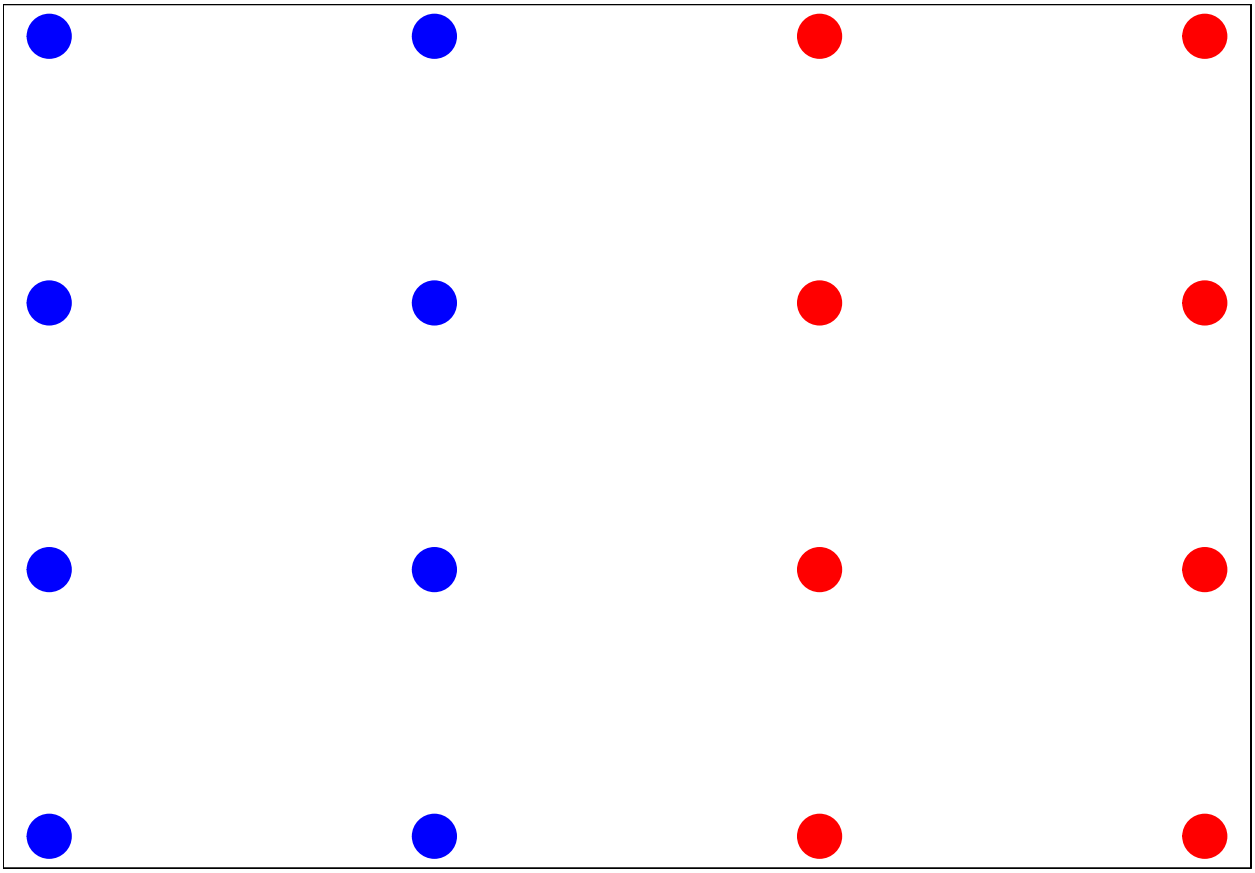
```
par(mar=c(0,0,0,0)); set.seed(1234); x = rep(1:4,each=4); y = rep(1:4,4)
plot(x,y,xaxt="n",yaxt="n",cex=3,col=c(rep("blue",15),rep("red",1)),pch=19)
```



- **Clasificación errónea:** $1/16 = 0.06$
- **Gini:** $1 - [(1/16)^2 + (15/16)^2] = 0.12$

- **Información:** $-[1/16 \times log2(1/16) + 15/16 \times log2(15/16)] = 0.34$

```
par(mar=c(0,0,0,0));
plot(x,y,xaxt="n",yaxt="n",cex=3,col=c(rep("blue",8),rep("red",8)),pch=19)
```



- **Misclassification:** $8/16 = 0.5$
- **Gini:** $1 - [(8/16)^2 + (8/16)^2] = 0.5$
- **Information:** $-[1/16 \times log2(1/16) + 15/16 \times log2(15/16)] = 1$

ejemplo con datos iris

```
library(caret)
data(iris); library(ggplot2)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

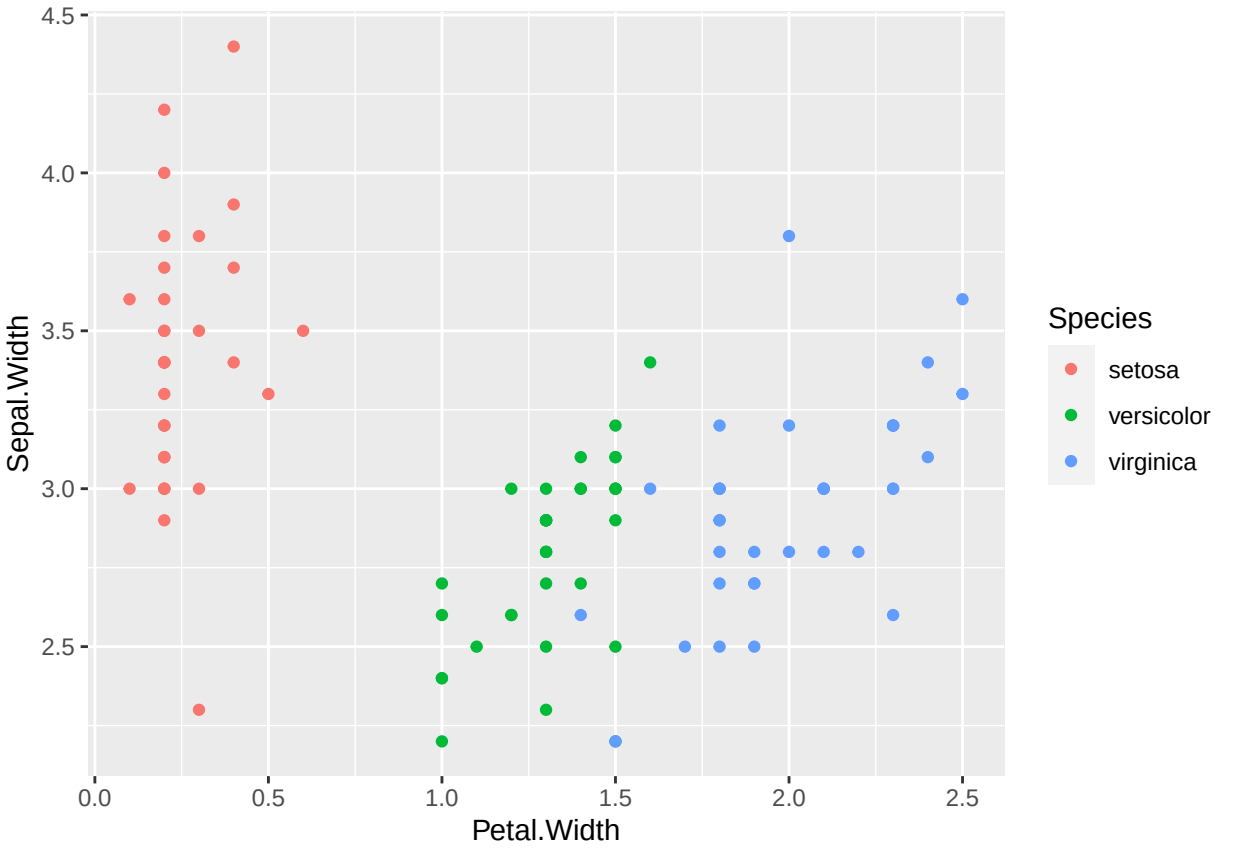
```
inTrain <- createDataPartition(y=iris$Species,
                               p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training); dim(testing)
```

```
## [1] 105   5
```

```
## [1] 45  5
```

hagaamos una grafica

```
qplot(Petal.Width,Sepal.Width,colour=Species,data=training)
```



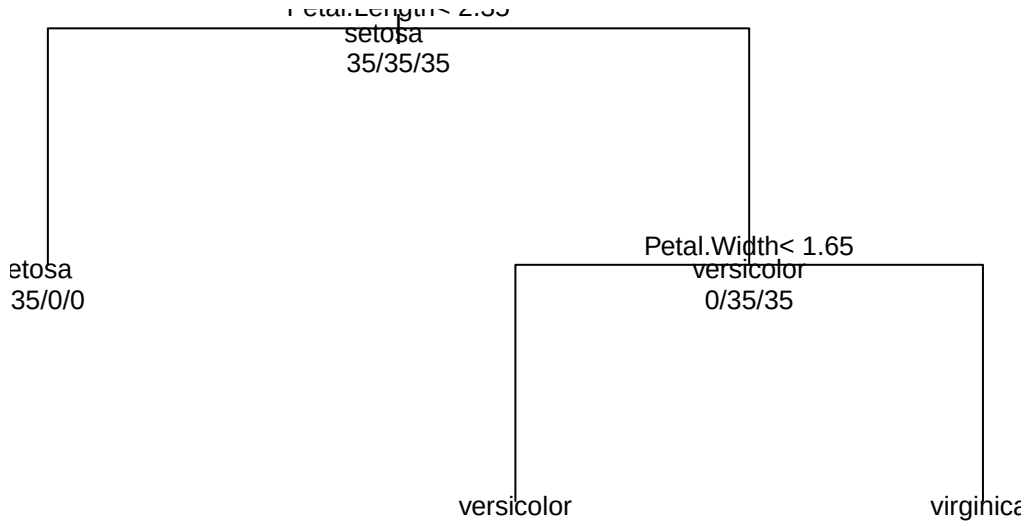
podemos ver claramento que podemos hacer diviciones en la variable petal width para predecir el tipo de especie, ahora hagamos un modelo para predecir especies

```
modFit <- train(Species ~ .,method="rpart",data=training)
print(modFit$finalModel)
```

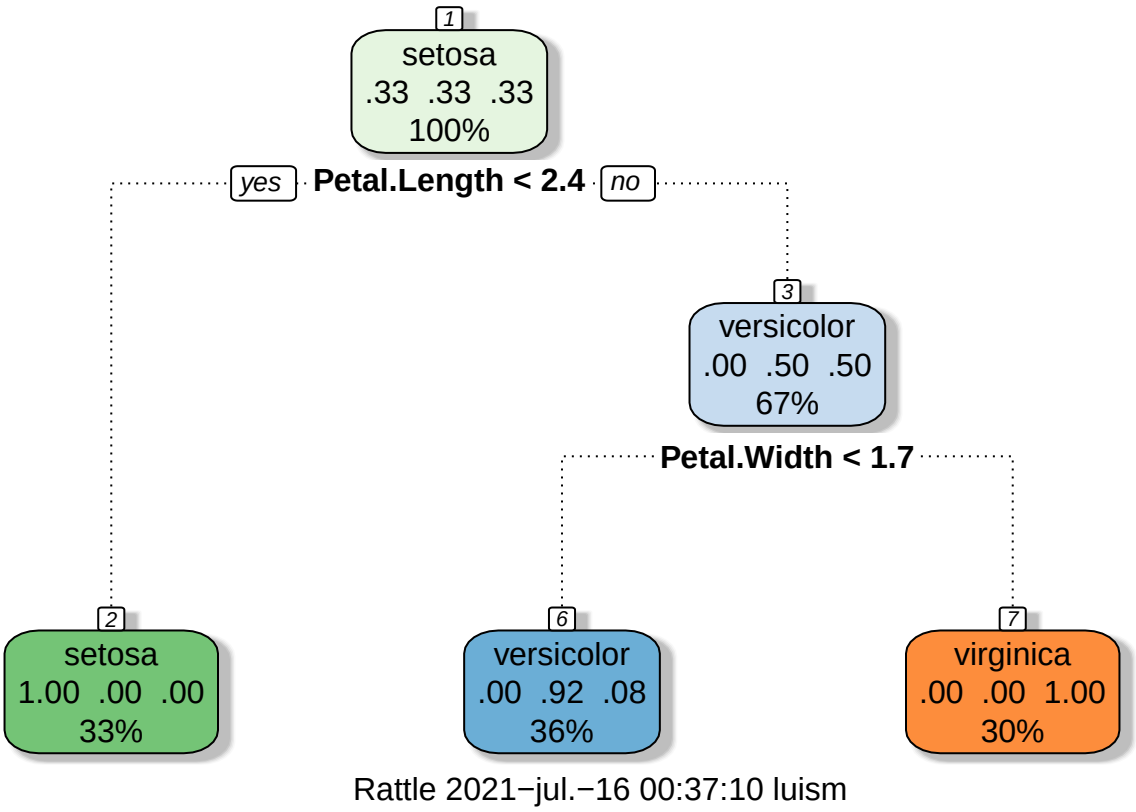
```
## n= 105
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 105 70 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.35 35  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.35 70 35 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.65 38  3 versicolor (0.00000000 0.92105263 0.07894737) *
##     7) Petal.Width>=1.65 32  0 virginica (0.00000000 0.00000000 1.00000000) *
```

no es muy entendible que digamos, por eso lo graficamos

```
plot(modFit$finalModel, uniform=TRUE,
      main="Classification Tree")
text(modFit$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```




```
library(rattle)
fancyRpartPlot(modFit$finalModel)
```



prediciendo nuevos valores

```
predict(modFit,newdata=testing)
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     setosa     versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor virginica  versicolor
## [25] versicolor virginica  versicolor versicolor versicolor versicolor
## [31] virginica  virginica  virginica  virginica  virginica  virginica
## [37] versicolor virginica  virginica  virginica  virginica  virginica
## [43] virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Notas y recursos adicionales

- Los árboles de clasificación son modelos no lineales
  - Usan interacciones entre variables
  - Las transformaciones de datos pueden ser menos importantes (transformaciones monótonas)
  - Los árboles también se pueden utilizar para problemas de regresión (resultado continuo)
- Tenga en cuenta que hay varias opciones de construcción de árboles en R tanto en el paquete de intercalación: party, rpart y fuera del paquete caret - tree
- Introducción al aprendizaje estadístico
- Elementos de aprendizaje estadístico
- Árboles de clasificación y regresión # bagging

Idea básica:

1. Volver a muestrear casos y volver a calcular las predicciones
2. Voto medio o mayoritario

Notas:

- Sesgo similar
- Varianza reducida
- Más útil para funciones no lineales

ejemplo con datos ozono

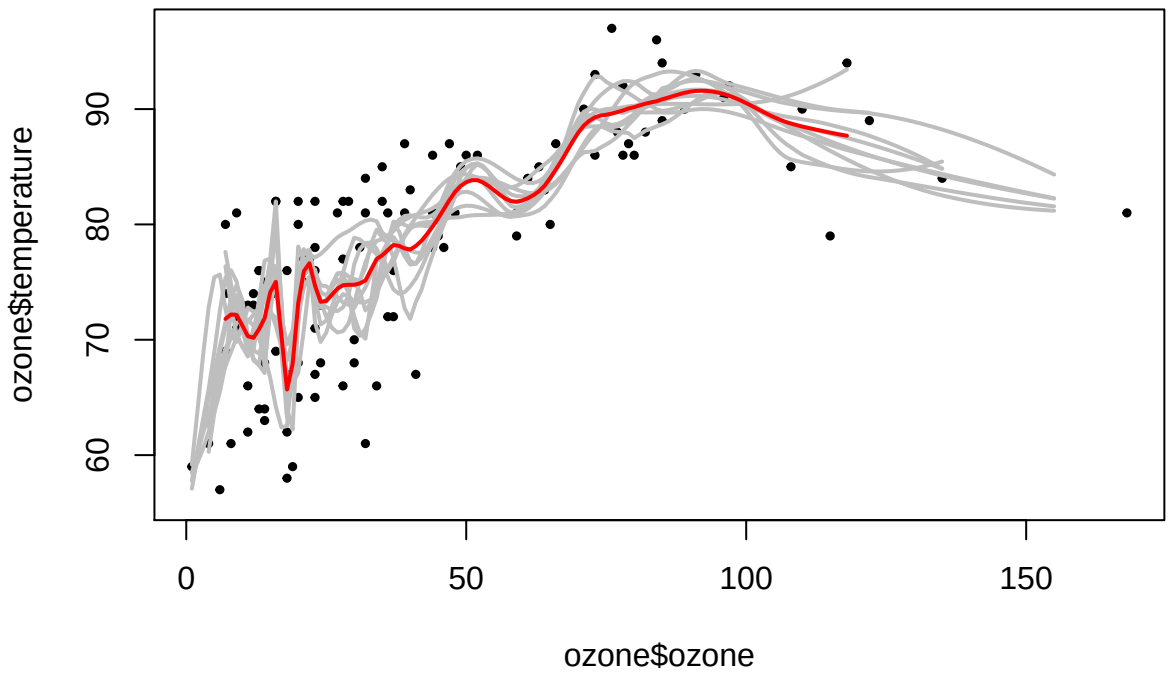
```
data(airquality)
ozone<-airquality
names(ozone)<-c("ozone","Solar.R", "Wind", "temperature", "Month","Day")
head(ozone)
```

```
##   ozone Solar.R Wind temperature Month Day
## 1    41     190  7.4          67     5   1
## 2    36     118  8.0          72     5   2
## 3    12     149 12.6          74     5   3
## 4    18     313 11.5          62     5   4
## 5    NA      NA 14.3          56     5   5
## 6    28      NA 14.9          66     5   6
```

```
ll <- matrix(NA,nrow=10,ncol=155)
for(i in 1:10){
  ss <- sample(1:dim(ozone)[1],replace=T)
  ozone0 <- ozone[ss,]; ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess(temperature ~ ozone,data=ozone0,span=0.2)
  ll[i,] <- predict(loess0,newdata=data.frame(ozone=1:155))
}
```

graficando, la roja es la "media"

```
plot(ozone$ozone,ozone$temperature,pch=19,cex=0.5)
for(i in 1:10){lines(1:155,ll[i,],col="grey",lwd=2)}
lines(1:155,apply(ll,2,mean),col="red",lwd=2)
```

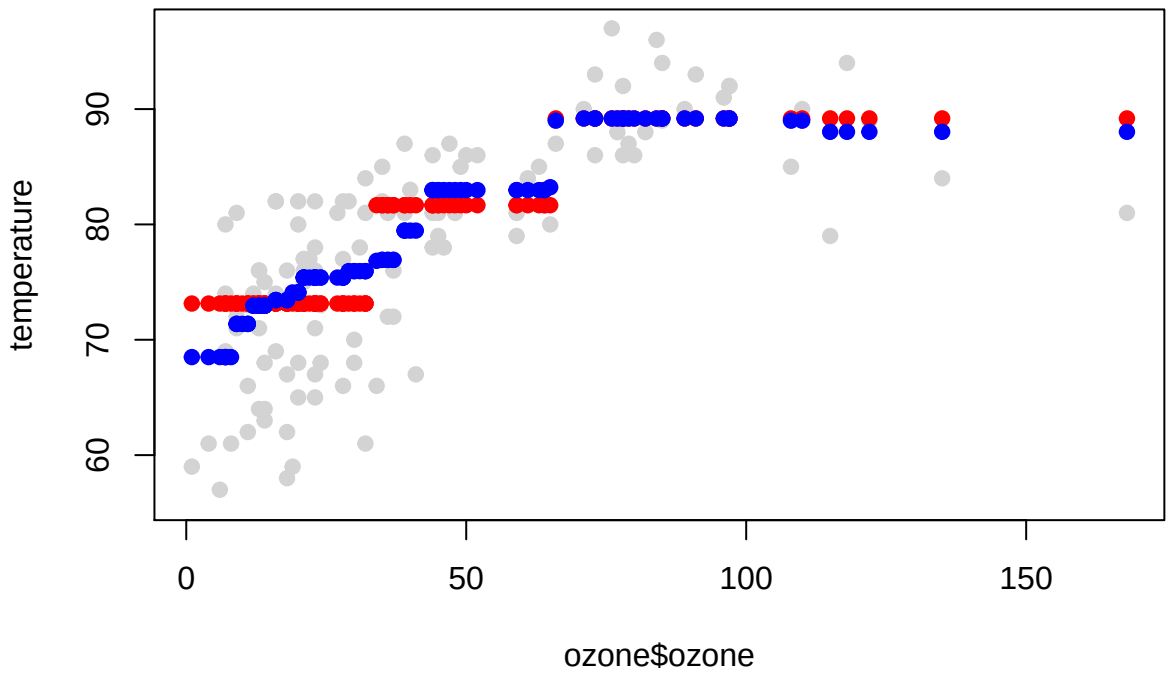


con caret

- Algunos modelos realizan el baggin por usted, en la función `train` considere las opciones de `method`
  - `bagEarth`
  - `treebag`
  - `bagFDA`
- Alternativamente, puede hacer baggin en cualquier modelo que elija utilizando la función `bag`

```
predictors = data.frame(ozone=ozone$ozone)
temperature = ozone$temperature
treebag <- bag(predictors, temperature, B = 10,
               bagControl = bagControl(fit = ctreeBag$fit,
                                       predict = ctreeBag$pred,
                                       aggregate = ctreeBag$aggregate))

plot(ozone$ozone,temperature,col='lightgrey',pch=19)
points(ozone$ozone,predict(treebag$fits[[1]]$fit,predictors),pch=19,col="red")
points(ozone$ozone,predict(treebag,predictors),pch=19,col="blue")
```



partes de baggin con caret

```
ctreeBag$fit
```

```
## function (x, y, ...)
## {
##     loadNamespace("party")
##     data <- as.data.frame(x, stringsAsFactors = TRUE)
##     data$y <- y
##     party::ctree(y ~ ., data = data)
## }
## <bytecode: 0x0000000027565b88>
## <environment: namespace:caret>
```

```
ctreeBag$pred
```

```
## function (object, x)
## {
##     if (!is.data.frame(x))
##         x <- as.data.frame(x, stringsAsFactors = TRUE)
##     obsLevels <- levels(object@data@get("response")[, 1])
##     if (!is.null(obsLevels)) {
##         rawProbs <- party::treeresponse(object, x)
##         probMatrix <- matrix(unlist(rawProbs), ncol = length(obsLevels),
##             byrow = TRUE)
##         out <- data.frame(probMatrix)
##         colnames(out) <- obsLevels
##         rownames(out) <- NULL
```

```
##     }
##     else out <- unlist(party::treeresponse(object, x))
##     out
## }
## <bytecode: 0x00000000275665d0>
## <environment: namespace:caret>
```

```
ctreeBag$aggregate
```

```
## function (x, type = "class")
## {
##     if (is.matrix(x[[1]]) | is.data.frame(x[[1]])) {
##         pooled <- x[[1]] & NA
##         classes <- colnames(pooled)
##         for (i in 1:ncol(pooled)) {
##             tmp <- lapply(x, function(y, col) y[, col], col = i)
##             tmp <- do.call("rbind", tmp)
##             pooled[, i] <- apply(tmp, 2, median)
##         }
##         if (type == "class") {
##             out <- factor(classes[apply(pooled, 1, which.max)],
##                 levels = classes)
##         }
##         else out <- as.data.frame(pooled, stringsAsFactors = TRUE)
##     }
##     else {
##         x <- matrix(unlist(x), ncol = length(x))
##         out <- apply(x, 1, median)
##     }
##     out
## }
## <bytecode: 0x0000000027564560>
## <environment: namespace:caret>
```

Notas y otros recursos

Notas:

- El bagging es más útil para modelos no lineales
- Se usa a menudo con árboles: una extensión son bosques aleatorios
- Varios modelos utilizan bagging en la función *train* de caret

Otros recursos:

- Embolsado
- Embolsado y boosting
- Elementos de aprendizaje estadístico

Random forests

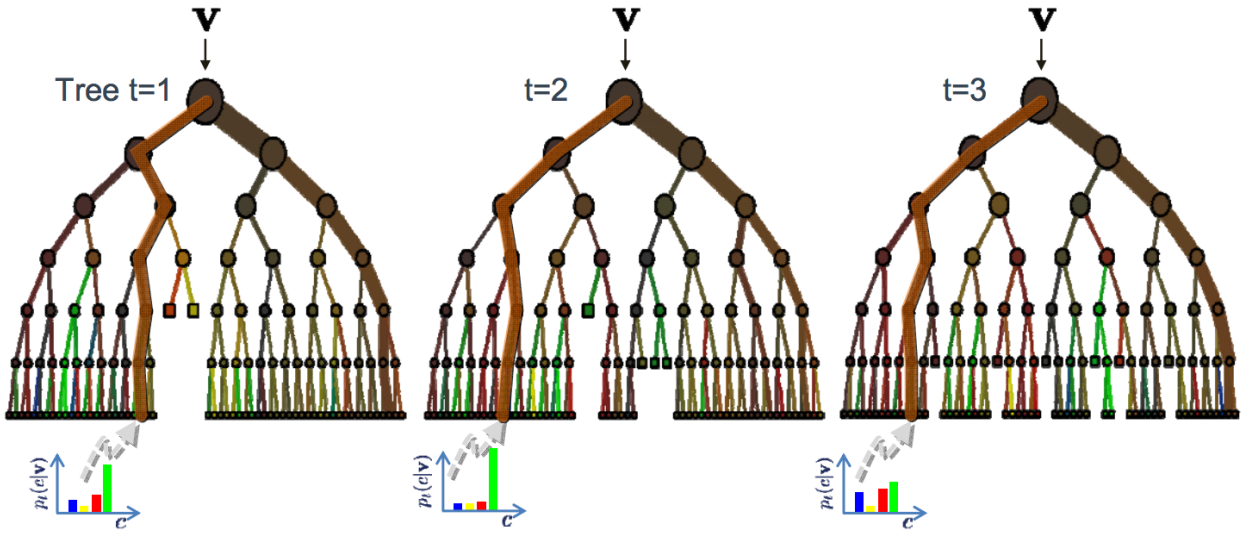
1. Muestras de Bootstrap
2. En cada división, las variables de arranque
3. Cultiva varios árboles y vota

Pros:

1. Precisión

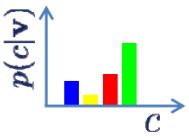
Contras:

1. Velocidad
2. Interpretabilidad
3. Sobreajuste



The ensemble model

Forest output probability $p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$



ejemplo con datos iris

```
data(iris); library(ggplot2);library(caret)
inTrain <- createDataPartition(y=iris$Species,
                               p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]


modFit <- train(Species~ .,data=training,method="rf",prox=TRUE)
modFit
```

```
## Random Forest
##
## 105 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
```

```
##   2     0.9423389  0.9118121
##   3     0.9404694  0.9090803
##   4     0.9377160  0.9050166
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

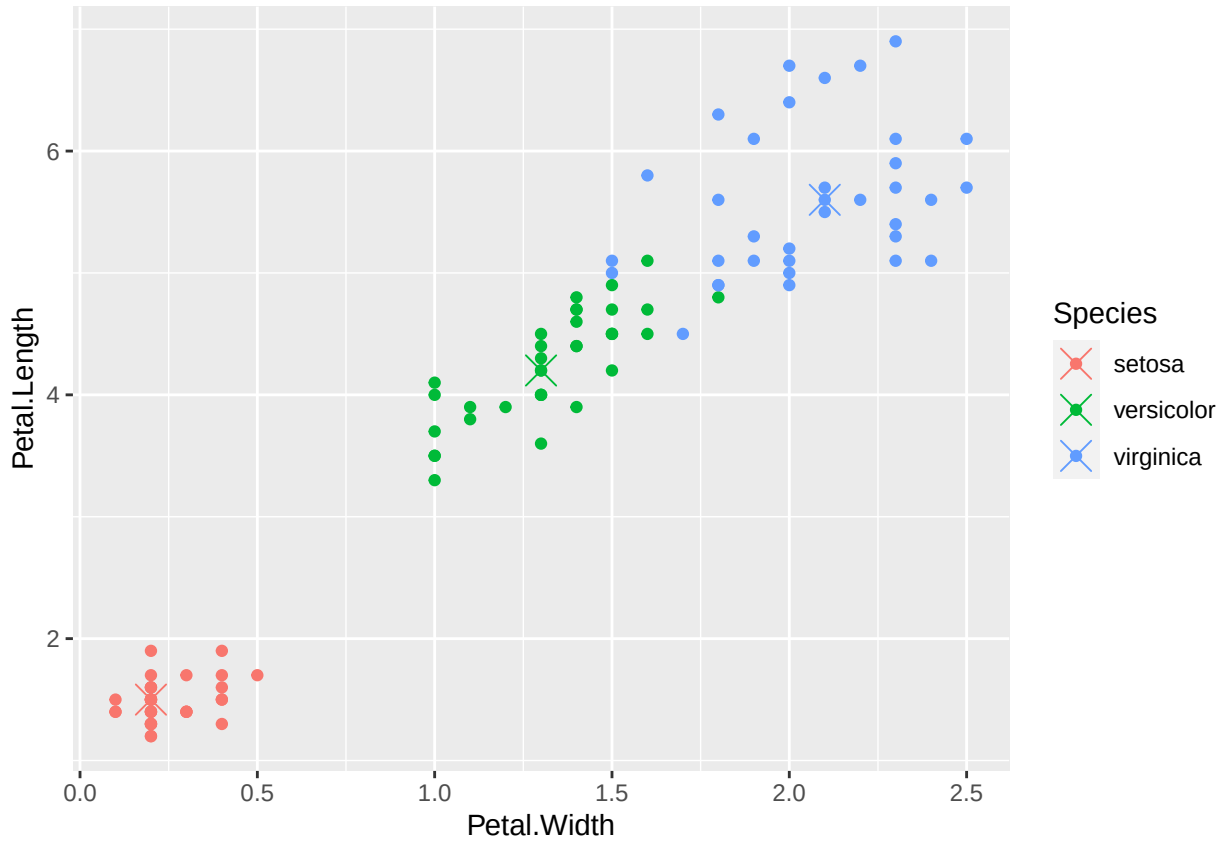
Conseguir un solo árbol

```
library(randomForest)
getTree(modFit$finalModel,k=2)
```

```
##    left daughter right daughter split var split point status prediction
## 1              2              3         3        2.50      1          0
## 2              0              0         0        0.00     -1          1
## 3              4              5         3        4.85      1          0
## 4              6              7         4        1.65      1          0
## 5              8              9         4        1.70      1          0
## 6              0              0         0        0.00     -1          2
## 7              0              0         0        0.00     -1          3
## 8             10             11         4        1.55      1          0
## 9              0              0         0        0.00     -1          3
## 10            12             13         3        4.95      1          0
## 11             0              0         0        0.00     -1          2
## 12             0              0         0        0.00     -1          2
## 13             0              0         0        0.00     -1          3
```

Creando prototipos (son los taches), Los prototipos son casos "representativos" de un grupo de puntos de datos, dada la matriz de similitud entre los puntos. Son muy similares a los medoides.

```
irisP <- classCenter(training[,c(3,4)], training$Species, modFit$finalModel$prox)
irisP <- as.data.frame(irisP); irisP$Species <- rownames(irisP)
p <- qplot(Petal.Width, Petal.Length, col=Species,data=training)
p + geom_point(aes(x=Petal.Width,y=Petal.Length,col=Species),size=5,shape=4,data=irisP)
```

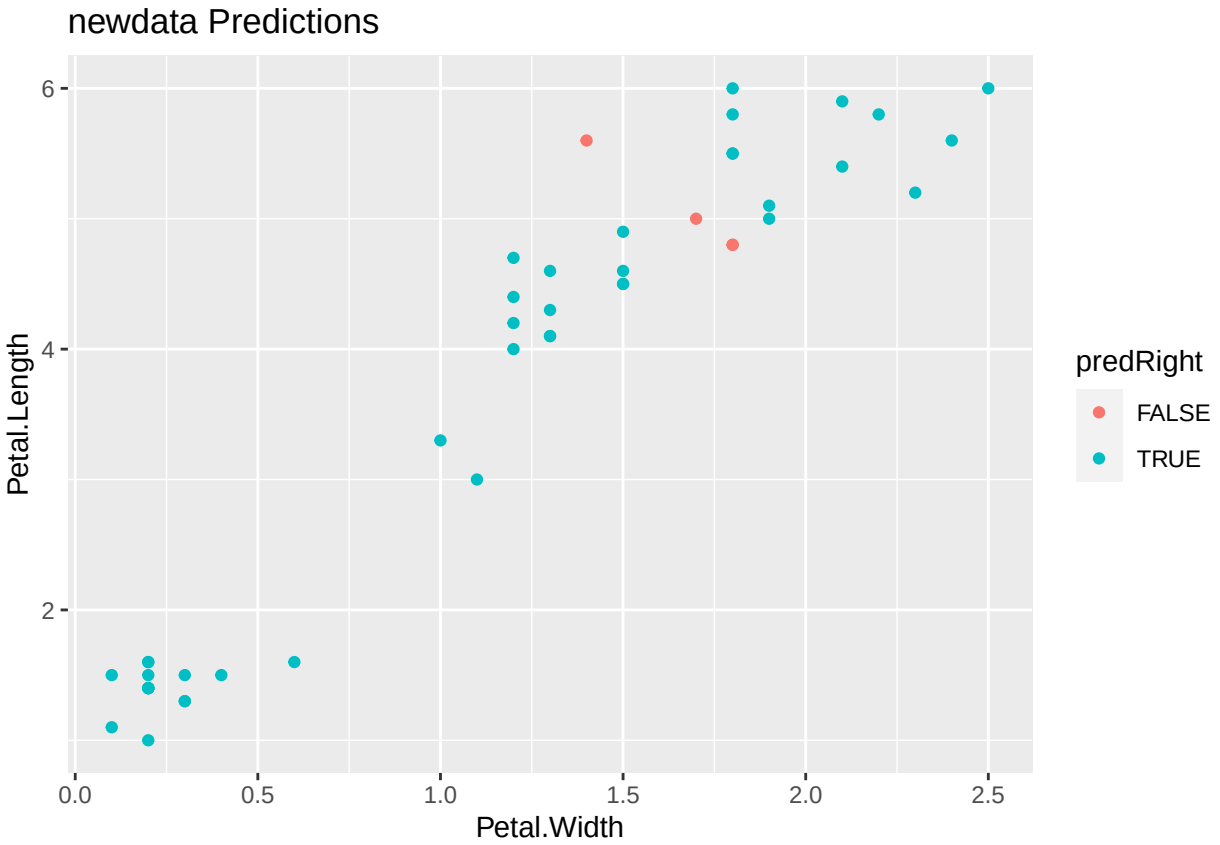


prediciendo nuevos valores

```
pred <- predict(modFit,testing); testing$predRight <- pred==testing$Species
table(pred,testing$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         14         3
##   virginica       0          1        12
```

```
qplot(Petal.Width,Petal.Length,colour=predRight,data=testing,main="newdata Predictions")
```



Notas y otros recursos

Notas:

- Los bosques aleatorios suelen ser uno de los dos algoritmos de mejor rendimiento junto con el boosting en los concursos de predicción.
- Los bosques aleatorios son difíciles de interpretar pero a menudo muy precisos.
- Se debe tener cuidado para evitar el sobreajuste (consulte la función rfcv)

Otros recursos:

- Bosques aleatorios
- Wikipedia del bosque aleatorio
- Elementos de aprendizaje estadístico

boosting

Idea básica

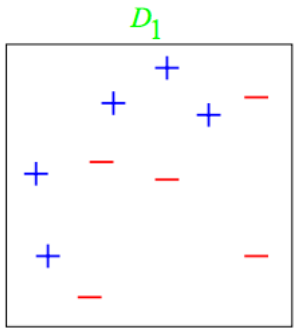
1. Tome muchos predictores (posiblemente) débiles
2. Péselos y súmelos
3. Obtenga un predictor más fuerte

Idea básica detrás del iboosting

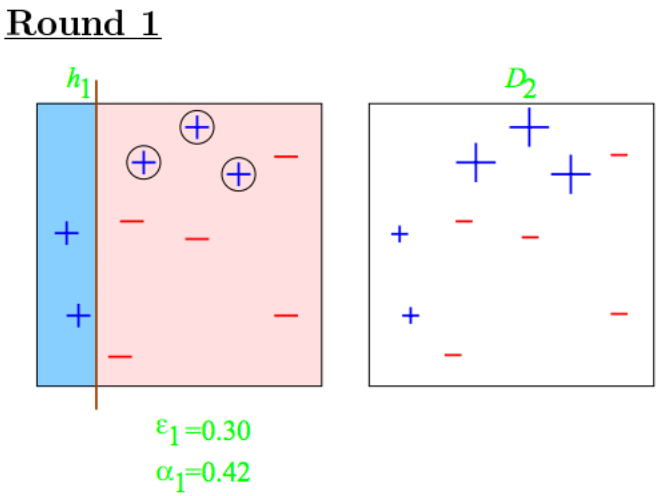
1. Comience con un conjunto de clasificadores $h_1, \ldots, h_k$
    - Ejemplos: todos los árboles posibles, todos los modelos de regresión posibles, todos los cortes posibles.

2. Cree un clasificador que combine funciones de clasificación: $f(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$..
   - El objetivo es minimizar el error (en el conjunto de entrenamiento)
   - Iterativo, seleccione un h en cada paso
   - Calcular pesos basados en errores
   - Subir clasificaciones perdidas y seleccionar los siguientes h

ejemplo simple, se trada de crear un clasificador que clasifique entre - y +

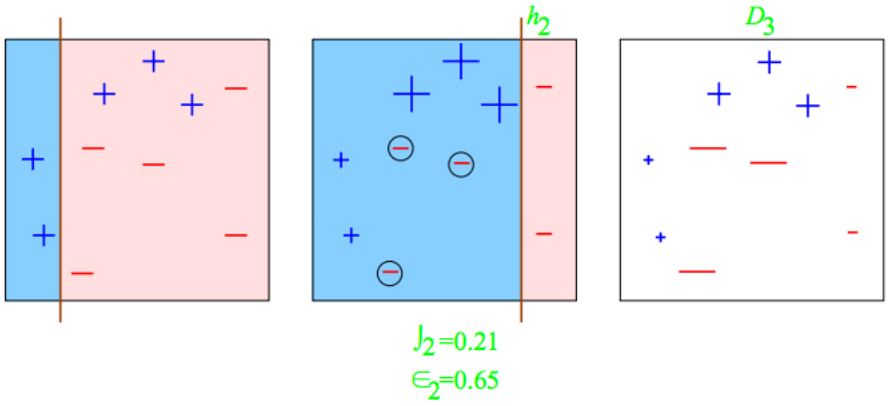


el primer clasificador es una linea vertical

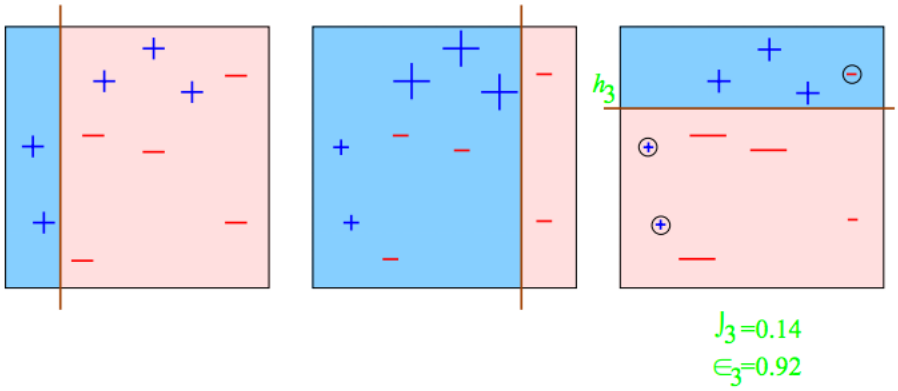


para el segundo y tercer clasificador se crea una linea vertical y horizontal

Round 2

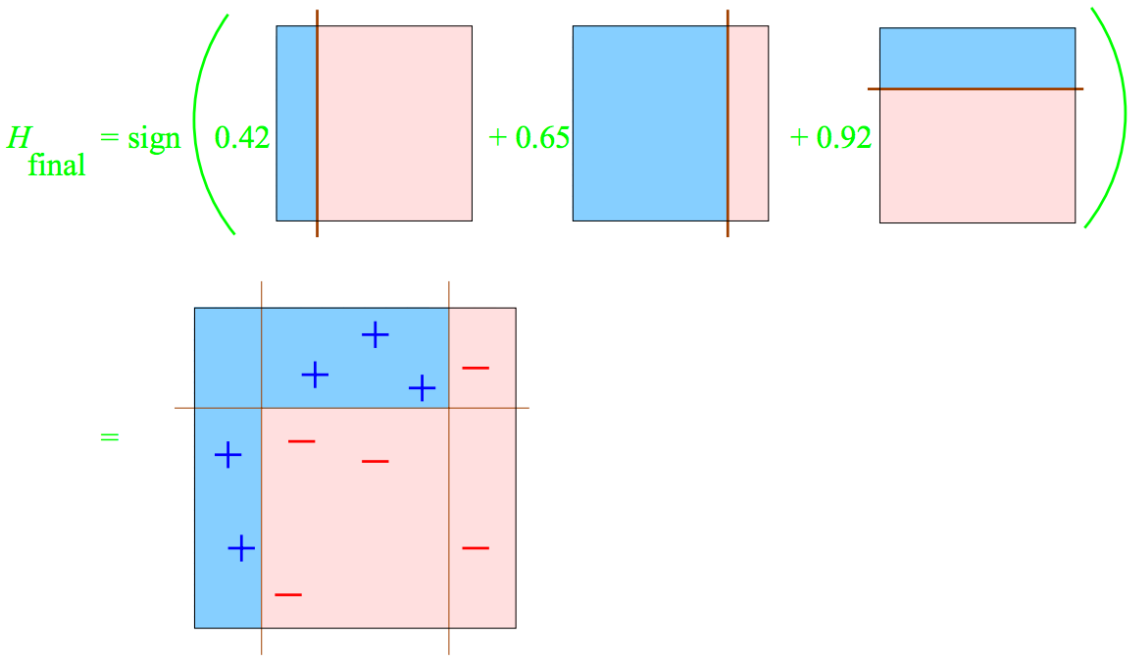


Round 3



se combinan los 3 predictores

Final Hypothesis



boosting en R

- El boosting se puede usar con cualquier subconjunto de clasificadores
- Una gran subclase es aumento de gradiente
- R tiene múltiples bibliotecas de boosting. Las diferencias incluyen la elección de funciones básicas de clasificación y reglas de combinación.
  - gbm: boosting con árboles.
  - mboost: boosting basado en modelos
  - ada - boosting estadístico basado en regresión logística aditiva
  - gamBoost para impulsar modelos aditivos generalizados
- La mayoría de estos están disponibles en el paquete de caret.

ejemplo con datos de sueldo

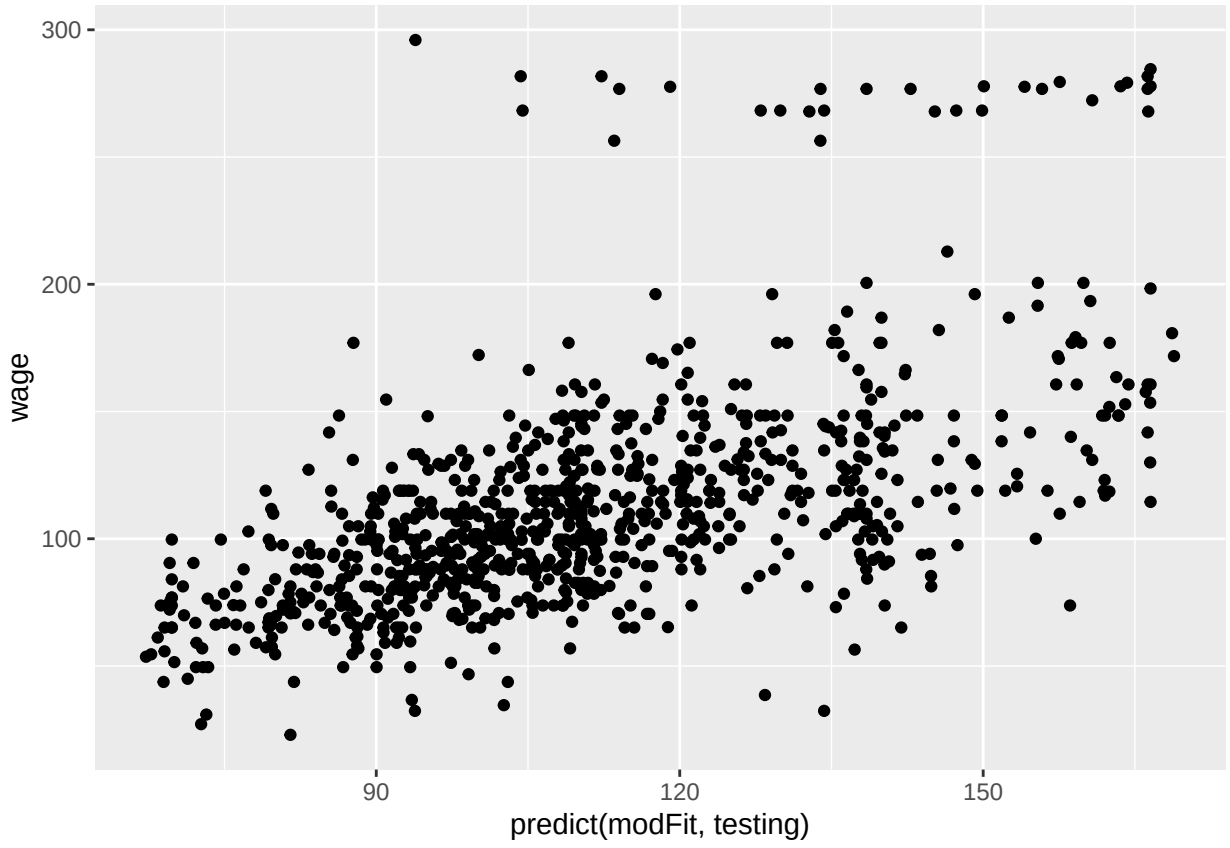
```
library(ISLR); data(Wage); library(ggplot2); library(caret);
Wage <- subset(Wage,select=-c(logwage))
inTrain <- createDataPartition(y=Wage$wage,
                               p=0.7, list=FALSE)
training <- Wage[inTrain,]; testing <- Wage[-inTrain,]

modFit <- train(wage ~ ., method="gbm",data=training,verbose=FALSE)
print(modFit)
```

```
## Stochastic Gradient Boosting
##
## 2102 samples
```

```
##    9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  RMSE      Rsquared   MAE
##   1                   50      34.02735  0.3164369  23.17244
##   1                  100      33.48048  0.3268483  22.80521
##   1                  150      33.41638  0.3283001  22.81330
##   2                   50      33.53075  0.3260401  22.81065
##   2                  100      33.43372  0.3280820  22.86149
##   2                  150      33.49621  0.3264654  22.95321
##   3                   50      33.39058  0.3299187  22.74033
##   3                  100      33.56036  0.3242165  22.96235
##   3                  150      33.77585  0.3179243  23.15347
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
qplot(predict(modFit,testing),wage,data=testing)
```



Notas y lectura adicional

- Un par de buenos tutoriales para impulsar
  - Freund y Shapire - http://www.cc.gatech.edu/~thad/6601-gradAI-fall2013/boosting.pdf
  - Ron Meir- http://webee.technion.ac.il/people/rmeir/BoostingTutorial.pdf
- boosting, bosques aleatorios y conjuntos de modelos son las herramientas más comunes que ganan Kaggle y otros concursos de predicción.
  - http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf
  - https://kaggle2.blob.core.windows.net/wiki-files/327/09ccf652-8c1c-4a3d-b979-ce2369c985e4/Willem%20Mestrom%20-%20Milestone%201%20Description%20V2%202.pdf *Adaboost on Wikipedia

prediccion basada en modelos

Idea básica

1. Suponga que los datos siguen un modelo probabilístico
2. Utilice el teorema de Bayes para identificar clasificadores óptimos

Pros:

- Puede aprovechar la estructura de los datos.
- Puede ser conveniente computacionalmente
- Son razonablemente precisos en problemas reales.

Contras:

- Haga suposiciones adicionales sobre los datos
- Cuando el modelo es incorrecto, es posible que se reduzca la precisión

1. Nuestro objetivo es construir un modelo paramétrico para distribución condicional $P(Y = k|X = x)$
2. Un enfoque típico es aplicar el teorema de Bayes:

$$Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k)Pr(Y = k)}{\sum_{\ell=1}^{K} Pr(X = x|Y = \ell)Pr(Y = \ell)}$$

$$Pr(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}$$

3. Normalmente, las probabilidades priori π_k se establecen de antemano.
4. Una elección común para $f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{(x-\mu_k)^2}{\sigma_k^2}}$, la distribucion normal
5. Estimar los parametros de (μ_k,σ_k^2)
6. Clasifique en la clase con el valor más alto de $P(Y = k|X = x)$

Una variedad de modelos utilizan este enfoque

- El análisis discriminante lineal supone que $f_k(x)$ es gaussiano multivariante con las mismas covarianzas
- El análisis discriminatorio cuadrático asume que $f_k(x)$ es gaussiano multivariante con diferentes covarianzas
- Predicción basada en modelos asume versiones más complicadas para la matriz de covarianza
- Naive Bayes asume la independencia entre características para la construcción de modelos

http://statweb.stanford.edu/~tibs/ElemStatLearn/

Análisis de discriminante lineal

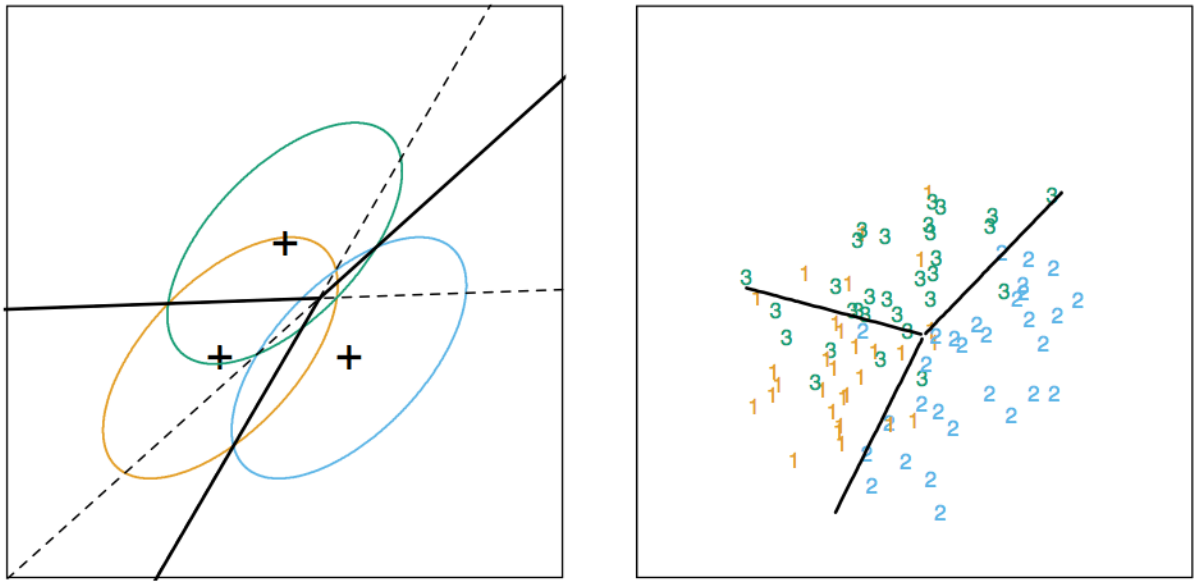
Análisis Discriminante Lineal (ADL, o LDA por sus siglas en inglés) es una generalización del discriminante lineal de Fisher, un método utilizado en estadística, reconocimiento de patrones y aprendizaje de máquinas para encontrar una combinación lineal de rasgos que caracterizan o separan dos o más clases de objetos o eventos. La combinación resultante puede ser utilizada como un clasificador lineal, o, más comúnmente, para la reducción de dimensiones antes de la posterior clasificación.

LDA está estrechamente relacionado con el análisis de varianza (ANOVA) y el análisis de regresión, el cual también intenta expresar una variable dependiente como la combinación lineal de otras características o medidas. Sin embargo, ANOVA usa variables independientes categóricas y una variable dependiente continua, mientras que el análisis discriminante tiene variables independientes continuas y una variable dependiente categórica (o sea, la etiqueta de clase).

¿por qué discriminante lineal?

$$
\begin{aligned}
& log\frac{Pr(Y=k|X=x)}{Pr(Y=j|X=x)} \\
& = log\frac{f_k(x)}{f_j(x)} + log\frac{\pi_k}{\pi_j} \\
& = log\frac{\pi_k}{\pi_j} - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k - \mu_j^T\Sigma^{-1}\mu_j) \\
& + x^T\Sigma^{-1}(\mu_k - \mu_j)
\end{aligned}
$$

asi funciona



funcion discriminante

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k + log(\mu_k)$$

- Decidir la clase en función de $\hat{Y}(x) = argmax_k\delta_k(x)$
- Solemos estimar los parámetros con la máxima verosimilitud

Nive bayes

En términos simples, un clasificador de Naive Bayes asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra característica, dada la clase variable. Por ejemplo, una fruta puede ser considerada como una manzana si es roja, redonda y de alrededor de 7 cm de diámetro. Un clasificador de Naive Bayes considera que cada una de estas características contribuye de manera independiente a la probabilidad de que esta fruta sea una manzana, independientemente de la presencia o ausencia de las otras características.

Supongamos que tenemos muchos predictores, querríamos modelar: $P(Y = k|X_1, \ldots, X_m)$

Podríamos usar el Teorema de Bayes para obtener:

$$P(Y = k|X_1, \ldots, X_m) = \frac{\pi_k P(X_1, \ldots, X_m|Y = k)}{\sum_{\ell=1}^{K} P(X_1, \ldots, X_m|Y = k)\pi_\ell}$$
$$\propto \pi_k P(X_1, \ldots, X_m|Y = k)$$

esto puede ser escrito

$$P(X_1, \ldots, X_m, Y = k) = \pi_k P(X_1|Y = k)P(X_2, \ldots, X_m|X_1, Y = k)$$
$$= \pi_k P(X_1|Y = k)P(X_2|X_1, Y = k)P(X_3, \ldots, X_m|X_1, X_2, Y = k)$$
$$= \pi_k P(X_1|Y = k)P(X_2|X_1, Y = k) \ldots P(X_m|X_1 \ldots, X_{m-1}, Y = k)$$

Podríamos hacer una suposición para escribir esto:

$$\approx \pi_k P(X_1|Y = k)P(X_2|Y = k) \ldots P(X_m|, Y = k)$$

ejemplo con datos iris

```
data(iris); library(ggplot2)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
inTrain <- createDataPartition(y=iris$Species,
                               p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training); dim(testing)
```

```
## [1] 105   5
```

```
## [1] 45  5
```

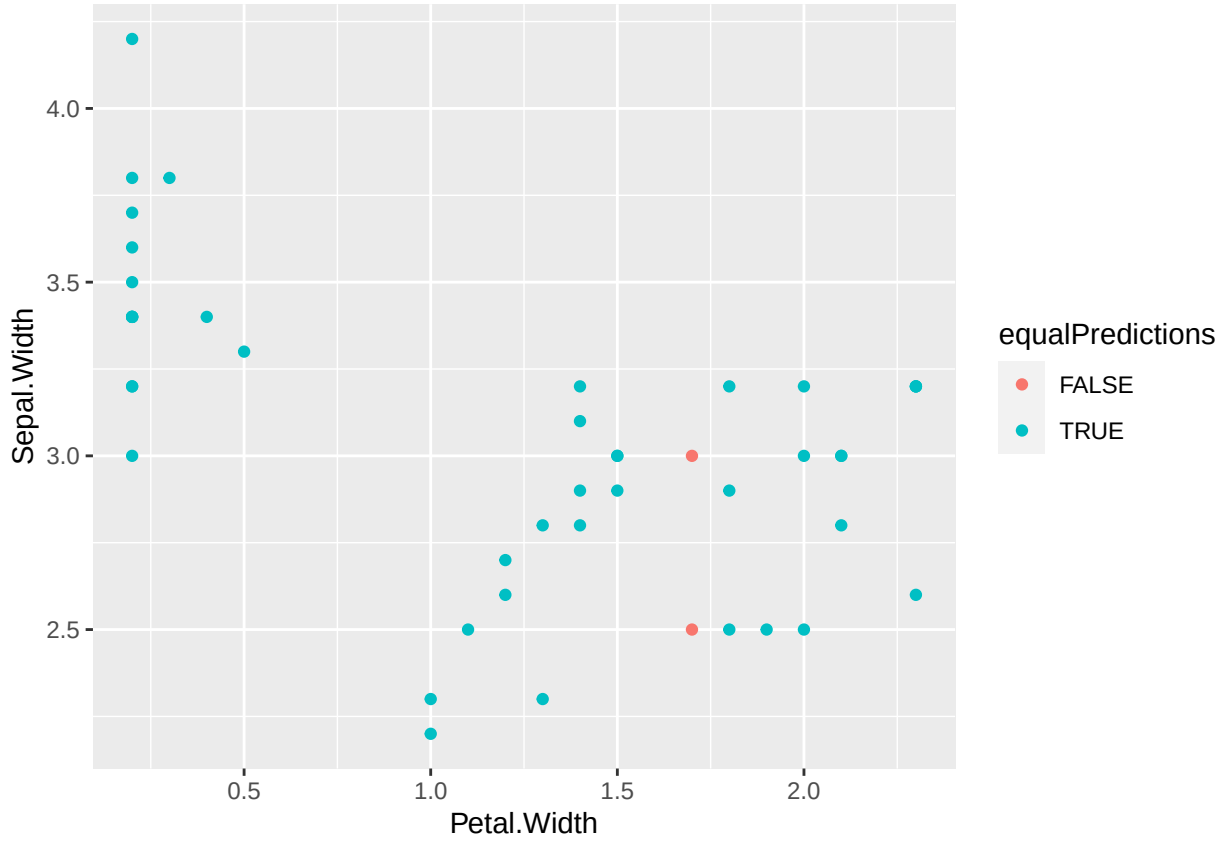
construyendo los predictores

```
#analisis de discriminante lineal
modlda = train(Species ~ .,data=training,method="lda")
#nive bayes
modnb = train(Species ~ ., data=training,method="nb")
plda = predict(modlda,testing); pnb = predict(modnb,testing)
table(plda,pnb)
```

```
##              pnb
## plda          setosa versicolor virginica
##   setosa          15          0         0
##   versicolor       0         14         1
##   virginica        0          1        14
```

comparacion de resultados

```
equalPredictions = (plda==pnb)
qplot(Petal.Width,Sepal.Width,colour=equalPredictions,data=testing)
```



notas y futuras lecturas

- Introduction to statistical learning
- Elements of Statistical Learning
- Model based clustering
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis