Validación cruzada

Contents

1 Introducción

Si el modelo se ajusta muy bien a los datos que hemos utilizado para estimarlo quiere decir que va a predecir bien dichos datos. Pero en la práctica, queremos el modelo para predecir datos que no conocemos. El objetivo es analizar la predicción del modelo frente a datos no conocidos.

Los métodos explicados a continuación se emplean para comparar modelosdesde el punto de vista de la predicción.

2 El método del subconjunto de validación (validation set approach)

- Dividimos en dos partes los datos (50%-50%, 40%-60%,...).
- En el **training set** estimamos el modelo.
- En el **test set** (**validation set**) predecimos la respuesta y calculamos el Mean Squared Error (MSE) entre la variable respuesta observada y la predicha.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
datos = read.csv("datos/Advertising.csv")
str(datos)
```

```
## 'data.frame':    200 obs. of  5 variables:
##  $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ TV       : num  230.1 44.5 17.2 151.5 180.8 ...
##  $ radio    : num  37.8 39.3 45.9 41.3 10.8 48.9 32.8 19.6 2.1 2.6 ...
##  $ newspaper: num  69.2 45.1 69.3 58.5 58.4 75 23.5 11.6 1 21.2 ...
##  $ sales    : num  22.1 10.4 9.3 18.5 12.9 7.2 11.8 13.2 4.8 10.6 ...
```

- Dividimos los datos en training set y test set. En general, el training test ha de ser mayor o igual que el test set:

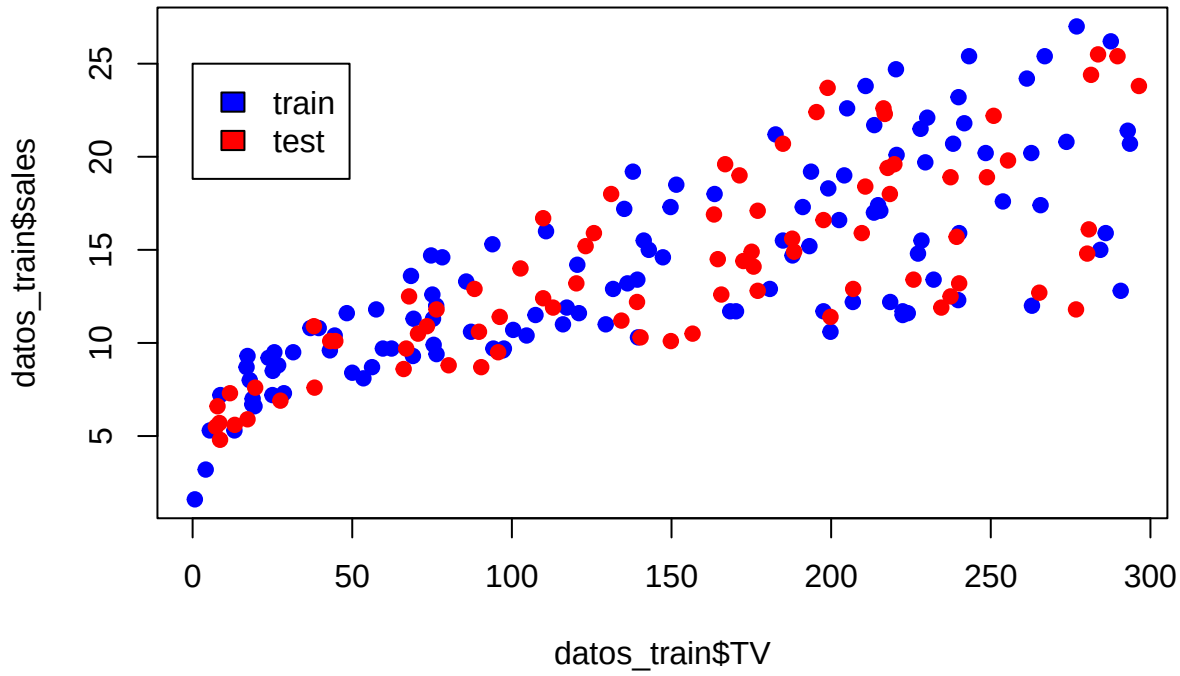
```
n = nrow(datos)
n_train = round(0.6*n)
n_test = n - n_train

set.seed(115)
```

```
pos = 1:n
pos_train = sample(pos, n_train, replace = F) # muestreo sin reemplazamiento
pos_test = pos[-pos_train]

# dividimos los datos en training set y validation set
datos_train = datos[pos_train,]
datos_test = datos[pos_test,]

plot(datos_train$TV, datos_train$sales, pch = 19, col = "blue")
points(datos_test$TV, datos_test$sales, pch = 19, col = "red")
legend(x=0,y=25, legend=c("train","test"), fill=c("blue","red"))
```



- En el training set estimamos el modelo:

```
m1 = lm(sales ~ TV, data = datos_train)

# error en el training set
y_train = datos_train$sales
y_train_p = predict(m1,datos_train)
```

- Para calcular el MSE se puede utilizar la función (descargar):

```
source("funciones/MSE.R")
MSE # se muestra la definición de la funcion
```

```
## function(y,yp){
##
##   y = as.numeric(y)
##   yp = as.numeric(yp)
##
##   n = length(y)
##   d = (y - yp)^2
##   suma = sum(d)
```

```
##   MSE = 1/n*suma
##   return(MSE)
## }
```

```
(mse_train = MSE(y_train,y_train_p))
```

```
## [1] 10.62459
```

- Error en el test set:

```
y_test = datos_test$sales
y_test_p = predict(m1,datos_test)

(mse_test = MSE(y_test,y_test_p))
```

```
## [1] 10.38168
```

Problemas:

- El MSE cambia en función de como se elija el training set y el validation set.
- Sólo se incluye una parte de los datos para estimar el modelo, por lo que la estimación es peor que incluyéndolos a todos.

3 Validación cruzada (k-fold Cross-Validation)

Se divide aleatoriamente los datos en **k** partes (o folds). Cada parte tiene n1 = n/k datos.

- Para i = 1:k
  - la parte i constituye el test set.
  - las otras partes constituyen el train set.
  - se calcula el MSE(i)
- El error total es:

$$MSE_{TOTAL} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

- Se ha programado una funcion que calcula las posiciones de train y test dado el numero de folds (descargar):

```
source("funciones/cross_val_pos.R")
cross_val_pos # se muestra el contenido de la función
```

```
## function(num_datos,num_folds){
##   # ---------------------------------------------------------------------
##   # esta funcion calcula las posiciones que corresponden al train
##   # y las posiciones que corresponden al test en cross validation
##   #
##   # num_datos: numero de datos
##   # num_folds: numero de folds
##   #
##   # javier.cara@upm.es, version 2019.10
##   # ---------------------------------------------------------------------
##
##   # numero de datos de cada fold
##   n1 = trunc(num_datos/num_folds)
##
##   v = sample(1:num_datos,num_datos,replace = F)
```

```
##   train = list()
##   test = list()
##   for (k in 1:(num_folds-1)){
##     pos_test = ((k-1)*n1+1):(k*n1)
##     test[[k]] = v[pos_test]
##     train[[k]] = v[-pos_test]
##   }
##   # el ultimo puede tener un numero diferente de datos (por trunc)
##   pos_test = ((num_folds-1)*n1+1):num_datos
##   test[[num_folds]] = v[pos_test]
##   train[[num_folds]] = v[-pos_test]
##
##   return(list(train = train, test = test))
## }
```

Por ejemplo, supongamos que tenemos 10 datos y definimos 3 folds:

```
set.seed(342)
num_datos = 10
num_folds = 3
cross_val_pos(num_datos,num_folds)
```

```
## $train
## $train[[1]]
## [1] 7 4 1 9 8 6 3
##
## $train[[2]]
## [1] 10  5  2  9  8  6  3
##
## $train[[3]]
## [1] 10  5  2  7  4  1
##
##
## $test
## $test[[1]]
## [1] 10  5  2
##
## $test[[2]]
## [1] 7 4 1
##
## $test[[3]]
## [1] 9 8 6 3
```

Con los dados del ejemplo:

```
# numero de folds
num_folds = 5
n = nrow(datos)

# se definen las posiciones de test de cada fold
pos = cross_val_pos(n,num_folds)

error_k = rep(0,num_folds)
for (k in 1:num_folds){
  # datos de training y de test de cada fold
  pos_test = pos$test[[k]]
```

```
  pos_train = pos$train[[k]]
  datos_train = datos[pos_train,]
  datos_test = datos[pos_test,]

  # estimamos el modelo
  mk = lm(sales ~ TV, data = datos_train)

  # error de prediccion
  yk = datos_test$sales
  yk_p = predict(mk,datos_test)

  error_k[k] = MSE(yk,yk_p)
}
(errork_media = mean(error_k))
```

```
## [1] 11.03013
```