Validación de modelos de regresión: Cross-validation, OneLeaveOut, Bootstrap

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Noviembre 2016

Tabla de contenido

Formato PDF: https://github.com/JoaquinAmatRodrigo/Estadistica-con-R

Introducción

Una vez seleccionados los predictores adecuados, generado el modelo y comprobado que se cumplen las condiciones necesarias del método de ajuste empleado, el siguiente paso es evaluar la capacidad de dicho modelo para predecir la variable respuesta.

Definiciones

Training data set o set de entrenamiento: datos/observaciones con las que se genera el modelo estadístico.

Test data set o set de validación: datos/observaciones del mismo tipo que las que forman el *training data set* pero que no se han empleado en la creación del modelo. Son datos que el modelo no ha "visto".

Training error rate: error que comete el modelo al predecir observaciones que pertenecen al *training data set*.

Test error rate: error que comete el modelo al predecir observaciones de un *test data set* y que por lo tanto el modelo no ha "visto".

Los métodos de *resampling* son muy útiles a la hora de evaluar los modelos estadísticos ya que permiten ajustar un modelo múltiples veces usando diferentes subsets del *training data set*. Dos de los métodos de *resampling* más empleados con este fin son el *cross-validation* y el *bootstraping*.

Cross-validation: Se emplea para estimar el *test error rate* de un modelo y así evaluar su capacidad predictiva, a este proceso se le conoce como *model assessment*. También se puede emplear para seleccionar el nivel de flexibilidad adecuado (grado del polinomio, número de *K-Nearest-Neighbors*...), lo que se conoce como *model selection*.

Bootstrap: Se emplea por lo general para medir la precisión de los parámetros estimados por un modelo estadístico (crear intervalos de confianza).

A la hora de evaluar la capacidad predictiva de un modelo estadístico es importante diferenciar entre dos tipos de error. El *training error rate* es el error promedio que tiene un modelo al intentar predecir las mismas observaciones que se han empleado para crear el modelo, el *training data set*. El *test error rate* es el error promedio que comete un modelo al intentar predecir nuevas observaciones que no estaban incluidas entre las que se emplearon para generar el modelo.

Por lo general, en el caso de variables cuantitativas, el error se mide mediante *mean square error (MSE)*. Cuando se trata de variables cualitativas (clasificación), el error se mide como la proporción de predicciones incorrectas (clasificación en el nivel erróneo) respecto al total de predicciones.

El *training error rate* siempre se puede calcular ya que consiste en predecir los datos con los que se ha generado el modelo y por lo tanto se conoce el verdadero valor de cada una de estas observaciones. Sin embargo, al emplear datos que el modelo ya ha "visto", esta medida de error es por lo general muy optimista y sobrevalora la capacidad predictiva del modelo. El *test error rate* es mucho más informativo ya que refleja el error que tiene el modelo al realizar su finalidad principal, predecir nuevas observaciones para las que se conocen los predictores pero no la variable respuesta. El problema radica en que no siempre se dispone de nuevos sets de datos con los que poder cuantificar este error.

Cross-Validation

El término *Cross-Validation* abarca distintas estrategias que permiten estimar el *test error rate*. Para ello se excluyen una serie de observaciones del *training data set* disponible, se ajusta el modelo y finalmente se evalúa con los datos excluidos.

Validación simple

El método más sencillo consiste en dividir aleatoriamente las observaciones disponibles en dos grupos, uno se emplea para entrenar al modelo y otro para evaluarlo. Si bien es la opción más simple, tiene dos problemas importantes:

- La estimación del *test error rate* es altamente variable dependiendo de qué observaciones se incluyan como set de entrenamiento y cuáles como set de validación (problema de varianza).
- Al excluir parte de las observaciones disponibles como datos de entrenamiento (generalmente la mitad), se dispone de menos información con la que crear el modelo y por lo tanto se reduce su capacidad. Esto suele tener como consecuencia una sobrestimación del *test error* comparado al que se obtendría si se emplearan todas las observaciones para el entrenamiento (problema de *bias*).

Leave One Out Cross-Validation (LOOCV)

El método LOOCV en un método iterativo que se inicia empleando como *training data set* todas las observaciones disponibles excepto una, que se excluye para emplearla como *test*. Si se emplea una única observación para calcular el *test error*, este varía mucho dependiendo de qué observación se haya seleccionado. Para evitarlo, el proceso se repite tantas veces como observaciones disponibles, excluyendo en cada iteración una observación distinta, ajustando el modelo con el resto y calculando el error con dicha observación. Finalmente, el *test error rate* estimado por el LOOCV es el promedio de todos lo *i* errores calculados.

En el caso de variables continuas en las que el error se mide como *MSE*:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(MSE_i)$$

En el caso de variables cualitativas en las que el error se mide por número de errores de clasificación:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(Err_i)$$

siendo $Err_1 = I(y_1 \neq \hat{y}_i)$.

El método LOOCV permite reducir la variabilidad que se origina si se divide aleatoriamente las observaciones únicamente en dos grupos, *training* y *test*. Esto es así porque al final del proceso de LOOCV se acaban empleando todos los datos disponibles tanto como entrenamiento como validación. Al no haber una separación aleatoria de los datos, los resultados de LOOCV son totalmente reproducibles.

La principal desventaja de este método es su coste computacional. El proceso requiere que el modelo sea reajustado y validado tantas veces como observaciones disponibles (n) lo que en algunos casos puede ser muy complicado. Excepcionalmente, en la regresión por mínimos cuadrados y regresión polinomial, por sus características matemáticas, solo es necesario un ajuste, lo que agiliza mucho el proceso.

LOOCV es un método de validación muy extendido ya que puede aplicarse para evaluar cualquier tipo de modelo. Sin embargo, los autores de *An Introduction to Statistical Learning* consideran que, al emplearse todas las observaciones como entrenamiento, se puede estar cayendo en *overfitting*, por lo que, aun considerándolo muy aceptable, recomiendan emplear *K-Fold Cross-Validation*.

K-Fold Cross-Validation

El método *K-Fold Cross-Validation* es también un proceso iterativo. Consiste en dividir los datos de forma aleatoria en *k* grupos de aproximadamente el mismo tamaño. *k-1* grupos se emplean para entrenar el modelo y uno de los grupos se emplea como test, este proceso se repite *k* veces utilizando un grupo distinto como *test* en cada iteración. El proceso genera *k* estimaciones del *test error* cuyo promedio se emplea como estimación final.

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k}(MSE_i)$$

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k}(Err_i)$$

En el libro *Introduction to Statistical Learning* consideran 2 ventajas del método *K-Fold Cross-Validation* frente al *LOOCV*:

- Requerimientos computacionales: el número de iteraciones necesarias viene determinado por el valor *k* escogido. Por lo general se recomienda un *k* entre 5 y 10. LOOCV es un caso particular de *K-Fold Cross-Validation* en el que *k = nº observaciones*, si el data set es muy grande o el modelo muy complejo, se requiere muchas más iteraciones.

- Balance entre *bias* y varianza: La principal ventaja de *K-fold CV* es que consigue una estimación precisa del *test error rate* gracias a un mejor Balance entre bias y varianza. LOOCV emplea *n-1* observaciones para entrenar el modelo, lo que es prácticamente todo el set de datos disponible, maximizando así el ajuste del modelo a los datos disponibles y reduciendo el *bias*. Sin embargo, para la estimación final del *test error rate* se promedian las estimaciones de *n* modelos entrenados con prácticamente los mismos datos (solo hay un dato de diferencia entre cada *training set*), por lo que están altamente correlacionados. Esto se traduce en un mayor riesgo de *overfitting* y por lo tanto varianza. En el método *K-fold CV* los *k* grupos empleados como entrenamiento son mucho menos solapantes, lo que se traduce en menor varianza al promediar las estimaciones de error.

Aunque emplea menos observaciones como entrenamiento que LOOCV, son un número suficiente como para no tener un *bias* excesivo, por lo que el método *K-fold CV* con valores de *k*= [5, 10] consigue un mejor balance final.

Ejemplos de validación

Validación simple: estimación de *test error* y selección de flexibilidad del modelo

En el siguiente ejemplo se emplea el dataset `Auto` *del paquete* `ISLR` *para mostrar el método de validación simple consistente en dividir las observaciones de forma aleatoria en dos grupos, uno de ellos se emplea como set de entrenamiento y otro como set de validación. Se pretende generar un modelo que permita predecir el consumo de un vehículo (mpg) a partir de la potencia del motor (horse power), estimar su test error rate y el grado de flexibilidad (polinomio) más adecuado.*

```
library(ISLR)
require(knitr)
data(Auto)
kable(head(Auto, n = 3), align = "c")
```

mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
18	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
15	8	350	165	3693	11.5	70	1	buick skylark 320
18	8	318	150	3436	11.0	70	1	plymouth satellite

```
dim(Auto)
```

```
## [1] 392   9
```

En primer lugar se separan aleatoriamente las observaciones en dos grupos de 196 observaciones (mitad de las observaciones para cada set).

```
# Se seleccionan 196 índices aleatorios que formarán el training set.
set.seed(1)
train <- sample(x = 1:392, 196)
```

Se genera un modelo lineal que relacione el consumo (*mpg*) con la potencia (*horsepower*) empleando únicamente los datos de *training*. Se recurre a la función `lm()` para generar el modelo y el argumento `subset` para identificar las observaciones que se tienen que emplear como entrenamiento.

```
modelo <- lm(mpg ~ horsepower, data = Auto, subset = train)
summary(modelo)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.698  -3.085  -0.216   2.680  16.770
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.340377   1.002269   40.25   <2e-16 ***
## horsepower  -0.161701   0.008809  -18.36   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.692 on 194 degrees of freedom
## Multiple R-squared:  0.6346, Adjusted R-squared:  0.6327
## F-statistic: 336.9 on 1 and 194 DF,  p-value: < 2.2e-16
```

Una vez generado el modelo se emplea la función `predict()` para estimar el consumo de las 196 observaciones restantes no empleadas como entrenamiento.

```
predicciones <- predict(object = modelo, newdata = Auto[-train, ])
```

Dado que se conoce el valor real de consumo de los 196 coches empleados como *test* se puede estimar el error de predicción del modelo. Al tratarse de una variable continua se emplea como medida de error el *MSE (mean square error)*.

```
mean((Auto$mpg[-train] - predicciones)^2)
```

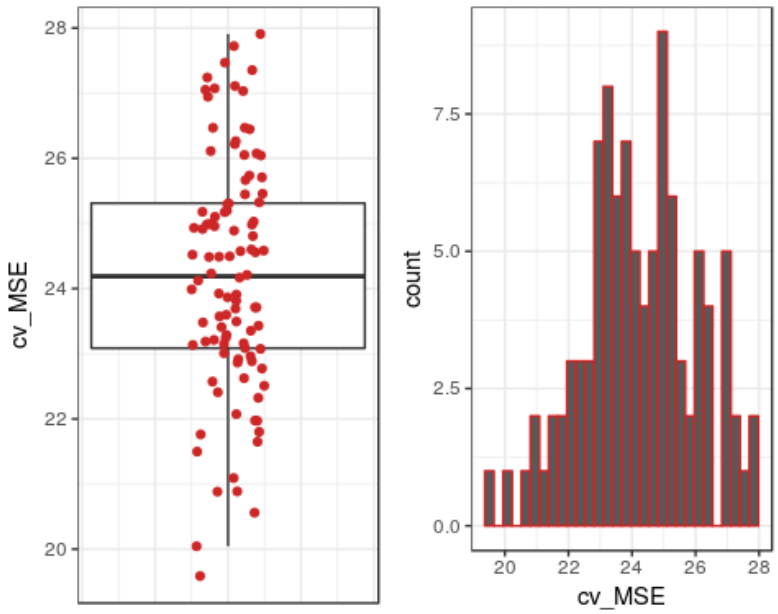
```
## [1] 26.14142
```

La estimación de *test error rate* del modelo de regresión lineal creado es de 26.14.

Como se ha comentado previamente, una de las principales desventajas de *cross-validation simple* es que la estimación de error varía mucho dependiendo de cómo se haya repartido los datos entre set de entrenamiento y set de validación. A continuación se calcula el *test error rate* para 100 repeticiones de la validación, repartiendo en cada una de ellas los datos de forma aleatoria.

```
library(ggplot2)
library(gridExtra)
```

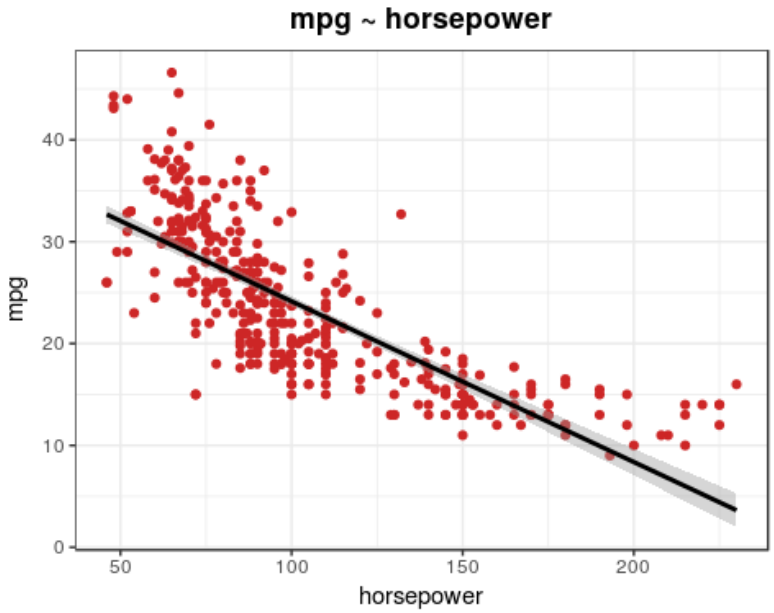
```
cv_MSE <- rep(NA, 100)
for (i in 1:100) {
    train <- sample(x = 1:392, 196)
    modelo <- lm(mpg ~ horsepower, data = Auto, subset = train)
    predicciones <- predict(object = modelo, newdata = Auto[-train, ])
    cv_MSE[i] <- mean((Auto$mpg[-train] - predicciones)^2)
}
p1 <- ggplot(data = data.frame(cv_MSE = cv_MSE), aes(x = 1, y = cv_MSE)) +
      geom_boxplot(outlier.shape = NA) +
      geom_jitter(colour = c("firebrick3"), width = 0.1) +
      theme_bw() +
      theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
      axis.ticks.x = element_blank())

p2 <- ggplot(data = data.frame(cv_MSE = cv_MSE), aes(cv_MSE)) +
      geom_histogram(colour = "firebrick3") +
      theme_bw()
      grid.arrange(p1, p2, ncol = 2)
```



Se observa que la estimación del *test error rate* oscila entre 19.58 y 27.91, con media de 24.20 y sd 1.8.

La representación gráfica de las observaciones y del modelo, muestra que la relación entre las variables *mpg* y *horsepower* no es del todo lineal.

```
ggplot(data = Auto, aes(x = horsepower, y = mpg)) +
geom_point(colour = c("firebrick3")) +
geom_smooth(method = "lm", colour = "black") +
theme_bw() +
labs(title = "mpg ~ horsepower") +
theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

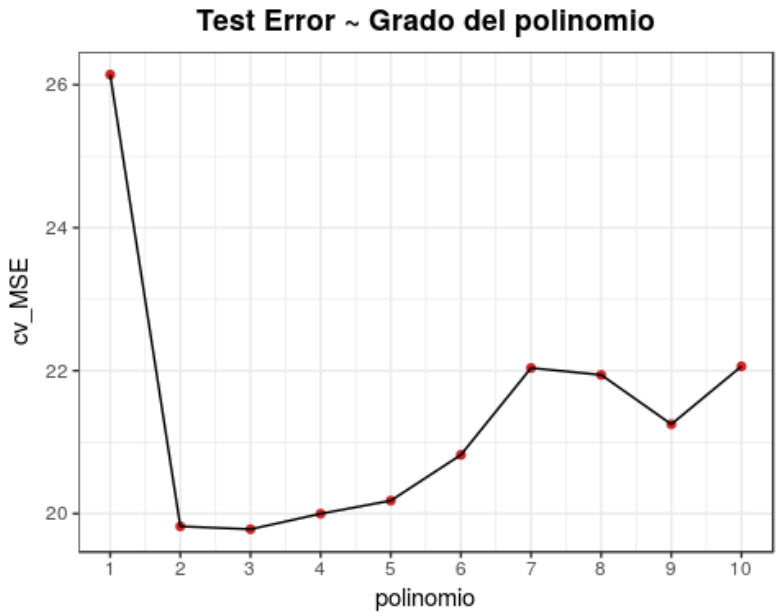


Introduciendo flexibilidad en el modelo mediante polinomios de mayor grado podría reducir el error de predicción. La validación cruzada nos permite identificar con qué grado de polinomio se consigue el mejor modelo (menor *test cv-MSE*).

A continuación se ajustan 10 modelos distintos empleando polinomios de grado 1 hasta 10 y se registra el cv-MSE para cada uno. La función `poly()` permite determinar el grado de polinomio empleado.

```
cv_MSE <- rep(NA, 10)
set.seed(1)
train <- sample(x = 1:392, 196)
```

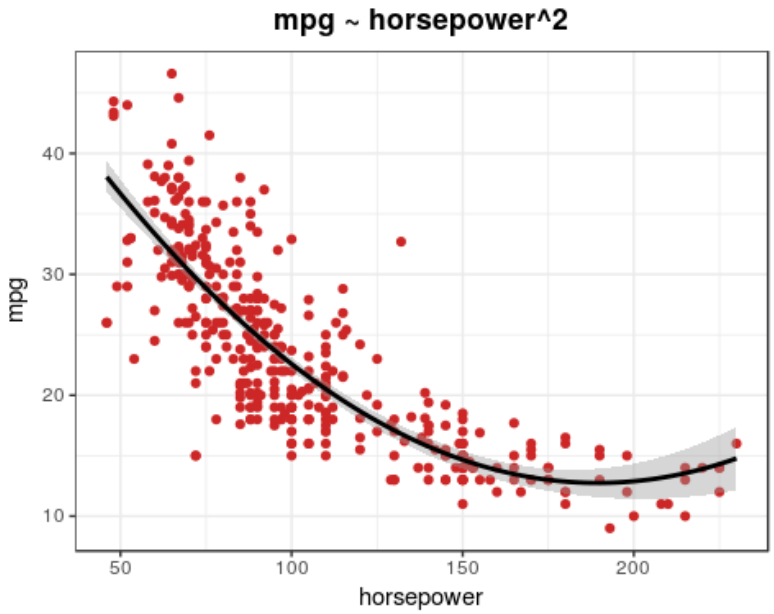
```
for (i in 1:10) {
    modelo <- lm(mpg ~ poly(horsepower, i), data = Auto, subset = train)
    predicciones <- predict(object = modelo, newdata = Auto[-train, ])
    cv_MSE[i] <- mean((Auto$mpg[-train] - predicciones)^2)
}
ggplot(data = data.frame(polinomio = 1:10, cv_MSE = cv_MSE),
       aes(x = polinomio, y = cv_MSE)) +
geom_point(colour = c("firebrick3")) +
geom_path() +
scale_x_continuous(breaks = c(0:10)) +
theme_bw() +
labs(title = "Test Error ~ Grado del polinomio") +
theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



Mediante validación cruzada simple se identifica que un modelo que emplee una función cuadrática o cúbica de *horsepower* minimiza el *test error*, es decir, captura mejor la relación existente entre las variables y por lo tanto tiene mayor precisión en sus predicciones. Dada la mínima mejoría que consiguen los polinomios de grado 3 o superior, siguiendo el principio de parsimonia, el de grado 2 es el más adecuado.

```
ggplot(data = Auto, aes(x = horsepower, y = mpg)) +
```

```
geom_point(colour = c("firebrick3")) +
stat_smooth(method = "lm", formula = y ~ poly(x, 2), color = "black") +
theme_bw() +
labs(title = "mpg ~ horsepower^2") +
theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



LOOCV: estimación de *test error* y selección de flexibilidad del modelo

En `R` se puede realizar LOOCV de cualquier *generalized linear model* creado mediante `glm()` empleando la función `cv.glm()` del paquete `boot`. La función `glm()` engloba diferentes tipos de modelos lineales que se especifican mediante el argumento `family`. En el caso de regresión lineal no se especifica ningún tipo de familia y es equivalente a emplear la función `lm()`. Para regresión logística se emplea `family=binomial`.

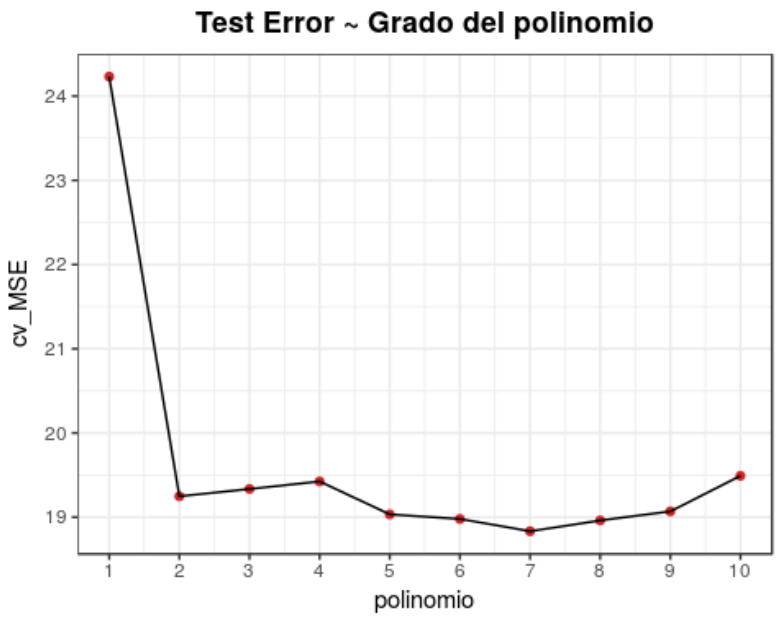
La función `cv.glm()` calcula el error de predicción mediante *cross-validation*. Si el argumento *k* no se especifica, se le asigna automáticamente el número de observaciones empleadas para crear el modelo, por lo que se realiza un *leave-one-out cross-validation*. La función devuelve una lista con múltiples componentes, el resultado de la validación se almacena dentro del vector `delta` que contiene la estimación de error con y sin corrección.

```
# Se genera el modelo lineal, dado que se va a emplear LOOCV no es necesario
# dividir las observaciones en dos grupos
modelo <- glm(mpg ~ horsepower, data = Auto)
# se emplea la función cv.glm() para la validación LOOCV
library(boot)
cv_error <- cv.glm(data = Auto, glmfit = modelo)
cv_error$delta
```

```
## [1] 24.23151 24.23114
```

La estimación de error del modelo lineal mediante LOOCV es de 24.23. Se ha obtenido un valor cercano al estimado mediante CV-simple pero eliminando el problema de la variabilidad. Al igual que se ha hecho en CV-simple, se puede emplear la validación LOOCV para identificar el grado de flexibilidad que permite obtener el mejor modelo.

```
cv_MSE <- rep(NA, 10)
for (i in 1:10) {
    modelo <- glm(mpg ~ poly(horsepower, i), data = Auto)
    cv_MSE[i] <- cv.glm(data = Auto, glmfit = modelo)$delta[1]
}
ggplot(data = data.frame(polinomio = 1:10, cv_MSE = cv_MSE),
       aes(x = polinomio, y = cv_MSE)) +
       geom_point(colour = c("firebrick3")) +
       geom_path() +
       scale_x_continuous(breaks = c(0:10)) +
       theme_bw() +
       labs(title = "Test Error ~ Grado del polinomio") +
       theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



Los resultados de la validación indican que polinomios superiores a grado 2 o 3 no aportan una mejora sustancial.

K-fold Cross-Validation: estimación de *test error* y selección de flexibilidad del modelo

La función `cv.glm()` del paquete `boot` puede emplearse para realizar *K-fold Cross-Validation* de cualquier modelo creado mediante la función `glm()`, especificando el número de grupos en el argumento `k`.

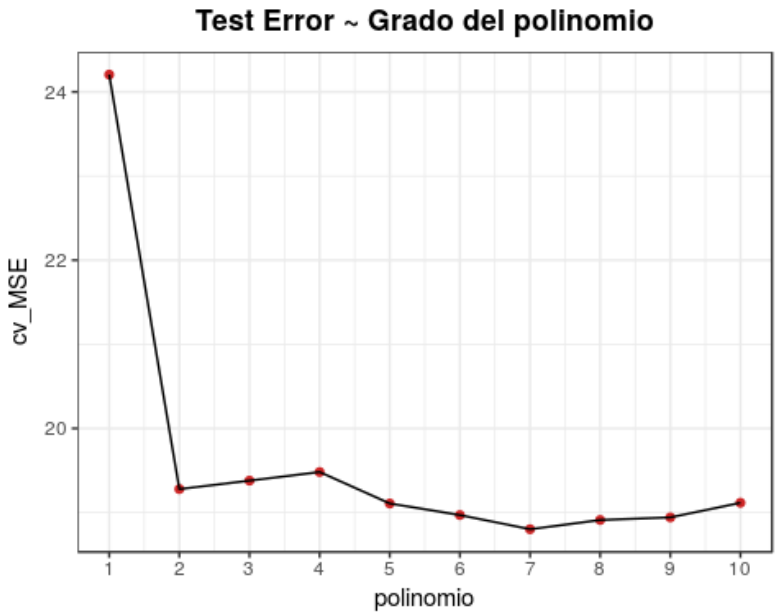
```
# Se genera el modelo lineal
modelo <- glm(mpg ~ horsepower, data = Auto)
# se emplea la función cv.glm() para la validación, empleando en este caso k=10
set.seed(1)
cv_error <- cv.glm(data = Auto, glmfit = modelo, K = 10)
cv_error$delta
```

```
## [1] 24.10716 24.09865
```

La estimación de error del modelo lineal mediante *K-fold Cross-Validation k=10* es de 24.1, un valor muy próximo al estimado mediante LOOCV, pero empleando mucho menos tiempo de computación. Cuando se especifica el número de grupos *K* a emplear en la validación, la función `cv.glm()` devuelve dos resultados, uno con corrección de continuidad y otro sin ella.

Si se estudia la influencia de la flexibilidad del modelo, los resultados muestran la misma tendencia que LOOCV. Se obtiene una mejora importante empleando una función cuadrática o cubica en comparación al modelo lineal. Para polinomios de grado > 3 la mejora es mínima.

```
cv_MSE_k10 <- rep(NA, 10)
for (i in 1:10) {
    modelo <- glm(mpg ~ poly(horsepower, i), data = Auto)
    set.seed(17)
    cv_MSE_k10[i] <- cv.glm(data = Auto, glmfit = modelo, K = 10)$delta[1]
}
ggplot(data = data.frame(polinomio = 1:10, cv_MSE = cv_MSE_k10),
       aes(x = polinomio, y = cv_MSE)) +
       geom_point(colour = c("firebrick3")) +
       geom_path() +
       scale_x_continuous(breaks = c(0:10)) +
       theme_bw() +
       labs(title = "Test Error ~ Grado del polinomio") +
       theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```



Bootstrap: estimación de la precisión de un modelo de regresión lineal

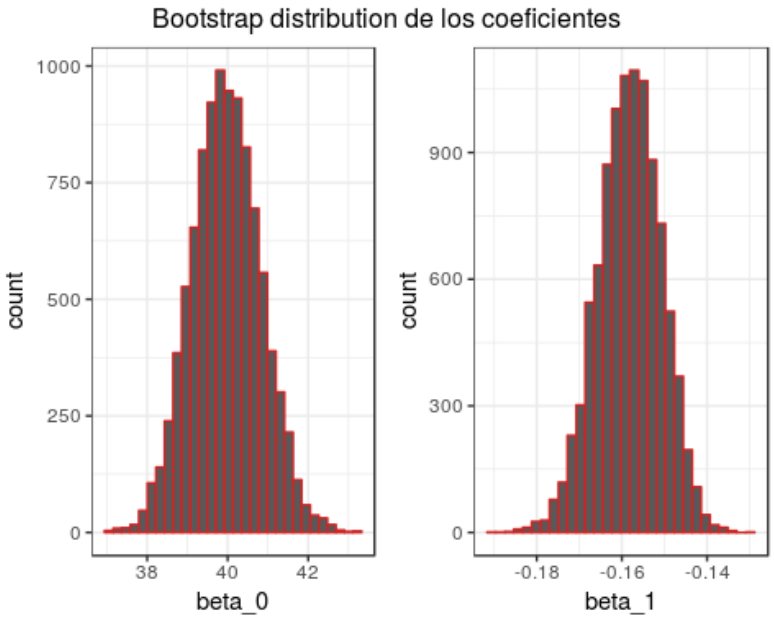
Para más información sobre *bootstrapping* consultar Resampling: Test de permutación, Simulación de Monte Carlo y Bootstrapping.

En el siguiente ejemplo se emplea el método bootstrap para estudiar la variabilidad en la estimación de los coeficientes de regresión β_0 y β_1 (intersección y pendiente) de un modelo de regresión lineal. El modelo en cuestión emplea la variable horsepower para predecir el consumo de un vehículo mpg, los datos empleados son del data set `Auto`.

En el caso de regresión lineal, es posible estimar la variabilidad de los coeficientes de regresión mediante fórmulas matemáticas (esta es la forma en la que lo calcula `R` por defecto y que se puede ver mediante `summary(modelo)`). Sin embargo, para que sean válidos los resultados, los residuos se tienen que distribuir de forma normal. La estimación mediante *bootstrap* no requiere de ninguna condición por lo que suele ser más precisa.

El proceso de *bootstraping* consiste en generar de forma iterativa diferentes modelos lineales, empleando en cada caso una *bootstrap-sample* creada mediante *resampling* del mismo tamaño que la muestra inicial. Para cada modelo ajustado se registran los valores de los coeficientes β_0 y β_1 y finalmente se estudia su distribución.

```
# Se define la función que devuelve el estadístico de interés, los coeficientes de
# regresión
fun_coeficientes <- function(data, index) {
    return(coef(lm(mpg ~ horsepower, data = data, subset = index)))
}
# Se implementa un bucle que genere los modelos de forma iterativa y
# almacene los coeficientes. El data frame Auto tiene 392 observaciones
beta_0 <- rep(NA, 9999)
beta_1 <- rep(NA, 9999)
for (i in 1:9999) {
    coeficientes <- fun_coeficientes(data = Auto,index = sample(1:392, 392,
                                     replace = TRUE))
    beta_0[i] <- coeficientes[1]
    beta_1[i] <- coeficientes[2]
}
# Se muestra la distribución de los coeficientes
p5 <- ggplot(data = data.frame(beta_0 = beta_0), aes(beta_0)) +
geom_histogram(colour = "firebrick3") +
theme_bw()
p6 <- ggplot(data = data.frame(beta_1 = beta_1), aes(beta_1)) +
geom_histogram(colour = "firebrick3") +
theme_bw()
grid.arrange(p5, p6, ncol = 2, top = "Bootstrap distribution de los coeficientes")
```



El valor del estadístico estimado mediante *bootstrapping* es la media de la *bootstrap-distribution* y la desviación estándar del estadístico la desviación estándar de la distribución:

- $\hat{\beta}_0$ = mean(beta_0) =39.9652841
- $SE(\hat{\beta}_0)$ = sd(beta_0) = 0.8568472
- $\hat{\beta}_1$ = mean(beta_1) = -0.1582257
- $SE(\hat{\beta}_1)$ = sd(beta_1) = 0.007442

Para este caso, si se comparan los valores estimados mediante *bootstapping* con los devueltos por la función `lm()` (obtenidos por *t-test*), se observa una diferencia muy pequeña.

```
summary(lm(mpg ~ horsepower, data = Auto))$coef
```

```
##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

El mismo resultado se puede obtener empleando la función `boot()`.

```
boot(data = Auto, statistic = fun_coeficientes, R = 9999)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = fun_coeficientes, R = 9999)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0314138953 0.856868372
## t2* -0.1578447 -0.0003515545 0.007397476
```

Bibliografía

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

Points of Significance: Sampling distributions and the bootstrap by Anthony Kulesa, Martin Krzywinski, Paul Blainey & Naomi Altman