Mayo 2018

Cristina Gil Martínez

MÉTODOS DE REGRESIÓN NO LINEAL

Apuntes personales sobre métodos de regresión no lineal (polinomios, step functions, regression splines, regresión local y GAMs) con ejemplos en R.

CONTENIDO

INTRODUCCIÓN

Los modelos lineales son relativamente simples de describir e implementar, y poseen la ventaja frente a otros modelos de ser fáciles de interpretar. Sim embargo, pueden estar limitados en cuanto a capacidad predictiva si la relación entre variables es más compleja. Algunos de los modelos que permiten modelar relaciones de tipo no lineal a la vez que intentan mantener un nivel de interpretabilidad alto son:

Regresión no lineal con un predictor

- **Regresión polinomial**: extensión del modelo lineal mediante la incorporación de predictores extra, obtenidos elevando a distintas potencias los predictores originales.
- ***Step functions:*** dividen el rango de la variable en *K* regiones distintas para generar una variable cualitativa. Tiene el efecto de ajustar una función constante a "trozos".
- ***Splines* de regresión:** más flexibles que la regresión polinomial y las *step functions* (de hecho, es una extensión de ambas). Consiste en dividir el rango de X en *K* intervalos distintos, y en cada intervalo se ajusta una función polinomial, restringidas para que estén unidas por los límites de cada intervalo o *knots*. Pueden producir un ajuste muy flexible.
- ***Smoothing splines:*** similar a *regression splines*, minimizan RSS sujeta a una penalización de suavizado.
- **Regresión local:** similar a las *splines* en cuanto al ajuste por regiones, pero difiere en que los intervalos pueden solaparse.

Regresión no lineal con múltiples predictores

- **Modelos aditivos generalizados:** Extensión de los métodos anteriores para modelar múltiples predictores.

REGRESIÓN POLINÓMICA

Se reemplaza el modelo lineal estándar

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

con una función polinomial

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_d x_i^d + \epsilon_i$$

Para un polinomio de grado *d* lo suficientemente elevado, la regresión polinómica permite ajustar una curva extremadamente no lineal, aunque no es habitual utilizar un polinomio mayor de grado 3 o 4 para evitar una flexibilidad excesiva u oscilaciones indeseables, sobre todo en los extremos del predictor *X*.

Los coeficientes β_0, ..., β_d se estiman mediante regresión lineal por mínimos cuadrados, ya que la ecuación no deja de ser una ecuación lineal.

La función polinómica también puede utilizarse para modelos de regresión logística (predecir una respuesta binaria).

$$\mathrm{P}\,(y_i > Y \mid x_i = X) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}$$

STEP FUNCTIONS

Las *step functions* evitan tener que imponer una función global a los datos, como en el caso de la regresión polinomial, ya que el rango del predictor *X* se subdivide en intervalos (**c_1, c_2, ..., c_K**), ajustándose una **constante** distinta a cada región. Esto equivale a convertir una variable continua en una variable categórica ordenada. Con ***K* intervalos** obtendríamos ***K + 1 nuevas variables***:

$$\begin{aligned} C_0(X) &= I(X < c_1), \\ C_1(X) &= I(c_1 \leq X < c_2), \\ C_2(X) &= I(c_2 \leq X < c_3), \\ &\vdots \\ C_{K-1}(X) &= I(c_{K-1} \leq X < c_K), \\ C_K(X) &= I(c_K \leq X) \end{aligned}$$

donde *I (·)* es una función indicadora donde las **variables *dummy*** que se crean para cada intervalo (***$C_0(X)$*, ..., $C_K(X)$**) toman el valor 1 si la X se encuentra dentro del correspondiente intervalo, y 0 si esto no se cumple. Ya que cada valor de *X* debe encontrarse en solo uno de los *K* + 1 intervalos, solo una de las variables *dummy* tomará el valor de 1 para cada valor del predictor.

Interpretación de los coeficientes

Cuando $X < c_1$, todos los predictores son 0, por lo que **β_0** se interpreta como el valor medio de Y cuando $X < c_1$, mientras que **β_j** representa el incremento medio en la respuesta para X en $c_j \leq X < c_{j+1}$ en relación a $X < c_1$.

El hecho de que el predictor no cuente con puntos de corte naturales puede suponer una **desventaja** a la hora de aplicar *step functions*, ya que la relación real entre predictor y variable respuesta puede no modelarse correctamente.

SPLINES DE REGRESIÓN

Las *splines* de regresión o *regression splines* agrupan un conjunto de métodos que suponen una extensión de la regresión polinómica y *step functions*.

Piecewise polynomials

En lugar de ajustar un polinomio de grado *d* en todo el rango del predictor X, se ajusta un polinomio de grado menor sobre diferentes intervalos del mismo, es decir, se ajustan diferentes funciones polinómicas a los datos.

Los coeficientes β_d difieren entre las diferentes regiones. Los puntos que delimitan los intervalos se denominan ***knots*** (c_1, c_2, ..., c_K). Por ejemplo, un *piecewise polynomial* con un solo *knot* en el punto *c* tomaría la siguiente forma

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c \end{cases}$$

donde las observaciones $x_i < c$ se ajustarán mediante una función polinómica con coeficientes β_{01}, β_{11}, β_{21}, β_{31}; y las observaciones $x_i \geq c$ se ajustarán con una función polinómica con coeficientes β_{02}, β_{12}, β_{22}, β_{32}. Cada una de estas funciones se ajustan usando **mínimos cuadrados**.

Con el aumento del número de *knots*, se consigue un ajuste más flexible. Teniendo *K knots*, ajustaremos *K* + 1 *piecewise polynomials*.

El **problema** o desventaja con la que cuentan las *piecewise polynomials* es la discontinuidad de las funciones en los *knots*.

Restricciones y *splines*

Para resolver el problema de la discontinuidad de las funciones en las regiones de los *knots*, se aplica la restricción de que la curva de ajuste final entre intervalos sea continua. Sin embargo, la transición entre *knots* puede ser abrupta o poco suave. Para solucionarlo puede aplicarse la restricción adicional de que las ***d* -1 derivadas** sean **continuas** en los puntos de corte.

Definición de *spline*: curva definida a trozos mediante polinomios. Se utilizan para aproximar curvas con formas complicadas. Producen buenos resultados con polinomios de grado bajo evitando así las oscilaciones, indeseables en la mayoría de las aplicaciones, que se producen con polinomios de grado elevado.

En general, un ***spline* de grado *d***, o lo que es lo mismo, un *piecewise polynomial* de grado *d*, con *K knots* usa un total de ***d* + *K* grados de libertad**. Cada restricción que se impone al modelo libera un grado de libertad.

Ejemplo de *spline* muy utilizado: *cubic spline* (*spline* de grado 3):

Para ajustar un *cubic spline* a los datos con *K knots*, se lleva a cabo una regresión por mínimos cuadrados con ordenada en el origen y 3 + *K* predictores, de la forma X, X_2, X_3, $h(X, \xi_1)$, ..., $h(X, \xi_K)$ donde ξ_1, ..., ξ_K son los *knots*. Esto equivale a estimar un total de *K* + 4 coeficientes de regresión o grados de libertad.

Los *splines* pueden sufrir de **alta varianza en los extremos** del predictor (en las regiones superior e inferior solo hay restricción de continuidad en un extremo). En este caso, puede solucionar el problema un ***spline natural,*** un *spline* de regresión con constricciones adicionales en los extremos: la función ha de ser lineal en los extremos o regiones donde *X* es menor que el *knot* más pequeño, y mayor que el *knot* más grade. Esto hace que los *splines* naturales produzcan generalmente estimaciones más estables en los extremos del rango.

Elección de posición y número de *knots*

Posición:

El *spline* de regresión se hace más flexible en aquellas zonas que contienen más *knots*, ya que los coeficientes polinómicos cambian rápidamente. En la práctica es común situar *knots* de manera uniforme a lo largo de la función, en lugar de situar más donde la función pueda variar más. Esto se hace especificando el número de grados de libertad, siendo el software el que automáticamente coloca los *knots* en cuantiles uniformes de los datos.

Número:

El número de *knots* será equivalente a los grados de libertad del *spline*. Una opción es probar diferentes números de *knots* y analizar con cuantos se obtiene la mejor curva, pero una opción más objetiva supone recurrir a la validación cruzada (donde escogeríamos el número de *knots* que minimice la suma de residuos al cuadrado, RSS).

Splines de regresión vs regresión polinómica

Los *splines* de regresión suelen dar mejores resultados que la regresión polinómica, esto es así porque a diferencia de los polinomios (que deben usar un grado alto para aumentar la flexibilidad), los *splines* introducen flexibilidad al modelo aumentando el número de *knots* y manteniendo fijos los grados de libertad. Generalmente, este enfoque produce estimaciones más estables. A diferencia de los polinomios, los *splines* también nos permiten aumentar la flexibilidad solo en aquellas regiones donde la función *f* parezca cambiar rápidamente.

SPLINES DE SUAVIZADO

Los *splines* de suavizado o *smoothing splines*, utilizan un enfoque diferente a las *splines* de regresión para ajustar el *spline* (en este caso **mínimos cuadrados no es aplicable**). El objetivo es obtener un ajuste o función *g(x)* que minimice RSS pero que sea suave, sin sobreajustarse a los datos ni ser demasiado flexible. La forma de llevar esto a cabo es encontrar la función *g* que minimice

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

donde **λ** es el **parámetro de penalización** o *tuning parameter* (no negativo), que penaliza la variabilidad en *g,* y *g"(t)* el término que indica la segunda derivada de la función *g.* Esta ecuación toma la forma de "Pérdida (RSS) + Penalización" que podemos encontrar en el contexto de *ridge regression* y *lasso*. Cuanto mayor sea λ, más suave o menos variable será la función. Si λ = 0, el término de penalización no tiene efecto, por lo que la función *g* será muy variable, llegando a interpolar los datos. Conforme aumente λ, más suave será la función (λ -> ∞ equivaldría a un ajuste lineal por mínimos cuadrados). Por lo tanto, λ juega el papel de controlar el equilibrio *bias*-varianza y los **grados de libertad efectivos**, definidos mediante

$$df_\lambda = \sum_{i=1}^{n} \{\mathbf{S}_\lambda\}_{ii}$$

lo que equivale a la suma de los elementos de la diagonal de la matriz S_λ (matriz n x n correspondiente a S_λ veces (para la cual hay una fórmula) el vector respuesta *y*).

Interpretación de la derivada e integral de una función:

La primera derivada mide la pendiente de la función en *t*, mientras que la **segunda derivada** mide el cambio de la pendiente: un valor alto indica que la función *g(t)* es muy ondulada o variante cerca de *t*. Por ejemplo, la segunda derivada de una línea recta es 0. La notación ∫ hace referencia a la integral, que podemos interpretar como la suma sobre el rango de t. Por lo tanto, ∫ $g"(t)^2$ mide el cambio total en la función *g'(t)* a lo largo de todo su rango. Si *g* es muy suave o poco variable, *g'(t)* se aproximará a constante y ∫ $g"(t)^2$ tomará un valor pequeño. Por el contrario, si *g* es muy variable, *g'(t)* variará significativamente y ∫ $g"(t)^2$ tomará un valor alto.

En definitiva, un *spline* de suavizado sería equivalente a un *spline* natural con *knots* en cada valor de x_i (por lo que no es necesario elegir el número ni la posición de los mismos), con restricción de variabilidad impuesta por λ (valor que sí debemos elegir).

Elección del parámetro de penalización λ

Podemos encontrar el valor óptimo de λ mediante **validación cruzada** (valor de λ que minimice RSS con los datos de test). Para las *splines* de suavizado, el método LOOCV puede calcularse de manera muy efectiva usando la siguiente fórmula:

$$\mathrm{RSS}_{cv}(\lambda) = \sum_{i=1}^{n}(y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^{n}\left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}}\right]^2.$$

REGRESIÓN LOCAL

La regresión local o *local regression* (*LOESS*) utiliza un enfoque distinto para el ajuste de curvas, que supone calcular el ajuste en cada punto x_0 usando solamente las observaciones de entrenamiento próximas.

El algoritmo se describe a continuación:

1. Obtener la fracción de datos de entrenamiento $s = k / n$ cuyos x_i están más próximos a x_0.
2. Asignar una función de peso $K_{i0} = K(x_i, x_0)$ a cada punto en esta vecindad, de manera que el punto más alejado de x_0 tenga peso 0, y el más próximo tenga el mayor peso. A excepción de estos k vecinos más cercanos, el resto tienen peso 0.
3. Ajustar una regresión ponderada por mínimos cuadrados de y_i sobre x_i usando los pesos citados anteriormente, obteniendo estimadores de β_0 y β_1 que minimicen

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$$

(en este caso, correspondiente a una regresión lineal)

4. El valor ajustado en x_0 está dado por

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

Aunque la selección del **grado de polinomio** para cada ajuste individual es algo a tener en cuenta, la elección más importante en este algoritmo es el ***span*** (*s*), que juega un papel como el de λ en las *splines* de suavizado ya que controla el nivel de flexibilidad del ajuste no lineal. Cuanto menor sea el valor de *s* (**porcentaje de observaciones en la vecindad**), más local y variable será el ajuste ya que se tendrán en cuenta menor observaciones vecinas. Por el contrario, un valor muy alto de *s* llevaría a un ajuste global de los datos usando todas las observaciones de entrenamiento. De nuevo podemos recurrir a validación cruzada para escoger el valor óptimo de *s*.

La regresión local puede generalizarse a un escenario con múltiples predictores, sin embargo resulta ineficiente en casos con más de 3 o 4 predictores por la falta de observaciones cercanas a x_0.

*Un punto a tener en cuenta respecto a la regresión local es la baja interpretabilidad del modelo.

MODELOS ADITIVOS GENERALIZADOS (GAM)

Los modelos aditivos generalizados o *generalized additive models* suponen una extensión de la regresión lineal múltiple de igual manera que los métodos descritos anteriormente suponen una extensión de la regresión lineal simple, para ajuste de modelos flexibles. Los GAMs también pueden aplicarse con **variables respuesta cuantitativas o cualitativas**.

GAMs para problemas de regresión

Una manera de extender el modelo de regresión lineal múltiple con *p* predictores

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

para incorporar relaciones no lineales entre cada predictor y la variable respuesta es el reemplazo de cada componente lineal $\beta_p x_{ip}$ con una función no lineal $f_j(x_{ij})$:

$$\begin{aligned} y_i &= \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i \\ &= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i \end{aligned}$$

Se denomina **modelo aditivo** porque se calcula una f_j para cada X_j, sumándose todas las contribuciones. De hecho, para cada predictor podemos emplear un método de ajuste distinto, de los descritos anteriormente (regresión polinómica, *splines*, regresión local, etc.), aplicables a un solo predictor.

GAMs para problemas de clasificación

Los modelos GAM también se pueden usar en situaciones donde *Y* es una variable cualitativa. A partir del modelo logístico

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

podemos recordar que la transformación *logit* corresponde al logaritmo de *odds* de $P(Y = 1 \mid X)$ versus $P(Y = 0 \mid X)$. Incorporando funciones no lineales $f_p(X_p)$ entre variables obtenemos el modelo el modelo de regresión logística GAM

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$$

Ventajas e inconvenientes de los GAMs

- Los modelos GAM nos permiten ajustar una función no lineal a cada X_j, por lo que no es necesario incorporar transformaciones a cada variable manualmente.
- Los ajustes no lineales que nos permite el modelo GAM pueden llegar a conseguir predicciones más precisas para la variable respuesta.
- Podemos examinar el efecto de cada X_j sobre Y de manera individual, manteniendo el resto de variables fijas, ya que es un modelo aditivo.
- La suavidad de la función f_j para la variable X_j puede resumirse mediante los grados de libertad.
- La mayor limitación de los GAMs es que el modelo está restringido a ser aditivo, lo que supone que en el caso de contar con muchas variables, ciertas interacciones importantes pueden perderse. Sin embargo, al igual que con la regresión lineal, podemos incluir manualmente términos de interacción incluyendo predictores adicionales de la forma $X_j \times X_k$.

Consejo: analizar por separado cada predictor, identificando qué modelo (función) de ajuste funciona mejor, para finalmente combinarlos todos. Tener en cuenta que no podemos estar seguros de que, la función que se ajusta bien para un predictor solo, también lo haga en presencia de otros una vez se combinan, por lo que es aconsejable probar varias opciones para intentar obtener la mejor.

EJEMPLOS EN R

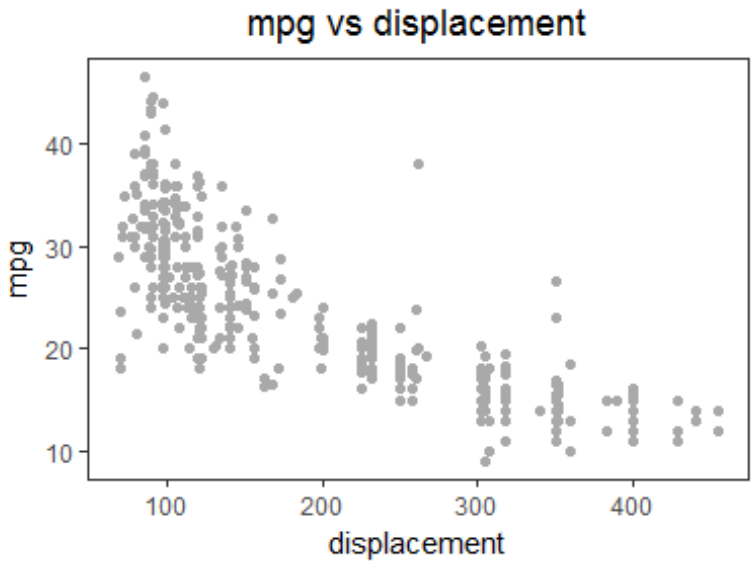
Con el set de datos *Auto* (información sobre vehículos), trataremos de predecir *mpg* (consumo de combustible, en millas/galón) en función de la variable predictora *displacement* (desplazamiento del motor, en pulgadas cúbicas), ajustando y comparando distintos tipos de modelo no lineal aplicables a un solo predictor. Junto con la variable *horsepower* (potencia del motor), aplicaremos un modelo GAM.

(Ver capítulo Regresión lineal múltiple para análisis adicionales con este set de datos.)

```
library(ISLR)
str(Auto)

## 'data.frame':    392 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 1
4 161 141 54 223 241 2 ...

library(ggplot2)
ggplot(data = Auto, aes(x = displacement, y = mpg)) +
geom_point(col = "darkgrey") +
ggtitle("mpg vs displacement") +
theme_bw() +
theme(panel.background = element_blank(), panel.grid.major = element_blank(),
      panel.grid.minor = element_blank()) +
theme(plot.title = element_text(hjust = 0.5))
```



Antes de comenzar ajustando los modelos, dividiremos el set de datos en un conjunto de entrenamiento (80%) y test (20%):

```
set.seed(1)
# Índice observaciones de entrenamiento
train <- sample(nrow(Auto), 0.8*nrow(Auto), replace = FALSE)
datosA.train <- Auto[train, ]
datosA.test <- Auto[-train, ]

nrow(datosA.train) + nrow(datosA.test)

## [1] 392
```

Regresión polinómica

AJUSTE DEL MODELO

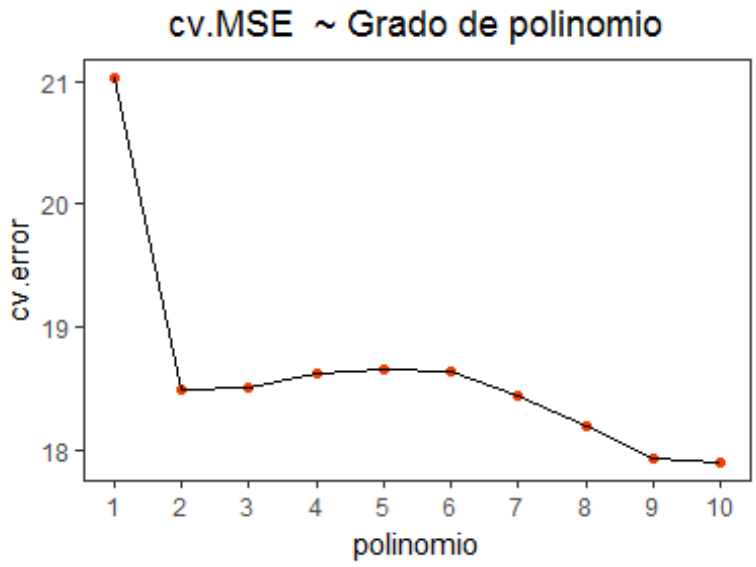
SELECCIÓN DEL MEJOR GRADO DE POLINOMIO MEDIANTE VALIDACIÓN CRUZADA

En lugar de ajustar el modelo polinómico con un valor de polinomio escogido al azar, recurriremos a la validación cruzada (10-*fold cross validation*) para escoger el valor óptimo en función del error de validación.

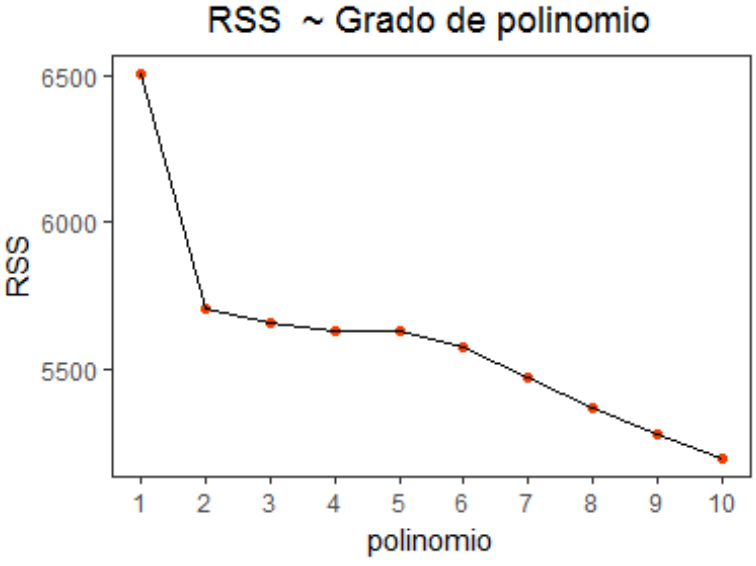
NOTA: en casos de regresión logística, utilizar I( ) para la variable respuesta en la fórmula dentro de la función glm(), indicando además el argumento family = "binomial".

```
library(boot)
# Vector para almacenar el error de validación de cada polinomio
cv.error <- rep(NA, 10)
# Vector para almacenar el RSS de cada polinomio
rss <- rep (NA, 10)
for (i in 1:10){
modelo.poli <- glm(mpg ~ poly(displacement, i), data = datosA.train)
set.seed(2)
cv.error[i] <- cv.glm(datosA.train, modelo.poli, K = 10)$delta[1]
rss[i] <- sum(modelo.poli$residuals^2)
}

ggplot(data = data.frame(polinomio = 1:10, cv.error = cv.error),
       aes(x = polinomio, y = cv.error)) +
geom_point(color = "orangered2") +
geom_path() +
scale_x_continuous(breaks = 0:10) +
labs(title = "cv.MSE  ~ Grado de polinomio") +
theme_bw() +
theme(panel.background = element_blank(), panel.grid.major = element_blank(), pane
l.grid.minor = element_blank()) +
theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot(data = data.frame(polinomio = 1:10, RSS = rss), aes(x = polinomio,
                                                           y = RSS)) +
geom_point(color = "orangered2") +
geom_path() +
scale_x_continuous(breaks=0:10) +
labs(title = "RSS  ~ Grado de polinomio") +
theme_bw() +
theme(panel.background = element_blank(), panel.grid.major = element_blank(), pane
l.grid.minor = element_blank()) +
theme(plot.title = element_text(hjust = 0.5))
```



Como es de esperar, el RSS disminuye conforme aumenta el grado del polinomio, ya que el modelo se ajusta cada vez más a los datos.

```r
# Polinomio con menor error de validación
which.min(cv.error)

## [1] 10
```

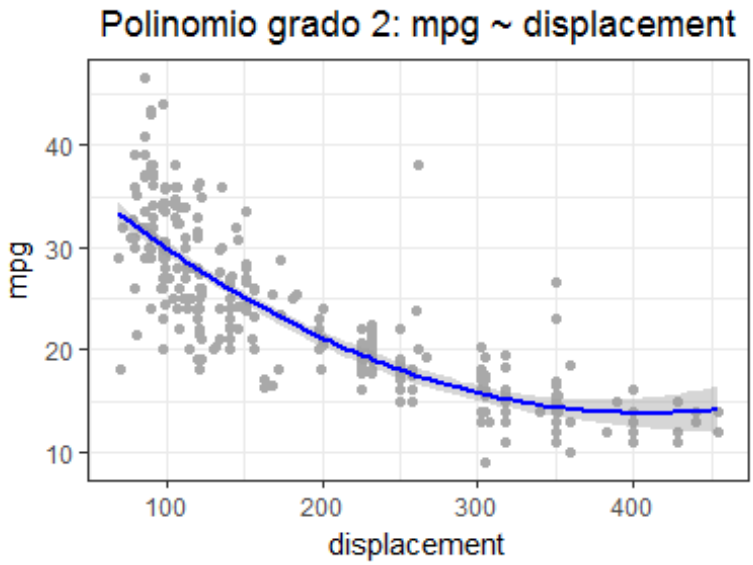
El polinomio que en este caso minimiza el error de validación (MSE) es el de grado 10. Sin embargo, observando el gráfico de la evolución del error de validación, decidiremos ajustar el modelo con el valor de polinomio 2 para evitar sobreajustar el modelo a los datos con un polinomio tan elevado.

```r
modelo.poli <- lm(mpg ~ poly(displacement, 2), data = datosA.train)
summary(modelo.poli)

##
## Call:
## lm(formula = mpg ~ poly(displacement, 2), data = datosA.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1848  -2.1741  -0.3295   2.0427  20.5989
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               23.2463     0.2426  95.838  < 2e-16 ***
## poly(displacement, 2)1 -109.8592     4.2913 -25.600  < 2e-16 ***
## poly(displacement, 2)2   28.2263     4.2913   6.578 2.03e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.291 on 310 degrees of freedom
## Multiple R-squared:  0.6927, Adjusted R-squared:  0.6907
## F-statistic: 349.3 on 2 and 310 DF,  p-value: < 2.2e-16
```

Todos los grados de polinomio resultan ser significativos en el ajuste del modelo (*p-value* < 0,05).

```r
ggplot(data = datosA.train, aes(x = displacement, y = mpg)) +
geom_point(col = "darkgrey") +
geom_smooth(method = "lm", formula = y ~ poly(x, 2), color = "blue", se = TRUE,
            level = 0.95) +
labs(title = "Polinomio grado 2: mpg ~ displacement") +
theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```



SELECCIÓN DEL MEJOR GRADO DE POLINOMIO MEDIANTE ANOVA

Una alternativa al uso de la validación cruzada para seleccionar el mejor modelo de entre un conjunto con distinto grado de polinomio, es el contraste de hipótesis ANOVA (test F). En este contraste se evalúa la hipótesis nula de que el modelo M1 es suficiente para explicar los datos, frente a la hipótesis alternativa de que un modelo más complejo M2 es mejor. Para poder aplicar ANOVA en estos casos, los **modelos** han de ser **anidados**.

```
modelo1 <- lm(mpg ~ displacement, data = datosA.train)
modelo2 <- lm(mpg ~ poly(displacement, 2), data = datosA.train)
modelo3 <- lm(mpg ~ poly(displacement, 3), data = datosA.train)

anova(modelo1, modelo2, modelo3)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ displacement
## Model 2: mpg ~ poly(displacement, 2)
## Model 3: mpg ~ poly(displacement, 3)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1    311 6505.5
## 2    310 5708.8  1    796.73 43.5156 1.824e-10 ***
## 3    309 5657.5  1     51.30  2.8019   0.09516 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

El contraste de ANOVA muestra que el modelo polinómico de grado 2 es superior al modelo lineal. Además, aumentar el polinomio a grado 3 no hace que el modelo sea significativamente superior.

EVALUACIÓN DEL MODELO

```
pred.modelo.poli <-  predict(modelo.poli, datosA.test)
test.error.poli <- mean((pred.modelo.poli - datosA.test$mpg)^2)
test.error.poli

## [1] 21.58356
```

Spline de regresión

```
    library(splines)
```

- **bs()** -> Genera la matriz base para un spline polinómico. El grado por defecto es 3 (cubic spline).
- **ns()** -> Genera la matriz base para un spline cúbico.
- **smooth.spline()** -> Ajusta un spline de suavizado cúbico.

AJUSTE DEL MODELO

Para ajustar un *spline* hemos de contar con una matriz de ecuaciones para su generación (proceso que lleva a cabo la función *bs()*), con un número de *knots* especificados. Por defecto se generan *splines* cúbicas. En la función podemos especificar el número de *knots*, o los grados de libertad *df*. Con esta última opción, los *knots* son elegidos automáticamente por R (posicionados en cuantiles uniformes de los datos). Además, con el argumento *degree* podemos especificar el grado de polinomio para el *spline*, en lugar del grado 3 que se usa por defecto.

SELECCIÓN DE LOS GRADOS DE LIBERTAD MEDIANTE VALIDACIÓN CRUZADA

Emplearemos el método de 10-*fold cross validation* para encontrar el valor óptimo de grados de libertad, probando valores de entre 3 y 16. Puesto que el valor óptimo de polinomio escogido fue de 2, será necesario especificar el argumento *degree*, ya que el *spline* se ajusta automáticamente con un grado de polinomio 3.

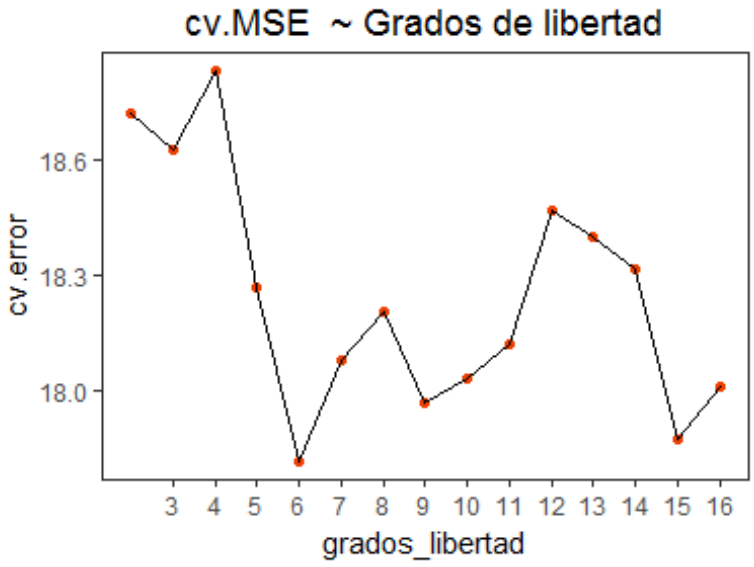
```
library(splines)
# Vector donde se almacenarán los errores de validación
cv.error <- rep(NA, 16)
# K-fold cross validation para cada valor de df
for(i in 2:16){
modelo.spline <- glm(mpg ~ bs(displacement, degree = 2, df = i),
                     data = datosA.train)
set.seed(3)
cv.error[i] = cv.glm(datosA.train, modelo.spline, K = 10)$delta[1]
```

```
}
```

```
# El primer valor del vector cv.error son NA ya que no se ha evaluado en el for lo
op el valor 1
head(cv.error)
```

```
## [1]       NA 18.72337 18.62538 18.83074 18.27122 17.81813
```

```
ggplot(data = data.frame(grados_libertad = 2:16, cv.error = cv.error[-1]),
       aes(x = grados_libertad, y = cv.error)) +
geom_point(color = "orangered2") +
geom_path() +
scale_x_continuous(breaks=3:16) +
labs(title = "cv.MSE  ~ Grados de libertad") +
theme_bw() +
theme(panel.background = element_blank(), panel.grid.major = element_blank(), pane
l.grid.minor = element_blank()) +
theme(plot.title = element_text(hjust = 0.5))
```




```
which.min(cv.error)
```

```
## [1] 6
```

En este caso, 6 son los grados de libertad que minimizan el error de validación, lo que en este caso corresponderá a 4 *knots*.

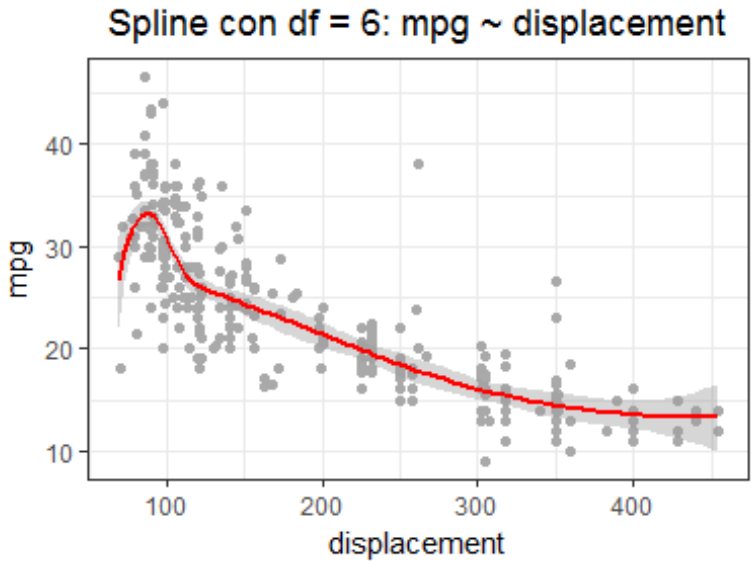
```
# Número y posición de los knots en función de los df
attr(x = bs(datosA.train$displacement, degree = 2, df = 6), which = "knots")
```

```
## 20% 40% 60% 80%
##  98 122 225 304
```

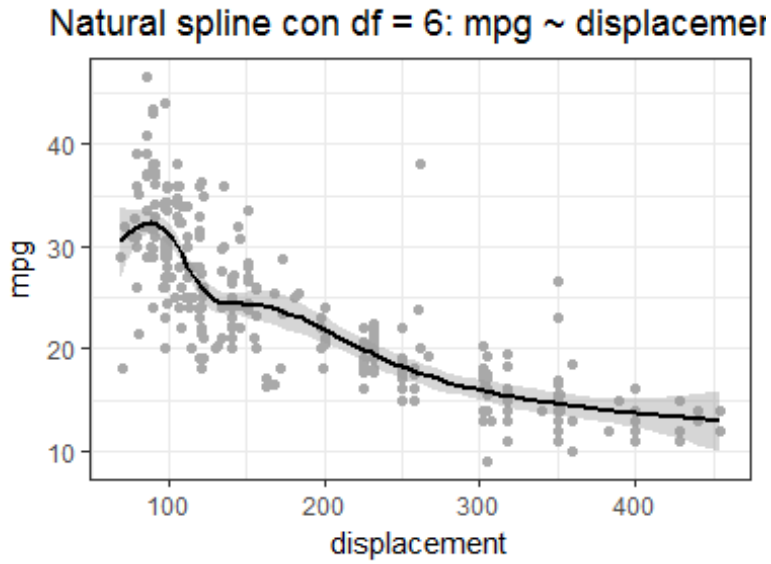
```
# Modelo spline
modelo.spline <- lm(mpg ~ bs(displacement, degree = 2, df = 6),
                    data = datosA.train)
summary(modelo.spline)
```

```
##
## Call:
## lm(formula = mpg ~ bs(displacement, degree = 2, df = 6), data = datosA.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.6694  -2.3010  -0.4294   2.1306  20.2003
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                           26.5191     2.2309  11.887  < 2e-16 ***
## bs(displacement, degree = 2, df = 6)1  10.5802     2.9502   3.586 0.000390 ***
## bs(displacement, degree = 2, df = 6)2   0.1433     2.2149   0.065 0.948467
## bs(displacement, degree = 2, df = 6)3  -3.4700     2.6569  -1.306 0.192521
## bs(displacement, degree = 2, df = 6)4  -9.0744     2.3698  -3.829 0.000156 ***
## bs(displacement, degree = 2, df = 6)5 -13.4888     2.6519  -5.087 6.37e-07 ***
## bs(displacement, degree = 2, df = 6)6 -13.2385     2.7761  -4.769 2.87e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.143 on 306 degrees of freedom
## Multiple R-squared:  0.7172, Adjusted R-squared:  0.7116
## F-statistic: 129.3 on 6 and 306 DF,  p-value: < 2.2e-16
```

```
# SPLINE
ggplot(data = datosA.train, aes(x = displacement, y = mpg)) +
geom_point(col = "darkgrey") +
geom_smooth(method = "lm", formula = y ~ bs(x, degree = 2, df = 6),
            color = "red", se = TRUE, level = 0.95) +
labs(title = "Spline con df = 6: mpg ~ displacement") +
theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```




```
# SPLINE NATURAL (por defecto, cúbico)
ggplot(data = datosA.train, aes(x = displacement, y = mpg)) +
geom_point(col = "darkgrey") +
geom_smooth(method = "lm", formula = y ~ ns(x, df = 6), color = "black",
            se = TRUE, level = 0.95) +
labs(title = "Natural spline con df = 6: mpg ~ displacement") +
theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```



Si queremos ajustar un *spline* de suavizado, hemos de aportar a la función el valor de λ, que podemos seleccionar mediante validación cruzada:

```
# SPLINE DE SUAVIZADO
modelo.spline.s <- smooth.spline(x = datosA.train$displacement,
                                 y = datosA.train$mpg, cv = TRUE)
```

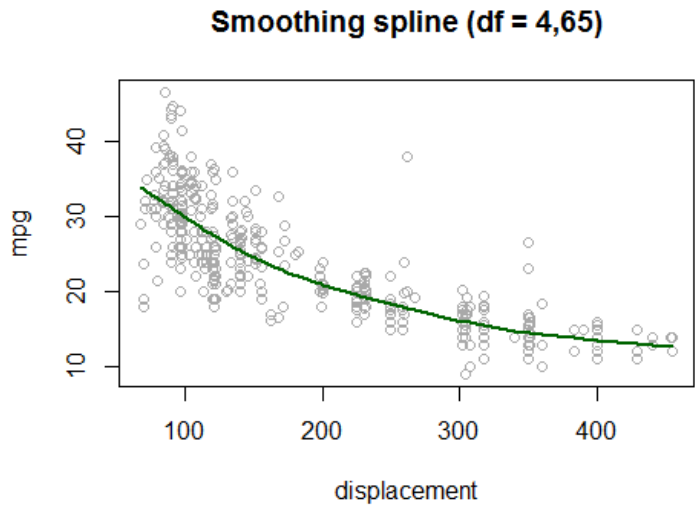
```r
# Grados de libertad y lambda correspondientes al modelo ajustado
modelo.spline.s$df

## [1] 4.651136

modelo.spline.s$lambda

## [1] 0.02108725

plot(mpg ~ displacement, data = Auto, col = "darkgrey")
title("Smoothing spline (df = 4,65)")
lines(modelo.spline.s, col = "darkgreen", lwd = 2)
```



EVALUACIÓN DEL MODELO

```r
pred.modelo.spline <-  predict(modelo.spline, datosA.test)
test.error.spline <- mean((pred.modelo.spline - datosA.test$mpg)^2)
test.error.spline

## [1] 18.66799
```

Regresión local

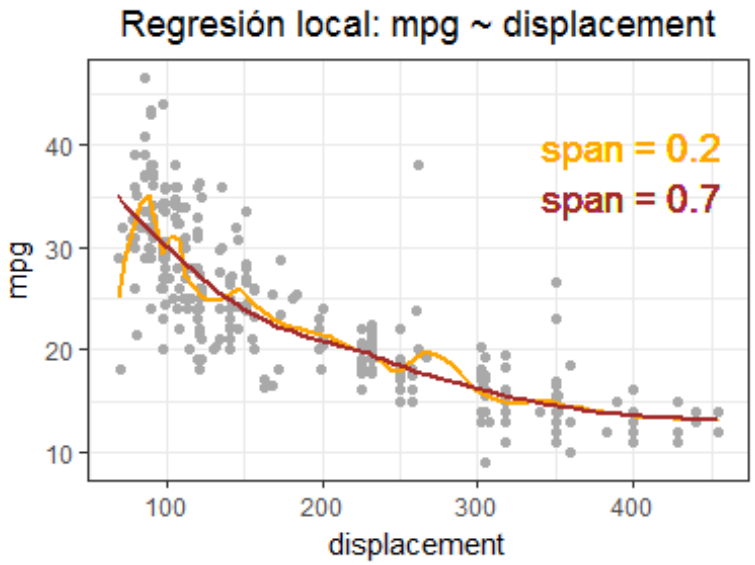
- **loess()** -> Ajuste local de un modelo polinómico

AJUSTE DEL MODELO

El parámetro a ajustar en regresión local es el *span* (% de observaciones vecinas a considerar en cada ajuste). Cuanto mayor sea el *span*, más suave será el ajuste.

```
# REGRESIÓN LOCAL
modelo.local <- loess(mpg ~ displacement, span = 0.7, data = datosA.train)

ggplot(data = datosA.train, aes(x = displacement, y = mpg)) +
geom_point(col = "darkgrey") +
geom_smooth(method = "loess", formula = y ~ x, span = 0.2,
            color = "orange", se = F) +
geom_smooth(method = "loess", formula = y ~ x, span = 0.7,
            color = "brown", se = F) +
labs(title = "Regresión local: mpg ~ displacement") +
theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
geom_text(aes(label = "span = 0.2", x = 400, y = 40), size = 5,
          colour = "orange") +
geom_text(aes(label = "span = 0.7", x = 400, y = 35), size = 5,
          colour = "brown")
```



EVALUACIÓN DEL MODELO

Evaluamos el modelo de regresión local con *span* = 0,7:

```
pred.modelo.local <-  predict(modelo.local, datosA.test)
test.error.local <- mean((pred.modelo.local - datosA.test$mpg)^2)
test.error.local
```

```
## [1] 22.95368
```

Step function

- `lm(y ~ cut(x, i)` -> Divide el rango del predictor en intervalos, y codifica sus valores según el intervalo en el que se encuentran.

Dada una variable numérica e indicando el número de cortes *n*, *cut()* establece los puntos de corte en el vector, convirtiéndola en una variable categórica con intervalos, y la función *lm()* creará una variable *dummy* a partir de esta conversión. Si deseamos especificar nuestros propios puntos de corte directamente, podemos indicarlo con el argumento *breaks*.

SELECCIÓN DEL NÚMERO DE CORTES MEDIANTE VALIDACIÓN CRUZADA

En este ejemplo, evaluaremos hasta un total de 7 puntos de corte mediante *k-fold cross validation* para escoger el número óptimo de los mismos:

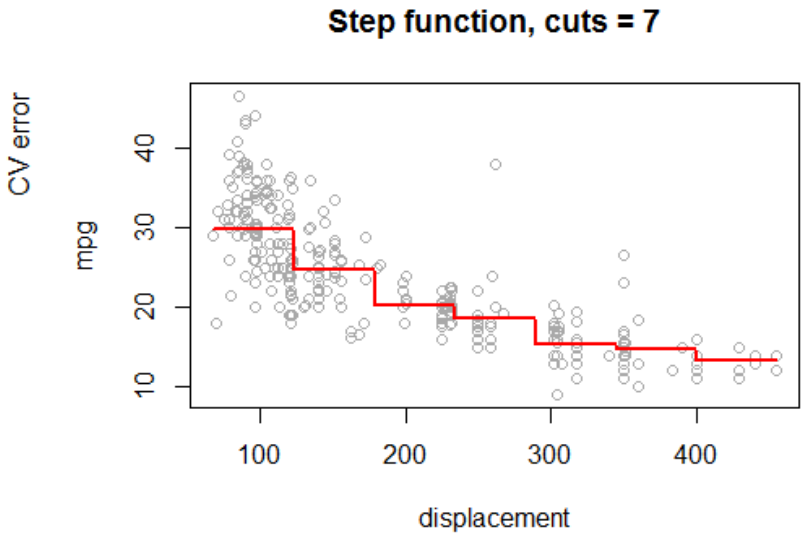
```
cv.error <- rep(NA, 7)

for (i in 2:7) {
datosA.train$dis.cut <- cut(datosA.train$displacement, i )
modelo.step <- glm(mpg ~ dis.cut, data = datosA.train)
set.seed(4)
cv.error[i] <- cv.glm(datosA.train, modelo.step, K = 10)$delta[1]
}

cv.error
```

```
## [1]       NA 37.29260 24.80680 24.06264 22.02786 21.79731 20.93323
```

```
plot(2:7, cv.error[-1], xlab="Número de cortes", ylab="CV error", type="l",
     pch=20, lwd=2)
points(which.min(cv.error), cv.error[which.min(cv.error)], col = "red", cex = 2,
       pch = 20)
```



El mínimo error de validación se consigue con 7 cortes.

```
modelo.step = lm(mpg ~ cut(displacement, 7), data = datosA.train)
summary(modelo.step)

##
## Call:
## lm(formula = mpg ~ cut(displacement, 7), data = datosA.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.8571  -2.3529  -0.3529   2.2333  19.4520
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     29.8571     0.4058  73.568  < 2e-16 ***
## cut(displacement, 7)(123,179]  -5.2245     0.7848  -6.657 1.29e-10 ***
## cut(displacement, 7)(179,234]  -9.7046     0.8268 -11.738  < 2e-16 ***
## cut(displacement, 7)(234,289] -11.3091     0.9974 -11.338  < 2e-16 ***
## cut(displacement, 7)(289,344] -14.6009     0.9018 -16.191  < 2e-16 ***
## cut(displacement, 7)(344,400] -15.1905     0.9661 -15.724  < 2e-16 ***
## cut(displacement, 7)(400,455] -16.5042     1.1771 -14.021  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.556 on 306 degrees of freedom
## Multiple R-squared:  0.6581, Adjusted R-squared:  0.6514
## F-statistic: 98.17 on 6 and 306 DF,  p-value: < 2.2e-16

lim.dis <- range(datosA.train$displacement)
nuevos.dis <- seq(from = lim.dis[1], to = lim.dis[2])
pred <- predict(modelo.step, data.frame(displacement = nuevos.dis))
plot(mpg ~ displacement, data = datosA.train, col="darkgrey")
lines(nuevos.dis, pred, col="red", lwd=2)
title("Step function, cuts = 7")
```

GAM

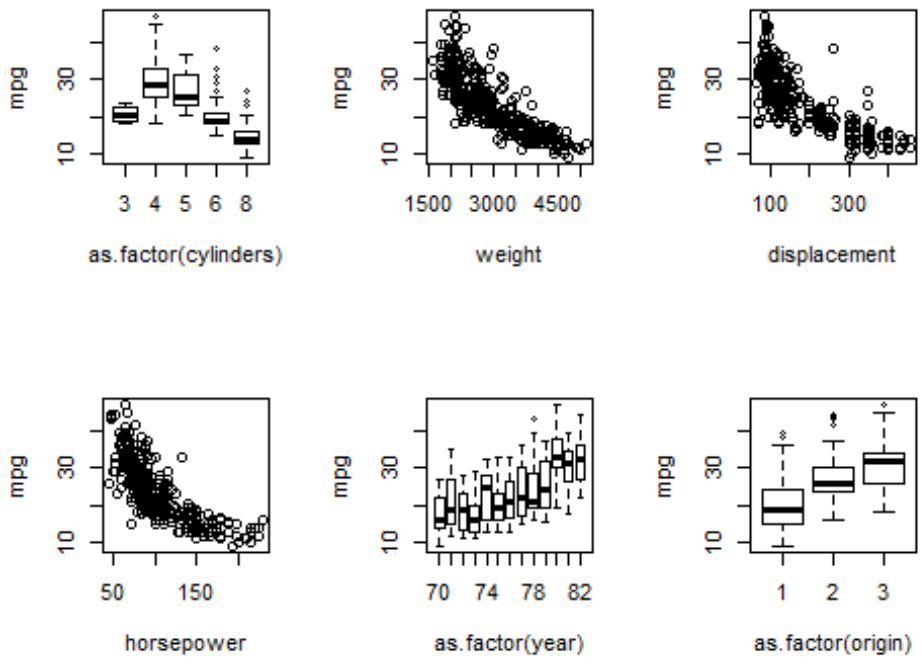
```
library(gam)
```

- **gam()** -> Ajuste de modelos aditivos generalizados. Usa el algoritmo backfitting para combinar diferentes métodos de ajuste o suavizado. Los métodos actualmente soportados son regresión local **lo()** y splines de suavizado **s()**.

AJUSTE DEL MODELO

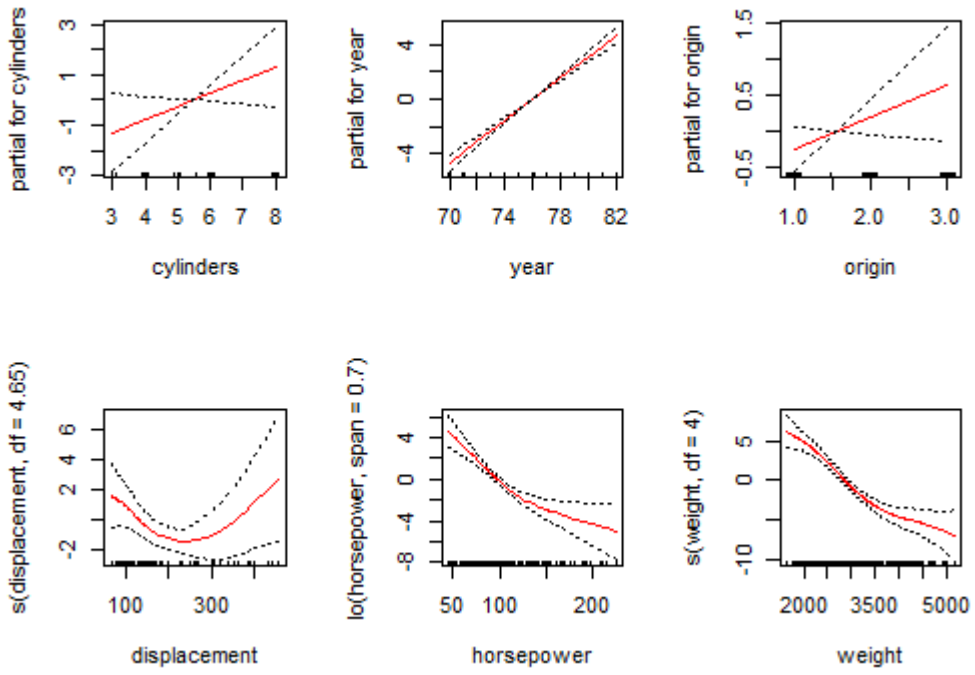
Con un modelo GAM podemos incluir múltiples predictores, pudiendo incluir distintos tipos de relación lineal o no lineal entre predictores.

```
par(mfrow=c(2,3))
attach(Auto)
plot(mpg ~ as.factor(cylinders))
plot(mpg ~ weight)
plot(mpg ~ displacement)
plot(mpg ~ horsepower)
plot(mpg ~ as.factor(year))
plot(mpg ~ as.factor(origin))
```




```
library(leaps)
library(gam)
# Modelo GAM
modelo.gam <- gam(mpg ~ cylinders + year + origin + s(displacement, df = 4.65) + l
o(horsepower, span = 0.7) + s(weight, df = 4), data = datosA.train)
```

```r
par(mfrow = c(2, 3))
plot(modelo.gam, se = T, col = "red")
```




```r
summary(modelo.gam)
```

```
##
## Call: gam(formula = mpg ~ cylinders + year + origin + s(displacement,
##     df = 4.65) + lo(horsepower, span = 0.7) + s(weight, df = 4),
##     data = datosA.train)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -8.70353 -1.61375  0.01309  1.56555 11.91471
##
## (Dispersion Parameter for gaussian family taken to be 7.8735)
##
##     Null Deviance: 18574.54 on 312 degrees of freedom
## Residual Deviance: 2343.78 on 297.6798 degrees of freedom
## AIC: 1551.064
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                                Df  Sum Sq Mean Sq  F value    Pr(>F)
## cylinders                    1.00 10401.9 10401.9 1321.132 < 2.2e-16 ***
## year                         1.00  2849.2  2849.2  361.874 < 2.2e-16 ***
## origin                       1.00   112.5   112.5   14.288 0.0001895 ***
## s(displacement, df = 4.65)   1.00   904.9   904.9  114.935 < 2.2e-16 ***
## lo(horsepower, span = 0.7)   1.00   495.7   495.7   62.953 4.328e-14 ***
## s(weight, df = 4)            1.00   496.4   496.4   63.044 4.168e-14 ***
## Residuals                  297.68  2343.8     7.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Anova for Nonparametric Effects
##                              Npar Df Npar F     Pr(F)
## (Intercept)
## cylinders
## year
## origin
## s(displacement, df = 4.65)      3.7 6.6055 7.716e-05 ***
## lo(horsepower, span = 0.7)      1.7 8.9229 0.0004395 ***
## s(weight, df = 4)               3.0 7.6752 5.875e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

El contraste de ANOVA muestra una fuerte evidencia de relación no lineal de *displacement*, *horsepower* y *weight* frente a *mpg*. (Los *p-values* corresponden a la H_0 de una relación lineal).

EVALUACIÓN DEL MODELO

Por último, con la función *predict ()* obtenemos el test error del modelo evaluándolo con el conjunto de datos de test:

```
pred.modelo.gam <-  predict(modelo.gam, datosA.test)
test.error.gam <- mean((pred.modelo.gam - datosA.test$mpg)^2)
test.error.gam
```

```
## [1] 10.86041
```

Con la ecuación para el cálculo de R^2

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

$$SST = \sum (y - \bar{y})^2$$

$$SSE = \sum (y - y')^2$$

podemos obtener el valor del test R^2 para el ajuste de nuestro modelo GAM:

```
SST <- mean((datosA.test$mpg - mean(datosA.test$mpg))^2)
SSE <- test.error.gam
R2 <- 1 - SSE / SST
R2
```

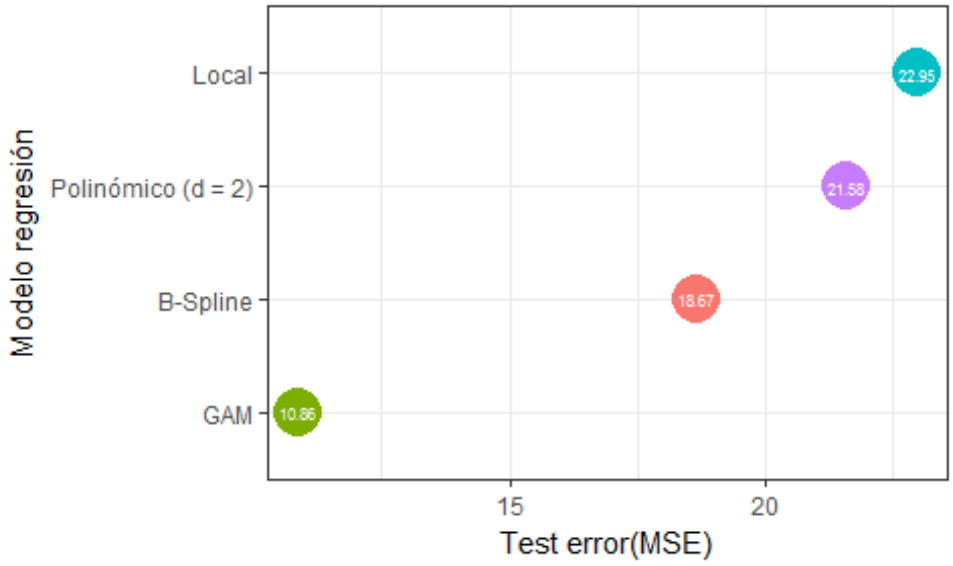
```
## [1] 0.8344508
```

El modelo GAM es capaz de explicar un 83,44% de la varianza en *mpg*.

Comparación de modelos

```
modelo <- c("Polinómico (d = 2)", "B-Spline", "Local", "GAM")
test.MSE <- c(test.error.poli, test.error.spline, test.error.local,
              test.error.gam)
comparacion <- data.frame(modelo = modelo, test.MSE = test.MSE)

ggplot(data = comparacion, aes(x = reorder(x = modelo, X = test.MSE),
                               y = test.MSE, color = modelo,
                               label = round(test.MSE,2))) +
geom_point(size = 8) +
geom_text(color = "white", size = 2) +
labs(x = "Modelo regresión", y = "Test error(MSE)") + theme_bw() +
coord_flip() + theme(legend.position = "none")
```



Mientras que el modelo de regresión local, el *spline* y el modelo polinómico comparten un test error similar, el modelo GAM es el que mejor resultado obtiene (en cuanto al *test error*).

BIBLIOGRAFÍA

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