Clasificación IV - Software R

Seminario Permanente de Formación en Inteligencia Artificial Aplicada a la Defensa

Francisco Charte Ojeda - fcharte.com Universidad de Jaén - DaSCI

19 abril 2020

Contents

Introducción

Este es un *notebook* R creado con RStudio, mediante la opción **File>New File>R Notebook**. Se trata de un tipo de documento que permite combinar texto y formato con código en lenguaje R y los resultados que produce dicho código.

Aunque en el presente *notebook* se recurrirá a representaciones a gráficas para llevar a cabo un análisis exploratorio de los datos cuando sea preciso, los detalles sobre cómo confeccionar gráficas y obtener indicadores estadísticos de los datos **se facilitan en un documento separado**. En este se describen únicamente los mecanismos para construir, evaluar y usar diferentes tipos de clasificadores.

Paquetes R para clasificación

En CRAN existe un *Task View* específico sobre aprendizaje automático con un centenar de paquetes que implementan una gran parte de los algoritmos de clasificación propuestos en la literatura, desde los más clásicos como los árboles de decisión o kNN a métodos basados *ensembles* y aprendizaje profundo.

Por regla general, todos los paquetes ofrecen un método `train()`, encargado de ejecutar el algoritmo de aprendizaje sobre las muestras facilitadas y devolver el correspondiente modelo, y un método `test()` que, a partir de dicho modelo, predice la clase para nuevas instancias.

Clasificación de conjuntos de datos muy simples

Supongamos que nuestros datos tienen una estructura muy sencilla, contando con un par de variables predictoras, además del atributo que actúa como etiqueta de clase, y que se trata de un problema de clasificación binario. Podemos construir este conjunto de datos a partir de `iris`, quedándonos con dos variables y solo dos de las tres clases de flor:

```
data(iris)
set.seed(73)

# Tomamos tres de los cinco atributos y las muestras que no sean de la familia `setosa`
datos <- iris[iris$Species != "setosa", c("Petal.Length", "Petal.Width", "Species")]

# Convertimos el atributo de clase en un número
datos$Species <- as.numeric(datos$Species) - 1

datos[sample(nrow(datos), 10), ] # Vistazo a 10 muestras tomadas aleatoriamente
```

```
##     Petal.Length Petal.Width Species
## 111          5.1         2.0       2
## 121          5.7         2.3       2
## 138          5.5         1.8       2
## 67           4.5         1.5       1
## 97           4.2         1.3       1
## 57           4.7         1.6       1
## 69           4.5         1.5       1
## 134          5.1         1.5       2
## 81           3.8         1.1       1
## 77           4.8         1.4       1
```

Aunque tenemos muy pocos datos, solo 100 muestras para entrenar y evaluar el clasificador, vamos a usar el enfoque de particionamiento *hold-out* por simplicidad:

```
set.seed(7)
test.ind <- sample(nrow(datos), size = nrow(datos) * 0.25)
```

```
train <- datos[-test.ind, ]
test <- datos[test.ind, ]

str(train)
```

```
## 'data.frame':    75 obs. of  3 variables:
##  $ Petal.Length: num  4.7 4.5 4.9 4 4.6 4.7 4.6 3.9 3.5 4 ...
##  $ Petal.Width : num  1.4 1.5 1.5 1.3 1.5 1.6 1.3 1.4 1 1 ...
##  $ Species     : num  1 1 1 1 1 1 1 1 1 1 ...
```

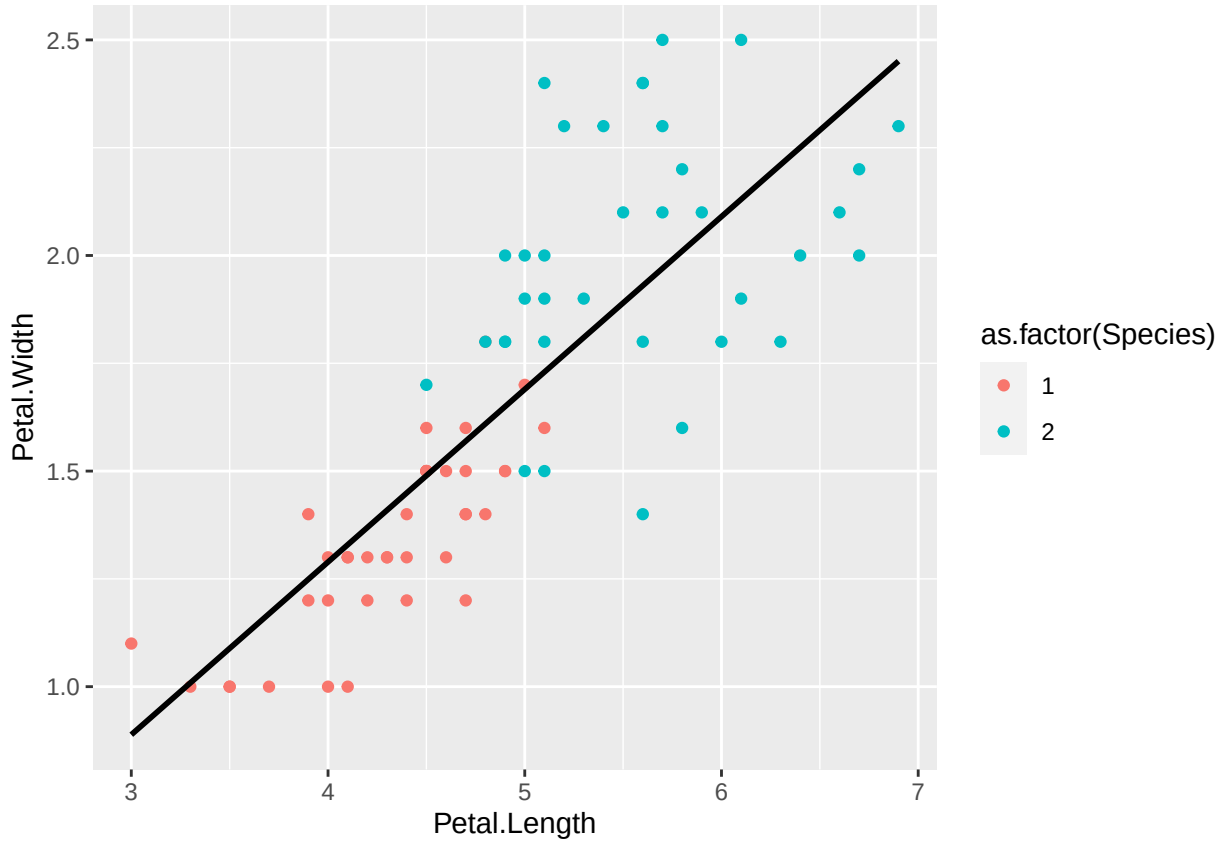
```
str(test)
```

```
## 'data.frame':    25 obs. of  3 variables:
##  $ Petal.Length: num  4.6 5.6 3.8 5.1 5.3 3.6 5.4 3.3 5.5 5.5 ...
##  $ Petal.Width : num  1.4 2.2 1.1 2.3 2.3 1.3 2.1 1 1.8 1.8 ...
##  $ Species     : num  1 2 1 2 2 1 2 1 2 2 ...
```

Por último, vamos a examinar la distribución de las muestras de test recurriendo a una gráfica de dispersión:

```
library(ggplot2)

ggplot(train, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(color = as.factor(Species))) +
  stat_smooth(method = lm, color= "black", se = FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Como puede apreciarse, las muestras pertenecientes a cada familia podrían separarse casi perfectamente

si usáramos como frontera una línea perpendicular a la mostrada y que la cruzase aproximadamennte por el punto central. Esto denota que un clasificador lineal simple, creado a partir de la línea regresión, sería suficiente.

Clasificación mediante regresión lineal

Vamos a ajustar un regresor lineal con el objeto de predecir la especie, que es un valor numérico en nuestro conjunto de datos, usando el resto de variables disponibles:

```
modelo <- lm(Species ~ ., data = train)
modelo
```

```
##
## Call:
## lm(formula = Species ~ ., data = train)
##
## Coefficients:
##  (Intercept)  Petal.Length   Petal.Width
##      -0.5752        0.1962        0.6570
```

El modelo resultando nos facilita los coeficientes de regresión para cada variable, así como el *intercept* o punto de corte de la recta con el eje Y o de ordenadas. Estos coeficientes son accesibles individualmente como `modelo$coefficients[N]`. Esto nos permite componer la fórmula de predicción y, con ella, construir nuestro clasificador:

```
prediccion <-
  modelo$coefficients[1] +
  modelo$coefficients[2] * test$Petal.Length +
  modelo$coefficients[3] * test$Petal.Width

test$Species
```

```
##  [1] 1 2 1 2 2 1 2 1 2 2 1 1 1 2 2 1 2 1 2 2 2 1 1 2 2
```

```
prediccion
```

```
##  [1] 1.2467774 1.9684889 0.8927716 1.9361088 1.9753389 0.9849317 1.8635637
##  [8] 0.7290012 1.6860933 1.6860933 1.0633919 1.0633919 1.1026220 1.6733283
## [15] 1.7449385 1.2340123 2.2440345 0.9123866 2.0930292 1.7586384 1.9027938
## [22] 1.1614672 1.2075473 1.5487880 2.1322593
```

```
round(prediccion)
```

```
##  [1] 1 2 1 2 2 1 2 1 2 2 1 1 1 2 2 1 2 1 2 2 2 1 1 2 2
```

Como puede apreciarse, la predicción es un valor real dado que estamos usando una fórmula de regresión. No obstante, podemos convertirla en una etiqueta de clase simplemente redondeando al entero más próximo. Esto sería lo mismo que decir que aplicamos un umbral de corte, de forma que los valores inferiores a `1.5` se consideran de la clase 1 y los mayores a ese valor de clase 2.

Comparando los valores predichos con los reales podemos calcular el ratio de acierto *Accuracy* que, como se aprecia a continuación, es perfecto sobre los datos de test. Esto se debe a la sencillez del conjunto de datos original.

```
accuracy <- sum(test$Species == round(prediccion)) / nrow(test) * 100
paste0(accuracy, "% de acierto")
```

```
## [1] "100% de acierto"
```

Clasificación mediante agrupamiento

Las técnicas de *clustering* son no supervisadas, es decir, no precisan de una etiqueta de clase para asignar cada muestra del conjunto de datos a un grupo u otro, guiándose exclusivamente por las distancias entre los puntos en el espacio n-dimensional generado por las variables que describen cada instancia.

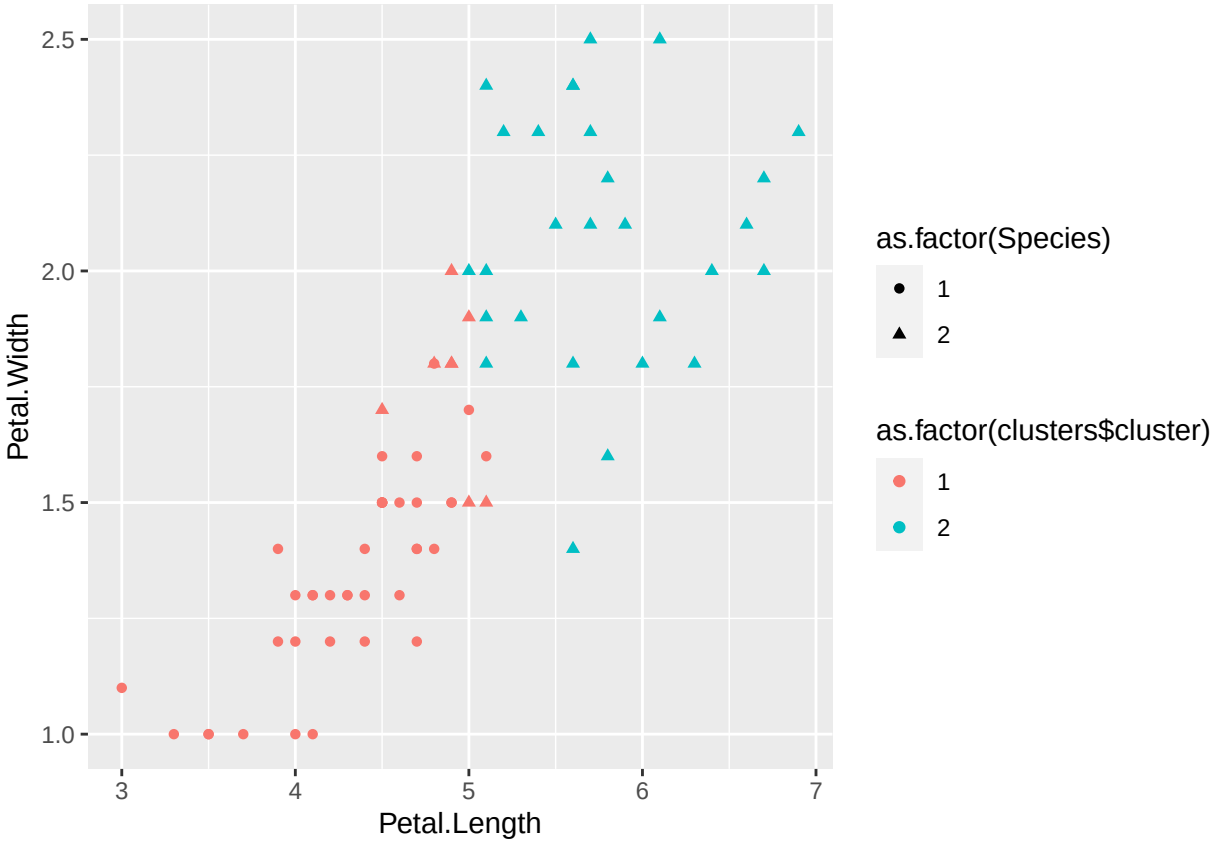
Por ello, usamos solo las dos variables que indican la longitud y anchura de pétal para ejecutar K-medias, el algoritnmo de agrupamiento más conocido y simple:

```
library(cluster)

set.seed(73)
clusters <- kmeans(train[,1:2], 2)

clusters
```

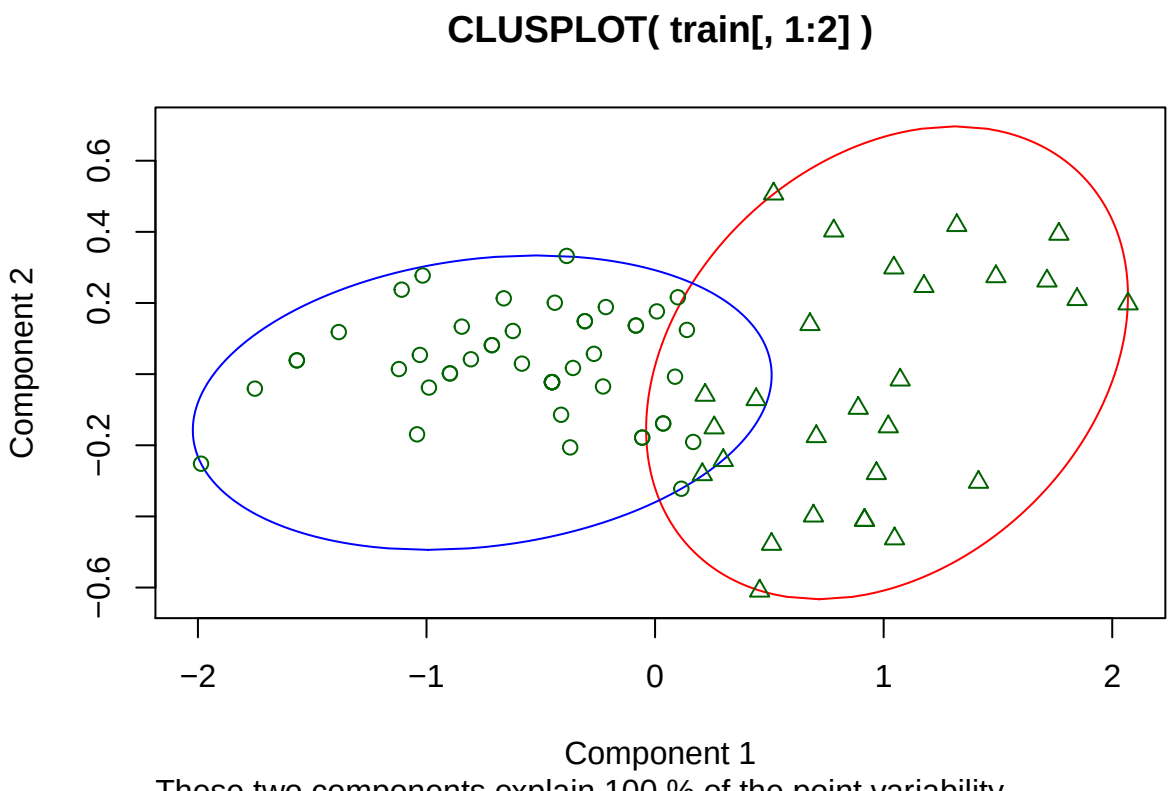
```
## K-means clustering with 2 clusters of sizes 47, 28
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     4.417021    1.410638
## 2     5.771429    2.075000
##
## Clustering vector:
##  51  52  53  54  55  57  59  60  61  63  64  67  68  69  71  73  74  75  76  77
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  78  79  80  82  83  84  85  86  87  88  89  91  93  94  95  96  98  99 100 102
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2
## 103 104 105 106 107 108 110 111 112 113 114 115 118 119 120 121 122 123 124 125
##   2   2   2   2   1   2   2   2   2   2   2   2   2   2   1   2   1   2   1   2
## 126 127 128 130 131 132 134 135 137 141 145 146 147 149 150
##   2   1   1   2   2   2   1   2   2   2   2   2   1   2   2
##
## Within cluster sum of squares by cluster:
## [1] 13.991064  9.949643
##  (between_SS / total_SS =  62.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

En la información devuelta observamos las coordenadas que actúan como centro de los dos *cluster* generados, la asignación de las muestras a uno u otro y la suma de error cuadrático interno que indica el grado de similitud de los puntos. Podemos representar los datos para apreciar el trabajo hecho por K-medias.

```
ggplot(train, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(color = as.factor(clusters$cluster), shape = as.factor(Species)))
```




```
clusplot(train[,1:2], clusters$cluster, color = TRUE)
```



En la nube de puntos se aprecia cómo en la parte central, donde los dos *cluster* se entremezclan, tenemos puntos de diferente color (que denota el grupo al que pertenece) y forma (que denota la especie, no usada para hacer el agrupamiento). La segunda representación gráfica nos permite apreciar las fronteras que separan a un grupo de otro.

Si bien la finalidad del agrupamiento es más descriptivo que predictivo, podemos usar los centroides de los *cluster* como referencia, calculando la distancia de las nuevas muestras (las de test) a cad centro y quedándonos con la menor como etiqueta de clase. Para ello comenzamos escribiendo una función que, recibiendo comoparémtros las muestras de test y los *cluster*, devuelve como resultado la predicción:

```
predict.kmeans <- function(muestras, clusters) {
  diferencias <- t(apply(muestras, 1, function(f)
    sqrt(rowSums((f - clusters$centers)^2))))
  ifelse(diferencias[,1] < diferencias[,2], 1, 2)
}
```

Usando la anterior función obtenemos las predicciones tanto para las muestras de entrenamiento como las de test y mostrando el ratio de acierto obtenido.

```
train$Prediccion <- predict.kmeans(train[,1:2], clusters)
test$Prediccion <- predict.kmeans(test[,1:2], clusters)

accuracy <- sum(train$Species == train$Prediccion) / nrow(train) * 100
paste0(accuracy, "% de acierto - entrenamiento")
```

```
## [1] "82.6666666666667% de acierto - entrenamiento"
```

```
accuracy <- sum(test$Species == test$Prediccion) / nrow(test) * 100
paste0(accuracy, "% de acierto - test")
```

```
## [1] "92% de acierto - test"
```

Predicción con árboles de decisión

Los árboles de decisión son modelos de clasificación altamente interpretables, a partir de los cuales es fácil extraer un conjunto de reglas que una persona puede aplicar directamente. En R hay disponibles múltiples algoritos para la generación de árboles, entre ellos el paquete `rpart`. El objetivo será predecir qué pasajeros del Titanic sobrevivirán a partir de datos como su sexo, edad o la clase del pasaje con el que viajaban.

El conjunto de datos `titanic`

Comencemos explorando el conjunto de datos con el que vamos a trabajar, alojado en la variable `titanic_train` del paquete `titanic`. El primer paso será examinar las variables que tiene dicho conjunto de datos:

```
library(titanic)
datos <- titanic_train

str(datos)
```

```
## 'data.frame':    891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
##  $ Sex        : chr  "male" "female" "female" "female" ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  "" "C85" "" "C123" ...
##  $ Embarked   : chr  "S" "C" "S" "S" ...
```

De los atributos con que cuenta cada muestra hay varios que no tienen valor como variables predictoras. Es el caso de `PassengerId`, que identifica de forma única a cada muestra; `Name`, con el nombre de los pasajeros; `Ticket`, `Cabin` o `Embarked`. Las variables que nos serán útiles son las siguientes:

- `Pclass`: 1, 2 o 3 indicando la clase del pasaje
- `Sex`: valores `female` y `male` indicando el sexo
- `Age`: edad de cada pasajero
- `Fare`: precio del pasaje
- `SibSp`: número de hermanos, esposo/esposa a bordo
- `Parch`: número de padres/hijos a bordo
- `Survived`: 0 si el pasajero no sobrevivió, 1 en caso contrario

Nos quedamos solo con esas variables:

```
variables <- c("Pclass", "Sex", "Age", "Fare", "SibSp", "Parch", "Survived")

datos <- datos[, variables]

str(datos)
```

```
## 'data.frame':    891 obs. of  7 variables:
```

```
##  $ Pclass  : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Sex     : chr  "male" "female" "female" "female" ...
##  $ Age     : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ Fare    : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ SibSp   : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch   : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Survived: int  0 1 1 1 0 0 0 0 1 1 ...
```

Análisis de correlaciones entre variables

Dado que tenemos siete variables, repersentar estos datos como puntos en un plano no es fácil. No obstante, podemos analizar las correlaciones existentes entre ellas de una forma mucho más cómoda, mediante un diagrama de correlaciones:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

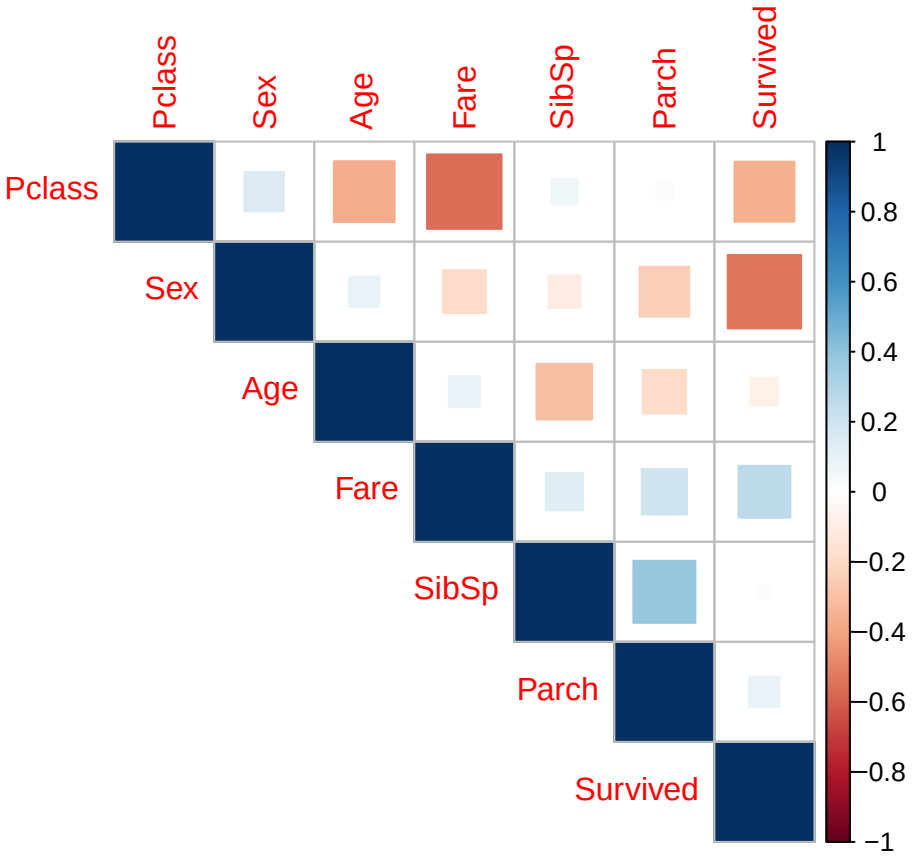
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
datos %>%
  na.omit %>%
  mutate(Sex = as.numeric(as.factor(Sex))) %>%
  cor %>%
  corrplot(method = "square", type = "upper")
```



Fijándonos en la última columna, que es la actúa como clase de las muestras, observamos que hay una alta correlación negativa con la variable `Sex` y algo menor con la variable `Pclass`. También existe un grado de correlación importante, en este caso positiva, respecto a la variable `Fare`.

Generación del árbol y evaluación

Cargamos el paquete `rpart`, dividimos los datos en un subconjunto de entrenamiento y otro de test, generamos el árbol de decisión y mostramos información sobre su estructura:

```
library(rpart)

set.seed(73)
test.ind <- sample(nrow(datos), size = nrow(datos) * 0.25)
train <- datos[-test.ind, ]
test <- datos[test.ind, ]

modelo <- rpart(Survived ~ ., data = train, method = "class")
summary(modelo)
```

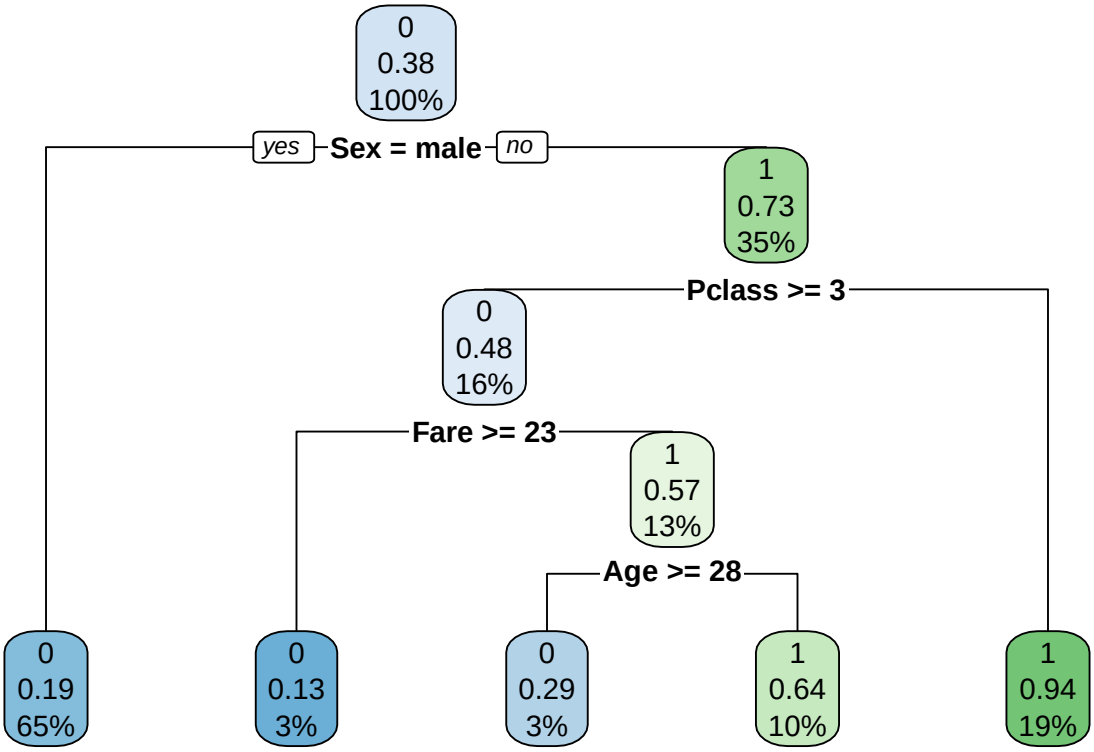
```
## Call:
## rpart(formula = Survived ~ ., data = train, method = "class")
##   n= 669
##
##           CP nsplit rel error    xerror       xstd
## 1 0.42125984      0 1.0000000 1.0000000 0.04941902
## 2 0.03346457      1 0.5787402 0.5787402 0.04216452
## 3 0.02755906      3 0.5118110 0.5748031 0.04206109
```

```
## 4 0.01000000      4 0.4842520 0.5433071 0.04120406
##
## Variable importance
##    Sex   Fare Pclass  Parch  SibSp    Age
##     53     17     15      7      5      3
##
## Node number 1: 669 observations,    complexity param=0.4212598
##   predicted class=0  expected loss=0.3796712  P(node) =1
##     class counts:   415   254
##    probabilities: 0.620 0.380
##   left son=2 (434 obs) right son=3 (235 obs)
##   Primary splits:
##       Sex    splits as  RL,           improve=87.733170, (0 missing)
##       Pclass < 2.5      to the right, improve=33.813280, (0 missing)
##       Fare   < 10.81665 to the left,  improve=27.415860, (0 missing)
##       Age    < 6.5      to the right, improve= 6.757799, (130 missing)
##       SibSp  < 2.5      to the right, improve= 5.237678, (0 missing)
##   Surrogate splits:
##       Fare  < 56.7125  to the left,  agree=0.673, adj=0.068, (0 split)
##       Parch < 1.5      to the left,  agree=0.671, adj=0.064, (0 split)
##       SibSp < 0.5      to the left,  agree=0.656, adj=0.021, (0 split)
##
## Node number 2: 434 observations
##   predicted class=0  expected loss=0.1912442  P(node) =0.6487294
##     class counts:   351    83
##    probabilities: 0.809 0.191
##
## Node number 3: 235 observations,    complexity param=0.03346457
##   predicted class=1  expected loss=0.2723404  P(node) =0.3512706
##     class counts:    64   171
##    probabilities: 0.272 0.728
##   left son=6 (110 obs) right son=7 (125 obs)
##   Primary splits:
##       Pclass < 2.5      to the right, improve=24.997150, (0 missing)
##       SibSp  < 2.5      to the right, improve= 8.920153, (0 missing)
##       Fare   < 48.2     to the left,  improve= 7.985674, (0 missing)
##       Parch  < 0.5      to the right, improve= 3.797474, (0 missing)
##       Age    < 14.75    to the left,  improve= 1.687599, (40 missing)
##   Surrogate splits:
##       Fare  < 25.69795 to the left,  agree=0.796, adj=0.564, (0 split)
##       SibSp < 1.5      to the right, agree=0.600, adj=0.145, (0 split)
##       Parch < 0.5      to the right, agree=0.583, adj=0.109, (0 split)
##       Age   < 16.5     to the left,  agree=0.570, adj=0.082, (0 split)
##
## Node number 6: 110 observations,    complexity param=0.03346457
##   predicted class=0  expected loss=0.4818182  P(node) =0.1644245
##     class counts:    57    53
##    probabilities: 0.518 0.482
##   left son=12 (23 obs) right son=13 (87 obs)
##   Primary splits:
##       Fare  < 23.35    to the right, improve=7.181146, (0 missing)
##       SibSp < 2.5      to the right, improve=3.749537, (0 missing)
##       Parch < 1.5      to the right, improve=3.077456, (0 missing)
##       Age   < 27.5     to the right, improve=2.624733, (30 missing)
```

```
##   Surrogate splits:
##       Parch < 1.5      to the right, agree=0.882, adj=0.435, (0 split)
##       SibSp < 2.5      to the right, agree=0.873, adj=0.391, (0 split)
##
## Node number 7: 125 observations
##   predicted class=1  expected loss=0.056  P(node) =0.186846
##     class counts:     7   118
##    probabilities: 0.056 0.944
##
## Node number 12: 23 observations
##   predicted class=0  expected loss=0.1304348  P(node) =0.03437967
##     class counts:    20     3
##    probabilities: 0.870 0.130
##
## Node number 13: 87 observations,    complexity param=0.02755906
##   predicted class=1  expected loss=0.4252874  P(node) =0.1300448
##     class counts:    37    50
##    probabilities: 0.425 0.575
##   left son=26 (17 obs) right son=27 (70 obs)
##   Primary splits:
##       Age   < 27.5     to the right, improve=2.74886900, (22 missing)
##       Fare  < 7.8417   to the right, improve=1.80616300, (0 missing)
##       SibSp < 1.5      to the left,  improve=0.54139390, (0 missing)
##       Parch < 1.5      to the left,  improve=0.04455842, (0 missing)
##
## Node number 26: 17 observations
##   predicted class=0  expected loss=0.2941176  P(node) =0.02541106
##     class counts:    12     5
##    probabilities: 0.706 0.294
##
## Node number 27: 70 observations
##   predicted class=1  expected loss=0.3571429  P(node) =0.1046338
##     class counts:    25    45
##    probabilities: 0.357 0.643
```

En este resumen se aprecian aspectos como la importancia relativa de las variables a la hora de tomar decisiones, el número de muestras en cada nodo, reparto por clases y otros parámetros. En general, resulta mucho más sencillo analizar esa información si la representamos gráficamente tal y como se muestra a continuación:

```
library(rpart.plot)
rpart.plot(modelo)
```



A continuación evaluamos el clasificador usando las muestras de test que habíamos reservado anteriormente. La matriz de confusión nos permite calcular fácilmente diferentes métricas de evaluación, entre ellas *Accuracy*.

```
prediccion <- predict(modelo, test, type = "class")
matriz.confusion <- table(test$Survived, prediccion)
accuracy <- sum(diag(matriz.confusion)) / sum(matriz.confusion)

matriz.confusion
```

```
##    prediccion
##       0   1
##   0 124  10
##   1  29  59
```

```
accuracy
```

```
## [1] 0.8243243
```

Extracción de las reglas de clasificación

Aunque el modelo devuelto por `rpart()` puede emplearse directamente para obtener predicciones futuras, simplemente facilitando los valores de las variables predictoras, puede interesarnos tener un conjunto de reglas que haga posible aplicar el modelo manualmente. Con esa idea, usamos el paquete `tidyrules` a fin de extraer las reglas y generar una tabla con ellas:

```
library(tidyrules)
library(pander)
```

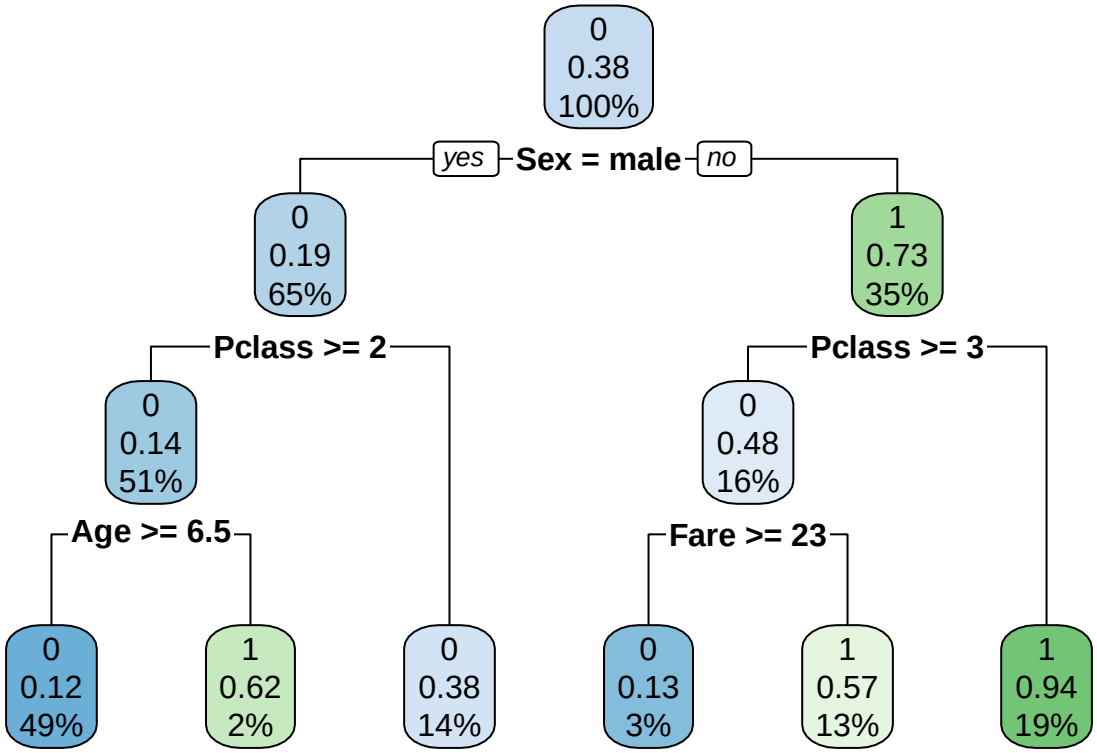
```
reglas <- tidyRules(modelo)
pandoc.table(reglas)
```

```
##
## ------------------------------------------------------------------------
##  id                LHS                RHS   support   confidence   lift
## ---- ------------------------------- ----- --------- ------------ -------
##  1          Sex %in% c('male')          0      434       0.8073     1.301
##
##  2    Sex %in% c('female') & Pclass     0      23         0.84      1.354
##           >= 2.5 & Fare >= 23.35
##
##  3    Sex %in% c('female') & Pclass     0      17        0.6842     1.103
##       >= 2.5 & Fare < 23.35 & Age >=
##                     27.5
##
##  4    Sex %in% c('female') & Pclass     1      70        0.6389     1.683
##       >= 2.5 & Fare < 23.35 & Age <
##                     27.5
##
##  5    Sex %in% c('female') & Pclass     1      125       0.937      2.468
##                     < 2.5
## ------------------------------------------------------------------------
```

Parámetros de control del árbol

En general, al ejecutar un algoritmo que construye un modelo a partir de datos de ejemplo podemos controlar ciertos aspectos alterando los valores por defecto para ciertos parámetros. En el caso de los árboles de decisión, por ejemplo, podemos modificar el número mínimo de muestras que deberá contener cada nodo hoja, la profundidad máxima del árbol, etc. En el caso de `rpart()` esos parámetros se facilitan como una lista `rpart.control`, tal y como se muestra en el siguiente ejemplo:

```
modelo <- rpart(Survived ~., data = train, method = "class",
                control = rpart.control(
                  cp = 0.001,
                  minbucket = 10,
                  maxdepth = 3))
rpart.plot(modelo)
```




```
prediccion <- predict(modelo, test, type = "class")
matriz.confusion <- table(test$Survived, prediccion)
accuracy <- sum(diag(matriz.confusion)) / sum(matriz.confusion)

matriz.confusion
```

```
##    prediccion
##       0   1
##   0 119  15
##   1  22  66
```

```
accuracy
```

```
## [1] 0.8333333
```

```
reglas <- tidyRules(modelo)
pandoc.table(reglas)
```

```
##
## ----------------------------------------------------------------------------
##  id                LHS                 RHS   support   confidence    lift
## ---- ------------------------------- ----- --------- ------------ --------
##  1    Sex %in% c('male') & Pclass >=    0      326        0.8811      1.42
##               1.5 & Age >= 6.5
##
##  2    Sex %in% c('male') & Pclass >=    1      16         0.6111      1.61
##               1.5 & Age < 6.5
##
```

```
##  3    Sex %in% c('male') & Pclass <     0      92        0.617      0.9947
##                    1.5
##
##  4    Sex %in% c('female') & Pclass     0      23         0.84      1.354
##           >= 2.5 & Fare >= 23.35
##
##  5    Sex %in% c('female') & Pclass     1      87        0.573      1.509
##           >= 2.5 & Fare < 23.35
##
##  6    Sex %in% c('female') & Pclass     1      125       0.937      2.468
##                   < 2.5
## ---------------------------------------------------------------------------
```

Como puede comprobarse, el árbol es mucho más sencillo que el obtenido anteriormente. Su ratio de acierto ha descendido ligeramente, pero a cambio se tiene un modelo más fácilmente aplicable y que también genera un conjunto de reglas más reducido, lo cual también facilita su aplicación.

Predicción con kNN y estimación realista del rendimiento

La predicción de etiquetas para nuevas muestras de datos basándose en las que corresponden a sus vecinos más cercanos, aquellas instancias ya conocidas y cuya *cercanía* podemos determinar usando alguna medida de distancia, representa un enfoque fácil de entender. El algoritmo para llevar a cabo esta tarea, conocido como kNN, no construye en general un modelo a partir de los datos de entrenamiento, por lo que todo el trabajo recae en la fase de clasificación. En esta sección usaremos este tipo de clasificador para determinar el tipo de cristal a partir de un conjunto de características químicas. Además, también se explicará cómo llevar a cabo una estimación realista del rendimiento de este clasificador.

Carga y análisis exploratorio

El conjunto de datos `glass` está disponible en el repositorio UCI, por lo que podemos descargarlo directamente (debemos contar con una conexión a Internet) y alojarlo en memoria. Establecemos el nombre de las variables y procedemos con la fase de análisis exploratorio:

```
glass <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data")
colnames(glass) <- c("Id", "RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Class")

str(glass)
```

```
## 'data.frame':    213 obs. of  11 variables:
##  $ Id   : int  2 3 4 5 6 7 8 9 10 11 ...
##  $ RI   : num  1.52 1.52 1.52 1.52 1.52 ...
##  $ Na   : num  13.9 13.5 13.2 13.3 12.8 ...
##  $ Mg   : num  3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 3.46 ...
##  $ Al   : num  1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 1.56 ...
##  $ Si   : num  72.7 73 72.6 73.1 73 ...
##  $ K    : num  0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 0.67 ...
##  $ Ca   : num  7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 8.09 ...
##  $ Ba   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Fe   : num  0 0 0 0 0.26 0 0 0 0.11 0.24 ...
##  $ Class: int  1 1 1 1 1 1 1 1 1 1 ...
```

La variable `Id` es un identificador único para cada muestra de datos, por lo que no resulta de interés para realizar predicciones. La variable `Class` es la que determina la clase, el tipo de cristal, y contempla siete etiquetas distintas. Estamos, por tanto, ante un problema multiclase.

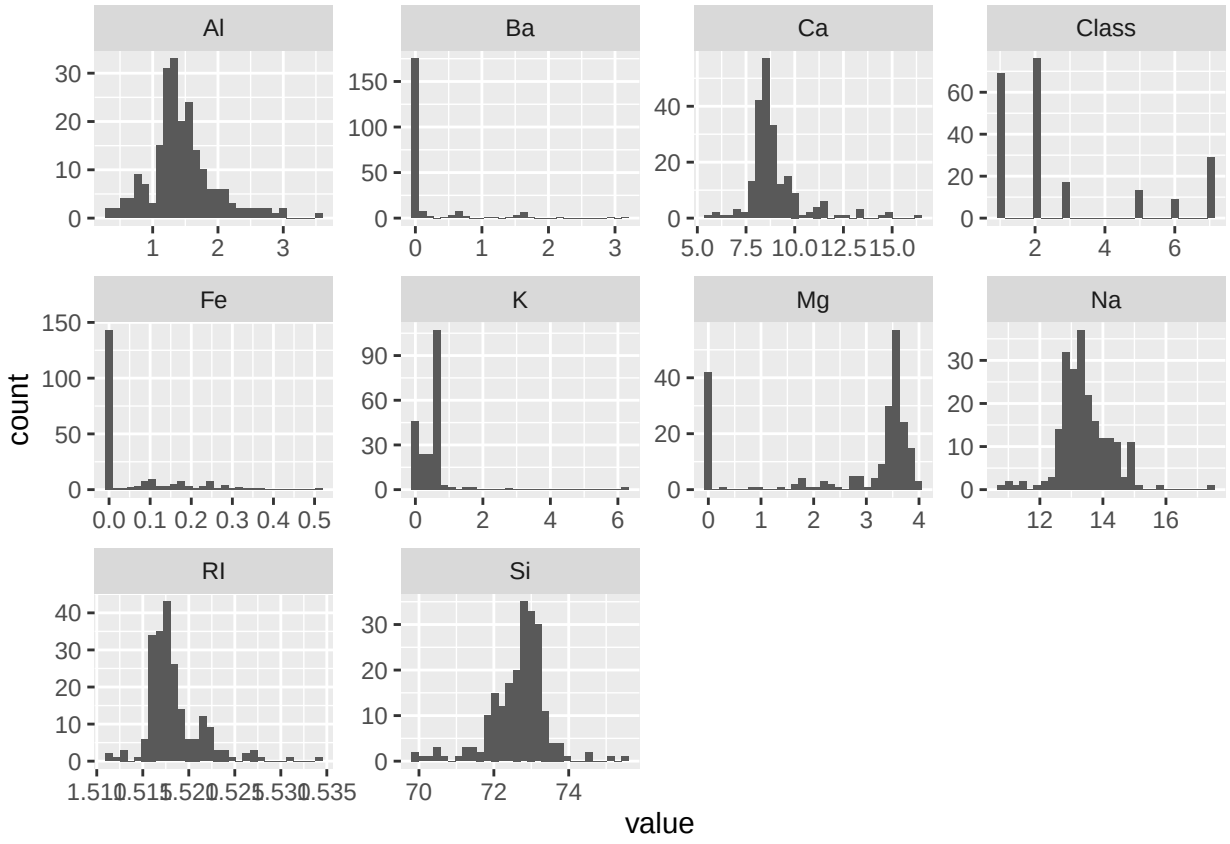
Quedándonos solo con las variables que nos interesan, únicamente vamos a usar un histograma por variable para completar el análisis exploratorio:

```
library (tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------------------
## v tibble  3.0.0     v purrr   0.3.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts ------------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
variables <- c( "RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Class")
datos <- glass[, variables]

datos %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Lo primero que podemos apreciar es que prácticamente ninguna de las variables sigue una distribución normal en sus valores. Por otra parte, la variable `Class` denota que hay un cierto desequilibro entre clases: se trata de un problema de **clasificación desbalanceada**.

Parámetros para controlar el particionamiento y evaluación

Aunque con R podríamos llevar a cabo de forma manual el particionamiento de los datos empleando las diferentes técnicas ya conocidas, como *hold-out*, validación cruzada o *leave-one-out*, esta es una tarea ya automatizada en el paquete `caret`. En él, además, también encontramos implementaciones de la mayoría de algoritmos de aprendizaje más populares, incluyendo kNN, por lo que podempos directamente entrenar y evaluar el predictor para obtener una estimación de su rendimiento:

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
glass$Class <- as.factor(glass$Class)

train_control <- trainControl(method = "cv", number = 5, savePredictions = TRUE)

modelo.knn <- train(Class ~ ., data = glass, method = 'kknn',
                    trControl = train_control)

modelo.knn$results
```

```
##   kmax distance  kernel  Accuracy     Kappa AccuracySD    KappaSD
## 1    5        2 optimal 0.8314942 0.7679482 0.05634331 0.08106022
## 2    7        2 optimal 0.8265000 0.7599685 0.05259673 0.07547314
## 3    9        2 optimal 0.8265000 0.7599371 0.05259673 0.07545171
```

```
summary(modelo.knn)
```

```
##
## Call:
## kknn::train.kknn(formula = .outcome ~ ., data = dat, kmax = param$kmax,     distance = param$distanc
##
## Type of response variable: nominal
## Minimal misclassification: 0.1267606
## Best kernel: optimal
## Best k: 5
```

Usando 5fcv (*5 folds cross validation*) se divide el conjunto de datos en cinco partes de igual número de muestras. En cada iteración se emplearán cuatro de esas partes como entrenamiento y una como test, obteniendo cinco evaluaciones distintas del clasificador. Los datos de rendimiento obtenidos son valores promedio de esas cinco evaluaciones, razón por la que también se incluye la desviación estándar.

En el resumen de resultados se informa de los valores que se han dado a algunos parámetros, como es el caso de `kmax` que determina el número de vecinos tomados en cuenta para realizar la predicción. En la salida resumida se concluye que `k = 5` es la mejor opción. No se han probado otras alternativas para este parámetro y otros potencialmente interesantes, como el tipo de distancia o la función *kernel* a usar para ponderar las distancias de los vecinos.

Ajuste de parámetros del algoritmo

Además de llevar a cabo la estimación del rendimiento mediante la técnica de validación que nos interese, con `caret` también podemos llevar a cabo un proceso de validación interno para la selección de los mejores

valores a emplear pra el clasificador. En el caso de kNN, por ejemplo, podemos probar distintos números de vecinos, funciones de distancia y *kernels* para los pesos:

```
parametros.knn <- expand.grid(kmax = c(1:5), distance = c(1, 2),
                        kernel = c('gaussian', 'epanechnikov', 'rectangular'))

modelo.knn <- train(Class ~ ., data = glass, method = 'kknn',
                  trControl = train_control, tuneGrid = parametros.knn)

modelo.knn$results
```

```
##    kmax distance       kernel  Accuracy     Kappa AccuracySD    KappaSD
## 1     1        1     gaussian 0.9106585 0.8783744 0.03411528 0.04697701
## 2     1        1 epanechnikov 0.9106585 0.8783744 0.03411528 0.04697701
## 3     1        1  rectangular 0.9106585 0.8783744 0.03411528 0.04697701
## 4     1        2     gaussian 0.8600088 0.8074566 0.05108477 0.07234266
## 5     1        2 epanechnikov 0.8600088 0.8074566 0.05108477 0.07234266
## 6     1        2  rectangular 0.8600088 0.8074566 0.05108477 0.07234266
## 7     2        1     gaussian 0.9106585 0.8783744 0.03411528 0.04697701
## 8     2        1 epanechnikov 0.9106585 0.8783744 0.03411528 0.04697701
## 9     2        1  rectangular 0.9106585 0.8783744 0.03411528 0.04697701
## 10    2        2     gaussian 0.8600088 0.8074566 0.05108477 0.07234266
## 11    2        2 epanechnikov 0.8600088 0.8074566 0.05108477 0.07234266
## 12    2        2  rectangular 0.8600088 0.8074566 0.05108477 0.07234266
## 13    3        1     gaussian 0.8960243 0.8583953 0.05035231 0.06739674
## 14    3        1 epanechnikov 0.9103259 0.8772369 0.03174055 0.04580352
## 15    3        1  rectangular 0.8960243 0.8583953 0.05035231 0.06739674
## 16    3        2     gaussian 0.8402697 0.7799269 0.05825756 0.07968488
## 17    3        2 epanechnikov 0.8505853 0.7943519 0.04426708 0.06189211
## 18    3        2  rectangular 0.8402697 0.7796264 0.05825756 0.07947383
## 19    4        1     gaussian 0.8960243 0.8587836 0.05035231 0.06773315
## 20    4        1 epanechnikov 0.9054478 0.8705622 0.03496599 0.04914485
## 21    4        1  rectangular 0.8960243 0.8583953 0.05035231 0.06739674
## 22    4        2     gaussian 0.8402697 0.7799269 0.05825756 0.07968488
## 23    4        2 epanechnikov 0.8688728 0.8199645 0.01730790 0.02341557
## 24    4        2  rectangular 0.8402697 0.7796264 0.05825756 0.07947383
## 25    5        1     gaussian 0.8960243 0.8587836 0.05035231 0.06773315
## 26    5        1 epanechnikov 0.9054478 0.8709505 0.03496599 0.04948600
## 27    5        1  rectangular 0.8960243 0.8583953 0.05035231 0.06739674
## 28    5        2     gaussian 0.8307405 0.7660923 0.05097448 0.07019838
## 29    5        2 epanechnikov 0.8735240 0.8264016 0.02409113 0.03272132
## 30    5        2  rectangular 0.8402697 0.7796264 0.05825756 0.07947383
```

```
summary(modelo.knn)
```

```
##
## Call:
## kknn::train.kknn(formula = .outcome ~ ., data = dat, kmax = param$kmax,     distance = param$distanc
##
## Type of response variable: nominal
## Minimal misclassification: 0.08450704
## Best kernel: gaussian
## Best k: 1
```

El ajuste de estos parámetros nos permite elevar el nivel de acierto del clasificador desde el algo más del 86% previo a más del 93%.

Predicción con máquinas de vectores soporte

Las conocidas como SVM son una familia de modelos predictivos que, si bien resultan menos interpretables que las opciones previas, suelen destacar por su gran rendimiento. Originalmente se diseñaron para abordar problemas de clasificación binaria, por lo que a continuación veremos cómo aplicarlas con el objetivo de identificar automáticamente el correo basura o *spam*.

Comenzamos, como es habitual, llevando a cabo un proceso de análisis exploratorio de los datos para, a continuación, crear un modelo de clasificación que en este caso será una SVM.

Carga de datos y análisis exploratorio

Desde el repositorio UCI podemos descargar un archivo conteniendo un *dataset* correspondiente a varios miles de correos electrónicos clasificados como *spam* y no *spam*. Se trata de un conjunto con 4600 muestras recopilado en Hewlett-Packard Labs. En R podemos importar este mismo *dataset* desde el paquete `kernlab`:

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
data(spam)
```

En la documentación en línea encontraremos información sobre la procedencia del conjunto de datos. También podemos echar un vistazo a sus primeras muestras, así como obtener un resumen de la distribución estadística de los valores que toman sus variables. De esta forma componemos una visión general de la estructura del *dataset*:

```
?spam   # Documentación integrada sobre el dataset
```

```
## starting httpd help server ... done
```

```
head(spam)
```

```
##   make address  all num3d  our over remove internet order mail receive will
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00 0.64
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21 0.79
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38 0.45
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00 0.00
##   people report addresses free business email  you credit your font num000
## 1   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0   0.00
## 2   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0   0.43
## 3   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0   1.16
## 4   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 5   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 6   0.00   0.00      0.00 0.00     0.00  0.00 0.00   0.00 0.00    0   0.00
##   money hp hpl george num650 lab labs telnet num857 data num415 num85
## 1  0.00  0   0      0      0   0    0      0      0    0      0     0
## 2  0.43  0   0      0      0   0    0      0      0    0      0     0
```

```
## 3  0.06  0   0      0      0   0    0      0      0    0      0     0
## 4  0.00  0   0      0      0   0    0      0      0    0      0     0
## 5  0.00  0   0      0      0   0    0      0      0    0      0     0
## 6  0.00  0   0      0      0   0    0      0      0    0      0     0
##   technology num1999 parts pm direct cs meeting original project   re  edu
## 1          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 2          0    0.07     0  0   0.00  0       0     0.00       0 0.00 0.00
## 3          0    0.00     0  0   0.06  0       0     0.12       0 0.06 0.06
## 4          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 5          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 6          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
##   table conference charSemicolon charRoundbracket charSquarebracket
## 1     0          0          0.00            0.000                 0
## 2     0          0          0.00            0.132                 0
## 3     0          0          0.01            0.143                 0
## 4     0          0          0.00            0.137                 0
## 5     0          0          0.00            0.135                 0
## 6     0          0          0.00            0.223                 0
##   charExclamation charDollar charHash capitalAve capitalLong capitalTotal type
## 1           0.778      0.000    0.000      3.756          61          278 spam
## 2           0.372      0.180    0.048      5.114         101         1028 spam
## 3           0.276      0.184    0.010      9.821         485         2259 spam
## 4           0.137      0.000    0.000      3.537          40          191 spam
## 5           0.135      0.000    0.000      3.537          40          191 spam
## 6           0.000      0.000    0.000      3.000          15           54 spam
```

```
summary(spam)
```

```
##       make              address               all              num3d         
##  Min.   :0.0000   Min.   : 0.000   Min.   :0.0000   Min.   : 0.00000  
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.: 0.00000  
##  Median :0.0000   Median : 0.000   Median :0.0000   Median : 0.00000  
##  Mean   :0.1046   Mean   : 0.213   Mean   :0.2807   Mean   : 0.06542  
##  3rd Qu.:0.0000   3rd Qu.: 0.000   3rd Qu.:0.4200   3rd Qu.: 0.00000  
##  Max.   :4.5400   Max.   :14.280   Max.   :5.1000   Max.   :42.81000  
##       our              over             remove           internet      
##  Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median : 0.0000   Median :0.0000   Median :0.0000   Median : 0.0000  
##  Mean   : 0.3122   Mean   :0.0959   Mean   :0.1142   Mean   : 0.1053  
##  3rd Qu.: 0.3800   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000  
##  Max.   :10.0000   Max.   :5.8800   Max.   :7.2700   Max.   :11.1100  
##      order             mail             receive            will        
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.00000   Median : 0.0000   Median :0.00000   Median :0.1000  
##  Mean   :0.09007   Mean   : 0.2394   Mean   :0.05982   Mean   :0.5417  
##  3rd Qu.:0.00000   3rd Qu.: 0.1600   3rd Qu.:0.00000   3rd Qu.:0.8000  
##  Max.   :5.26000   Max.   :18.1800   Max.   :2.61000   Max.   :9.6700  
##      people            report          addresses           free        
##  Min.   :0.00000   Min.   : 0.00000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median :0.00000   Median : 0.00000   Median :0.0000   Median : 0.0000  
##  Mean   :0.09393   Mean   : 0.05863   Mean   :0.0492   Mean   : 0.2488  
##  3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.1000  
```

```
##  Max.   :5.55000   Max.   :10.00000   Max.   :4.4100   Max.   :20.0000
##     business          email             you              credit
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   : 0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 0.00000
##  Median :0.0000   Median :0.0000   Median : 1.310   Median : 0.00000
##  Mean   :0.1426   Mean   :0.1847   Mean   : 1.662   Mean   : 0.08558
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 2.640   3rd Qu.: 0.00000
##  Max.   :7.1400   Max.   :9.0900   Max.   :18.750   Max.   :18.18000
##       your              font             num000           money
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.00000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.00000
##  Median : 0.2200   Median : 0.0000   Median :0.0000   Median : 0.00000
##  Mean   : 0.8098   Mean   : 0.1212   Mean   :0.1016   Mean   : 0.09427
##  3rd Qu.: 1.2700   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.00000
##  Max.   :11.1100   Max.   :17.1000   Max.   :5.4500   Max.   :12.50000
##        hp               hpl              george           num650
##  Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median :0.0000
##  Mean   : 0.5495   Mean   : 0.2654   Mean   : 0.7673   Mean   :0.1248
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:0.0000
##  Max.   :20.8300   Max.   :16.6600   Max.   :33.3300   Max.   :9.0900
##       lab               labs             telnet            num857
##  Min.   : 0.00000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   : 0.09892   Mean   :0.1029   Mean   : 0.06475   Mean   :0.04705
##  3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :14.28000   Max.   :5.8800   Max.   :12.50000   Max.   :4.76000
##       data              num415            num85          technology
##  Min.   : 0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.00000   Median : 0.0000   Median :0.00000
##  Mean   : 0.09723   Mean   :0.04784   Mean   : 0.1054   Mean   :0.09748
##  3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.00000
##  Max.   :18.18000   Max.   :4.76000   Max.   :20.0000   Max.   :7.69000
##     num1999          parts              pm               direct
##  Min.   :0.000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median :0.000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   :0.137   Mean   :0.0132   Mean   : 0.07863   Mean   :0.06483
##  3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :6.890   Max.   :8.3300   Max.   :11.11000   Max.   :4.76000
##        cs              meeting          original         project
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.0000
##  Median :0.00000   Median : 0.0000   Median :0.0000   Median : 0.0000
##  Mean   :0.04367   Mean   : 0.1323   Mean   :0.0461   Mean   : 0.0792
##  3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000
##  Max.   :7.14000   Max.   :14.2800   Max.   :3.5700   Max.   :20.0000
##        re               edu              table            conference
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.000000   Min.   : 0.00000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.: 0.00000
##  Median : 0.0000   Median : 0.0000   Median :0.000000   Median : 0.00000
```

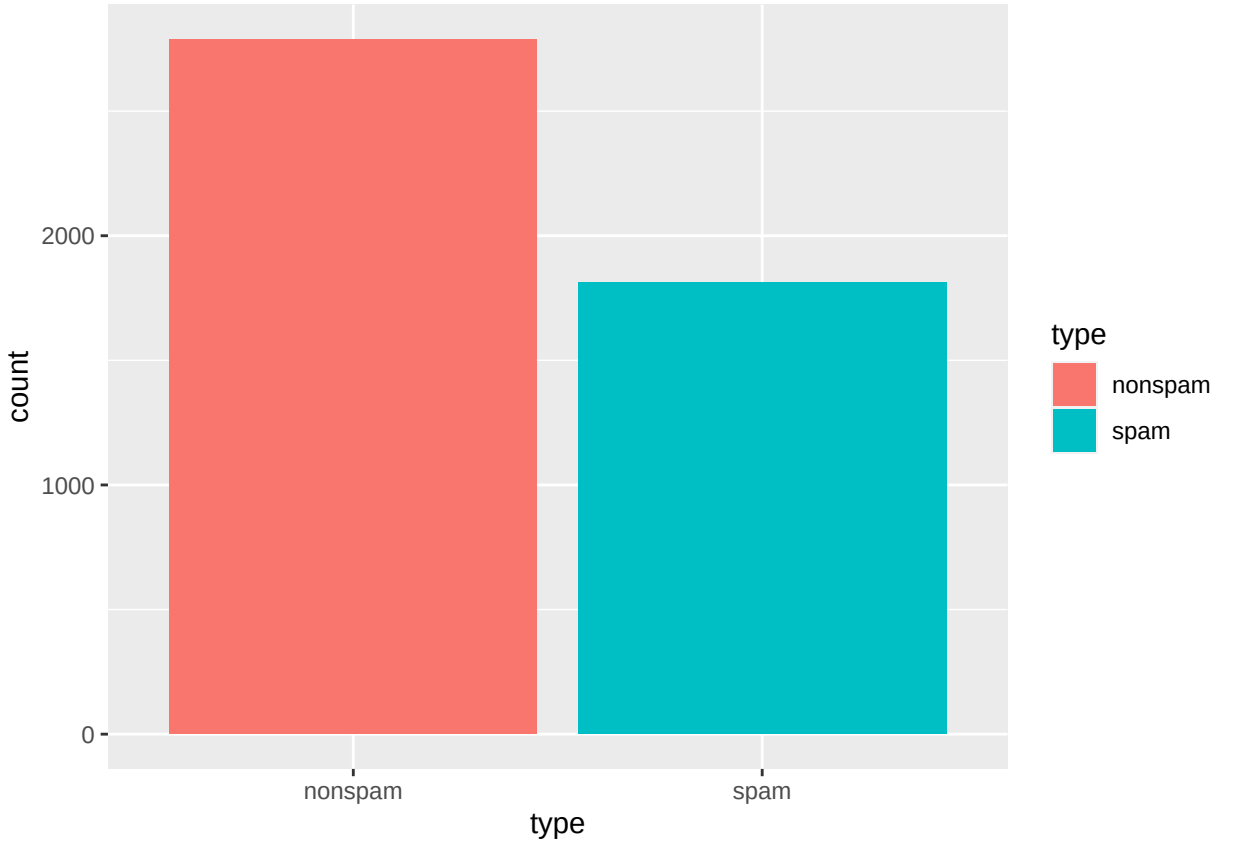
```
##  Mean   : 0.3012   Mean   : 0.1798   Mean   :0.005444   Mean   : 0.03187
##  3rd Qu.: 0.1100   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.: 0.00000
##  Max.   :21.4200   Max.   :22.0500   Max.   :2.170000   Max.   :10.00000
##  charSemicolon     charRoundbracket charSquarebracket charExclamation
##  Min.   :0.00000   Min.   :0.000    Min.   :0.00000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.:0.000    1st Qu.:0.00000   1st Qu.: 0.0000
##  Median :0.00000   Median :0.065    Median :0.00000   Median : 0.0000
##  Mean   :0.03857   Mean   :0.139    Mean   :0.01698   Mean   : 0.2691
##  3rd Qu.:0.00000   3rd Qu.:0.188    3rd Qu.:0.00000   3rd Qu.: 0.3150
##  Max.   :4.38500   Max.   :9.752    Max.   :4.08100   Max.   :32.4780
##    charDollar        charHash          capitalAve        capitalLong
##  Min.   :0.00000   Min.   : 0.00000   Min.   :   1.000   Min.   :   1.00
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:   1.588   1st Qu.:   6.00
##  Median :0.00000   Median : 0.00000   Median :   2.276   Median :  15.00
##  Mean   :0.07581   Mean   : 0.04424   Mean   :   5.191   Mean   :  52.17
##  3rd Qu.:0.05200   3rd Qu.: 0.00000   3rd Qu.:   3.706   3rd Qu.:  43.00
##  Max.   :6.00300   Max.   :19.82900   Max.   :1102.500   Max.   :9989.00
##   capitalTotal          type
##  Min.   :    1.0   nonspam:2788
##  1st Qu.:   35.0   spam   :1813
##  Median :   95.0
##  Mean   :  283.3
##  3rd Qu.:  266.0
##  Max.   :15841.0
```

A continuación recurrimos a distintas representaciones gráficas con el objetivo de conocer mejor ciertos aspectos, como la distribución de las clases, frecuencia de ciertos símbolos y palabras en los correos electrónicos según su tipo:

Cantidad de correos según tipo: spam/legítimo

Comenzamos comprobando la cantidad de correos que hay en el conjunto de datos de cada tipo, para lo cual es suficiente con una gráfica de barras:

```
library(ggplot2)

ggplot(spam, aes(x = type)) + geom_bar(aes(fill = type))
```



Calculamos la ratio de desbalanceo

Es fácil apreciar que existe un cierto desequilibrio en el número de muestras de cada clase. Calculamos el conocido com IR (*Imbalance Ratio*) para determinar exactamente la medida en que el *dataset* está desbalanceado:

```
library(dplyr)
conteo <- spam %>% group_by(type) %>% count
conteo$n[1] / conteo$n[2]
```

```
## [1] 1.537783
```

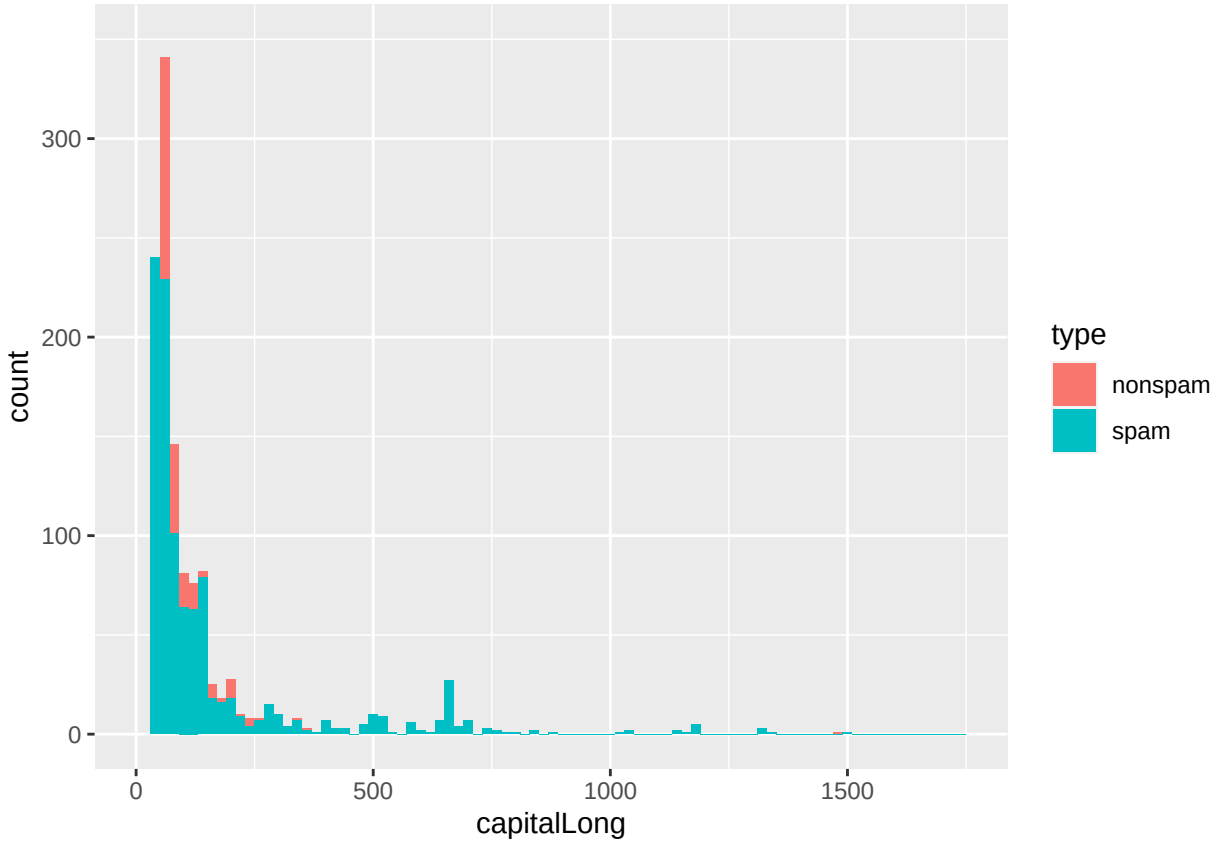
Frecuencias de mayúsculas y uso del símbolo $

En los correos basura es habitual abusar de las mayúsculas, así como del uso de ciertos símbolos. Podemos comprobarlo examinando la frecuencia de algunos de los atributos del conjunto de datos diferenciando por tipo de correo:

```
ggplot(spam, aes(x = capitalLong)) +
  geom_histogram(aes(fill = type), binwidth = 20) +
  scale_x_continuous(limits = c(0, 1750)) +
  scale_y_continuous(limits = c(0, 350))
```

```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```

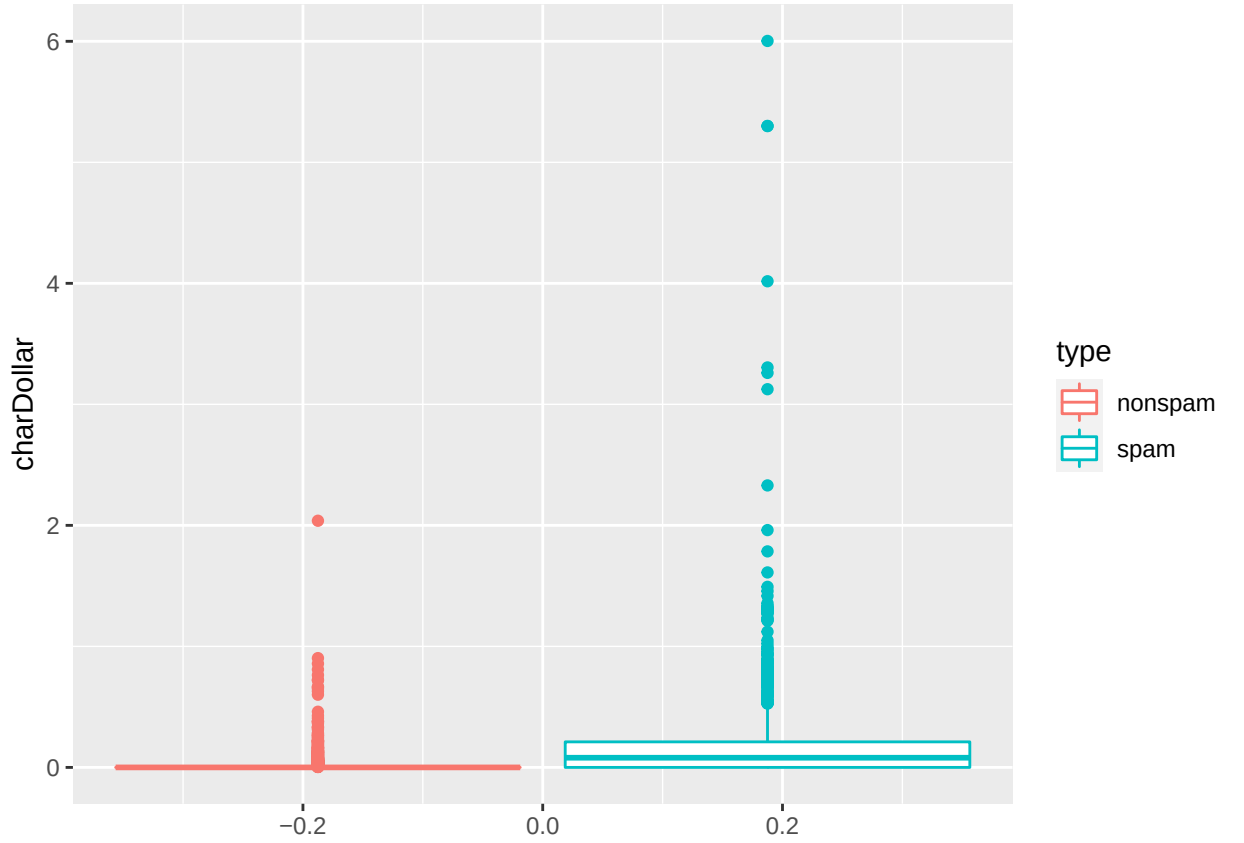
```
## Warning: Removed 5 rows containing missing values (geom_bar).
```



Aquí comprobamos cómo las secuencias largas de texto en mayúsculas corresponden casi por completo a correos basura, mientas que los legítimos destacan por un uso moderado.

Veamos qué ocurre con el uso del símbolo *$*:

```
ggplot(spam, aes(y = charDollar)) +
  geom_boxplot(aes(color = type))
```




```
spam %>% group_by(type) %>% summarize(Apariciones = sum(charDollar), Promedio = mean(charDollar))
```

```
## # A tibble: 2 x 3
##   type    Apariciones Promedio
##   <fct>         <dbl>    <dbl>
## 1 nonspam        32.5   0.0116
## 2 spam          316.    0.174
```

En los diagramas de cajas y bigotes se aprecia que su uso es mucho más frecuente en los correos basura, algo qe confirman los valores absolutos y relativos de apariciones calculados a continuación.

Frecuencias de algunas palabras según el tipo de correo

Es habitual que en los correos basura aparezcan ciertas palabras que son mucho menos corrientes en mensajes legítimos. Vamos a tomar parte de los términos del dataset, concretamente los primeros veinte, y a representarlos en gráficas de barras según el tipo de correo:

```
library(reshape2)
```

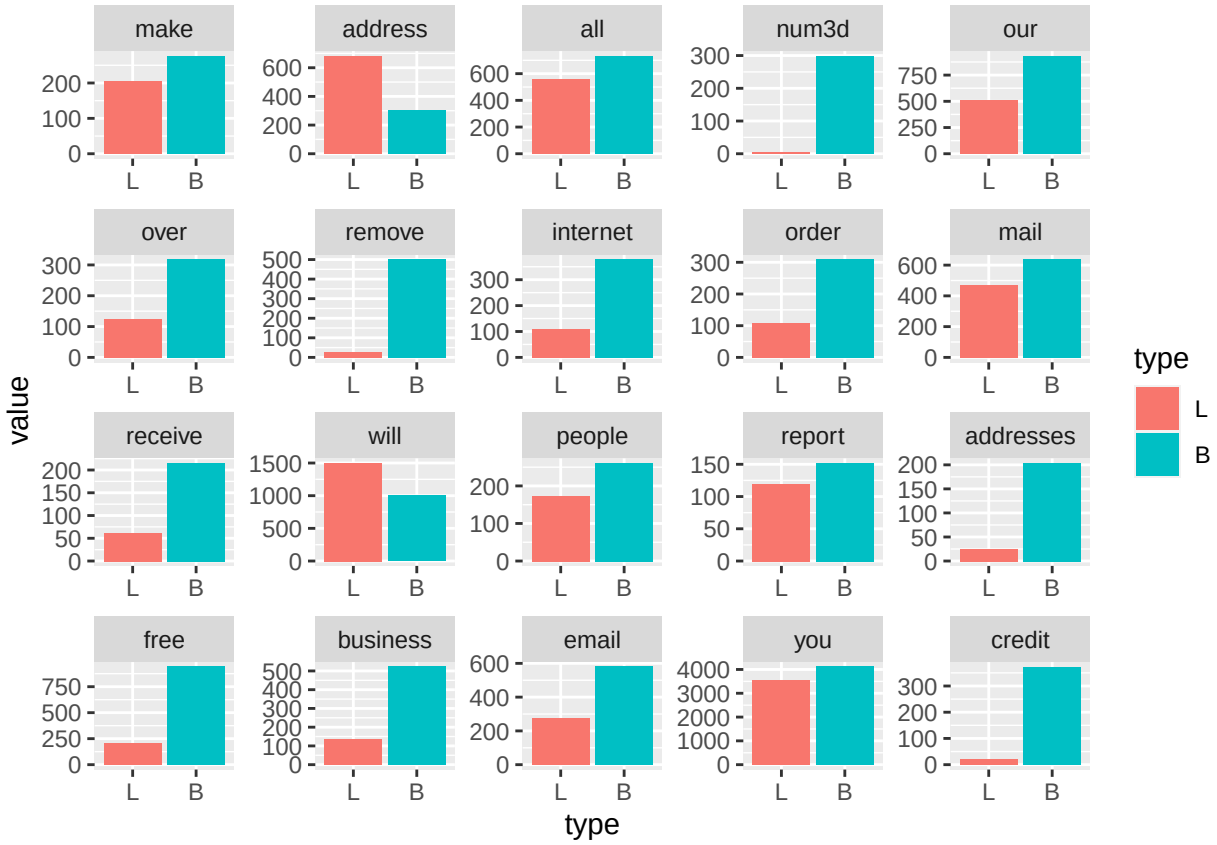
```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
datos <- melt(spam, measure.vars = 1:20, id.vars = "type")
levels(datos$type) <- c("L", "B")
str(datos)
```

```
## 'data.frame':    92020 obs. of  3 variables:
##  $ type    : Factor w/ 2 levels "L","B": 2 2 2 2 2 2 2 2 2 2 ...
##  $ variable: Factor w/ 20 levels "make","address",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ value   : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
```

```
ggplot(datos, aes(y = value, x = type, fill = type)) +
  geom_bar(stat = "identity") +
  facet_wrap(~variable, scales = "free")
```



Observando estas gráficas puede apreciarse que términos como *credit*, *business* o *receive* apenas aparecen en los mensajes legítimos pero sí en el correo basura. Lo inverso ocurre con otras palabras (aquí solo se han examinado algunas de las que aparecen en cada mensaje).

Introducción al paquete `caret`

Una vez que conocemos el conjunto de datos con el que estamos trabajando, procedemos a entrenar una SVM con la finalidad de generar un modelo de clasificación adecuado.

En este caso vamos a usar el paquete `caret`. Al igual que el `scikit-learn` de Python, este paquete R nos ofrece una metodología común para usar prácticamente cualquier algoritmo de aprendizaje automático. Para ello se apoya fundamentalmente en dos funciones:

- `train()`: se encarga de entrenar el modelo indicado por el parámetro `method` con el objetivo de crear un predictor acorde a la fórmula entregada como primer argumento. Es capaz de llevar a cabo el particionamiento y estimación del rendimiento del predictor, todo ello de forma automática. Actualmente `caret` nos ofrece más de 230 métodos, cuyos nombres es fácil obtener:

```
library(caret)
names(getModelInfo())
```

```
##   [1] "ada"                 "AdaBag"              "AdaBoost.M1"
##   [4] "adaboost"            "amdai"               "ANFIS"
##   [7] "avNNet"              "awnb"                "awtan"
##  [10] "bag"                 "bagEarth"            "bagEarthGCV"
##  [13] "bagFDA"              "bagFDAGCV"           "bam"
##  [16] "bartMachine"         "bayesglm"            "binda"
##  [19] "blackboost"          "blasso"              "blassoAveraged"
##  [22] "bridge"              "brnn"                "BstLm"
##  [25] "bstSm"               "bstTree"             "C5.0"
##  [28] "C5.0Cost"            "C5.0Rules"           "C5.0Tree"
##  [31] "cforest"             "chaid"               "CSimca"
##  [34] "ctree"               "ctree2"              "cubist"
##  [37] "dda"                 "deepboost"           "DENFIS"
##  [40] "dnn"                 "dwdLinear"           "dwdPoly"
##  [43] "dwdRadial"           "earth"               "elm"
##  [46] "enet"                "evtree"              "extraTrees"
##  [49] "fda"                 "FH.GBML"             "FIR.DM"
##  [52] "foba"                "FRBCS.CHI"           "FRBCS.W"
##  [55] "FS.HGD"              "gam"                 "gamboost"
##  [58] "gamLoess"            "gamSpline"           "gaussprLinear"
##  [61] "gaussprPoly"         "gaussprRadial"       "gbm_h2o"
##  [64] "gbm"                 "gcvEarth"            "GFS.FR.MOGUL"
##  [67] "GFS.LT.RS"           "GFS.THRIFT"          "glm.nb"
##  [70] "glm"                 "glmboost"            "glmnet_h2o"
##  [73] "glmnet"              "glmStepAIC"          "gpls"
##  [76] "hda"                 "hdda"                "hdrda"
##  [79] "HYFIS"               "icr"                 "J48"
##  [82] "JRip"                "kernelpls"           "kknn"
##  [85] "knn"                 "krlsPoly"            "krlsRadial"
##  [88] "lars"                "lars2"               "lasso"
##  [91] "lda"                 "lda2"                "leapBackward"
##  [94] "leapForward"         "leapSeq"             "Linda"
##  [97] "lm"                  "lmStepAIC"           "LMT"
## [100] "loclda"              "logicBag"            "LogitBoost"
## [103] "logreg"              "lssvmLinear"         "lssvmPoly"
## [106] "lssvmRadial"         "lvq"                 "M5"
## [109] "M5Rules"             "manb"                "mda"
## [112] "Mlda"                "mlp"                 "mlpKerasDecay"
## [115] "mlpKerasDecayCost"   "mlpKerasDropout"     "mlpKerasDropoutCost"
## [118] "mlpML"               "mlpSGD"              "mlpWeightDecay"
## [121] "mlpWeightDecayML"    "monmlp"              "msaenet"
## [124] "multinom"            "mxnet"               "mxnetAdam"
## [127] "naive_bayes"         "nb"                  "nbDiscrete"
## [130] "nbSearch"            "neuralnet"           "nnet"
## [133] "nnls"                "nodeHarvest"         "null"
## [136] "OneR"                "ordinalNet"          "ordinalRF"
## [139] "ORFlog"              "ORFpls"              "ORFridge"
## [142] "ORFsvm"              "ownn"                "pam"
## [145] "parRF"               "PART"                "partDSA"
## [148] "pcaNNet"             "pcr"                 "pda"
## [151] "pda2"                "penalized"           "PenalizedLDA"
```

```
## [154] "plr"                "pls"                 "plsRglm"
## [157] "polr"               "ppr"                 "PRIM"
## [160] "protoclass"         "qda"                 "QdaCov"
## [163] "qrf"                "qrnn"                "randomGLM"
## [166] "ranger"             "rbf"                 "rbfDDA"
## [169] "Rborist"            "rda"                 "regLogistic"
## [172] "relaxo"             "rf"                  "rFerns"
## [175] "RFlda"              "rfRules"             "ridge"
## [178] "rlda"               "rlm"                 "rmda"
## [181] "rocc"               "rotationForest"      "rotationForestCp"
## [184] "rpart"              "rpart1SE"            "rpart2"
## [187] "rpartCost"          "rpartScore"          "rqlasso"
## [190] "rqnc"               "RRF"                 "RRFglobal"
## [193] "rrlda"              "RSimca"              "rvmLinear"
## [196] "rvmPoly"            "rvmRadial"           "SBC"
## [199] "sda"                "sdwd"                "simpls"
## [202] "SLAVE"              "slda"                "smda"
## [205] "snn"                "sparseLDA"           "spikeslab"
## [208] "spls"               "stepLDA"             "stepQDA"
## [211] "superpc"            "svmBoundrangeString" "svmExpoString"
## [214] "svmLinear"          "svmLinear2"          "svmLinear3"
## [217] "svmLinearWeights"   "svmLinearWeights2"   "svmPoly"
## [220] "svmRadial"          "svmRadialCost"       "svmRadialSigma"
## [223] "svmRadialWeights"   "svmSpectrumString"   "tan"
## [226] "tanSearch"          "treebag"             "vbmpRadial"
## [229] "vglmAdjCat"         "vglmContRatio"       "vglmCumulative"
## [232] "widekernelpls"      "WM"                  "wsrf"
## [235] "xgbDART"            "xgbLinear"           "xgbTree"
## [238] "xyf"
```

Además de particionar los datos, entrenar y evaluar los modelos, el método `train()` también es capaz de ajustar los parámetros de funcionamiento de los algoritmos siempre que se le facilite, mediante el parámetro `tuneGrid`, la lista de parámetros e intervalos a probar. Asimismo, `train()` puede aplicar distintos preprocesamiento a los datos antes de usarlo, por ejemplo normalizando los valores de las variables.

- `predict()`: una vez que tenemos el modelo entrenado, la función `predict()` lo usará para, aplicándolo sobre nuevas muestras de datos, generar las predicciones correspondientes.

Clasificación con SVM

Procedemos particionando nuestro conjunto de datos. Para ello no realizaremos un particionamiento aleatorio, como en ejemplos previos, sino de tipo estratificado, de forma que la distribución de clases sea similar entre los datos de entrenamiento y prueba. De esto se encarga la función `createDataPartition()`. Nos reservamos un 30% de los correos para verificar al final el rendimiento del clasificador:

```
library(caret)
set.seed(73)

# Particionamos los datos de forma estratificada, reservando un 30% para test final
indices.test <- createDataPartition(y = spam$type, p = 0.3, list = FALSE)
entrenamiento <- spam[-indices.test, ]
test <- spam[indices.test, ]

conteo.entrenamiento <- entrenamiento %>% group_by(type) %>% count
conteo.test <- test %>% group_by(type) %>% count
```

```r
cat("Proporción de clases en entrenamiento:", conteo.entrenamiento$n[1] / conteo.entrenamiento$n[2], "\n
```

```
## Proporción de clases en entrenamiento: 1.537431
```

```r
cat("Proporción de clases en test:", conteo.test$n[1] / conteo.test$n[2])
```

```
## Proporción de clases en test: 1.538603
```

A continuación, usando solo los datos de entrenamiento, llevamos a cabo una validación cruzada con 5 porciones a fin de entrenar la SVM y obtener una estimación inicial de su rendimiento. Usamos un *kernel* de tipo RBF para la proyección en dimensiones superiores y optimizamos la métrica ROC durante el entrenamiento:

```r
# Entrenamos un modelo de SVM con kernel RBF y obtenemos el estimador
train_control <- trainControl(
  method = "cv", number = 5,
  classProbs = TRUE, savePredictions = TRUE,
  summaryFunction = twoClassSummary)

svm.spam <- train(type ~., data = entrenamiento, method = "svmRadial",
      trControl = train_control,
      metric = "ROC")
svm.spam
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 3220 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2576, 2575, 2576, 2576, 2577
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.25  0.9660571  0.9533517  0.8636768
##   0.50  0.9697422  0.9543800  0.8841679
##   1.00  0.9725213  0.9569441  0.8904703
##
## Tuning parameter 'sigma' was held constant at a value of 0.03016092
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03016092 and C = 1.
```

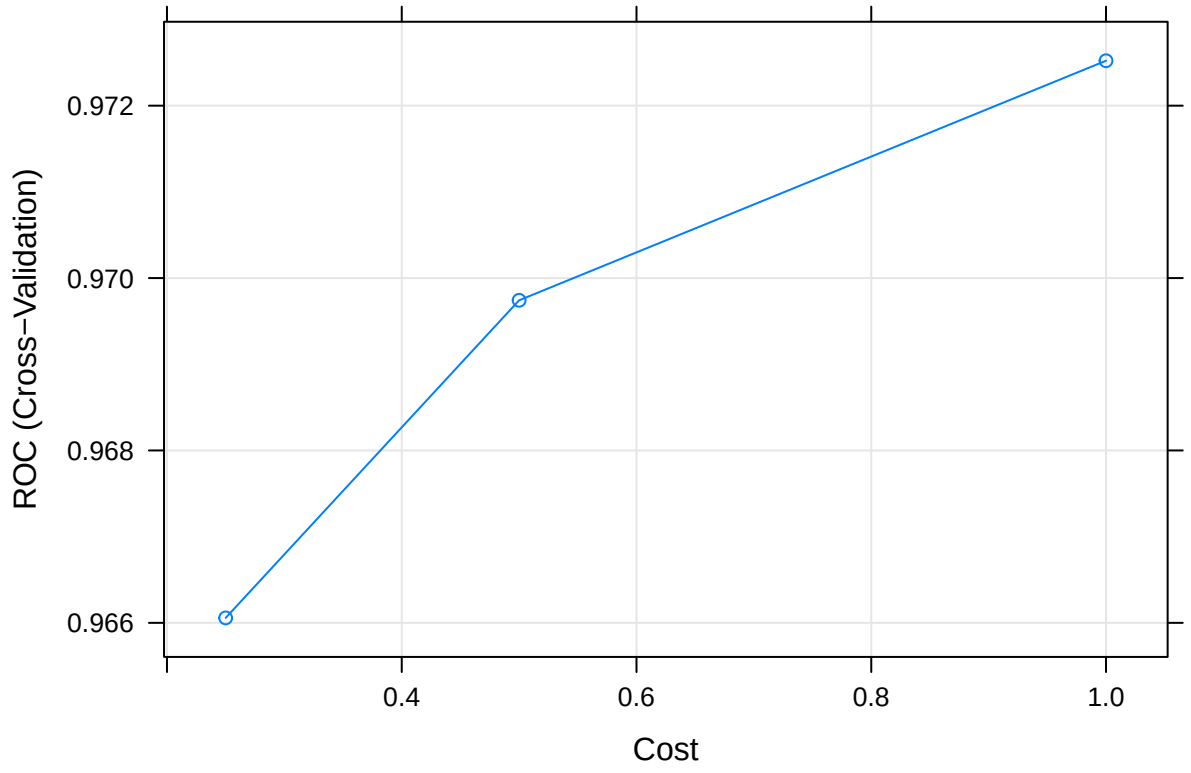
La salida del clasificador nos indica que la mejor configuración es con `C = 1`, alcanzando un ROC de casi el 97%. Podemos obtener métricas adicionales de rendimeinto con el método `multiClassSummary()`:

```r
multiClassSummary(svm.spam$pred, svm.spam$levels)
```

```
##           logLoss               AUC            prAUC          Accuracy
##         0.2141217         0.9692425         0.9630450         0.9251553
##             Kappa                F1       Sensitivity       Specificity
##         0.8418644         0.9392488         0.9548949         0.8794326
##    Pos_Pred_Value    Neg_Pred_Value         Precision            Recall
##         0.9241071         0.9269103         0.9241071         0.9548949
##    Detection_Rate Balanced_Accuracy
##         0.5785714         0.9171638
```

Podemos examinar cómo ha cambiado el rendimiento del clasificador en proporción al coste (parámetro C):

```
plot(svm.spam)
```

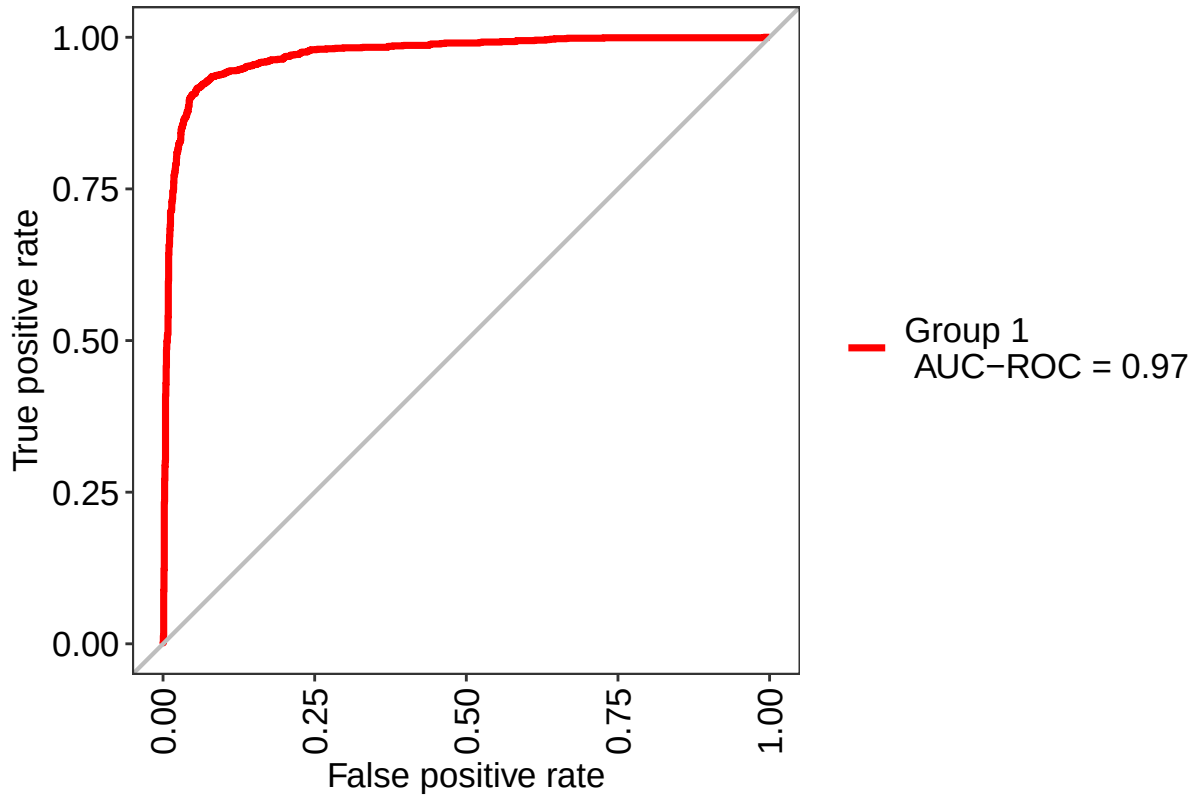


También annalizar la curva ROC del clasificador generado:

```
library(MLeval)
evalua <- evalm(svm.spam, showplots = FALSE)
```

```
## ***MLeval: Machine Learning Model Evaluation***
## Input: caret train function object
## Not averaging probs.
## Group 1 type: cv
## Observations: 3220
## Number of groups: 1
## Observations per group: 3220
## Positive: spam
## Negative: nonspam
## Group: Group 1
## Positive: 1269
## Negative: 1951
## ***Performance Metrics***
```

```
## Group 1 Optimal Informedness = 0.859812853847555
## Group 1 AUC-ROC = 0.97
evalua$roc
```



Asumiendo que tuviésemos nuevos correos entrantes, usaríamos la función `predict` para determinar si son *spam* o no. La matriz de confusión nos indica el nivel de acierto con que se realiza esa predicción usando muestras de datos antes no vistas:

```
predicciones <- predict(svm.spam, test, type = "prob")
predicciones$obs <- test$type
predicciones$pred <- predict(svm.spam, test)
multiClassSummary(predicciones, lev = levels(predicciones$obs))
```

```
##           logLoss               AUC              prAUC          Accuracy
##         0.2059498         0.9704213         0.9669517         0.9283128
##             Kappa                F1       Sensitivity       Specificity
##         0.8483550         0.9419355         0.9593787         0.8805147
##    Pos_Pred_Value    Neg_Pred_Value         Precision            Recall
##         0.9251152         0.9337232         0.9251152         0.9593787
##    Detection_Rate Balanced_Accuracy
##         0.5814627         0.9199467
```

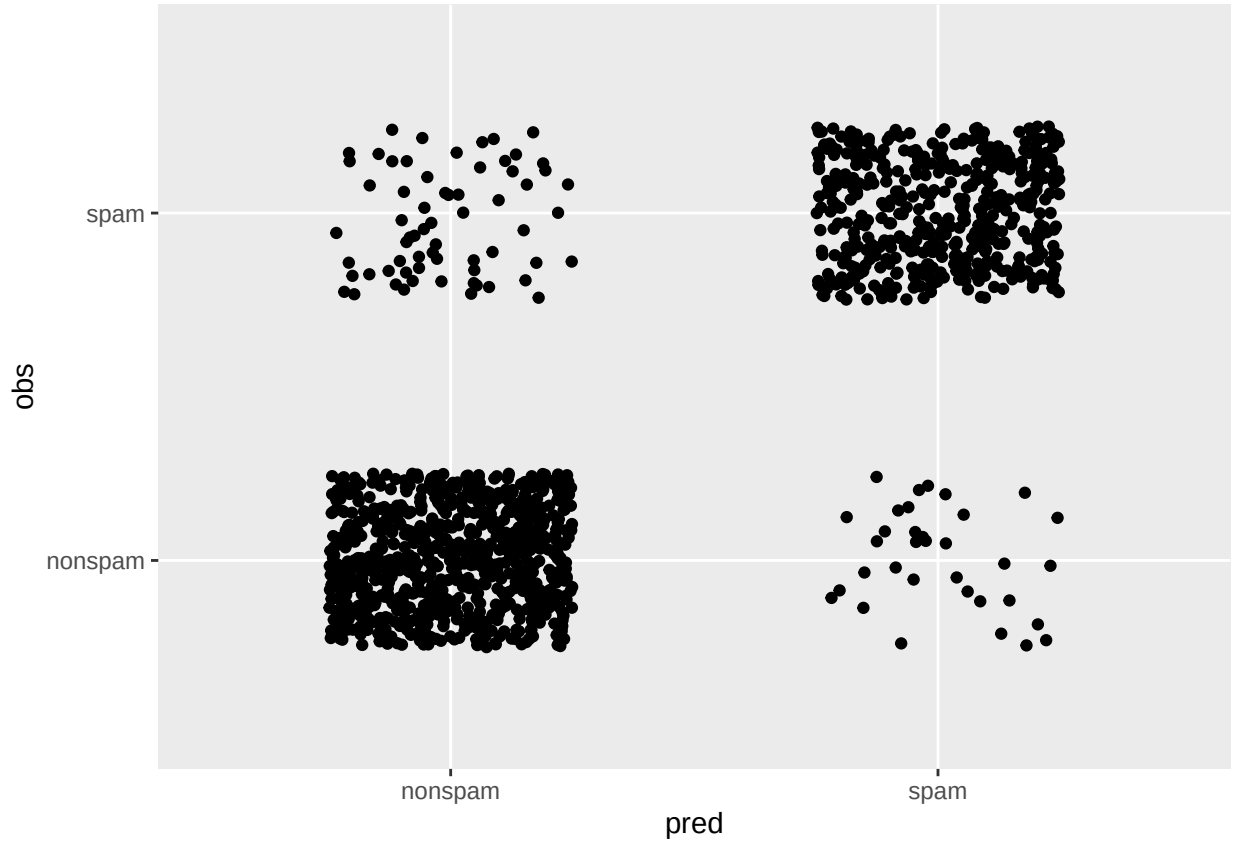
```
confusionMatrix(predicciones$pred, test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction nonspam spam
##    nonspam     803   65
##    spam         34  479
##
##                Accuracy : 0.9283
##                  95% CI : (0.9134, 0.9414)
##     No Information Rate : 0.6061
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8484
##
##  Mcnemar's Test P-Value : 0.002569
##
##             Sensitivity : 0.9594
##             Specificity : 0.8805
##          Pos Pred Value : 0.9251
##          Neg Pred Value : 0.9337
##              Prevalence : 0.6061
##          Detection Rate : 0.5815
##    Detection Prevalence : 0.6285
##       Balanced Accuracy : 0.9199
##
##        'Positive' Class : nonspam
##
```

Mediante una gráfica de nube de puntos podemos mostrar gráficamente el volumen de aciertos y fallos para cada clase. Básicamente es una visualización de la mamtriz de confusión:

```
pred <- data.frame(pred = predicciones$pred, obs = predicciones$obs)
ggplot(pred, aes(x = pred, y = obs)) +
  geom_jitter(position = position_jitter(width = 0.25, height = 0.25))
```



Ensembles de clasificadores

En sesiones previas conocimos distintos tipos de *ensembles*, conjuntos de múltiples clasificadores, entrenados con subconjuntos de muestras y/o variables para inducir diversidad, que combinan sus salidas mediante alguna estrategia de voto.

En esta última sección, usando el paquete `caret` que ya conocemos, se introducen varios de los ensembles disponibles en R.

Preparación de paquetes y datos

Comenzamos cargando los paquetes necesarios y particionamos los datos reservando un 30% de muestras para el test final, simulando que tenemos nuevos correos a clasificar:

```
library(caret)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
trellis.par.set(caretTheme())  # Aspecto para algunas gráficas

# Particionamos los datos
set.seed(73)
indices <- createDataPartition(spam$type, p = .70, list = FALSE)
training = spam[indices, ]
test = spam[-indices, ]
```

Entrenamos los modelos multi-clasificadores

Usando en este caso una 10fcv, entrenamos varios *ensembles* usando los enfoques de *bagging*, *boosting* y *random forest* que conocimos en una sesión previa, **Aviso**: este proceso puede tardar un cierto tiempo, al contar con varias miles de muestras que han de usarse para entrenar cientos de modelos distintos según el tipo de ensemble:

```
# Configuramos la selección del modelo durante entrenamiento
train10CV <- trainControl(method = "cv", number = 10, classProbs = TRUE)

# Bagging
bag <- train(type ~ ., data = training, method = "treebag", trControl = train10CV)
# Boosting
gbm <- train(type ~ ., data = training, method = "gbm", trControl = train10CV, verbose = FALSE)
# Random forest
rf <- train(type ~ ., data = training, method = "rf", trControl = train10CV)
```

Examinamos algunos aspectos del entrenamiento

Obtenemos un resumen de los modelos entrenados y una visualización de algunos detalles. Comenzamos con el *bagging*:

```
bag
```

```
## Bagged CART
##
## 3222 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2900, 2900, 2900, 2900, 2900, 2899, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9363796  0.8661402
```

La estimación nos dice que alcanza casi un 93% de acierto. Veamos en el caso de *random forest*:

```
rf
```

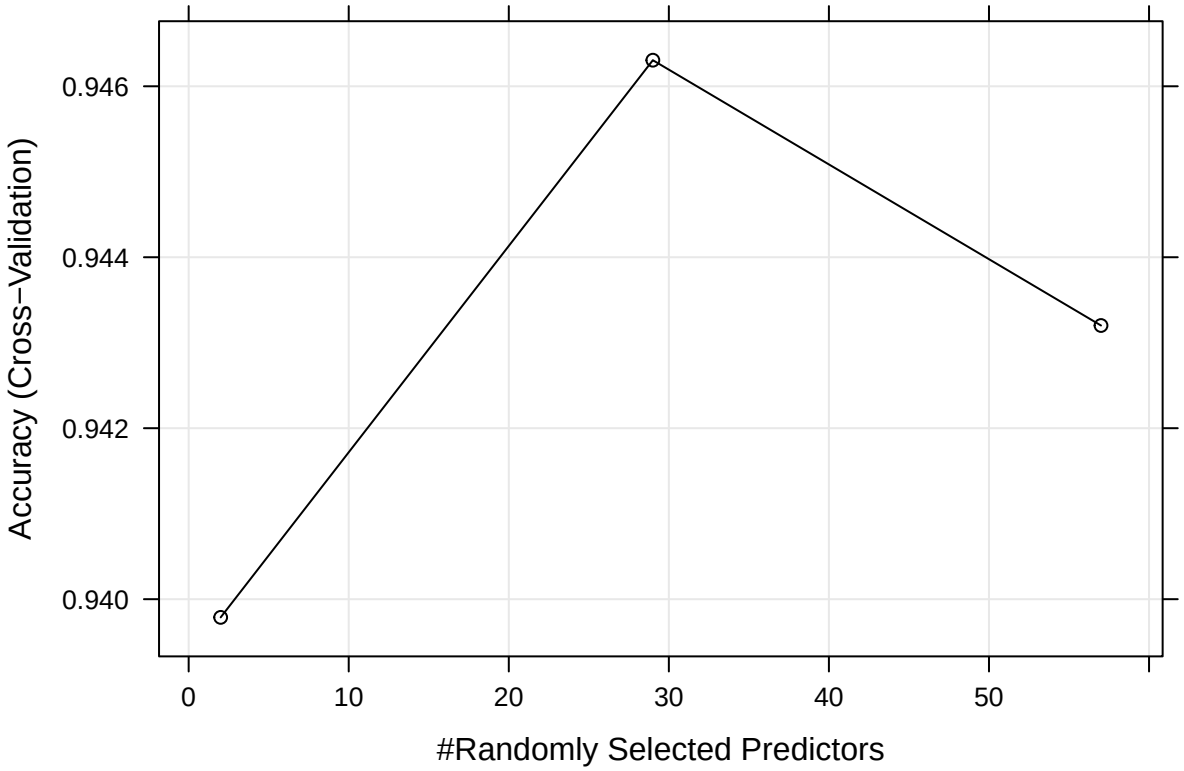
```
## Random Forest
##
## 3222 samples
```

```
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2900, 2899, 2900, 2900, 2899, 2900, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9397871  0.8728809
##   29    0.9463050  0.8870540
##   57    0.9432014  0.8806081
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 29.
```

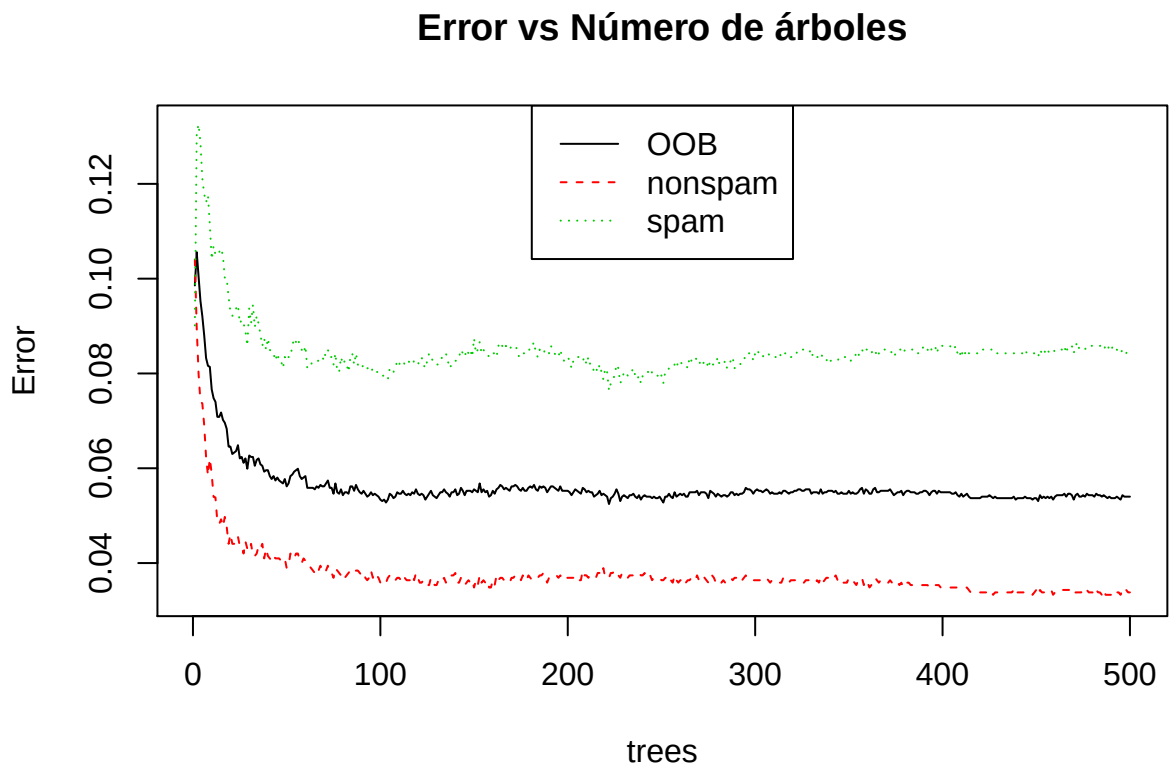
En este caso la ratio de acierto sube a algo más del 94%. Este nivel se alcanza cuando `mtry = 2`, descendiendo para valores superiores. Dicho parámetro determina el número de variables que de forma aleatoria se prueba como candidatas para ir dividiendo los árboles.

Este modelo nos permite, además, observar detalles como el cambio en rendimiento según el número de predictores seleccionados (el citado parámetro `mtry`) o el número de árboles empleados en el *ensemble*:

```
plot(rf)
```




```
leyenda <- attr(rf$finalModel$err.rate, "dimnames")[[2]]
plot(rf$finalModel, main = "Error vs Número de árboles")
legend("top", legend = leyenda, lty = 1:3, col = 1:3)
```



Examinemos ahora la estimación de rendimiento para el *boosting*:

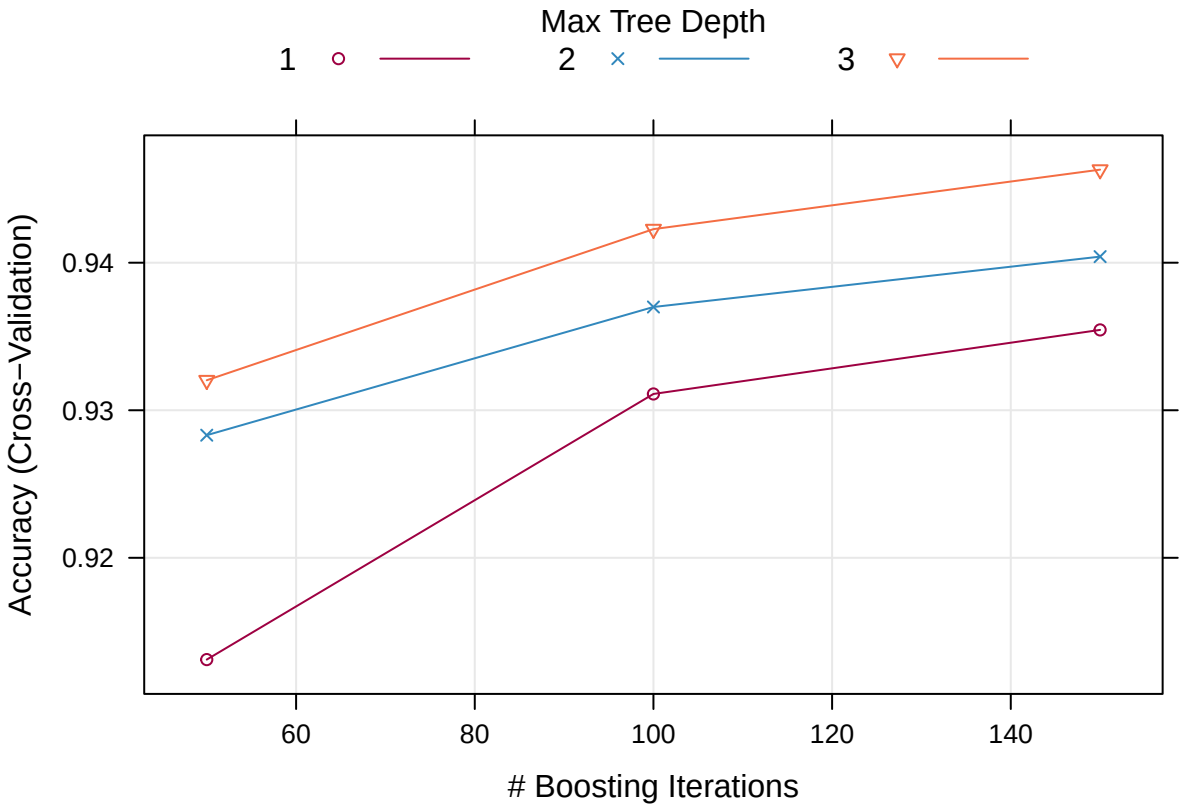
```
gbm
```

```
## Stochastic Gradient Boosting
##
## 3222 samples
##   57 predictor
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2900, 2900, 2899, 2900, 2900, 2899, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9131002  0.8143042
##   1                  100      0.9311040  0.8538996
##   1                  150      0.9354460  0.8634635
##   2                   50      0.9283099  0.8482462
##   2                  100      0.9369969  0.8670610
##   2                  150      0.9404111  0.8743851
##   3                   50      0.9320337  0.8560773
##   3                  100      0.9422754  0.8781721
##   3                  150      0.9463079  0.8868857
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
```

```
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

Como puede apreciarse, en este caso se han ido ajustando esencialmente dos parámetros: el número de árboles empleados y la profundidad de estos, llegando a alcanzarse algo más de un 94% de acierto, ligeramente por encima del *random forest*. Para verificar la influencia de los dos parámetros citados no tneemos para que reprsentar gráficamente el modelo:

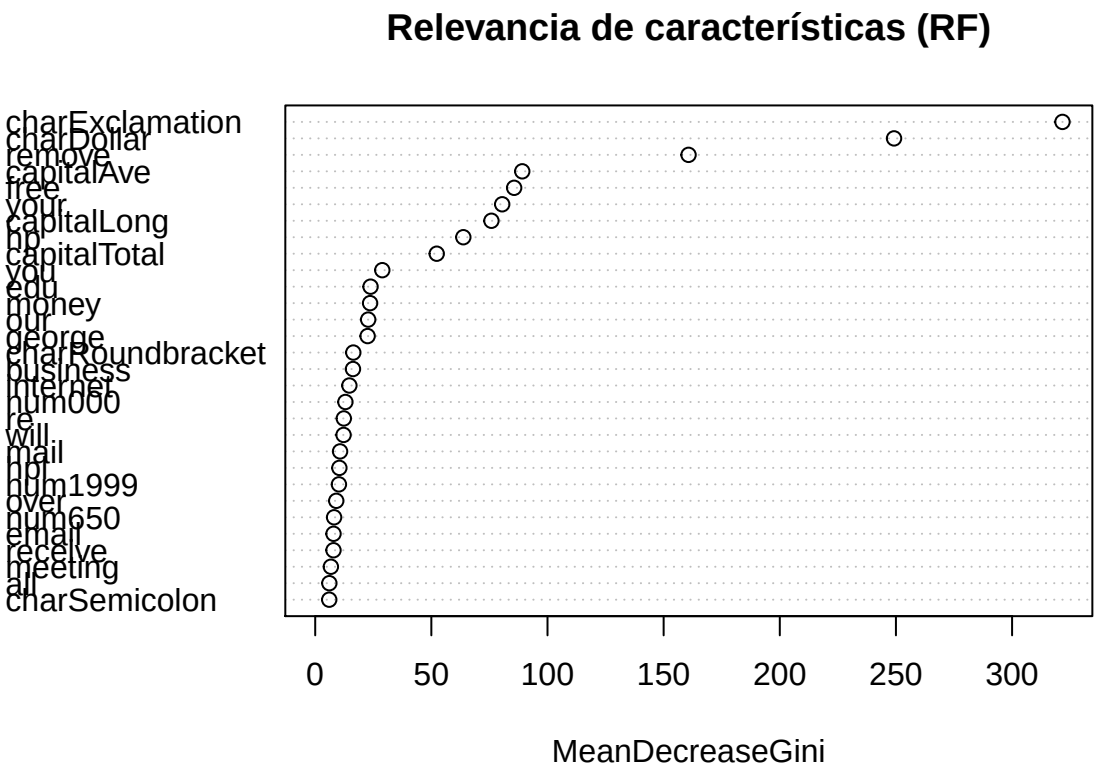
```
plot(gbm)  # Explorar la influencia de parámetros en el modelo
```



Importancia de las variables según RF

Un aspecto interesante de *random forest* es que nos permite recuperar un ranking de importancia de las características (las variables), lo cual aporta cierto nivel de interpretabilidad a un modelo que está compuesto por cientos de árboles y que, en consecuencia, generaría una base de reglas inmesa. La función `varImpPlot()`, tomando como parámetro el mejor modelo producido anteriormente, nos ofrece este resultado:

```
varImpPlot(rf$finalModel,
           main = "Relevancia de características (RF)")
```



Como podemos comprobar, la aparición en un correo electrónico de signos de exclamación, dólar y la palabra *remove* son los que más influencia tienen a la hora de determinar si es legítimo o basura.

Evaluación de los modelos con datos de test

Hasta ahora, en los apartados previos, hemos comprobado la estimación que el propio proceso de ajuste de parámetros ha llevado a cabo en cada *ensemble*. Dado que tenemos desde el inicio una partición con datos de test, veamos cuál es el rendimiento de cada modelo con datos que nunca ha visto antes:

Bagging

```
confusionMatrix(predict(bag, test), test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     799   39
##    spam         37  504
##
##                Accuracy : 0.9449
##                  95% CI : (0.9315, 0.9563)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8845
```

```
##
##  Mcnemar's Test P-Value : 0.9087
##
##             Sensitivity : 0.9557
##             Specificity : 0.9282
##          Pos Pred Value : 0.9535
##          Neg Pred Value : 0.9316
##              Prevalence : 0.6062
##          Detection Rate : 0.5794
##    Detection Prevalence : 0.6077
##       Balanced Accuracy : 0.9420
##
##        'Positive' Class : nonspam
##
```

Random Forest

```
confusionMatrix(predict(rf, test), test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction nonspam spam
##    nonspam     799   38
##    spam         37  505
##
##                Accuracy : 0.9456
##                  95% CI : (0.9323, 0.957)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.886
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9557
##             Specificity : 0.9300
##          Pos Pred Value : 0.9546
##          Neg Pred Value : 0.9317
##              Prevalence : 0.6062
##          Detection Rate : 0.5794
##    Detection Prevalence : 0.6070
##       Balanced Accuracy : 0.9429
##
##        'Positive' Class : nonspam
##
```

Boosting

```
confusionMatrix(predict(gbm, test), test$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction nonspam spam
##    nonspam     805   42
##    spam         31  501
##
##                Accuracy : 0.9471
##                  95% CI : (0.9339, 0.9583)
##     No Information Rate : 0.6062
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8887
##
##  Mcnemar's Test P-Value : 0.2418
##
##             Sensitivity : 0.9629
##             Specificity : 0.9227
##          Pos Pred Value : 0.9504
##          Neg Pred Value : 0.9417
##              Prevalence : 0.6062
##          Detection Rate : 0.5838
##    Detection Prevalence : 0.6142
##       Balanced Accuracy : 0.9428
##
##        'Positive' Class : nonspam
##
```

De acuerdo a datos de test el *ensemble* que mejor rendimiento alcanza es *random forest*, con más un de un 98% de acierto, seguido del *bagging* y del *boosting*. Esto nos demuestra que la estimación llevada a cabo mediante validación interna, a pesar de usarse 10fcv y evaluar promedios de múltiples ejecuciones, no implica que el modelo con mejor estimación sea el que luego mejor se comporta con datos reales.

Comparar los modelos entre sí

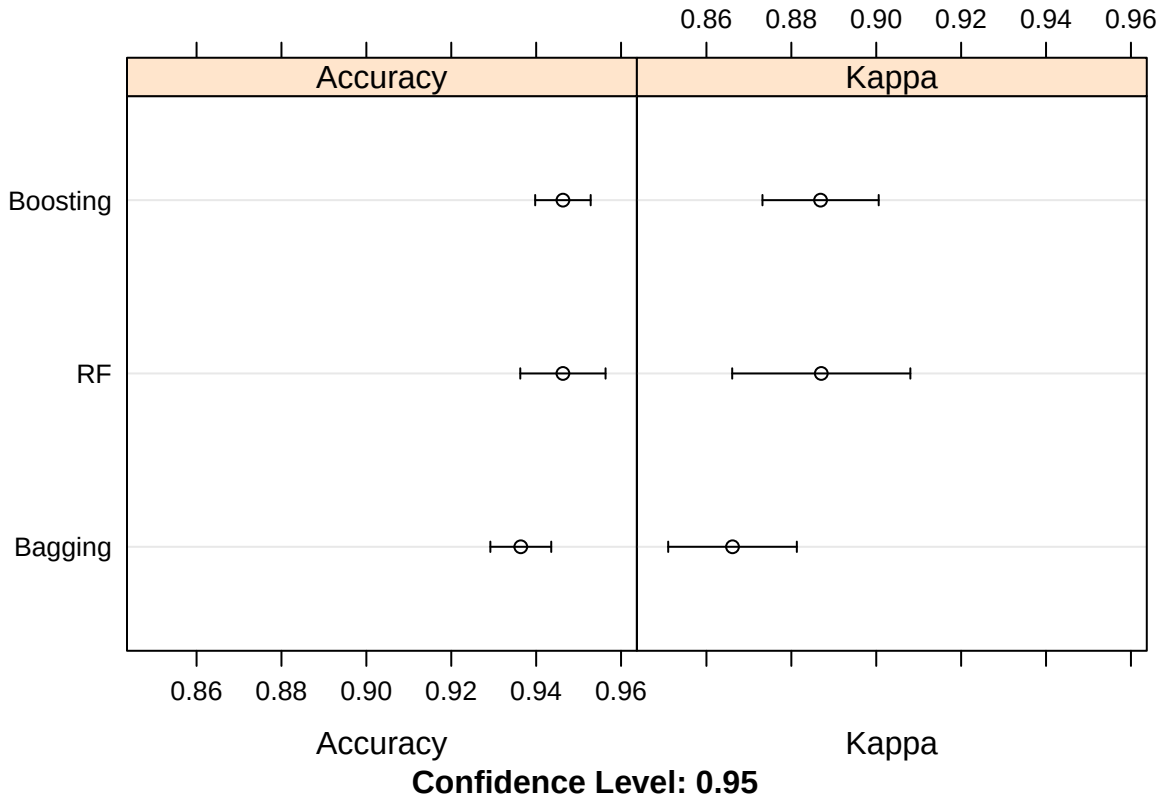
Podemos usar el método `resamples()` para llevar a cabo unan comparación, tanto numérica como gráfica, del rendimiento de los modelos basándonos en los datos de la estimación interna:

```
diff <- resamples(list(Boosting = gbm, Bagging = bag, RF = rf))
summary(diff)
```

```
##
## Call:
## summary.resamples(object = diff)
##
## Models: Boosting, Bagging, RF
## Number of resamples: 10
##
## Accuracy
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Boosting 0.9285714 0.9419072 0.9456522 0.9463079 0.9495750 0.9596273    0
## Bagging  0.9223602 0.9293478 0.9364364 0.9363796 0.9440994 0.9503106    0
## RF       0.9161491 0.9386646 0.9503106 0.9463050 0.9565217 0.9628483    0
##
## Kappa
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Boosting 0.8506795 0.8769776 0.8853626 0.8868857 0.8940199 0.9151360    0
## Bagging  0.8354389 0.8514225 0.8665910 0.8661402 0.8830588 0.8951183    0
```

```
## RF        0.8237441 0.8722623 0.8955520 0.8870540 0.9082285 0.9210494    0
```

```
dotplot(diff)
```



Fijándonos en la columna `Max.` para la métrica `Accuracy`, en el resumen numérico, podemos observar que *boosting* y *bagging* apenas difieren en este caso, mientras que *random forest* se comporta algo mejor.

Ejercicios adicionales

```
mnist <- read.csv("datos/mnist_test.csv", header = FALSE)
dim(mnist)
```

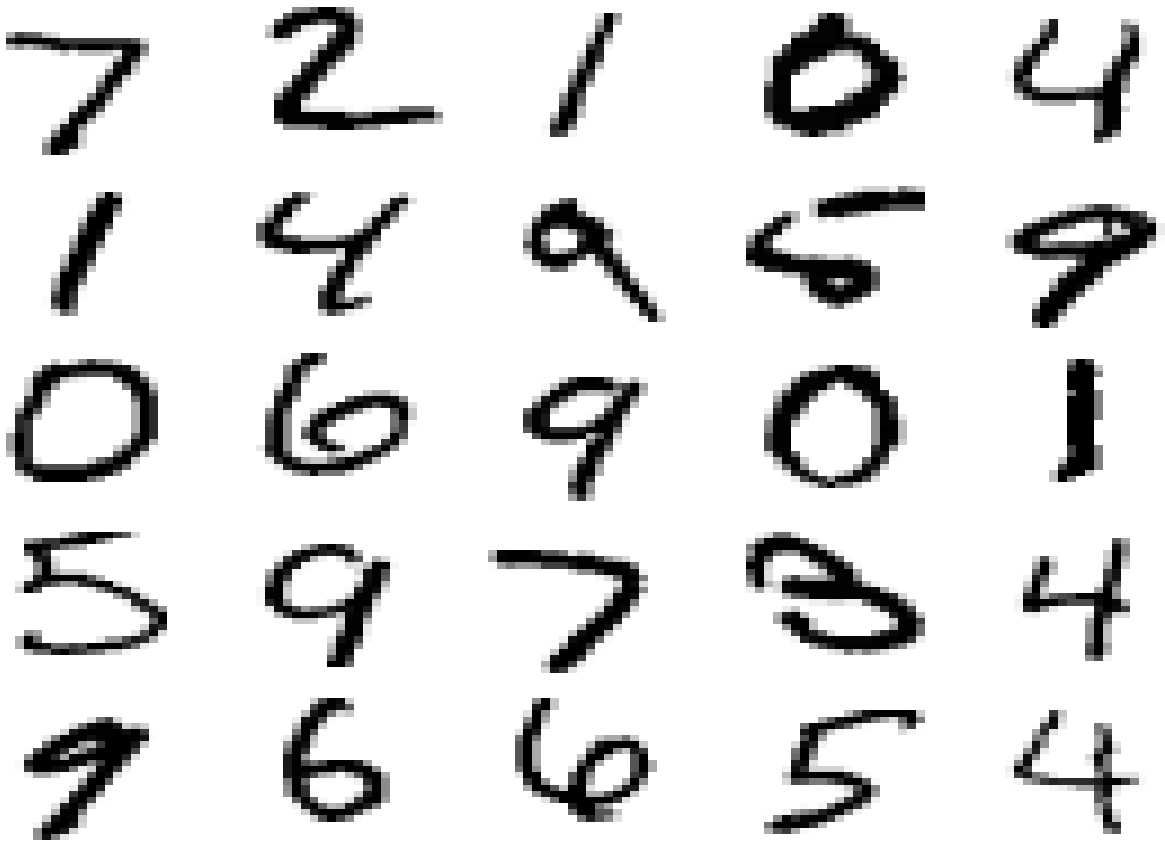
```
## [1] 10000   785
```

```
sample(mnist$V1, 25)
```

```
##  [1] 4 6 6 6 9 0 9 9 6 0 8 8 2 7 0 1 5 8 3 4 2 7 6 2 6
```

Examinamos el contenido de algunas muestras de MNIST

```
prev.conf <- par(mfrow = c(5, 5), mai = c(0,0,0,0))
digits <- lapply(1:25, function(row) {
  digit <- matrix(as.numeric(mnist[row, 2:785]), ncol = 28, byrow = FALSE)
  image(digit[ , 28:1], col = gray(255:0 / 255), axes = FALSE)
})
```



```r
par(prev.conf)
```

Tomamos una pequeña parte de MNIST

```r
set.seed(4242)
datos <- mnist[sample(1:nrow(mnist), 1000), ]
datos[ , 1] <- as.factor(datos[ , 1])
levels(datos$V1) <- paste0("D", 0:9)
table(datos[ , 1]) # Número de muestras por clase
```

```
##
##  D0  D1  D2  D3  D4  D5  D6  D7  D8  D9
## 106 126 105  93 101  79 101 103  99  87
```

Y la usamos para entrenar varios modelos

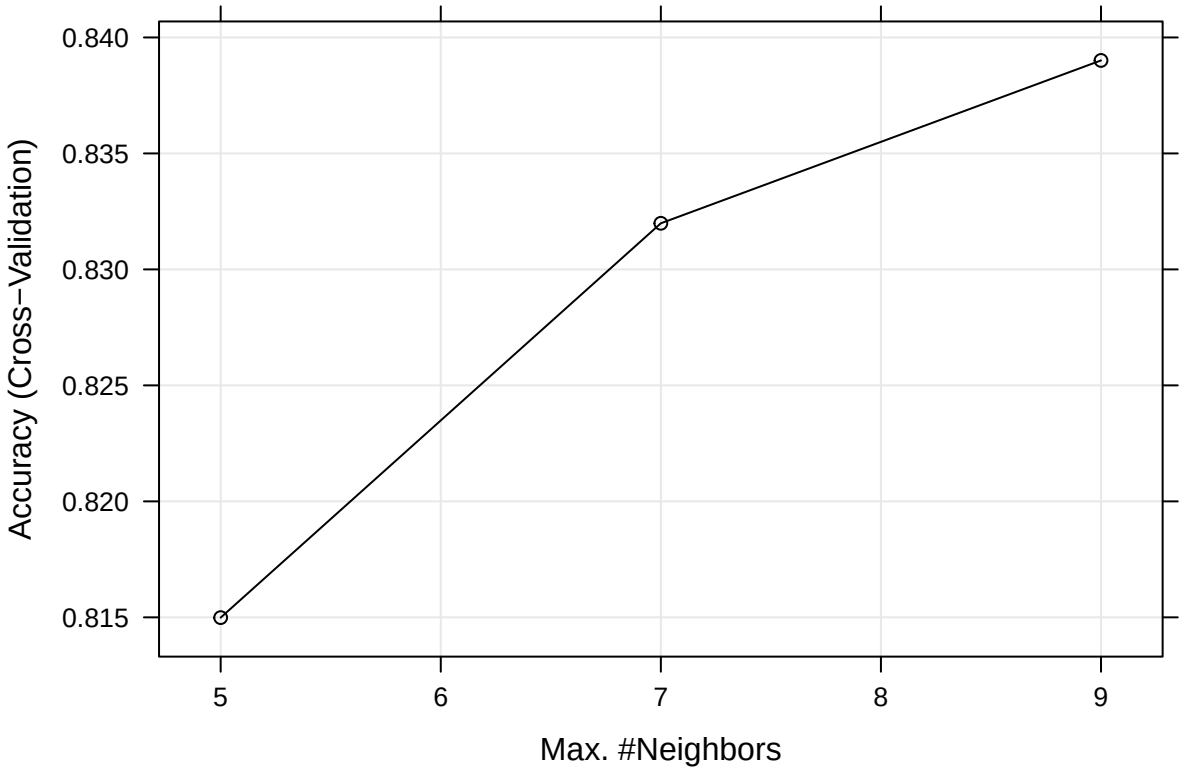
kNN (*k Nearest Neightbors*)

```r
train5fcv <- trainControl(method = "cv", number = 5)

knn.mnist <- train(V1 ~ ., data = datos, method = "kknn", trControl = train5fcv)
knn.mnist
```

```
## k-Nearest Neighbors
##
## 1000 samples
```

```
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 800, 800, 800, 799, 801
## Resampling results across tuning parameters:
##
##   kmax  Accuracy   Kappa
##   5     0.8149882  0.7939894
##   7     0.8319882  0.8128464
##   9     0.8390084  0.8206340
##
## Tuning parameter 'distance' was held constant at a value of 2
## Tuning
##  parameter 'kernel' was held constant at a value of optimal
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were kmax = 9, distance = 2 and kernel
##  = optimal.
```

```
plot(knn.mnist)
```

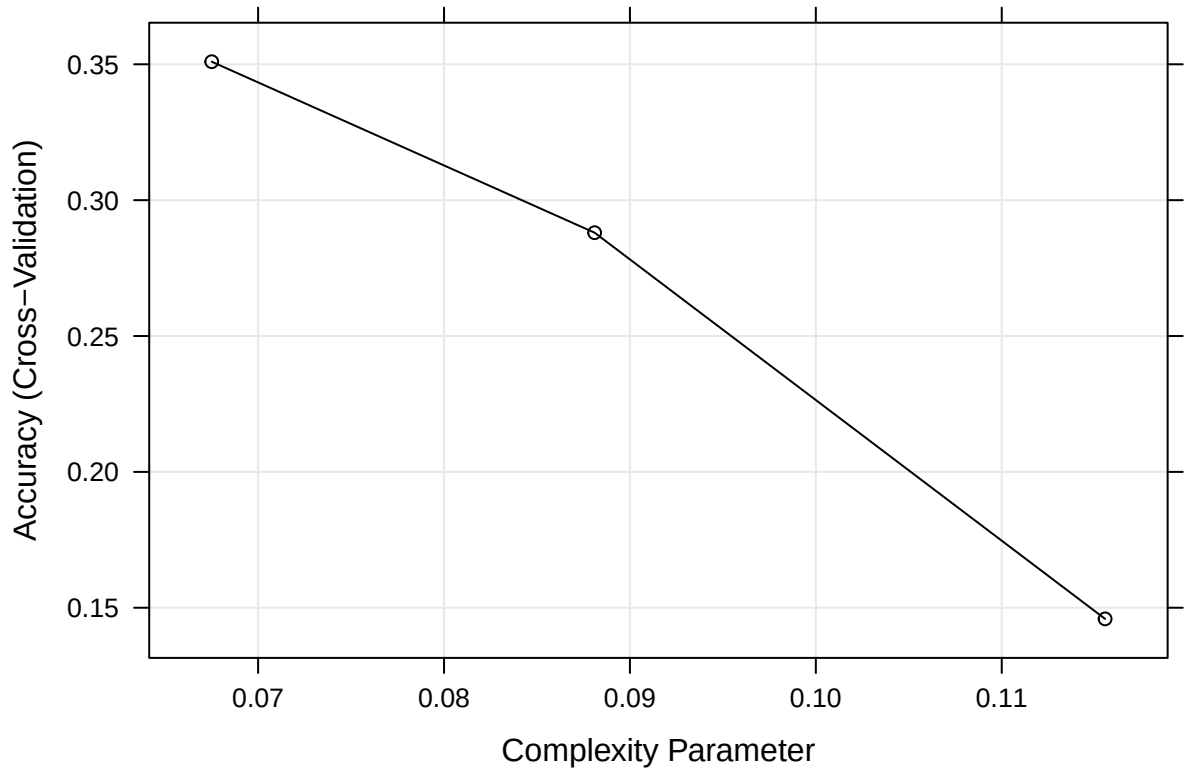


CART (*Classification and Regression Tree*)

```
rpart.mnist <- train(V1 ~ ., data = datos, method = "rpart", trControl = train5fcv)
rpart.mnist
```

```
## CART
##
## 1000 samples
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 801, 802, 799, 799, 799
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06750572  0.3509486  0.27200094
##   0.08810069  0.2880265  0.19897465
##   0.11556064  0.1459005  0.02573067
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06750572.
```

```
plot(rpart.mnist)
```



SVM (*Support Vector Machine*)

```
svm.mnist <- train(V1 ~ ., data = datos, method = "svmRadial", trControl = train5fcv)
```

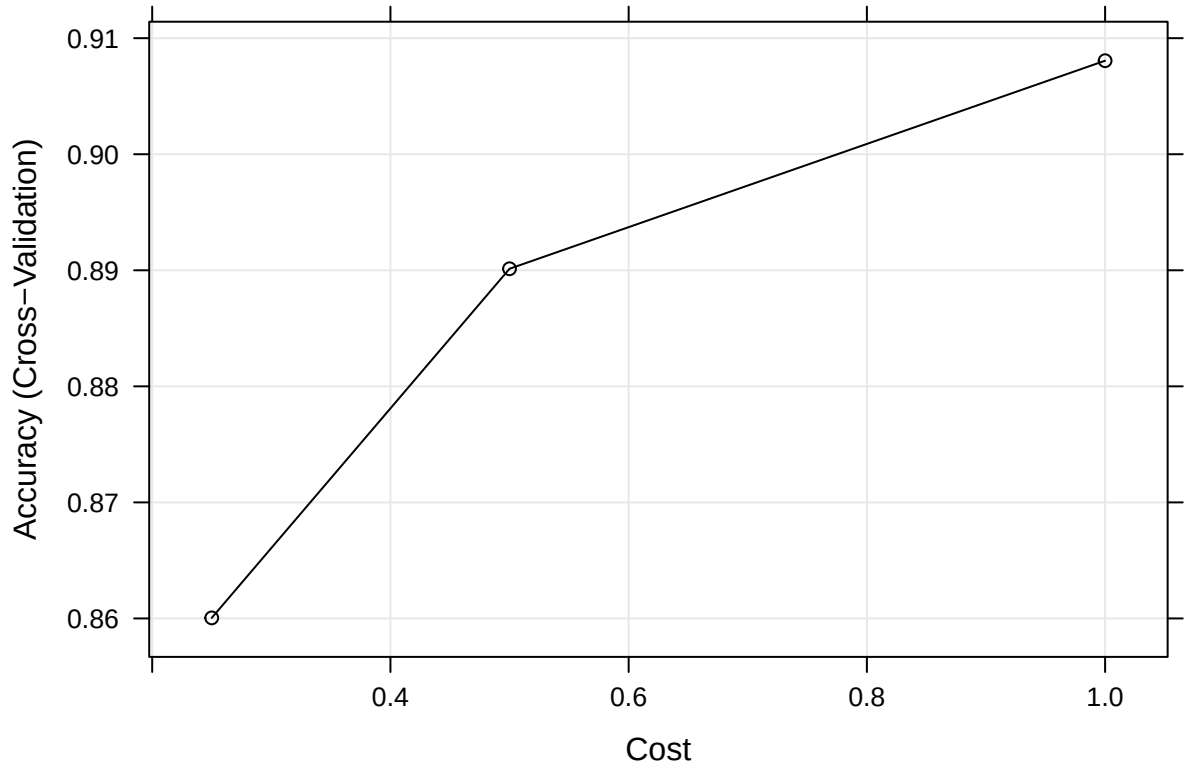
```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svm.mnist
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1000 samples
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 799, 798, 803, 800, 800
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.8600491  0.8440311
##   0.50  0.8901316  0.8776240
##   1.00  0.9080580  0.8976125
##
## Tuning parameter 'sigma' was held constant at a value of 1.654673e-07
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1.654673e-07 and C = 1.
```

```r
plot(svm.mnist)
```



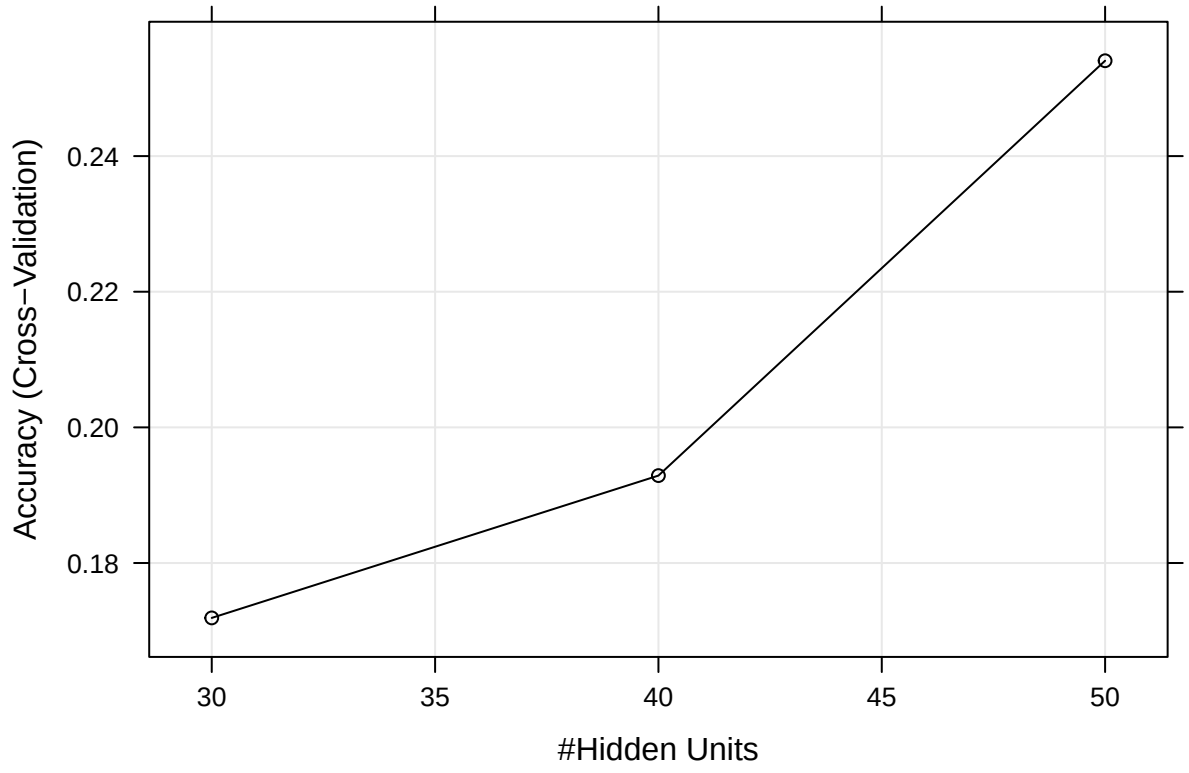
MLP (*MultiLayer Perceptron*)

```r
mlp.mnist <- train(V1 ~ ., data = datos, method = "mlp", trControl = train5fcv,
                   tuneGrid = expand.grid(size = c(30, 40, 50)))

mlp.mnist
```

```
## Multi-Layer Perceptron
##
## 1000 samples
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 799, 800, 799, 801, 801
## Resampling results across tuning parameters:
##
##   size  Accuracy   Kappa
##   30    0.1719033  0.0739299
##   40    0.1928887  0.1036850
##   50    0.2540603  0.1678120
##
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final value used for the model was size = 50.
```

```
plot(mlp.mnist)
```

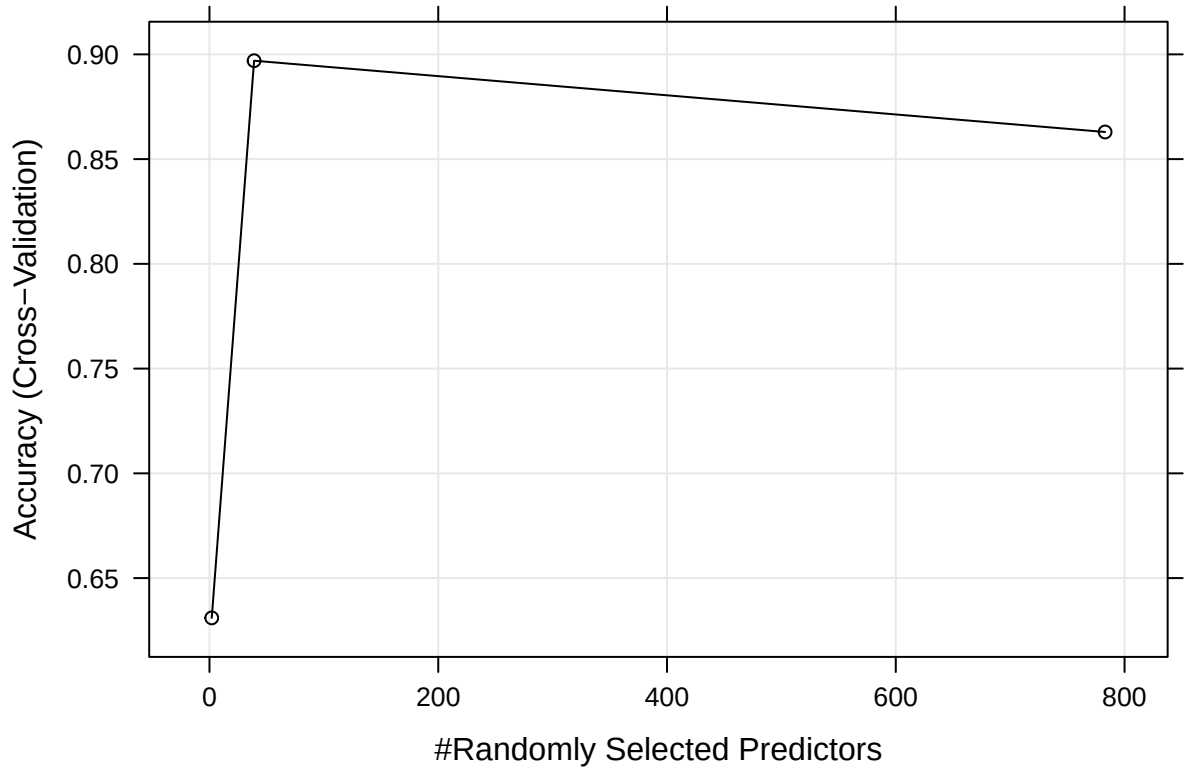


RF (*Random Forest*)

```
rf.mnist <- train(V1 ~ ., data = datos, method = "rf", trControl = train5fcv)
rf.mnist
```

```
## Random Forest
##
## 1000 samples
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 800, 800, 799, 801, 800
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   0.6310115  0.5850468
##    39   0.8969690  0.8852829
##   783   0.8629436  0.8474144
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 39.
```

```r
plot(rf.mnist)
```



XGB (*Extreme Gradiente Boosting*)

```r
xgb.mnist <- train(V1 ~ ., data = datos, method = "xgbTree", trControl = train5fcv)
xgb.mnist
```

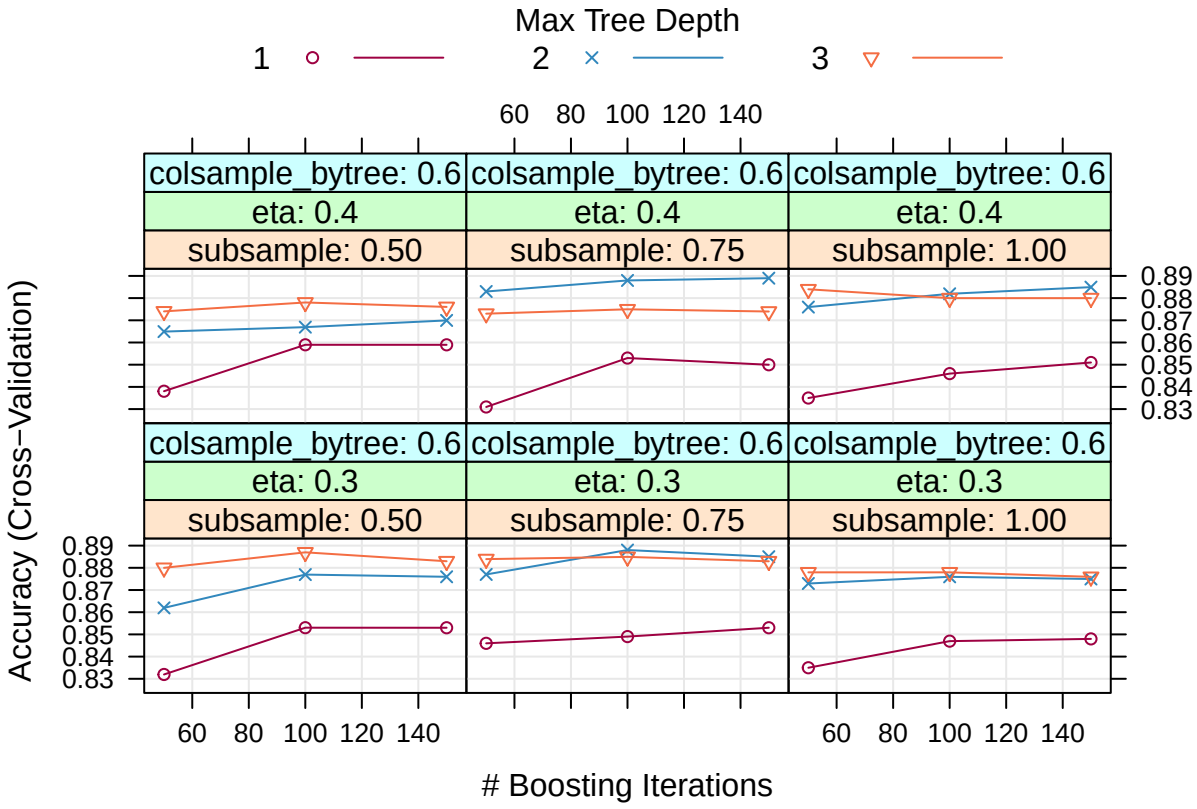
```
## eXtreme Gradient Boosting
##
## 1000 samples
##  784 predictor
##   10 classes: 'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 801, 800, 800, 798, 801
## Resampling results across tuning parameters:
##
##   eta  max_depth  colsample_bytree  subsample  nrounds  Accuracy   Kappa
##   0.3  1          0.6               0.50        50      0.8319552  0.8128064
##   0.3  1          0.6               0.50       100      0.8530155  0.8363480
##   0.3  1          0.6               0.50       150      0.8530005  0.8363170
##   0.3  1          0.6               0.75        50      0.8459855  0.8284822
##   0.3  1          0.6               0.75       100      0.8490153  0.8318548
##   0.3  1          0.6               0.75       150      0.8529954  0.8363077
##   0.3  1          0.6               1.00        50      0.8349803  0.8162062
```

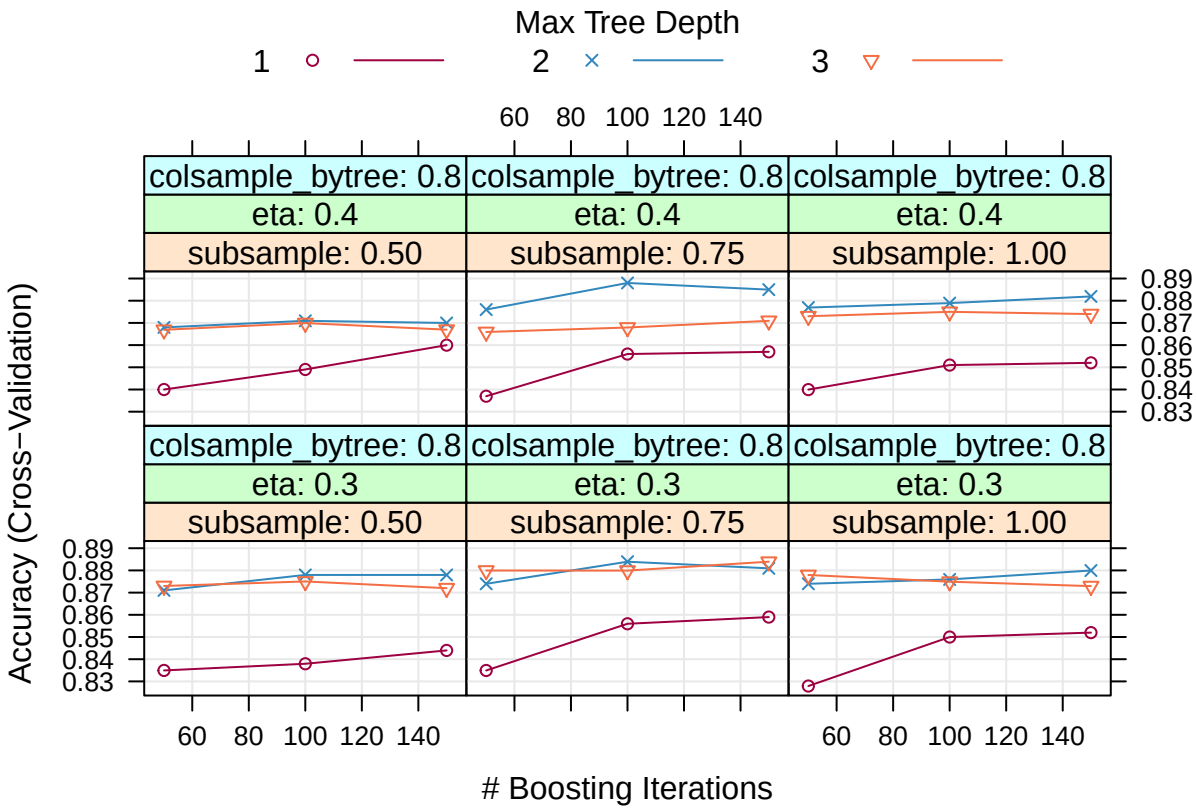
```
##   0.3  1          0.6              1.00        100      0.8469657  0.8295713
##   0.3  1          0.6              1.00        150      0.8479804  0.8307008
##   0.3  1          0.8              0.50         50      0.8349454  0.8161859
##   0.3  1          0.8              0.50        100      0.8379451  0.8195955
##   0.3  1          0.8              0.50        150      0.8439552  0.8262558
##   0.3  1          0.8              0.75         50      0.8349206  0.8161538
##   0.3  1          0.8              0.75        100      0.8559608  0.8396015
##   0.3  1          0.8              0.75        150      0.8589757  0.8429684
##   0.3  1          0.8              1.00         50      0.8279153  0.8083575
##   0.3  1          0.8              1.00        100      0.8499806  0.8329132
##   0.3  1          0.8              1.00        150      0.8519757  0.8351476
##   0.3  2          0.6              0.50         50      0.8619408  0.8462771
##   0.3  2          0.6              0.50        100      0.8769562  0.8629968
##   0.3  2          0.6              0.50        150      0.8759512  0.8618907
##   0.3  2          0.6              0.75         50      0.8770062  0.8630500
##   0.3  2          0.6              0.75        100      0.8880016  0.8752962
##   0.3  2          0.6              0.75        150      0.8850014  0.8719613
##   0.3  2          0.6              1.00         50      0.8729216  0.8585106
##   0.3  2          0.6              1.00        100      0.8759465  0.8618480
##   0.3  2          0.6              1.00        150      0.8749564  0.8607431
##   0.3  2          0.8              0.50         50      0.8709909  0.8563629
##   0.3  2          0.8              0.50        100      0.8779564  0.8641229
##   0.3  2          0.8              0.50        150      0.8779614  0.8641466
##   0.3  2          0.8              0.75         50      0.8739661  0.8596808
##   0.3  2          0.8              0.75        100      0.8839218  0.8707605
##   0.3  2          0.8              0.75        150      0.8809366  0.8674455
##   0.3  2          0.8              1.00         50      0.8739763  0.8596940
##   0.3  2          0.8              1.00        100      0.8759364  0.8618663
##   0.3  2          0.8              1.00        150      0.8799565  0.8663561
##   0.3  3          0.6              0.50         50      0.8799816  0.8663819
##   0.3  3          0.6              0.50        100      0.8869670  0.8741675
##   0.3  3          0.6              0.50        150      0.8829568  0.8697182
##   0.3  3          0.6              0.75         50      0.8839218  0.8707760
##   0.3  3          0.6              0.75        100      0.8849317  0.8718935
##   0.3  3          0.6              0.75        150      0.8829367  0.8696767
##   0.3  3          0.6              1.00         50      0.8779615  0.8641102
##   0.3  3          0.6              1.00        100      0.8779565  0.8641029
##   0.3  3          0.6              1.00        150      0.8759265  0.8618402
##   0.3  3          0.8              0.50         50      0.8729812  0.8585743
##   0.3  3          0.8              0.50        100      0.8750261  0.8608572
##   0.3  3          0.8              0.50        150      0.8719862  0.8574739
##   0.3  3          0.8              0.75         50      0.8799515  0.8663383
##   0.3  3          0.8              0.75        100      0.8799166  0.8662754
##   0.3  3          0.8              0.75        150      0.8839317  0.8707606
##   0.3  3          0.8              1.00         50      0.8779466  0.8640743
##   0.3  3          0.8              1.00        100      0.8749215  0.8607080
##   0.3  3          0.8              1.00        150      0.8729314  0.8585008
##   0.4  1          0.6              0.50         50      0.8380499  0.8196775
##   0.4  1          0.6              0.50        100      0.8589657  0.8429245
##   0.4  1          0.6              0.50        150      0.8589608  0.8429138
##   0.4  1          0.6              0.75         50      0.8309501  0.8117747
##   0.4  1          0.6              0.75        100      0.8529955  0.8362874
##   0.4  1          0.6              0.75        150      0.8499655  0.8329393
##   0.4  1          0.6              1.00         50      0.8349850  0.8162401
```

```
##   0.4  1          0.6              1.00       100      0.8459853  0.8284907
##   0.4  1          0.6              1.00       150      0.8509954  0.8340659
##   0.4  1          0.8              0.50        50      0.8399899  0.8218551
##   0.4  1          0.8              0.50       100      0.8490251  0.8319214
##   0.4  1          0.8              0.50       150      0.8599806  0.8441200
##   0.4  1          0.8              0.75        50      0.8369701  0.8184139
##   0.4  1          0.8              0.75       100      0.8559705  0.8395956
##   0.4  1          0.8              0.75       150      0.8569708  0.8407253
##   0.4  1          0.8              1.00        50      0.8399506  0.8217522
##   0.4  1          0.8              1.00       100      0.8510005  0.8340836
##   0.4  1          0.8              1.00       150      0.8520104  0.8351970
##   0.4  2          0.6              0.50        50      0.8649160  0.8495851
##   0.4  2          0.6              0.50       100      0.8669210  0.8518104
##   0.4  2          0.6              0.50       150      0.8699462  0.8551825
##   0.4  2          0.6              0.75        50      0.8829667  0.8696956
##   0.4  2          0.6              0.75       100      0.8879568  0.8752637
##   0.4  2          0.6              0.75       150      0.8889719  0.8763828
##   0.4  2          0.6              1.00        50      0.8759516  0.8618923
##   0.4  2          0.6              1.00       100      0.8819318  0.8685469
##   0.4  2          0.6              1.00       150      0.8849419  0.8718835
##   0.4  2          0.8              0.50        50      0.8679760  0.8529863
##   0.4  2          0.8              0.50       100      0.8709462  0.8563145
##   0.4  2          0.8              0.50       150      0.8699661  0.8551979
##   0.4  2          0.8              0.75        50      0.8759963  0.8619180
##   0.4  2          0.8              0.75       100      0.8879966  0.8752999
##   0.4  2          0.8              0.75       150      0.8849865  0.8719432
##   0.4  2          0.8              1.00        50      0.8769117  0.8629462
##   0.4  2          0.8              1.00       100      0.8789265  0.8651992
##   0.4  2          0.8              1.00       150      0.8819267  0.8685348
##   0.4  3          0.6              0.50        50      0.8740209  0.8597284
##   0.4  3          0.6              0.50       100      0.8780311  0.8642114
##   0.4  3          0.6              0.50       150      0.8760261  0.8619782
##   0.4  3          0.6              0.75        50      0.8729862  0.8586006
##   0.4  3          0.6              0.75       100      0.8749565  0.8607781
##   0.4  3          0.6              0.75       150      0.8739615  0.8596820
##   0.4  3          0.6              1.00        50      0.8839766  0.8708222
##   0.4  3          0.6              1.00       100      0.8799765  0.8663667
##   0.4  3          0.6              1.00       150      0.8799765  0.8663684
##   0.4  3          0.8              0.50        50      0.8669015  0.8518321
##   0.4  3          0.8              0.50       100      0.8699264  0.8552142
##   0.4  3          0.8              0.50       150      0.8669512  0.8519013
##   0.4  3          0.8              0.75        50      0.8659165  0.8507131
##   0.4  3          0.8              0.75       100      0.8679265  0.8529555
##   0.4  3          0.8              0.75       150      0.8709366  0.8563108
##   0.4  3          0.8              1.00        50      0.8730159  0.8586404
##   0.4  3          0.8              1.00       100      0.8749861  0.8608143
##   0.4  3          0.8              1.00       150      0.8739810  0.8596959
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
##  parameter 'min_child_weight' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 150, max_depth = 2, eta
##  = 0.4, gamma = 0, colsample_bytree = 0.6, min_child_weight = 1 and subsample
```

```
##  = 0.75.
```

```
plot(xgb.mnist)
```





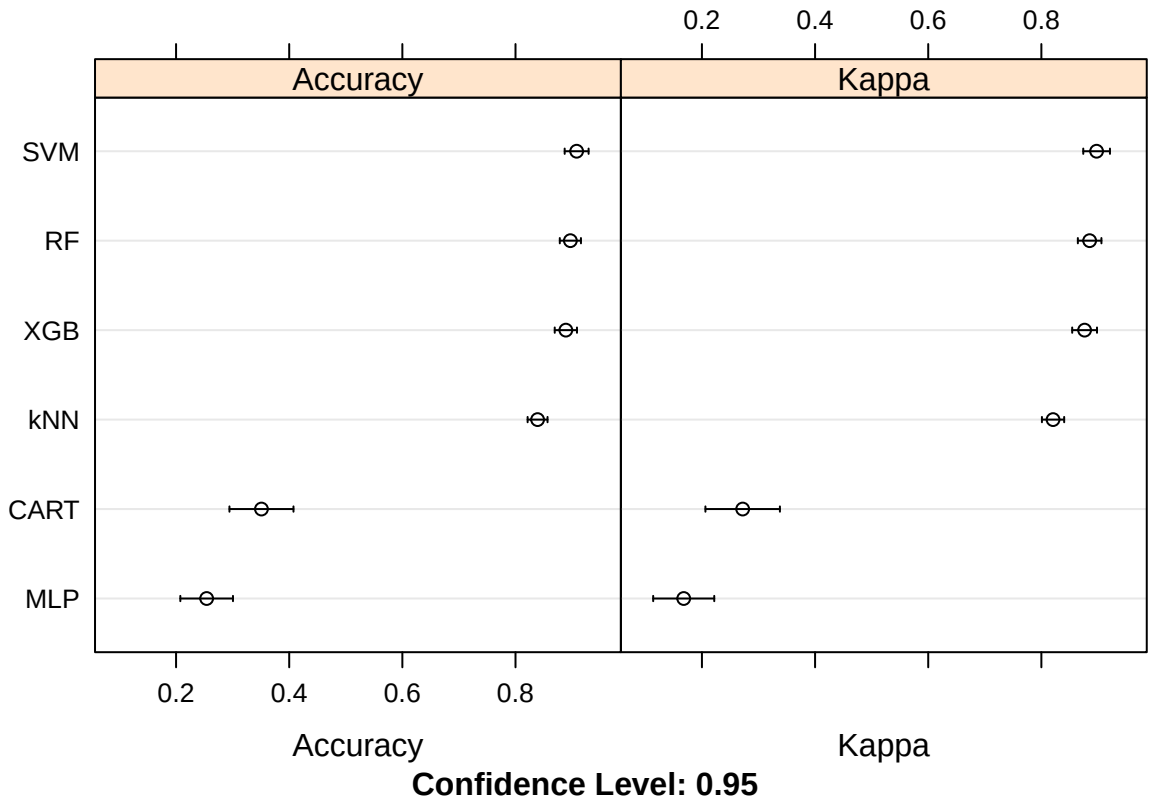
Comparación de los modelos

```
diff <- resamples(list(
  kNN = knn.mnist, CART = rpart.mnist, SVM = svm.mnist,
  MLP = mlp.mnist, RF = rf.mnist, XGB = xgb.mnist))

summary(diff)
```

```
##
## Call:
## summary.resamples(object = diff)
##
## Models: kNN, CART, SVM, MLP, RF, XGB
## Number of resamples: 5
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## kNN  0.8200000 0.8308458 0.8391960 0.8390084 0.8500000 0.8550000    0
## CART 0.3084577 0.3232323 0.3333333 0.3509486 0.3668342 0.4228856    0
## SVM  0.8811881 0.9054726 0.9086294 0.9080580 0.9200000 0.9250000    0
## MLP  0.2050000 0.2238806 0.2713568 0.2540603 0.2786070 0.2914573    0
## RF   0.8800000 0.8894472 0.8950000 0.8969690 0.9000000 0.9203980    0
## XGB  0.8693467 0.8750000 0.8950000 0.8889719 0.9009901 0.9045226    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
```

```
## kNN  0.7995043 0.8114238 0.8208216 0.8206340 0.8328272 0.8385930    0
## CART 0.2215814 0.2434991 0.2484304 0.2720009 0.2909739 0.3555199    0
## SVM  0.8675952 0.8947470 0.8982350 0.8976125 0.9109578 0.9165275    0
## MLP  0.1114837 0.1313165 0.1920762 0.1678120 0.1944444 0.2097392    0
## RF   0.8663957 0.8769464 0.8829888 0.8852829 0.8886786 0.9114050    0
## XGB  0.8544626 0.8608250 0.8831321 0.8763828 0.8897560 0.8937384    0
```

```
dotplot(diff)
```



Recursos de utilidad

- Libro Análisis exploratorio y visualización de datos con R del que soy autor y descargable desde mi página web
- En la sección Tutoriales de mi página personal se ofrecen múltiples tutoriales sobre cómo realizar diversas tareas con R, por ejemplo:
  - Redes neuronales con R
  - Intercambio de datos entre Excel y R
  - Análisis de rendimiento de tareas en R
- En mi perfil de Github hay distintos repositorios con contenidos de utilidad, desde paquetes R para acometer distintas tareas hasta el material que he creado para distintos cursos impartidos sobre R
- Data Carpentry Data Analysis and Visualization in R, con ejercicios paso a paso sobre análisis de datos y visualización
- El libro R for Data Science es una referencia gratuita (disponible en papel en Amazon) e imprescindible, escrito por el gurú de R Hadley Wickham
- En MLR Playground, una página web con R como motor, se pueden comparar las fronteras y resultados de distintos clasificadores aplicados a diversos conjuntos de datos