Aprendizaje, Generalización y Sobreajuste

Grupo de Meteorología

Contents

Como hemos visto en la sesión anterior, una de las razones por las cuales se retoman los modelos estadísticos en el entorno del "Machine learning" radica en la sólida base teórica, tanto para el desarrollo de los modelos como para el análisis de su capacidad de generalización y de la incertidumbre asociada a las predicciones desarrolladas por el modelo aprendido. Recordemos algunos conceptos introducidos en la sesión anterior que nos serán útiles para la correcta comprensión y desarrollo de la presente práctica. Durante la práctica usaremos los siguiente paquetes y librerías:

```
install.packages('ISLR') # Data for An Introduction to Statistical Learning with Applications in R
```

```
library(ISLR)
```

Elementos del Aprendizaje: Modelo de aprendizaje

Para que un problema sea adecuado para su resolución mediante técnicas de aprendizaje automático, debe cumplir inicialmente tres condiciones:

- Debe existir un patrón, el cual queremos aprender a través de las técnicas de aprendizaje adecuadas.
- No debe existir solución analítica a nuestro problema o, de existir, debe ser complicada de obtener por medios analíticos.
- Debe existir una base de datos suficiente para poder inferir el patrón de ellos.

De las tres condiciones anteriores, realmente la fundamental es la existencia de datos para alimentar el algoritmo de aprendizaje ya que, en ausencia de las otras dos condiciones las técnicas podrían aplicarse igualmente a pesar de que tuviese más o menos sentido hacerlo.

Partiendo de la definición de aprendizaje introducida -procedimiento (automático) por el cual se construye/ajusta un modelo particular (de la familia de modelos posibles) a partir de un conjunto de datos de entrenamiento que se considera representativo de la distribución que se quiere modelar-, nuestro modelo de aprendizaje viene dado por el algoritmo usado para aprender el patrón y la familia de funciones y datos que utilicemos para aproximar el patrón a aprender. De este modo, el proceso de aprendizaje dependerá de estos tres elementos, los cuales establecerán nuestras limitaciones e incertidumbres a la hora tanto de aprender el patrón como de realizar predicciones con el patrón aprendido.

Generalización y sobreajuste

Recordemos que el objetivo principal del modelo aprendido es que tenga la capacidad de generalizar, es decir, la capacidad de funcionar bien para nuevos datos que no forman parte de la muestra de entrenamiento (por ejemplo, una muestra de datos de test). En caso contario, diremos que el modelo está sobreajustado a la muestra de entrenamiento. Un ejemplo viene dado por la aproximación de una función que pase por `N` puntos. A nivel teórico, dados `N` puntos existe un polinomio de grado `N-1` el cual pasa por los `N` puntos. Por lo tanto, dicho polinomio `P(x)` tendrá error 0 para la muestra dada. Sin embargo, si consideramos otra muestra independiente la probabilidad de que dicha muestra quede ajustada por el polinomio obtenido es 0 ya que sólo existe un caso (que la función que quiero ajustar sea P(x)) favorable de todos los polinomios de grado `N-1` que puedo construir (infinitos). Por lo tanto, aún habiendo obtenido una solución con error 0 para el conjunto de entrenamiento, dicha solución carece de capacidad de generalización.

La introducción de grados de libertad en la familia de funciones consideradas en el aprendizaje suele dar lugar a modelos sobreajustados, por lo que suele ser conveniente partir de los modelos más simples e ir introduciendo grados de libertad progresivamente si fuera necesario.

Teniendo en cuenta que sólo podemos evaluar el error de nuestro modelo aprendido en la muestra de entrenamiento, ¿cómo analizar el sobreajuste del modelo?

Validación cruzada

Para responder a la pregunta planteada en la sección anterior se desarrollaron diferentes técnicas, englobadas básicamente en técnicas de validación cruzada y técnicas de remuestreo o bootstraping. Por el momento nos centraremos en las primeras, dejando la segunda familia para la siguiente sesión práctica. Para ilustrar los métodos utilizaremos el modelo de regresión lineal y el conjunto de datos `Pulsaciones` enlazados en el Moodle de la asignatura. Para la validación definiremos dos funciones de error: el error medio absoluto (mae, mean absolute error) y el error cuadrático medio (mse, mean square error), que pueden cargarse en R ejecutando las siguientes líneas:

```
# Functions
mae<-function(obs,est){
  return(mean(abs(obs-est)))
}

mse <-function(obs,est){
  return(mean((obs-est)^2))
}
```

Este modo de definir funciones es bastante útil en el entorno R.

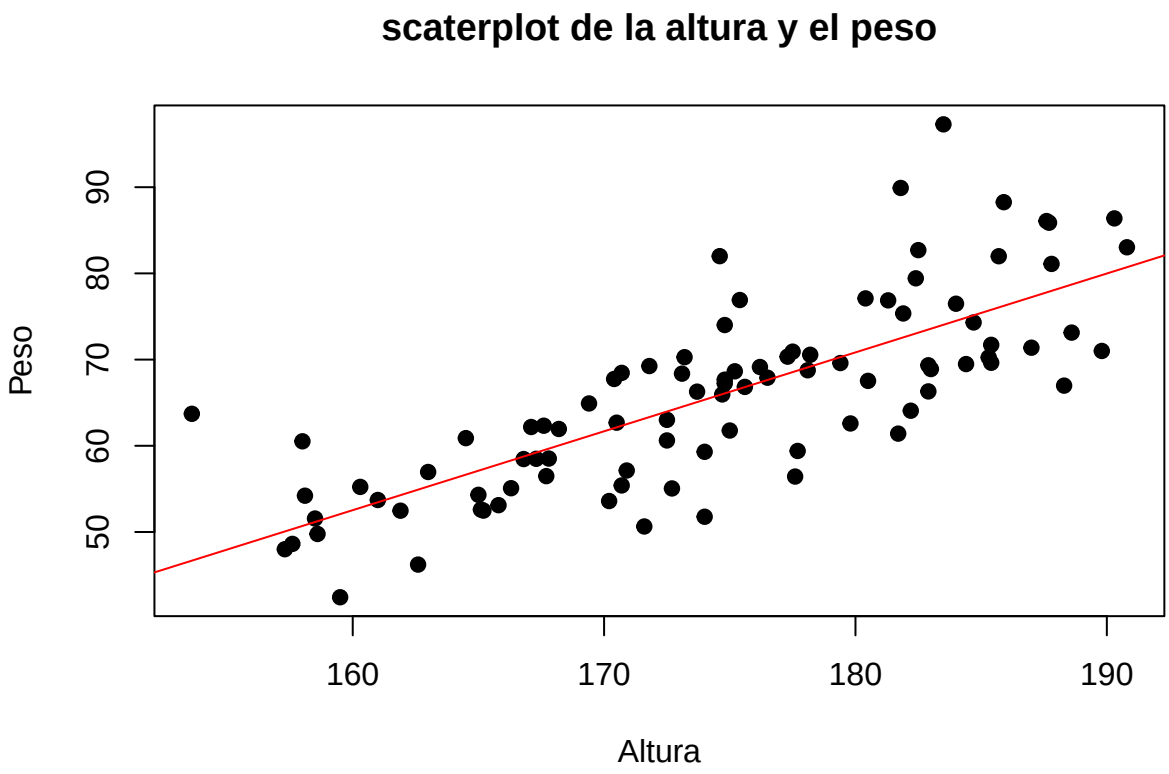
Para ir desarrollando el ejemplo, cargad los datos de vuestro directorio local:

```
load('~/Dropbox/M1966_DataMining/datasets/Pulsaciones.rda')
attach(Pulsaciones)
n <- length(Altura)
```

La validación cruzada se basa en separar la muestra en dos conjuntos disjuntos, uno de entrenamiento y otro de test, que permita analizar tanto el error muestral como la capacidad de generalización de nuestro modelo aprendido en la fase de entrenamiento a través del conjunto de test.

En esta ocasión trabajaremos con datos de `Altura` y `Peso`, los cuales podemos representar en un diagrama de dispersión usando funciones básicas de R.

```
# Usamos las funciones comunes de R
plot(Altura,Peso,main='scaterplot de la altura y el peso', pch=19) #plot(x,y)
# Aniadimos la linea de ajuste
abline(lm(Peso~Altura), col="red") # regression line (y~x)
```

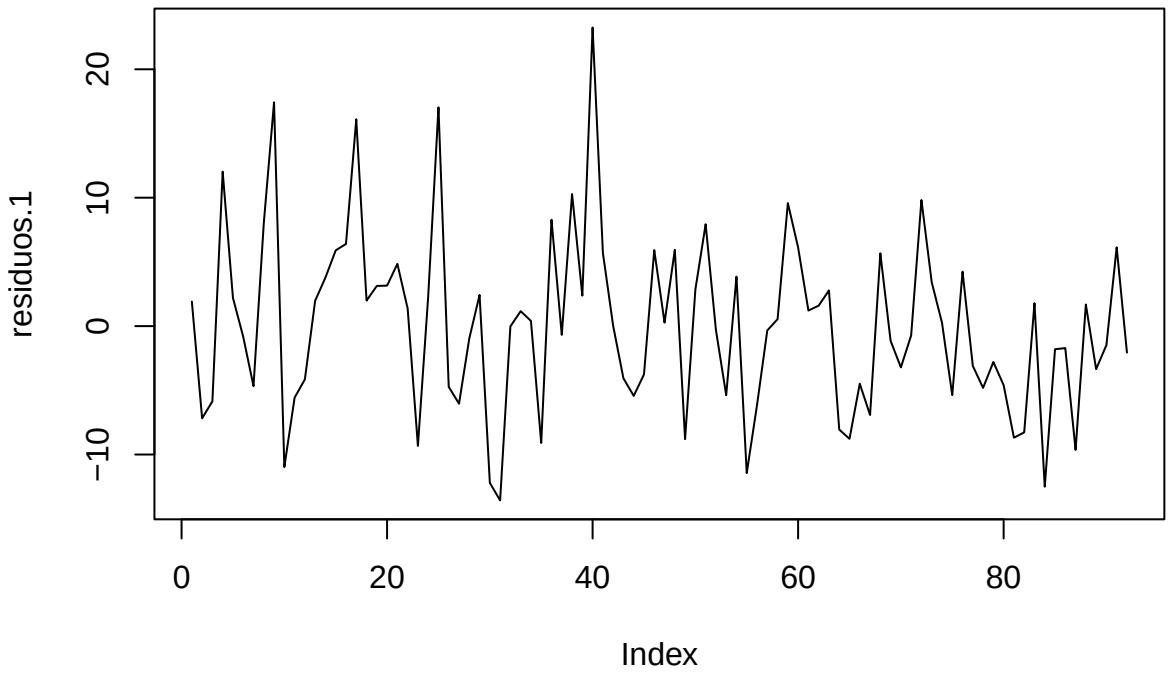


La función `lm` realiza el ajuste de un modelo lineal entre ambas variables, en este caso aplicando una regresión lineal ellas. Podemos explorar los diferentes parámetros que devuelve la función de ajuste

```
# Salida de la regresion
Reg.1<-lm(Peso~Altura) # regression model (y~x)
summary(Reg.1)
```

```
##
## Call:
## lm(formula = Peso ~ Altura)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5727  -4.7486  -0.0001   3.5080  23.2533
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -93.89492   13.88816  -6.761 1.33e-09 ***
## Altura        0.91516    0.07947  11.515  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.981 on 90 degrees of freedom
## Multiple R-squared:  0.5957, Adjusted R-squared:  0.5912
## F-statistic: 132.6 on 1 and 90 DF,  p-value: < 2.2e-16
```

```r
coeficientes.1<-coef(Reg.1)
residuos.1<-residuals(Reg.1)
# Pintamos el residuo
plot(residuos.1,type='l')
```




```r
mean(residuos.1) # Residuos: media cero y varianza constante
```

```
## [1] -2.986977e-17
```

```r
mean(Peso)
```

```
## [1] 65.81217
```

```r
sd(Peso)
```

```
## [1] 10.91776
```

ó aplicar el modelo ajustado a los datos para ver la estimación `yest`, de modo que podamos compararla con la real.

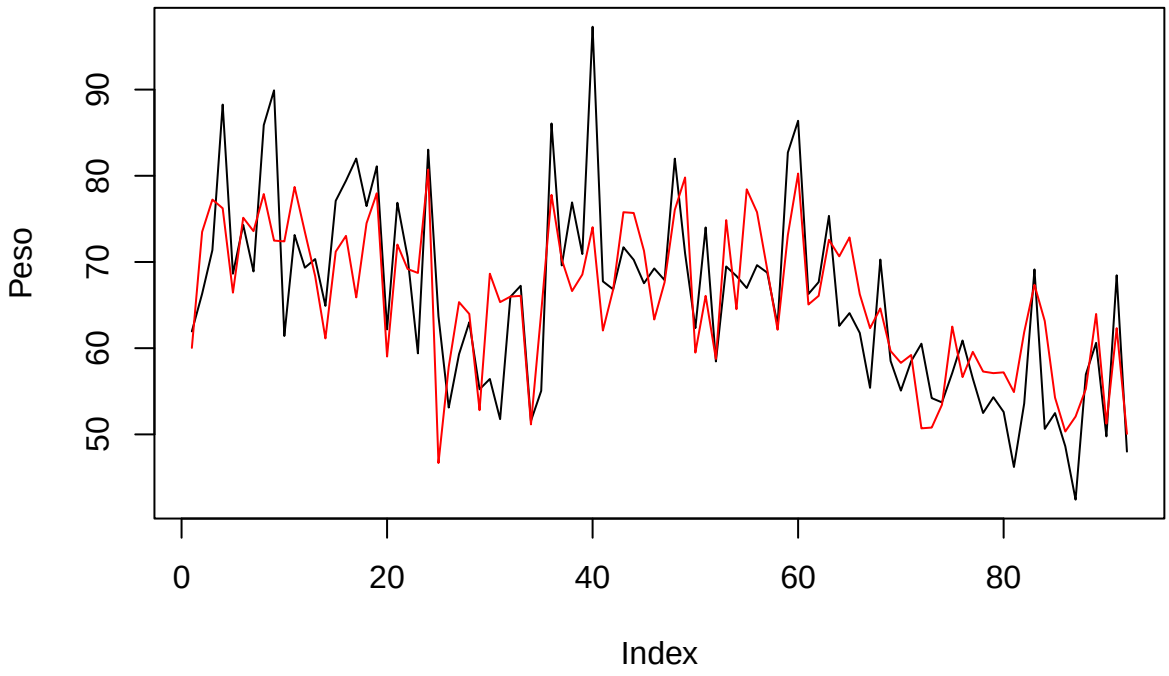
```r
# Comparamos el valor real con el estimado
yest.1<-fitted(Reg.1)
mae.Reg.1<- mae(Peso,yest.1); mae.Reg.1 # Error de validacion
```

```
## [1] 5.327923
```

```r
mse.Reg.1<-mse(Peso,yest.1); mse.Reg.1
```

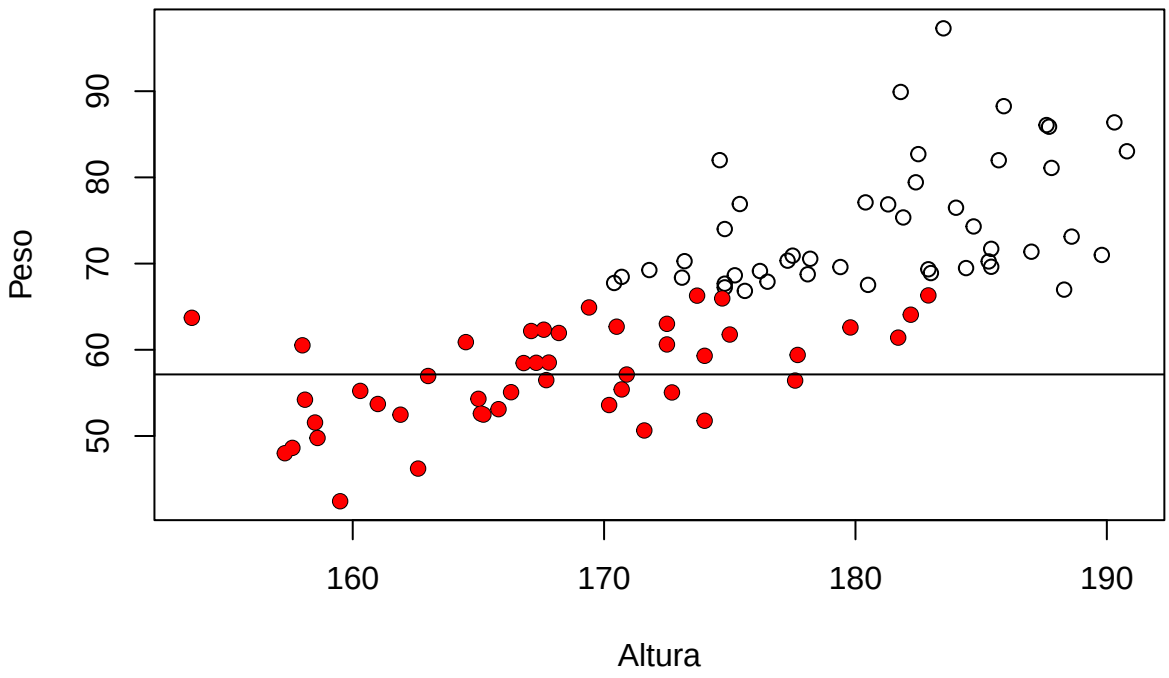
```
## [1] 47.66858
```

```
plot(Peso,type='l')
lines(yest.1,col='red')
```



Consideremos ahora las diferentes aproximaciones descritas en el paper. En primer lugar, el modo más básico de analizar el error de mi modelo es dividiendo la muestra en subconjuntos disjuntos (holdout). En este primer caso consideraremos sólo dos conjuntos, uno de train y otro de test. Para simplificar el modelo, supongamos que el modelo viene dado por la media de la variable. En este caso obtendremos

```
# Validation
# Tomamos primero el modelo y=cte, donde la cte es la media de la variable y
# para los elementos de train. Es decir para cualquier x, la prediccion de y
# es siempre la misma. En este caso entrenamos el modelo con la mitad de los datos
plot(Altura, Peso)
train <- 1:ceiling(n/2)
order.index <- order(Peso)
Peso.sort <- Peso[order.index]
Altura.sort <- Altura[order.index]
points(Altura.sort[train], Peso.sort[train], pch=16, col="red")
mean.peso <- mean(Peso.sort[train]) #y.est=cte esa cte es la media de la variable y selecionada en trai
abline(h=mean.peso)
```



Dado este ajuste, el error de test es un orden de magnitud mayor que el de test:

```
mse.train <- mse(Peso.sort[train],mean.peso); mse.train
```

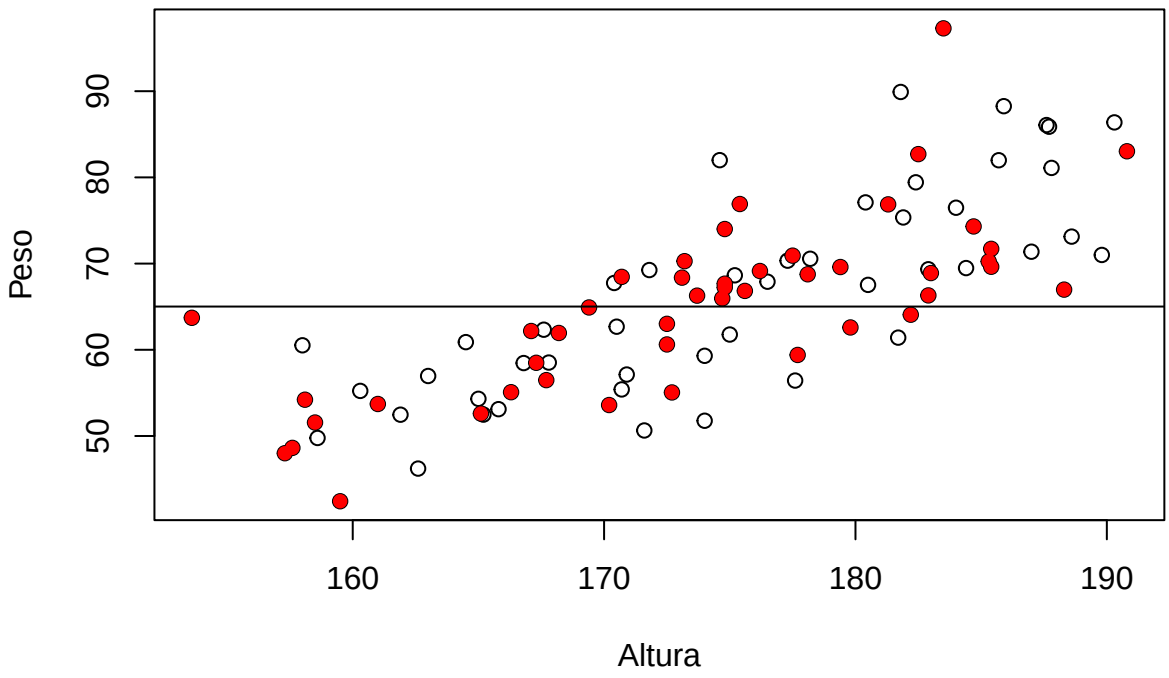
```
## [1] 31.42861
```

```
mse.test <- mse(Peso.sort[-train],mean.peso); mse.test # El error de test es mucho mayor ya que el mode
```

```
## [1] 354.6602
```

Notar que hemos entrenado con los valores de pesos menores, por lo que nuestra muestra no es representativa de la población. Una solución sería aleatorizar la elección de la muestra de entrenamiento. Si repetimos el proceso de este modo obtendremos.

```
# Repetimos el proceso con el mismo modelo y=cte pero cogiendo los datos de entrenamiento aleatoriament
set.seed(1) # to obtain the same results for all the users when a R's random number generator is used (
train <- sample(n,ceiling(n/2))
plot(Altura, Peso)
points(Altura[train], Peso[train], pch=16, col="red")
mean.peso <- mean(Peso[train]) #y.est=cte esa cte es la media de la variable y selecionada en train
abline(h=mean.peso)
```




```
mse.train <- mse(Peso[train],mean.peso); mse.train
```

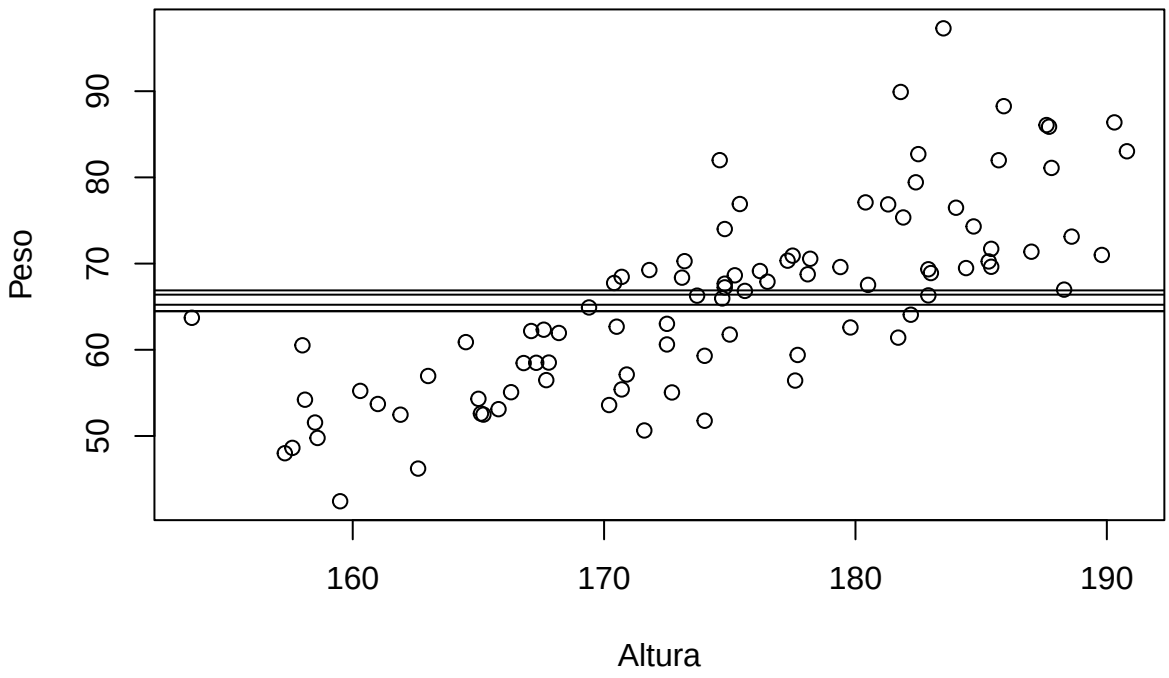
```
## [1] 99.36708
```

```
mse.test <- mse(Peso[-train],mean.peso); mse.test
```

```
## [1] 137.7091
```

Al considerar una muestra representativa de la variabilidad de la población ambos errores pasan a ser comparables. A pesar de ello, esta metodología tiene dos inconvenientes potenciales: 1.- La estimación del error en el conjunto de test puede variar mucho en función de la partición considerada.

```
plot(Altura, Peso)
for (i in c(1:5)){
  train <- sample(n,ceiling(n/2))
  mean.peso <- mean(Peso[train]) #y.est=cte esa cte es la media de la variable y selecionada en train
  abline(h=mean.peso)
  print(mse(Peso[-train],mean.peso))
}
```



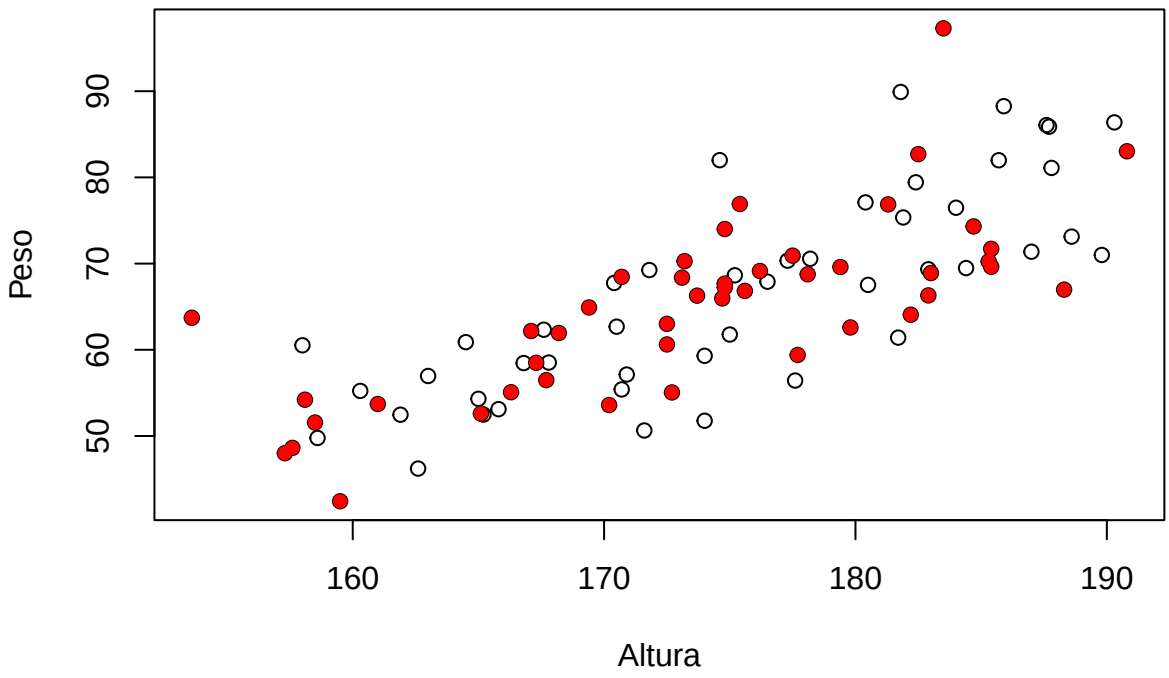

```
## [1] 97.35114
## [1] 144.1005
## [1] 106.466
## [1] 112.6597
## [1] 153.0071
```

2.- El error de test puede ser sobrestimado. Dado que el modelo se entrena en un subconjunto de la muestra y que los métodos estadísticos suelen comportarse peor cuando son entrenados con pocos datos, esto puede dar lugar a una sobrestimación del error de test respecto al obtenido considerando toda la muestra.

La validación cruzada que aplicaremos un poco más adelante considera estos dos problemas.

Vamos ahora a considerar un modelo un poco más complejo que el y=cte aplicado hasta ahora. Seleccionamos un modelo lineal combinado con un muestreo aleatorio. Para obtener valores equivalentes, fijaremos al semilla de la generación de números aleatorios a través del comando `set.seed`.

```
set.seed(1)
train <- sample(n,ceiling(n/2))
plot(Altura, Peso)
points(Altura[train], Peso[train], pch=16, col="red")
```




```
Reg.2<-lm(Peso~Altura, data=Pulsaciones, subset=train)
yest.2 <- predict(Reg.2, data.frame(Altura=Altura[-train]))
mse.Reg.2<-mse(Peso[-train],yest.2); mse.Reg.2 #If we choose different training set, the mse will be di
```

```
## [1] 52.29043
```

```
yest.2.train <- predict(Reg.2, data.frame(Altura=Altura[train]))
mse.Reg.2.train<-mse(Peso[train],yest.2.train); mse.Reg.2.train
```

```
## [1] 44.5575
```

Decimos que existe sobreajuste cuando mse.Reg.2 y mse.Reg.2.train son muy diferentes. Un modelo con capacidad de generalización, no sobreajustado, es aquel para el que ambos errores, en la muestra de entrenamiento y de test, son similares/comparables.

Como alternativa al muestreo aleatorio surge de forma natural para dar una primera solución a los dos problemas señalados anteriormente el método de validación cruzada denominado leave-one-out. En primer lugar, la selección de la muestra de entrenamiento no se hace aleatoriamente, eliminando la variabilidad del error de test, y en segundo lugar, la muestra de entrenamiento es la mayor posible considerando una muestra de test. Este método consiste en tomar `N` muestras de entrenamiento y de test, donde `N` es el tamaño muestral, donde la primera considera todos los datos menos uno, que forma la muestra de test. Por lo tanto, se ajustan `N` modelos, uno para cada muestra, obteniendo una estimación para cada uno de los puntos de la muestra obtenida con un modelo entrenado en una muestra independiente.

```
yest.3 <- rep(NA, length(train))
train <- 1:n
for (i in train){
  Reg.i<-lm(Peso~Altura, data=Pulsaciones, subset=train[-i])
```

```
  yest.3[i]<-predict(Reg.i,data.frame(Altura=Altura[i]))
}
mse.Reg.3<-mse(Peso,yest.3); mse.Reg.3
```

```
## [1] 50.02996
```

Como dijimos, si el tamaño muestral es grande este método es computacionalmente muy costoso. Como evolución del leave-one-out para evitar este coste surge otro método de validación cruzada denominado método k-fold, en el que se hace un leave-one-out por `bloques`. Es decir, se divide la muestra en `k` subconjuntos, de modo que se ajustan `k` modelos, considerando en cada caso un bloque como test y los `k-1` restante como muestra de entrenamiento. Al igual que en el primer método, la estimación dependerá de como se realice la partición de los datos, si bien esta variabilidad es mucho menor que en ese caso y con un número suficiente de subconjuntos, los resultados y conclusiones son las mismas que las obtenidas con un leave-one-out. Consideramos el ejemplo anterior con 10 subconjuntos.

```
# In this method, there is some variability in the cross validation estimates as a result of the
# variability in how the observations are divided into the k-folds. But this variability is typically m
# the variability in the test error estimates that results from the validation set approach.
idx.aleatorios <- sample(1:n,n,replace=F)
K <- 10
tam <- ceiling(n/K)
yest4 = c()
for (i in 0:(K-1)){
  idx.test <- idx.aleatorios[(i*tam+1):((i+1)*tam)]
  idx.test <- idx.test[!is.na(idx.test)]
  lm4 <- lm(Peso~Altura, subset=-idx.test)
  yest4[idx.test] <- predict(lm4, data.frame(Altura=Altura[idx.test]))
}
mse4 <- mse(Peso,yest4); mse4
```

```
## [1] 51.77176
```

Si consideramos diferentes Ks y lo repetimos varias veces los resultados del error resultan más estables:

```
for (K in c(2,5,10)){
  for (r in c(1:5)){
  idx.aleatorios <- sample(1:n,n,replace=F)
  tam <- ceiling(n/K)
  yest4 = c()
  for (i in 0:(K-1)){
    idx.test <- idx.aleatorios[(i*tam+1):((i+1)*tam)]
    idx.test <- idx.test[!is.na(idx.test)]
    lm4 <- lm(Peso~Altura, subset=-idx.test)
    yest4[idx.test] <- predict(lm4, data.frame(Altura=Altura[idx.test]))
  }
  mse4 <- mse(Peso,yest4); print(paste0(K,", ", r,": ",mse4))
  }
}
```

```
## [1] "2, 1: 50.4496700952768"
## [1] "2, 2: 50.8483026591664"
## [1] "2, 3: 53.6246601817439"
## [1] "2, 4: 49.1863859491232"
## [1] "2, 5: 47.7714788915106"
## [1] "5, 1: 49.0803650463122"
## [1] "5, 2: 50.9161789075285"
```

```
## [1] "5, 3: 51.5413588619798"
## [1] "5, 4: 48.8511086524317"
## [1] "5, 5: 51.4358274756512"
## [1] "10, 1: 51.019973392173"
## [1] "10, 2: 50.5397876805949"
## [1] "10, 3: 50.2573674553987"
## [1] "10, 4: 51.0470942991515"
## [1] "10, 5: 49.8715963817259"
```

Session Info:

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=es_ES.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=es_ES.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_ES.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ISLR_1.4
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.1.1    magrittr_2.0.1    tools_4.1.1       htmltools_0.5.1.1
##  [5] yaml_2.2.1        stringi_1.5.3     rmarkdown_2.11    highr_0.8
##  [9] knitr_1.31        stringr_1.4.0     xfun_0.27         digest_0.6.27
## [13] rlang_0.4.11      evaluate_0.14
```