Mayo 2018

Cristina Gil Martínez

MÉTODOS DE REMUESTREO Y VALIDACIÓN DE MODELOS: VALIDACIÓN CRUZADA Y BOOTSTRAP

Apuntes personales sobre Validación Cruzada (validación simple, k-fold cross validation y LOOCV) y Bootstrap con ejemplo en R

CONTENIDO

INTRODUCCIÓN

Los métodos de remuestreo (*resampling methods*) se basan en extraer muestras repetidamente a partir de un set de datos de entrenamiento, ajustando el modelo de interés para cada muestra. Se trata de métodos **no paramétricos**, que no requieren ninguna asunción sobre la distribución de la población.

Dos de los métodos más utilizados de remuestreo son la validación cruzada (*cross-validation*) y el *bootstrap*:

- **Validación cruzada**: puede aplicarse para estimar el test error asociado a un determinado método de aprendizaje estadístico (tanto regresión como clasificación) para **evaluar** el rendimiento del modelo (*model assessment*), pero también para **seleccionar** niveles apropiados de flexibilidad, como el grado de polinomio, etc. (*model selection*), usando el *test error rate*.
- **Bootstrap**: se puede usar en muchos contextos (como para el cálculo de la significancia estadística), comúnmente se aplica a la evaluación de la **precisión de la estimación de parámetros** de un determinado modelo de aprendizaje estadístico (error estándar, intervalos de confianza...).

Es importante diferenciar entre el *training error rate* y el *test error rate* a la hora de evaluar la capacidad predictiva de un modelo:

- ***Training error rate***: se obtiene a partir de los datos usados para el entrenamiento del modelo (error al predecir los mismos datos usados para su ajuste), sin embargo, subestima el verdadero error de predicción del modelo. Tiende a decrecer conforme la flexibilidad del modelo aumenta.
- ***Test error rate***: error promedio que resulta de usar el modelo para predecir nuevas observaciones no usadas para ajustar dicho modelo. Si este error es bajo, mejor la capacidad predictiva. Se puede calcular fácilmente si contamos con un set de datos de validación, sin embargo, no siempre disponemos de ellos. En ese caso, debemos utilizar los datos de entrenamiento disponibles y reservar un grupo de los mismos como set de validación, empleando el resto para el ajuste del modelo. En la práctica, para datos reales, tanto el límite de decisión de Bayes como el verdadero *test error rate* son **desconocidos**.

VALIDACIÓN CRUZADA

El término de Validación Cruzada incluye una clase de métodos que **estiman** el *test error rate* (no conocemos el verdadero *test error*) excluyendo una parte de las observaciones en el proceso de ajuste del modelo, usando luego dichas observaciones para evaluar la capacidad predictiva del modelo (en el cual

caso es de interés el valor específico del *test error rate*) o seleccionar un nivel de flexibilidad del mismo (en el cual caso interesa más determinar a qué nivel de flexibilidad se encuentra el mínimo en la curva de estimación del *test error rate*).

La diferencia principal entre un método u otro es la forma en la que se generan los grupos de entrenamiento y test.

Validación simple

El método de validación simple se basa en dividir de manera aleatoria (y tamaño comparable) el set de observaciones disponible en dos partes, el set de entrenamiento o *training set* y el set de validación o *test set*. El modelo es ajustado usando el set de entrenamiento, y este mismo modelo se usa para predecir las nuevas observaciones del set de validación, obteniendo así la estimación del *test error rate* (*Mean Squared Error* o *MSE* para variables respuesta cuantitativas, y proporción de predicciones incorrectas sobre las predicciones totales en el caso de variables respuesta cualitativas).

Aunque este método es simple y fácil de implementar, cuenta con dos **inconvenientes** en cuanto a varianza (1) y *bias* (2):

1. La estimación del *test error rate* por el método de validación simple puede ser **altamente variable**, dependiendo de qué observaciones hayan sido seleccionadas para formar parte del set de entrenamiento o validación.
2. Solo un subconjunto (aproximadamente la mitad) de observaciones se utiliza para entrenar el modelo, por lo que el error del set de validación puede **sobreestimar el *test error rate*** del modelo ajustado con el set completo de observaciones.

Leave-One-Out Cross Validation

El método de validación cruzada dejando uno fuera (LOOCV) está íntimamente relacionado con el de validación simple descrito anteriormente, además de que trata de resolver sus principales inconvenientes.

De nuevo, el set de datos se divide en dos partes. Sin embargo, en este caso la diferencia se encuentra en que en lugar de separar el set de datos de manera proporcional en entrenamiento y validación, **se utiliza una sola observación para el set de validación** (x_1, y_1), dejando al resto para el set de entrenamiento (x_2, y_2), ..., (x_n, y_n). Es decir, el modelo se ajusta utilizando las $\boldsymbol{n - 1}$ observaciones de entrenamiento,

obteniéndose una predicción $\hat{y}_1$ para la observación excluida usando su valor x_1. Ya que la observación (x_i, y_i) no se emplea en el ajuste del modelo, su *MSE*

$$MSE_i = (y_i - \hat{y}_i)^2$$

proporciona una estimación insesgada aproximada del *test error*. Pero aun así, resulta una estimación muy pobre debido a su alta variabilidad, ya que esta estimación se basa en una sola observación.

Repitiendo *n* veces el procedimiento descrito, se pueden obtener *n MSE*s (MSE_1, ..., MSE_n). Así, **la estimación** que hace el LOOCV **para el test MSE es el promedio de las *n* estimaciones**:

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MSE}_i$$

para problemas de regresión, y

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Err}_i$$

para problemas de clasificación, donde $\mathrm{Err}_i = I(y_i \neq \hat{y}_i)$ es el número de clasificaciones erróneas.

Podemos encontrar la excepción en el caso de modelos lineales ajustados por mínimos cuadrados, donde podemos usar la siguiente fórmula cuyo coste computacional iguala al ajuste de un único modelo:

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2$$

donde h_i es el estadístico de influencia de la *i*-ésima observación (un valor alto de h_i implica una observación con alta influencia).

Las **ventajas** principales del LOOCV frente a la Validación Simple son:

1. Sufre **menor *bias***, ya que el modelo se ajusta utilizando casi todas las observaciones ($n - 1$), a diferencia de la mayor cantidad de observaciones que se excluyen del modelo en el método de validación simple. Es decir, los estimadores serán aproximadamente insesgados. Además, permite **reducir la variabilidad** ya que al final, todos los datos se acaban usando tanto de entrenamiento como de test.
2. Tiende a **no sobreestimar el *test error rate*** tanto como lo pueda hacer la Validación Simple.

3. Llevar a cabo el LOOCV múltiples veces dará siempre los **mismos resultados** (resultados reproducibles): no existe aleatoriedad en la separación de los datos en entrenamiento y validación.

Por otro lado, el LOOCV sufre la **desventaja** del **coste computacional** de tener que ajustar *n* modelos, especialmente si *n* es elevado y cada ajuste individual es lento. Además, puede sufrir de ***overfitting***, ya que se acaban empleando todos los datos para ajustar el modelo.

LOOCV es un método que puede usarse con cualquier tipo de modelo predictivo.

LEAVE-GROUP-OUT CROSS VALIDATION

Leave Group Out cross-validation (LGOCV), también conocido como Monte Carlo CV, separa de manera aleatoria los subgrupos de datos entrenamiento-test (cuyo porcentaje se establece de antemano) múltiples veces. Se requieres muchas repeticiones para obtener resultados estables.

K-Fold Cross Validation

La Validación Cruzada de K iteraciones o *K-Fold Cross Validation* supone una alternativa al LOOCV. De hecho, LOOCV puede verse como un caso especial de K-Fold Cross Validation donde *K* = *n*. En este caso, **las observaciones se dividen aleatoriamente en k grupos de aproximadamente igual tamaño**. Uno de los k grupos se emplea como set de validación, mientras que el resto de ***k* – 1** grupos de emplean para entrenar el modelo. El *MSE* se calcula sobre el grupo *k* excluido del modelo, proceso que se repite *k* veces (cada vez un grupo *k* diferente es usado como set de validación). Se obtienen, por tanto, *k* estimaciones del test error (MSE_1, MSE_2, ..., MSE_k), calculándose la estimación global promediando los *k* valores:

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i$$

para problemas de regresión, y

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{Err}_i$$

para problemas de clasificación.

Las **ventajas** principales del *K-Fold Cross Validation* frente al LOOCV son:

1. Es **computacionalmente menos costoso**, ya que no se requiere ajustar *n* modelos.
2. **Equilibrio *Bias*-Varianza**: la variabilidad de las estimaciones debido a como se dividen las observaciones en los *k* grupos suele ser mucho menor que con LOOCV, por lo que las estimaciones son más precisas. También podemos obtener un **nivel intermedo de *bias*** (mayor con la validación simple y menor con el LOOCV), ya que contamos con menos observaciones para el set de entrenamiento que con LOOCV, pero más que con el método de validación simple, concretamente:

$$(k - 1)n / k$$

Aun así, desde la perspectiva de la reducción del *bias*, LOOCV es preferible a *k-Fold Cross Validation* (teniendo en cuenta el riesgo de sufrir *overfitting* con LOOCV).

Un valor común de *k* que puede dar buenos resultados en cuanto al equilibrio *bias*-varianza para estimar el *test error rate* es *K* = 5 o ***K* = 10**.

REPEATED K-FOLD CROSS VALIDATION

El proceso es el mismo que con *k-fold cross validation* simple, pero repitiendo el proceso *n* veces. Su uso es recomendado si el tamaño muestral es pequeño, ya que consigue un buen equilibrio *bias*-varianza.

Estadísticamente hablando, con el uso de *k-fold cross validation* las *k* repeticiones no son independientes unas de otras (los datos usados como entrenamiento en una determinada iteración también son usados como test en otras iteraciones), lo que significa obtener un estimador sesgado (aunque puede que no mucho). Llevar a cabo múltiples iteraciones puede reducir este sesgo, aunque la forma teórica de conseguir esto sería generar *n* set de datos independientes y llevar a cabo la división entrenamiento/test en cada una de ellas y luego obtener el promedio. Sin embargo, ello requeriría contar con muchos menos datos por grupo entrenamiento/test, por lo que el modelo obtenido puede no llegar a ser el mejor. *Repeated k-fold cross validation* supone una alternativa.

BOOTSTRAP

El método de *bootstrap* se puede aplicar para cuantificar la incertidumbre asociada a un determinado estimador o método de aprendizaje estadístico. Por ejemplo, se puede utilizar para estimar el error estándar (*SE*) de los coeficientes de un modelo de regresión lineal (en este caso se utilizan fórmulas

matemáticas que R implementa automáticamente, aunque el resultado sería válido siempre y cuando las condiciones para su aplicación se cumplan, mientras que el *bootstrap* no requiere de condiciones previas).

En la práctica, raramente se puede disponer reiteradamente de nuevas muestras o set de datos independientes de la población original. El *bootstrap* lo que nos permite es emular el proceso de obtención de nuevas muestras utilizando nuestro set de datos único con las que podemos por ejemplo **estimar la variabilidad/precisión asociada a un parámetro estimado**, sin la necesidad de repetir el muestreo (cuando esto no es posible). El muestreo se lleva a cabo con sustitución (*with replacement*), lo que implica que una misma observación puede ser seleccionada más de una vez dentro de un mismo set de datos de *bootstrap*, y el tamaño de cada pseudomuestra es del mismo tamaño que la muestra original.

EJEMPLOS EN R

Validación Simple:

ESTIMACIÓN DEL TEST ERROR

Usando como ejemplo el modelo logístico simple ajustado en el capítulo de Regresión Logística: Se trabajó con el set de datos *Weekly*, que contiene información sobre el rendimiento porcentual semanal del índice bursátil S&P 500 entre los años 1990 y 2010. Con ellos, se ha ajustado un modelo logístico simple para predecir el rendimiento (*Direction*: positivo o negativo), usando *Lag2* (porcentaje de retorno en las dos semanas previas) como el único predictor.

En aquel ejemplo se usó un método de separación de los datos en función del año, pero esta vez usando el método de validación simple para estimar el *test error* del modelo.

Para llevar a cabo el método de validación simple, generamos dos grupos separando los datos de manera aleatoria en un set de entrenamiento y otro de validación (50% en cada grupo):

```
library(ISLR)
set.seed(1)

# Índices aleatorios para el set de entrenamiento
indices.train <- sample(x = nrow(Weekly), size = 0.5*(nrow(Weekly)),
                        replace = FALSE)
# Subgrupos de entrenamiento y test
datos.entrenamiento <- Weekly[indices.train, ]
datos.test <- Weekly[-indices.train, ]
```

Es importante verificar que la distribución de la variable respuesta *Direction* se distribuye aproximadamente de manera similar entre el set de entrenamiento y test:

```
library(dplyr)

prop.table(table(datos.entrenamiento$Direction)) %>% round(digits = 3)

##
##  Down    Up
## 0.421 0.579

prop.table(table(datos.test$Direction)) %>% round(digits = 3)

##
##  Down    Up
## 0.468 0.532
```

En este caso, las proporciones son similares en ambos casos. A continuación, ajustamos el modelo logístico con los datos asignados a entrenamiento:

```
# Ajuste del modelo logístico con los datos de entrenamiento
modelo.logistico <- glm(Direction ~ Lag2, data = Weekly, family = "binomial",
                        subset = indices.train)

summary(modelo.logistico)

##
## Call:
## glm(formula = Direction ~ Lag2, family = "binomial", data = Weekly,
##     subset = indices.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4646  -1.3103   0.9979   1.0470   1.2716
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.31274    0.08707   3.592 0.000328 ***
## Lag2         0.05043    0.03915   1.288 0.197717
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 740.49  on 543  degrees of freedom
## Residual deviance: 738.82  on 542  degrees of freedom
## AIC: 742.82
##
## Number of Fisher Scoring iterations: 4
```

```
# Codificación de la variable respuesta para el modelo
attach(Weekly)
contrasts(Direction)

##      Up
## Down  0
## Up    1
```

Después de obtener el modelo, calculamos las predicciones que podemos obtener con el mismo a partir de la probabilidad a posteriori de *Direction* (estableciendo un *cutoff* de probabilidad de 0,5 para la clasificación).

```
# Cálculo de la probabilidad predicha por el modelo con los datos de test
prob.modelo <- predict(object = modelo.logistico, newdata = datos.test,
                       type = "response")

# Vector de caracteres "Down"
pred.modelo <- rep("Down", length(prob.modelo))
# Sustitución de "Down" por "Up" si la probabilidad a posteriori > 0,5
pred.modelo[prob.modelo > 0.5] <- "Up"

# Matriz de confusión
table(pred.modelo, datos.test$Direction)

##
## pred.modelo Down  Up
##        Down    3   3
##        Up    252 287

# Test error rate
mean(pred.modelo != datos.test$Direction)

## [1] 0.4678899
```

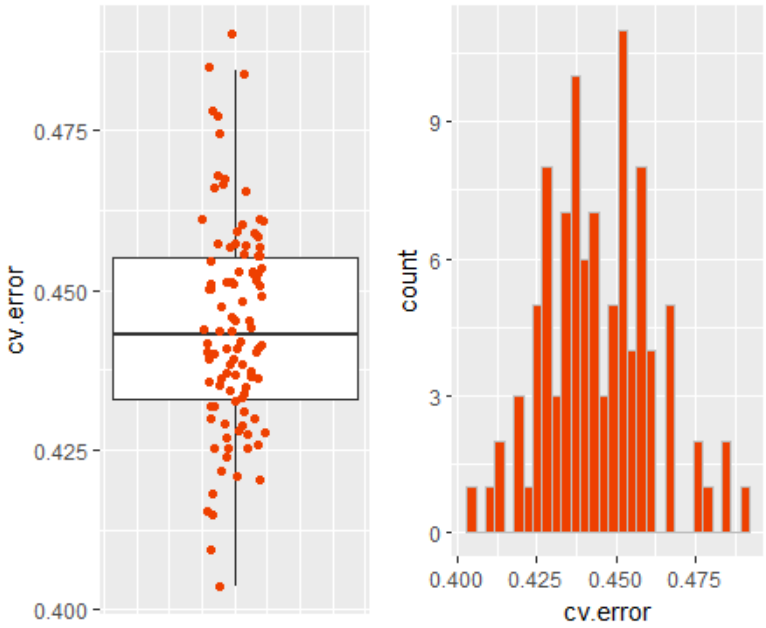
La estimación del *test error rate* del modelo mediante validación simple es del **46,78%**, por lo que el modelo acierta con sus predicciones en solo un 1 – 0,4678 = 53,2% de los casos.

Sin embargo, como se ha comentado en la explicación inicial, la estimación del *test error rate* mediante validación simple es propenso a sufrir alta variabilidad (depende de cómo se hayan distribuido las observaciones en los grupos de entrenamiento y test). A continuación, se muestra el mismo proceso llevado a cabo anteriormente, repitiéndolo 100 veces (en cada iteración los datos se van a repartir de manera distinta en entrenamiento y test).

```
# Vector donde se almacenarán los 100 test error estimados
cv.error <- rep(NA, 100)

for (i in 1:100){
# importante la creación de nuevos índices para cada iteración, de lo contrario, e
l test error obtenido siempre sería el mismo
indices.train <- sample(x = nrow(Weekly), size = 0.5*(nrow(Weekly)),
```

```
                        replace = FALSE)
datos.entrenamiento <- Weekly[indices.train, ]
datos.test <- Weekly[-indices.train, ]
modelo.logistico <- glm(Direction ~ Lag2, data = Weekly, family = "binomial",
                        subset = indices.train)
prob.modelo <- predict(object = modelo.logistico, newdata = datos.test,
                       type = "response")
pred.modelo <- rep("Down", length(prob.modelo))
pred.modelo[prob.modelo > 0.5] <- "Up"
cv.error[i] <- mean(pred.modelo != datos.test$Direction)
}

summary(cv.error)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4037  0.4330  0.4431  0.4442  0.4550  0.4899
```

```
library(ggplot2)
library(gridExtra)

boxplot <- ggplot(data = data.frame(cv.error = cv.error), aes(x = 1,
                                                              y = cv.error)) +
geom_boxplot(outlier.shape = NA) +
geom_jitter(colour = c("orangered2"), width = 0.1) +
theme(axis.title.x = element_blank(),axis.text.x = element_blank(),
      axis.ticks.x = element_blank())

histograma <- ggplot(data = data.frame(cv.error = cv.error), aes(x = cv.error)) +
geom_histogram(color = "grey", fill = "orangered2")

grid.arrange(boxplot, histograma, ncol = 2)
```



Como resultado de las 100 iteraciones, la estimación del *test error* oscila entre el 40,37% y el 48,99%, con una media del **44,43%,** por lo que el porcentaje de acierto es del 55,57%.

Para que el modelo predictivo se considere útil, debe acertar con las predicciones en un porcentaje superior a lo esperado por azar o respecto al nivel basal (el que se obtiene si se asignaran a la clase mayoritaria), que en el caso de este set de datos corresponde a "Up":

```
prop.table(table(Weekly$Direction)) %>% round(digits = 3)
```

```
##
##  Down    Up
## 0.444 0.556
```

Si siempre se predijera *Direction = "Up"*, el porcentaje de aciertos sería aproximadamente del 56%, por lo que este es el porcentaje mínimo que debería superar el modelo. En este caso, no se supera.

LOOCV

Al igual que con la validación simple, LOOCV se puede emplear para estimar el *test error* y para determinar el grado óptimo de flexibilidad de un modelo:

ESTIMACIÓN DEL TEST ERROR

Usando el ejemplo anterior, esta vez llevaremos a cabo el método LOOCV para estimar el test error. En este caso usamos $n - 1$ observaciones para entrenar el modelo en cada iteración (usando un *for loop*):

```
set.seed(1)
# Vector que contendrá los aciertos (0) y errores (1) del modelo
loocv.error <- rep(0, nrow(Weekly))
 # Las iteraciones serán igual al número de observaciones n
for (i in 1:nrow(Weekly)) {
modelo.logistico <- glm(Direction ~ Lag2, data = Weekly[-i, ],
                        family = "binomial")
# si la probabilidad a posteriori es > 0.5 -> TRUE
pred.up <- predict.glm(modelo.logistico, Weekly[i, ], type = "response") > 0.5
# si el valor Direction de la i-ésima observación es "Up" -> TRUE
true.up <- Weekly[i, ]$Direction == "Up"
# si los TRUE/FALSE de ambos objetos (predicción y realidad) no coinciden -> se as
igna error (1)
if (pred.up != true.up)
loocv.error[i] <- 1
}

sum(loocv.error)
```

```
## [1] 480
```

Ha habido en total 480 errores de predicción de un total de 1089 observaciones con las que cuenta el set de datos. Si calculamos la media, obtenemos la estimación del *test error*:

```
mean(loocv.error)
```

```
## [1] 0.4407713
```

LOOCV estima un *test error* del **44,07%,** valor muy próximo al estimado por validación simple**.**

SELECCIÓN DE FLEXIBILIDAD DEL MODELO

La función *cv.glm()* del paquete *boot* puede emplearse para llevar a cabo LOOCV (si el argumento *K* no se especifica) de cualquier modelo lineal generalizado creado mediante la función *glm()*.

La función devuelve una lista con múltiples componentes. El vector *delta* es el que contiene el resultado de la validación cruzada (contiene dos valores, el primero es el estimador estándar y el segundo es la versión corregida por sesgo).

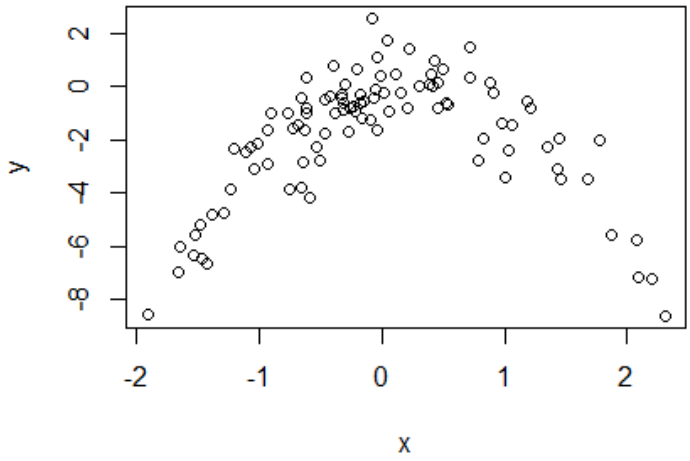
Para este ejemplo usaremos un set de datos simulados con dos variables ($p = 2$) y 100 observaciones ($n = 100$), según la siguiente ecuación:

$$Y = 2x - 2x^2 + \epsilon$$

```
set.seed(1)
y <- rnorm(100)
x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)

datos <- data.frame(x = x, y = y)
head(datos)
```

```
##             x          y
## 1 -0.62036668 -0.9806745
## 2  0.04211587  1.7274417
## 3 -0.91092165 -0.9838897
## 4  0.15802877 -0.2228252
## 5 -0.65458464 -3.7967823
## 6  1.76728727 -1.9816597
```

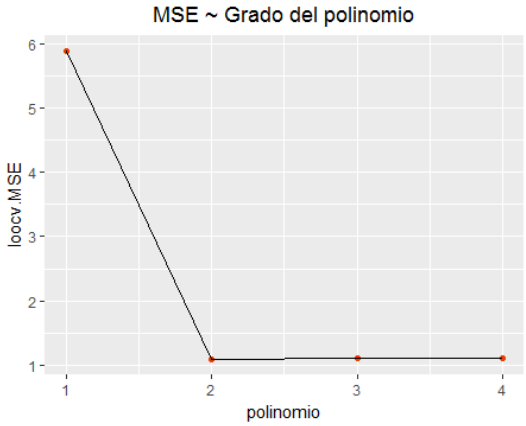
```
plot(x, y)
```



Al representar los datos podemos observar como la relación entre *x* e *y* no es lineal. Con lo cual, si quisiéramos ajustar un modelo, sería conveniente usar términos polinómicos de la variable *X*:

En este caso, al llevar a cabo LOOCV no es necesario dividir los datos en grupos.

```
library(boot)
# En LOOCV el establecer una u otra semilla no cambiará el resultado ya que no hay
 reparto aleatorio de las observaciones
loocv.MSE <- rep(NA, 4)
for (i in 1:4) {
modelo <- glm(y ~ poly(x, i), data = datos)
# Almacenamos en el vector los valores estándar (delta[1])
loocv.MSE[i] <- cv.glm(data = datos, glmfit = modelo)$delta[1]
}

loocv.MSE

## [1] 5.890979 1.086596 1.102585 1.114772


ggplot(data = data.frame(polinomio = 1:4, loocv.MSE = loocv.MSE),
       aes(x = polinomio, y = loocv.MSE))+
geom_point(color = "orangered2") +
geom_path() +
labs(title = "MSE ~ Grado del polinomio") +
theme(plot.title = element_text(hjust = 0.5))
```



El resultado de la validación cruzada indica que un polinomio de grado 2 mejora sustancialmente el ajuste del modelo (*test error* = 1,08%), no superándose esta mejora con grados mayores de polinomio. Esto es de esperar ya que la relación entre *x* e *y* (dada por la ecuación inicial) es cuadrática. Además, con el summary del modelo podemos obtener la significancia estadística de los coeficientes para cada grado de polinomio mediante el ajuste por mínimos cuadrados.

```
summary(lm(y ~ poly(x, 4), data = datos))
```

```
##
## Call:
## lm(formula = y ~ poly(x, 4), data = datos)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8914 -0.5244  0.0749  0.5932  2.7796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.1041 -17.549   <2e-16 ***
## poly(x, 4)1   2.3164     1.0415   2.224   0.0285 *
## poly(x, 4)2 -21.0586     1.0415 -20.220   <2e-16 ***
## poly(x, 4)3  -0.3048     1.0415  -0.293   0.7704
## poly(x, 4)4  -0.4926     1.0415  -0.473   0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.041 on 95 degrees of freedom
## Multiple R-squared:  0.8134, Adjusted R-squared:  0.8055
## F-statistic: 103.5 on 4 and 95 DF,  p-value: < 2.2e-16
```

Resulta significativo el ajuste hasta el grado de polinomio 2 (*p-value* = 2 x10^{-16}), con lo que las conclusiones coinciden.

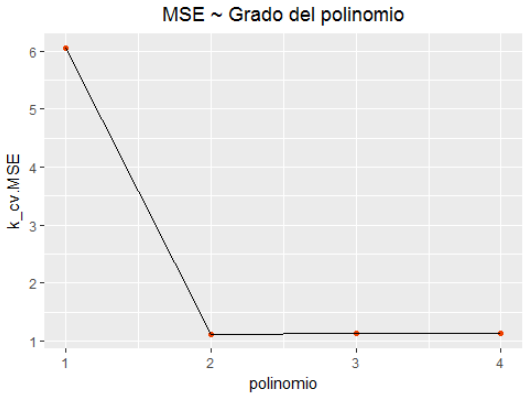
k-Fold Cross Validation

La función *cv.glm()* del paquete *boot* puede emplearse para llevar a cabo *k-fold cross validation* (especificando el argumento *K* correspondiente a los *folds*) de cualquier modelo lineal generalizado creado mediante la función *glm()*. Un valor común es *K* = 10.

ESTIMACIÓN DEL TEST ERROR

Siguiendo con el ejemplo anterior, incluyendo el argumento *K*, el resto del código sería el mismo:

```r
library(boot)
# Vector donde se almacenarán los 10 test error estimados
k_cv.MSE <- rep(NA, 4)

for (i in 1:4) {
modelo <- glm(y ~ poly(x, i), data = datos)
# Especificamos los folds con el argumento k. Almacenamos en el vector los valores
 estándar del test error (delta[1]).
set.seed(20)
k_cv.MSE[i] <- cv.glm(data = datos, glmfit = modelo, K = 10)$delta[1]
}

k_cv.MSE

## [1] 6.060910 1.111831 1.137608 1.141248



ggplot(data = data.frame(polinomio = 1:4, k_cv.MSE = k_cv.MSE),
       aes(x = polinomio, y = k_cv.MSE))+
geom_point(color = "orangered2") +
geom_path() +
labs(title = "MSE ~ Grado del polinomio") +
theme(plot.title = element_text(hjust = 0.5))
```



Hemos obtenido resultados muy similares al LOOCV en cuanto a la estimación del test error por cada grado de polinomio (siendo el grado 2 de nuevo, el mejor). En casos con un gran número de datos, *k-fold cross validation* supondría una ventaja al ser computacionalmente más rápido.

Bootstrap

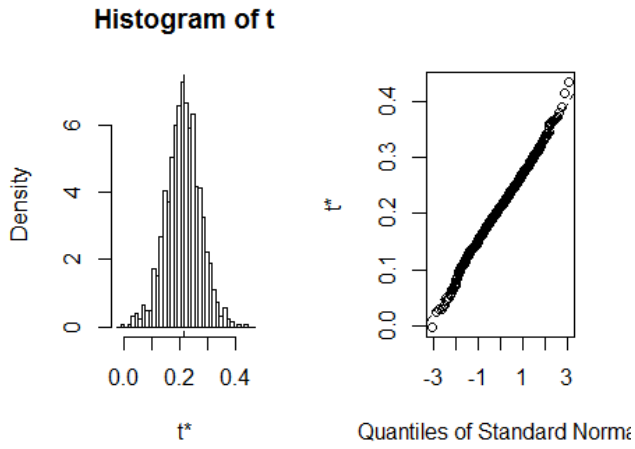
Continuando con el ejemplo del set de datos *Weekly* usado en los ejemplos anteriores de validación simple y LOOCV, obtendremos una estimación del error estándar del modelo logístico mediante *bootstrap,* usando la función *boot* del paquete *boot*.

```r
set.seed(1)
# Función que devuelve las estimaciones de los coeficientes del modelo. Con
"index" se seleccionan las observaciones para la muestra bootstrap
fun.boot <- function(data, index) {
modelo.glm <- glm(Direction ~ Lag2, data = data, family = "binomial",
                  subset = index)
return (coef(modelo.glm))
}
```

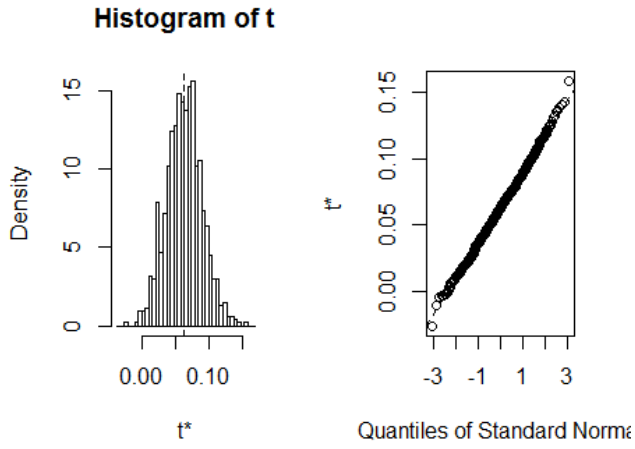
```r
# Bootstrap con 1000 pseudomuestras
library(boot)
estimacion.boot <- boot(data = Weekly, statistic = fun.boot, R = 1000)
estimacion.boot

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Weekly, statistic = fun.boot, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 0.21473151 -0.0031742583  0.06128018
## t2* 0.06279058  0.0001021562  0.02672193
```

Si compararemos los estimadores del error estándar para cada coeficiente obtenidos mediante *bootstrap* con los que se obtiene mediante la fórmula del error estándar usada por el modelo *glm(),* podemos ver que obtenemos valores casi iguales:

```r
summary(glm(Direction ~ Lag2, data = Weekly, family = "binomial"))$coef

##               Estimate Std. Error  z value     Pr(>|z|)
## (Intercept) 0.21473151 0.06122616 3.507185 0.0004528734
## Lag2        0.06279058 0.02636297 2.381772 0.0172295448


# Distribución bootstrap de Intercept
plot(estimacion.boot, index = 1)
```




```
# Distribución bootstrap de Lag2
plot(estimacion.boot, index = 2)
```



También podemos obtener los intervalos de confianza para los coeficientes:

```
# Intervalo de confianza para Intercept
boot.ci(boot.out = estimacion.boot, type = "norm", index = 1)

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = estimacion.boot, type = "norm", index = 1)
##
## Intervals :
## Level      Normal
## 95%   ( 0.0978,  0.3380 )
## Calculations and Intervals on Original Scale
```

```r
# Intervalo de confianza para Lag2
boot.ci(boot.out = estimacion.boot, type = "norm", index = 2)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = estimacion.boot, type = "norm", index = 2)
##
## Intervals :
## Level      Normal
## 95%   ( 0.0103,  0.1151 )
## Calculations and Intervals on Original Scale
```

BIBLIOGRAFÍA

An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)

http://www.stat.wisc.edu/~larget/stat302/chap3.pdf