KNN: Bias-Variance trade-off

Victor Gallego y Roi Naveiro

09/04/2019

```
library(class)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

En este ejercicio, entrenaremos un clasificador KNN para aprender a distinguir imágenes del dígito "8" de otras del dígito "9". Para ello, vamos a usar las proyecciones a 2D que nos daba el análisis de componentes principales.

Funciones auxiliares

- show_digit: Hace una gráfica del dígito en cuestión.
- load_image_file: Para cargar las imágenes de los dígitos
- load_label_file: Para cargar las etiquetas

```
show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784[-785]), nrow = 28)[, 28:1], col = col, ...)
}

load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow * ncol, byrow = TRUE))
}

load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  y = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
```

```
  close(f)
  y
}
```

Lectura de Datos

Cargamos el dataset MNIST.

```
df = load_image_file("src/t10k-images.idx3-ubyte")
df$y  = as.factor(load_label_file("src/t10k-labels.idx1-ubyte"))
```

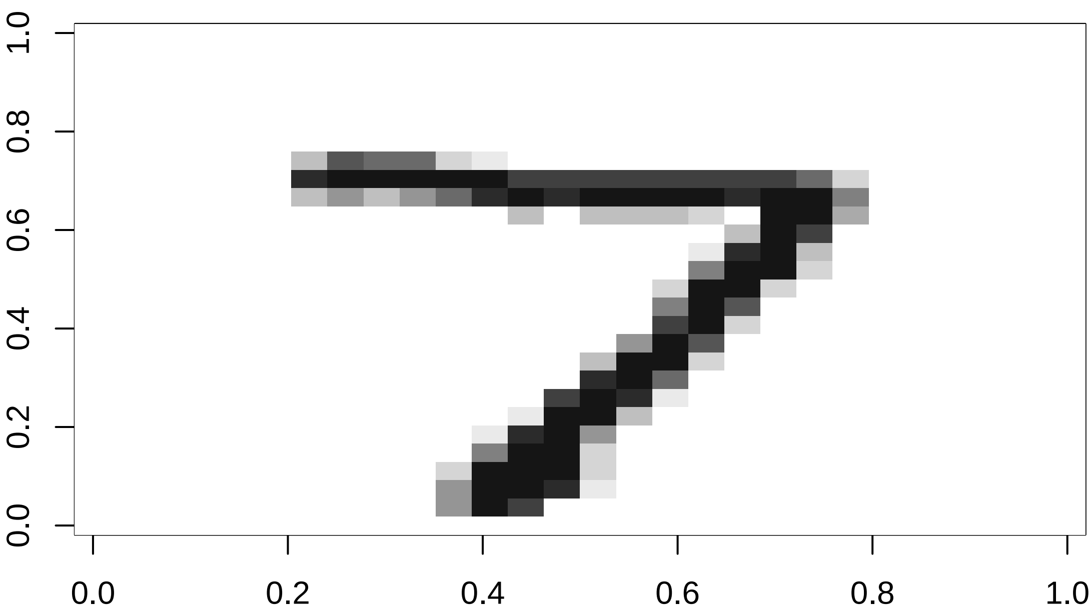
Esta base de datos consta de 10000 imágenes en escala de gris a 28 x 28, de los dígitos del 0 al 9 (escritos a mano).

```
dim(df)
```

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

Visualizamos algún ejemplo

```
show_digit(df[1, ])
```



Selecciona únicamente las imágens de los dígitos 8 y 9.

```
df2 = df[df$y == '8' | df$y == '9',]
```

Creación de conjuntos de train, test y validación.

Divide los datos en train y test, utilizando porcentajes 70, 30; respectivamente.

```
size_train = floor(0.7 * nrow(df2))
#size_test = floor(0.3 * nrow(df2))
#size_val = floor(0.2 * nrow(df2))
##
ind_train = sample(1:nrow(df2), size=size_train)
```

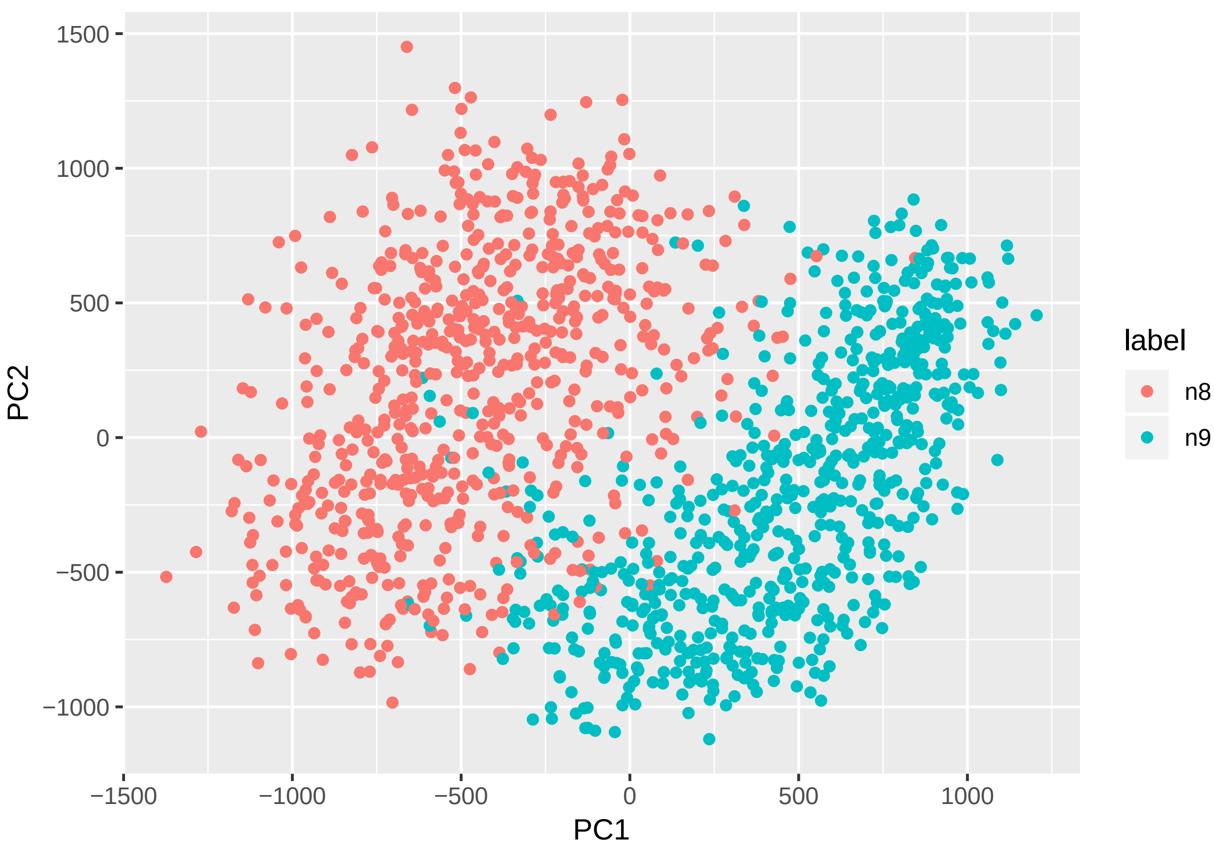
```
train = df2[ind_train,]
test = df2[-ind_train,]

#ind_val = sample(1:nrow(test_val), size=size_val)
#validation = test_val[ind_val,]
#test = test_val[-ind_val,]
```

Proyección a 2D usando PCA

Proyecta los datos de entrenamiento a dos dimensiones usando el paquete prcomp

```
proy_pca <- prcomp(train[, 1:28^2], retx = T) ## Ojo, quitar LABEL, sino son trampas

train_proy = data.frame(proy_pca$x[, 1:2])
train_proy$label = train$y
train_proy$label = factor(train_proy$label)
levels(train_proy$label) = c("n8", "n9")
```

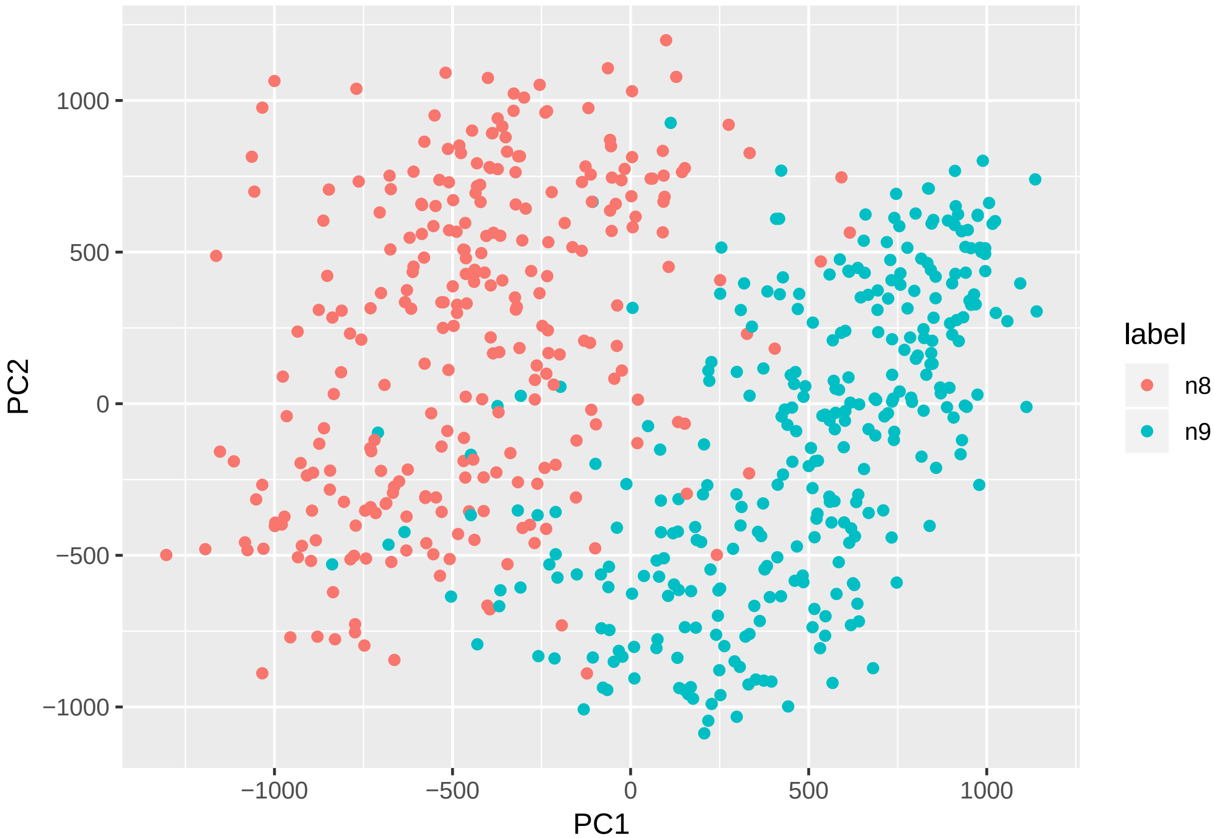
```
p = ggplot(train_proy, aes(x = PC1, y=PC2, colour=label) ) + geom_point()
p
```



Proyecta los datos de test y validación a 2D (OJO, usa las matrices de proyección generadas por el PCA del conjunto de train, de otra manera son trampas. Piensa por qué).

```
test_proy = scale(test[, 1:28^2], proy_pca$center, proy_pca$scale) %*% proy_pca$rotation
test_proy = data.frame(test_proy[, 1:2])
#test_proy = scale(test[, 1:28^2], proy_pca$center, proy_pca$scale) %*% proy_pca$rotation
```

```
#test_proy = data.frame(test_proy)

test_proy$label = test$y
test_proy$label = factor(test_proy$label)
levels(test_proy$label) = c("n8", "n9")
p = ggplot(test_proy, aes(x = PC1, y=PC2, colour=label) ) + geom_point()
p
```



Entrenamiento

Entrena un clasificador KNN usando el paquete caret. Usar validación cruzada con 5 folds y 3 repeticuines para estimar el número optimo de vecinos. Primero definir los controles del training.

```
# Setting up train controls
repeats = 3
numbers = 5
tunel = 100

x = trainControl(method = "repeatedcv",
                 number = numbers,
                 repeats = repeats,
                 classProbs = TRUE,
                 summaryFunction = twoClassSummary)
```

Una vez definidos, entrenar el algoritmo

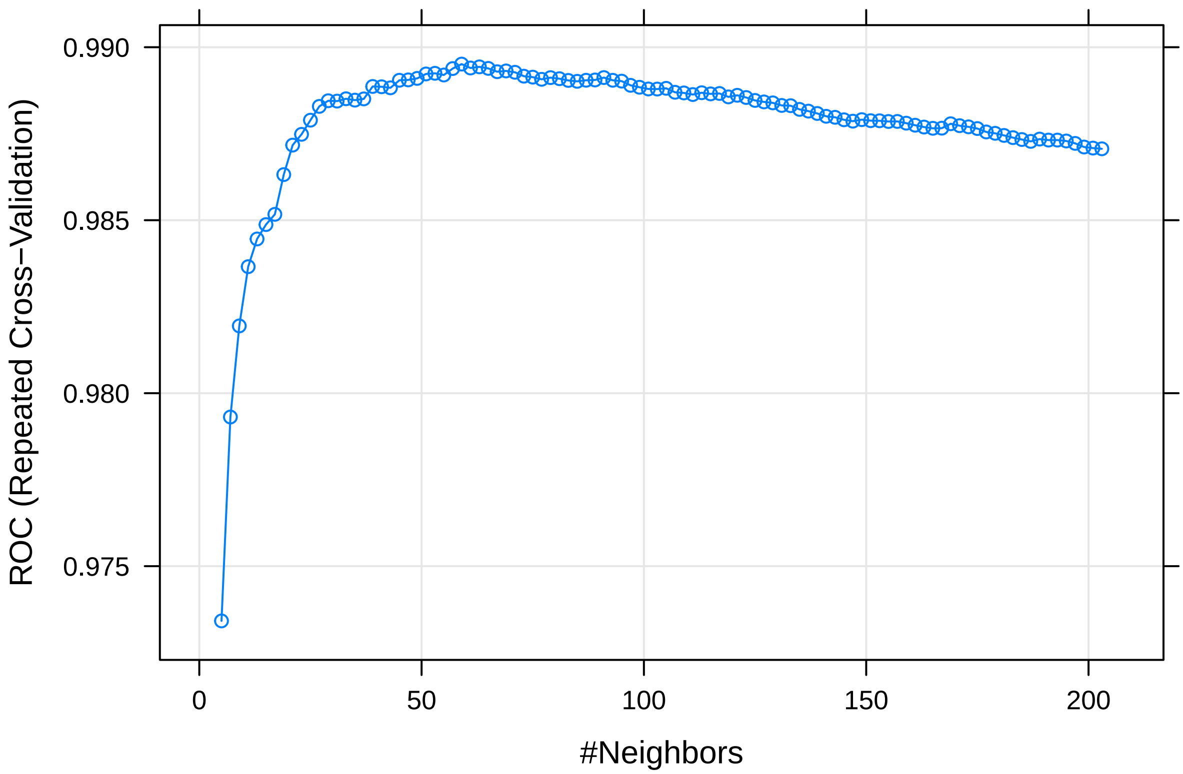
```r
model1 <- train(label ~ ., data = train_proy, method = "knn",
                trControl = x,
                preProcess = c("center", "scale"),
                metric = "ROC",
                tuneLength = tunel)

# Summary of model
model1
```

```
## k-Nearest Neighbors
##
## 1388 samples
##    2 predictors
##    2 classes: 'n8', 'n9'
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1111, 1110, 1109, 1111, 1111, 1111, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    5  0.9734174  0.9444467  0.9470128
##    7  0.9793111  0.9463515  0.9470128
##    9  0.9819440  0.9506749  0.9489487
##   11  0.9836568  0.9502021  0.9470128
##   13  0.9844581  0.9492395  0.9479790
##   15  0.9848738  0.9506817  0.9494213
##   17  0.9851699  0.9497191  0.9484586
##   19  0.9863185  0.9482905  0.9470128
##   21  0.9871700  0.9468482  0.9494213
##   23  0.9874824  0.9463720  0.9494179
##   25  0.9878914  0.9444536  0.9489348
##   27  0.9882906  0.9458890  0.9498940
##   29  0.9884527  0.9449298  0.9503736
##   31  0.9884425  0.9430147  0.9503701
##   33  0.9885103  0.9444536  0.9518125
##   35  0.9884703  0.9449332  0.9527752
##   37  0.9885070  0.9458924  0.9542210
##   39  0.9888660  0.9454162  0.9547006
##   41  0.9888573  0.9458959  0.9537379
##   43  0.9888260  0.9444536  0.9547006
##   45  0.9890455  0.9478040  0.9551802
##   47  0.9890560  0.9449298  0.9561464
##   49  0.9890993  0.9449366  0.9561499
##   51  0.9892281  0.9449332  0.9561464
##   53  0.9892465  0.9420589  0.9556633
##   55  0.9891966  0.9434978  0.9551837
##   57  0.9893831  0.9444570  0.9546971
##   59  0.9895125  0.9444570  0.9537344
##   61  0.9893999  0.9439774  0.9527752
##   63  0.9894326  0.9425420  0.9532583
##   65  0.9893879  0.9435012  0.9542210
##   67  0.9892927  0.9435012  0.9532583
##   69  0.9893150  0.9435012  0.9537379
```

```
##    71  0.9892752  0.9449400  0.9532583
##    73  0.9891629  0.9444604  0.9537414
##    75  0.9891352  0.9449400  0.9542175
##    77  0.9890730  0.9458993  0.9542140
##    79  0.9891216  0.9458993  0.9547006
##    81  0.9890904  0.9449400  0.9547006
##    83  0.9890403  0.9454197  0.9537379
##    85  0.9890144  0.9449400  0.9542175
##    87  0.9890454  0.9444639  0.9537379
##    89  0.9890533  0.9449400  0.9542210
##    91  0.9891228  0.9439808  0.9547041
##    93  0.9890444  0.9439808  0.9547041
##    95  0.9890198  0.9430250  0.9537379
##    97  0.9888986  0.9435046  0.9542175
##    99  0.9888430  0.9435046  0.9542175
##   101  0.9887931  0.9435046  0.9542210
##   103  0.9887912  0.9444639  0.9542210
##   105  0.9888123  0.9439842  0.9547006
##   107  0.9886981  0.9439808  0.9556633
##   109  0.9886791  0.9435012  0.9551837
##   111  0.9886322  0.9444604  0.9566330
##   113  0.9886843  0.9439808  0.9571160
##   115  0.9886514  0.9444604  0.9571160
##   117  0.9886634  0.9449400  0.9561499
##   119  0.9885645  0.9444639  0.9561499
##   121  0.9886108  0.9430250  0.9561499
##   123  0.9885469  0.9444639  0.9561499
##   125  0.9884619  0.9439842  0.9561499
##   127  0.9884197  0.9439842  0.9556668
##   129  0.9883922  0.9439842  0.9561499
##   131  0.9883176  0.9430250  0.9566330
##   133  0.9883105  0.9430250  0.9561499
##   135  0.9882030  0.9425454  0.9561499
##   137  0.9881528  0.9430250  0.9566330
##   139  0.9880852  0.9411065  0.9566330
##   141  0.9880035  0.9406269  0.9566330
##   143  0.9879740  0.9406269  0.9571160
##   145  0.9879028  0.9415862  0.9566330
##   147  0.9878626  0.9411100  0.9566330
##   149  0.9879077  0.9411100  0.9566330
##   151  0.9878763  0.9401507  0.9571160
##   153  0.9878744  0.9396677  0.9566330
##   155  0.9878536  0.9391881  0.9561499
##   157  0.9878550  0.9387119  0.9566330
##   159  0.9878050  0.9387119  0.9566330
##   161  0.9877459  0.9396711  0.9556702
##   163  0.9876924  0.9396711  0.9566364
##   165  0.9876564  0.9401507  0.9571160
##   167  0.9876600  0.9396711  0.9566364
##   169  0.9877883  0.9401507  0.9561533
##   171  0.9877328  0.9401507  0.9561533
##   173  0.9876983  0.9406304  0.9566330
##   175  0.9876497  0.9401507  0.9571126
##   177  0.9875510  0.9406304  0.9566330
```

```
##   179  0.9875112  0.9401507  0.9566330
##   181  0.9874504  0.9396711  0.9566330
##   183  0.9873864  0.9391915  0.9566330
##   185  0.9873325  0.9396711  0.9571126
##   187  0.9872790  0.9401473  0.9566330
##   189  0.9873450  0.9406269  0.9566330
##   191  0.9873155  0.9406269  0.9566330
##   193  0.9873155  0.9411065  0.9566330
##   195  0.9872894  0.9401473  0.9566330
##   197  0.9872199  0.9406269  0.9566330
##   199  0.9871158  0.9411065  0.9566330
##   201  0.9870848  0.9411065  0.9566330
##   203  0.9870639  0.9396677  0.9566330
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 59.
```

```
plot(model1)
```



Representar el número de vecinos frente al valor de AUC. ¿Cuál es el número óptimo de vecinos?

```
model1$bestTune
```

```
##     k
## 28 59
```

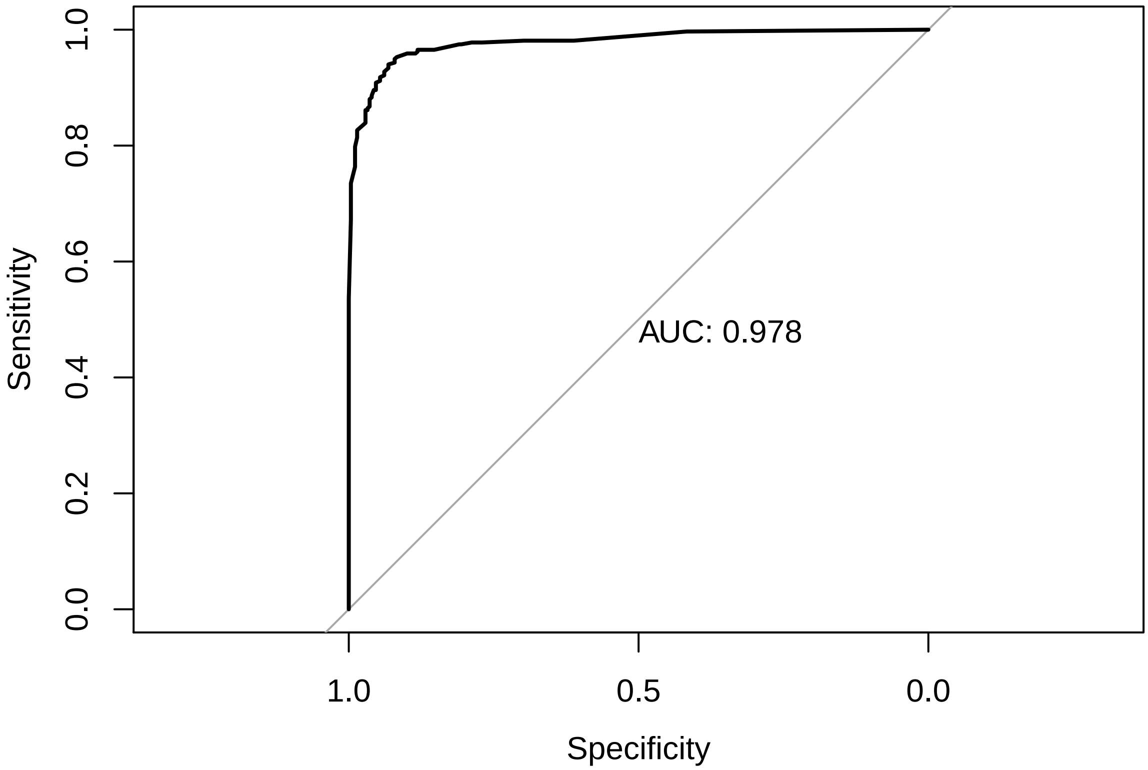
AUC en el conjunto de test

Estima el valor de la AUC en el conjunto de test y pinta la curva ROC

```
preds = predict(model1, newdata = test_proy, type = "prob")
roc_obj = roc(test_proy$label, preds[,1])
auc(roc_obj)
```

```
## Area under the curve: 0.9783
```

```
roc_full_resolution <- roc(test_proy$label, preds[,1])
plot(roc_full_resolution, print.auc=TRUE)
```



Overfitting

Juega con el valor del número de vecinos para entender el comportamiento observado en el gráfico anterior.

```
k = 95
title = paste0(k, "-nearest neighbour")
x <- train_proy[, c("PC1", "PC2")]
g <- train_proy$label
px1 <- seq(min(train_proy$PC1), max(train_proy$PC1), length.out = 20)
px2 <- seq(min(train_proy$PC2), max(train_proy$PC2), length.out = 20)
xnew <- expand.grid(px1, px2)
mod15 <- knn(x, xnew, g, k=k, prob=TRUE)
prob <- attr(mod15, "prob")
prob <- ifelse(mod15=="n8", prob, 1-prob)

prob15 <- matrix(prob, length(px1), length(px2))
par(mar=rep(2,4))
contour(px1, px2, prob15, levels=0.7, labels="", xlab="", ylab="", main=
        title, axes=FALSE)
points(x, col=ifelse(g=="n8", "coral", "cornflowerblue"))
```

```r
gd <- expand.grid(x=px1, y=px2)
points(gd, pch=".", cex=3.0, col=ifelse(prob15>0.5, "coral", "cornflowerblue"))
box()
```