GAMLSS: modelos aditivos generalizados para posición, escala y forma

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Marzo, 2020

Tabla de contenidos

Versión PDF: Github

Más sobre ciencia de datos: cienciadedatos.net o joaquinamatrodrigo.github.io

Introducción

¿Qué son los modelos GAMLSS?

Los modelos aditivos generalizados para posición, escala y forma **GAMLSS** (*Generalized Additive Models for Location, Scale and Shape*), son modelos de regresión semi-paramétricos. Paramétricos en cuanto a que requieren asumir que la variable respuesta sigue una determinada distribución paramétrica (normal, beta, gamma…) y *semi* porque los parámetros de esta distribución pueden ser modelados, cada uno de forma independiente, siguiendo funciones no paramétricas (lineales, aditivas o no lineales). Esta versatilidad hace de los *GAMLSS* una herramienta adecuada para modelar variables que siguen todo un abanico de distribuciones (no normales, asimétricas, con varianza no constante…).

Aunque en este documento se intentan resumir los principales aspectos prácticos, son muchos los aspectos teóricos (distribuciones, métodos de ajuste, penalizaciones…) que engloban de este tipo de modelos. Para profundizar en ellos, la mejor opción es el libro escrito por los propios autores de los modelos *GAMLSS*.

Stasinopoulos, Dm & Rigby, Robert & Heller, Gillian & Voudouris, Vlasios & De Bastiani, Fernanda. (2017). Flexible regression and smoothing: Using GAMLSS in R. 10.1201/b21973.

¿Cómo funcionan?

Los modelos *GAMLSS* asumen que la variable respuesta tiene una función de densidad definida por hasta 4 parámetros (μ, σ, ν, τ) que determinan su posición (p.ej. media), escala (p.ej. desviación estándar) y forma (p.ej. *skewness* y *kurtosis*), y que cada uno de ellos puede variar independientemente de los otros en función de los predictores. Estos modelos aprenden por lo tanto hasta 4 funciones, donde cada una establece la relación entre las variables predictoras y uno de los parámetros.

Son muchas las distribución que pueden emplearse para la variable respuesta y, $f(y_i|\mu_i, \sigma_i, \nu_i, \tau_i)$, la única condición es que el logaritmo de la función de densidad ($\log f(y_i|\mu_i, \sigma_i, \nu_i, \tau_i)$) y su primera derivada respecto a cada uno de los parámetros puedan calcularse.

Ventajas de los modelos GAMLSS

Para entender mejor las ventajas de este tipo de modelos, conviene repasar brevemente la evolución que han seguido los modelos estadísticos desde el modelo lineal (*LM*), modelo lineal generalizado (*GLM*), modelo aditivo generalizado (*GAM*) y finalmente los modelos *GAMLSS*.

Modelos de regresión lineal (LM)

El modelo de regresión lineal (Legendre, Gauss, Galton y Pearson) considera que, dado un conjunto de observaciones, la media μ de la variable respuesta Y se relaciona de forma lineal con la o las variables regresoras X.

$$\mu = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

o en notación matricial (incorporando β_0 en el vector $\boldsymbol{\beta}$:

$$\mu = \mathbf{X}^T \boldsymbol{\beta}$$

Ajustar el modelo consiste en estimar los valores de los coeficientes de regresión $\hat{\beta}$ y la varianza $\hat{\sigma}^2$ que maximizan la verosimilitud (*likelihood*) de los datos, es decir, los que dan lugar al modelo que con mayor probabilidad puede haber generado los datos observados. El método empleado con más frecuencia es el ajuste por mínimos cuadrados (*OLS*):

$$\hat{\beta} = \underset{\beta_0, \beta}{\operatorname{argmin}} (Y - \mathbf{X}^T \boldsymbol{\beta})^2$$

$$\hat{\mu} = \mathbf{X}^T \hat{\boldsymbol{\beta}}$$

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} \hat{\epsilon}_i^2}{n - p} = \frac{\sum_{i=1}^{n} (y_i - \hat{\mu})^2}{n - p}$$

donde $\hat{\epsilon}_i$ es la diferencia entre cada valor observado y la media estimada, n es el número de observaciones y p el número de predicciones.

Es importante resaltar una característica de estos modelos. La estimación de la media $\hat{\mu}$ depende directamente del valor de los predictores $\mathbf{X}$, sin embargo, la estimación de la varianza $\hat{\sigma}^2$ no depende directamente de los predictores, sino de $\hat{\mu}$. Para que sean válidas, deben cumplirse varias condiciones, dos de las cuales son:

- La variable respuesta Y se tiene que distribuir de forma normal.
- La varianza σ^2 debe de ser constante en todo el rango de la variable respuesta.

Modelos lineales generalizados (GLM)

El requisito de que la variable respuesta se distribuya de forma normal hace que el modelo de regresión lineal no sea aplicable a muchos problemas reales. Los modelos GLM (John Nelder y Robert Wedderburn) permiten superar esta limitación haciendo una abstracción a un modelo más genérico, en el que la variable respuesta puede seguir cualquier distribución de la familia exponencial (Gaussian, Poisson, binomial...). Para conseguirlo, la media de la variable respuesta ya no está relacionada directamente con los predictores, sino que se hace a través de una función *link* $g(.)$. Por ejemplo, el modelo lineal generalizado es equivale al modelo lineal por mínimos cuadrados cuando la función *link* es la identidad, o a la regresión logística cuando es binomial (*logit*).

$$g(\mu) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

$$g(\mu) = \mathbf{X}^T \boldsymbol{\beta}$$

Cuando se emplea una función *identidad*, la máxima verosimilitud equivale a minimizar la suma de errores cuadrados, lo que tiene una solución analítica. Para el resto de familias o cuando se emplea la regularización *L1*, no existe una solución analítica, por lo que se requiere un método de optimización iterativo como el *iteratively reweighted least squares (IRLSM)*, *L-BFGS*, método de Newton o descenso de gradiente.

Modelos de regresión lineal generalizados aditivos (GAM)

A pesar de las posibilidades que ofrecen los *GLM*, siguen teniendo como limitación que la relación entre los predictores y la media de la variable respuesta debe de ser lineal y constante. En 1990 Hastie y Tibshirani introdujeron los modelos GAM, una extensión de los modelos *GLM* que permite incorporar relaciones no lineales. En los modelos *GAM*, la relación de cada predictor x_i con la media de la variable respuesta $g(\mu)$ no es directa, sino que se hace a través de una función $f(x_i)$.

$$\eta = g(\mu) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

La función $f_i(x_i)$ puede ser cualquier función que recibe el valor del predictor x_i y devuelve otro valor, pudiendo ser esta transformación lineal o no lineal. En la práctica, las funciones más empleadas son funciones no lineales tipo *smooth*: *cubic regression splines*, *thin plate regression splines* o *Penalized splines*.

Modelos aditivos generalizados para posición, escala y forma (GAMLSS)

Los modelos lineales generalizados (*GLM*) y los modelos generalizados aditivos (*GAM*) asumen que la variable respuesta Y sigue una distribución de la familia exponencial, cuya media μ puede ser modelada en función de otras variables (predictores) y cuya varianza σ se calcula mediante una constante de dispersión y una función $v(\mu)$. Esto último significa que, la varianza, *skewness* y *kurtosis*, no se modelan directamente en función de las variables predictoras sino de forma indirecta a través de su relación con la media. ¿Qué impacto puede tener esto en la práctica? Si bien las estimación de la media obtenidas por estos modelos son buenas, no lo son tanto las incertidumbres asociadas y, en consecuencia, los intervalos de confianza y de predicción que se puedan calcular.

Los modelos *GAMLSS*, introducidos por Rigby y Stasinopoulos en 2005, permiten, además de incorporar distribuciones que no son de la familia exponencial, modelar explícitamente cada uno de sus parámetros en función de las variables predictoras empleando funciones lineales y no lineales. Los términos empleados dentro del marco de los *GAMLSS* para referirse a los parámetros de localización, escala y forma son (μ, σ, ν, τ) .

$$\mathbf{Y} \sim D(\mu, \sigma, \nu, \tau)$$

$$\mathbf{Y} = \mathbf{X}^T\boldsymbol{\beta}$$

$$\eta_1 = g_1(\mu) = \mathbf{X}^T\boldsymbol{\beta} + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

$$\eta_2 = g_2(\sigma) = \mathbf{X}^T\boldsymbol{\beta} + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

$$\eta_3 = g_3(\nu) = \mathbf{X}^T\boldsymbol{\beta} + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

$$\eta_4 = g_4(\tau) = \mathbf{X}^T\boldsymbol{\beta} + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

donde $\mathbf{Y} \sim D(\mu, \sigma, \nu, \tau)$ es la distribución de la variable respuesta (pueden ser menos parámetros), $\mathbf{X}$ contiene los términos lineales del modelo, $\boldsymbol{\beta}$ son los coeficientes lineales y $f_i(x_i)$ son funciones de suavizado no lineales (*smooth*) de cada predictor.

Los *GAMLSS* son una forma de superar las limitaciones de los modelos *GLM* y *GAM*. Como resultado del modelo, se consigue caracterizar la distribución completa, permitiendo generar intervalos probabilísticos y predicción de cuantiles.

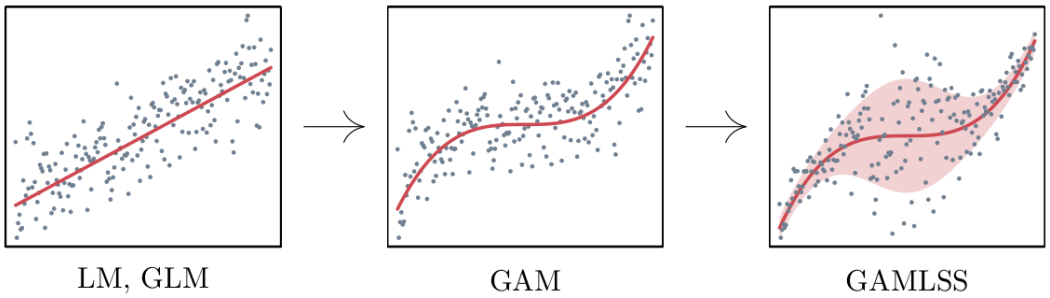


Imagen obtenida de: Schlosser, Lisa & Hothorn, Torsten & Stauffer, Reto & Zeileis, Achim. (2018). Distributional Regression Forests for Probabilistic Precipitation Forecasting in Complex Terrain. The Annals of Applied Statistics. 13. 10.1214/19-AOAS1247.

Ajuste de los modelos GAMLSS

Los modelos *GAMLSS* se ajustan mediante una adaptación del algoritmo *backfitting*, un algoritmo típicamente empleado para el ajuste de modelos aditivos. Por ejemplo, para el modelo aditivo

$$g(\mu) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

en términos generales, el algoritmo de ajuste por *backfitting* funciona de la siguiente forma:

1) Se inicia con un valor para todos los términos f_i del modelo, por ejemplo poniéndolos todos a cero.

2) Se estima el valor del primer término f_1 empleando los datos de entrenamiento, con el objetivo de predecir lo mejor posible la variable respuesta y.

3) Se estima el valor del segundo término f_2 empleando residuos $y - f_1(x_1)$.

4) Se repite el paso 3, ajustando cada término con los residuos del anterior.

5) Tras ajustar todos los términos, el valor del primer término se descarta y se reajusta empleando el residuo de todos los demás términos.

6) Repetir todos los pasos del 3 al 5 hasta que se alcance un criterio de parada (que los valores apenas cambien o que se alcance un número máximo de iteraciones).

Selección de modelos GAMLSS

Para identificar el mejor modelo *GAMLSS* es necesario comparar diferentes modelos candidatos, cada uno con una combinación de distribución para la variable respuesta, función *link* para cada uno de los parámetros, predictores e hiperparámetros. Existen varias estrategias para la selección del mejor modelo de entre los comparados:

- *Generalized Akaike information criterion* (*GAIC*): esta métrica emplea el *log likelihood* del modelo multiplicado por -2 y añade una penalización por cada parámetro que incluye el modelo. Más detalles sobre el *GAIC* en el apartado **métricas de ajuste**.
- *Generalized cross-validation* y cross-validation

Acorde a la comunidad de estadísticos, el segundo método es más recomendable, aunque supone mayor coste computacional.

gamlss

La implementación en **R** de los modelos *GAMLSS* está disponible en el paquete `gamlss` y todas sus extensiones:

- `gamlss`: paquete central con las principales funciones para ajustar modelos *GAMLSS*.
- `gamlss.dist`: extensión con distribuciones adicionales.
- `gamlss.cens`: extensión para variables respuesta censuradas.
- `gamlss.mx`: extensión para distribuciones mixtas (combinación de distribuciones).
- `gamlss.nl`: extensión para ajustar modelos no lineales.
- `gamlss.tr`: extensión para distribuciones truncadas.
- `gamlss.add`: extensión que permite incluir nuevas transformaciones no lineales a los predictores de un modelo aditivo (árboles, redes neuronales y *splines* multidimensionales).

La principal función del paquete es `gamlss()`, con la que se ajustan y generan los modelos. A lo largo de este documento se muestran, mediante ejemplos prácticos, como emplear estos paquetes para crear modelos *GAMLSS*.

LM, GLM, GAM y GAMLSS

En el siguiente ejemplo, se emplea el paquete `gamlss` para ajustar y comparar los resultados obtenidos con un modelo *LM*, *GLM*, *GAM* y *GAMLSS*, haciendo hincapié en las limitaciones de cada uno. Primero, se ajusta un modelo *LM* asumiendo que la distribución de la variable respuesta es de tipo normal y modelando únicamente la media de la distribución en función de los predictores. En segundo lugar, se mejora el ajuste con un *GLM*, que permite emplear una distribución *gamma* en lugar de normal. A continuación, se añaden relaciones de tipo no lineal empleando un modelo *GAM*. Finalmente, se ajusta un *GAMLSS* en el que la variable respuesta sigue una distribución *gamma*, la relación con los predictores no es lineal, y se modelan todos los parámetros de la distribución en función de los predictores, no solo la media.

Datos

El set de datos `rent` del paquete `gamlss.data` contiene información sobre la precio de 1969 viviendas situadas en Munich en el año 1993. Además del precio, incluye 9 variables adicionales:

- `R`: precio del alquiler
- `Fl`: metros cuadrados de la vivienda
- `A`: año de construcción
- `Sp`: si la calidad del barrio donde está situada la vivienda es superior la media (1) o no (0).
- `Sm`: si la calidad del barrio donde está situada la vivienda es inferior la media (1) o no (0).
- `B`: si tiene cuarto de baño (1) o no (0).
- `H`: si tiene calefacción central (1) o no (0).
- `L`: si el equipamiento de la cocina está por encima de la media (1) o no (0).
- `loc`: combinación de `Sp` y `Sm` indicando si la calidad del barrio donde está situada la vivienda es inferior (1), igual (2) o superior (3) a la media.

El objetivo es obtener un modelo capaz de predecir el precio del alquiler. Siguiendo el ejemplo de Stasinopoulos, Dm & Rigby, Robert & Heller, Gillian & Voudouris, Vlasios & De Bastiani, Fernanda, se emplean como predictores únicamente `Fl`, `A`, `H` y `loc`.

```
library(gamlss)
library(tidyverse)
library(ggpubr)
library(skimr)

data("rent")
datos <- rent
datos <- datos %>% select(R, Fl, A, H, loc)
# Se renombran las variables para que sean más explicativas
colnames(datos) <- c("precio", "metros", "anyo", "calefaccion", "situacion")
skim(datos)
```

Data summary

Name	datos
Number of rows	1969
Number of columns	5
_____	
Column type frequency:	
factor	2
numeric	3
_____	
Group variables	None

Variable type: factor

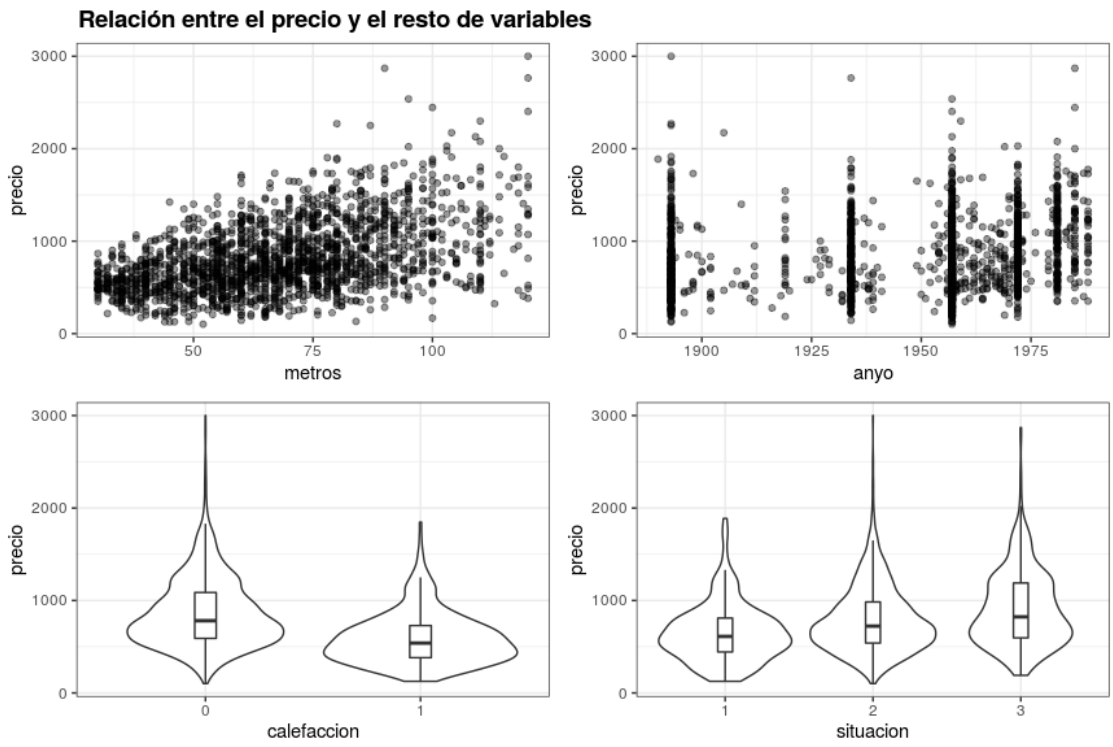
skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
calefaccion	0	1	FALSE	2	0: 1580, 1: 389
situacion	0	1	FALSE	3	2: 1247, 3: 550, 1: 172

Variable type: numeric

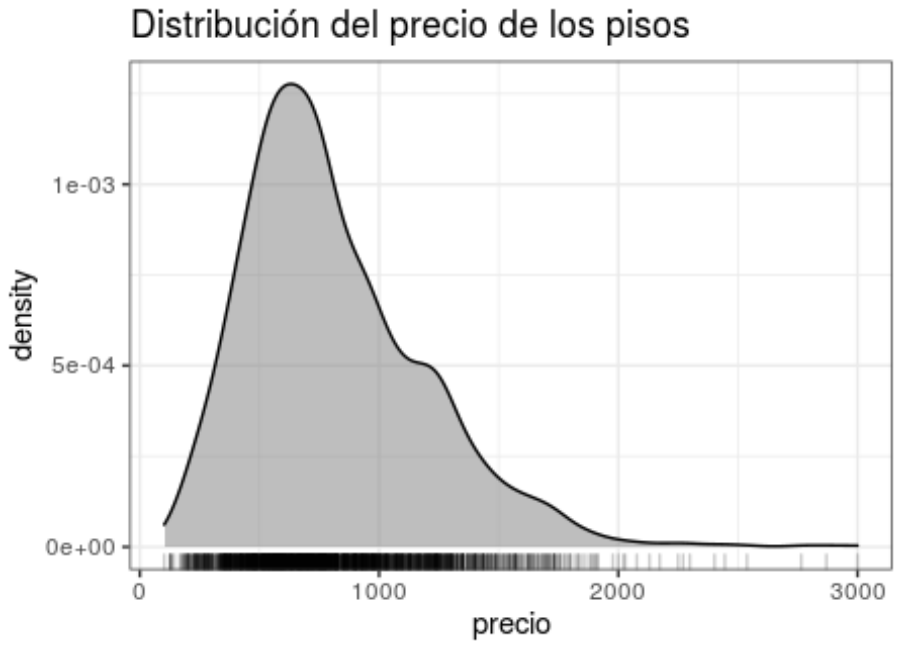
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
precio	0	1	811.88	379.00	101.7	544.2	737.8	1022	3000	▇▇▂▁▁
metros	0	1	67.73	20.86	30.0	52.0	67.0	82	120	▇▇▇▅▂
anyo	0	1	1948.48	29.02	1890.0	1934.0	1957.0	1972	1988	▂▁▂▃▇

Los siguientes gráficos muestran la distribución del precio de las viviendas y su relación con cada uno de los predictores.

```r
p1 <- ggplot(data = datos, aes(x = metros, y = precio)) +
      geom_point(alpha = 0.4) +
      labs(ttile = "Precio vs metros") +
      theme_bw()

p2 <- ggplot(data = datos, aes(x = anyo, y = precio)) +
      geom_point(alpha = 0.4) +
      labs(ttile = "Precio vs año") +
      theme_bw()

p3 <- ggplot(data = datos, aes(x = calefaccion, y = precio)) +
      geom_violin() +
      geom_boxplot(width = 0.1, outlier.shape = NA) +
      labs(ttile = "Precio vs calefacción") +
      theme_bw()

p4 <- ggplot(data = datos, aes(x = situacion, y = precio)) +
      geom_violin() +
      geom_boxplot(width = 0.1, outlier.shape = NA) +
      labs(ttile = "Precio vs situación") +
      theme_bw()

ggpubr::ggarrange(
  plotlist = list(p1, p2, p3, p4)
) %>%
ggpubr::annotate_figure(
  top = text_grob("Relación entre el precio y el resto de variables",
                  color = "Black",
                  face  = "bold",
                  size  = 14,
                  x     = 0.3)
  )
```




```
ggplot(data = datos, aes(x = precio)) +
  geom_density(alpha = 0.5, fill = "gray50") +
  geom_rug(alpha = 0.2) +
  labs(title = "Distribución del precio de los pisos") +
  theme_bw()
```



Con estos gráficos se evidencia que:

- La relación entre el precio y los metros cuadrados es lineal y positiva, con una varianza creciente a medida que aumentan los metros cuadrados. Esto significa que no se puede asumir la condición de varianza constante (homocedasticidad).
- La distribución del precio es asimétrica con una cola positiva, la variable respuesta no se distribuye de forma normal.
- El precio medio de pisos con calefacción es superior a los que no tienen calefacción.
- El precio de los pisos aumenta progresivamente si está en un barrio por debajo de la media, en la media o por encima.

Dadas estas características, se necesita un modelo que:

- Sea capaz de aprender relaciones complejas entre varios predictores, incluyendo relaciones no lineales.
- Sea capaz de modelar explícitamente la varianza en función de los predictores, ya que esta no es constante.
- Sea capaz de aprender distribuciones asimétricas con una marcada cola positiva.

Regresión Lineal (LM)

```
# El resultado es igual al obtenido con lm()
modelo_lm <- gamlss(
              formula = precio ~ metros + anyo + calefaccion + situacion,
              family  = NO,
              data    = datos,
              trace   = FALSE
             )

summary(modelo_lm)
```

```
## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:  gamlss(formula = precio ~ metros + anyo + calefaccion +
##     situacion, family = NO, data = datos, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2775.0388   470.1352  -5.903 4.20e-09 ***
```

```
## metros           8.8394     0.3370  26.228  < 2e-16 ***
## anyo             1.4808     0.2385   6.208 6.55e-10 ***
## calefaccion1  -204.7596    18.9858 -10.785  < 2e-16 ***
## situacion2     134.0523    25.1430   5.332 1.09e-07 ***
## situacion3     209.5815    27.1286   7.725 1.76e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.73165    0.01594   359.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  7
##       Residual Deg. of Freedom:  1962
##                       at cycle:  2
##
## Global Deviance:     28159
##             AIC:     28173
##             SBC:     28212.1
## ******************************************************************
```

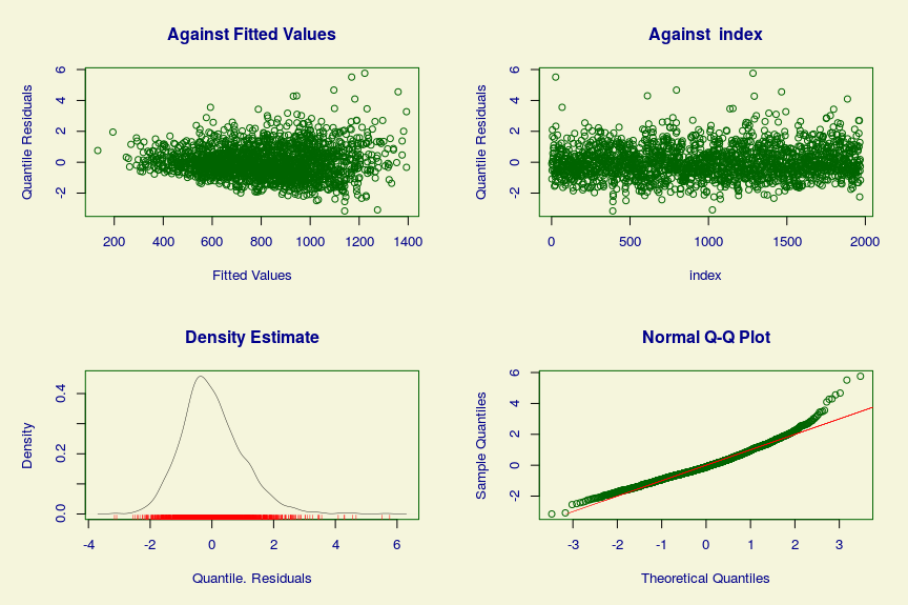
La función *link* para estimar la varianza es log, ya que la varianza no puede ser negativa. Para obtener el valor estimado de la varianza en la misma escala que la variable respuesta, hay que aplicar el exponente.

```
paste("El valor estimado de la varianza es:", exp(5.73165))
```

```
## [1] "El valor estimado de la varianza es: 308.477837124068"
```

Con la función `plot()` se muestra el diagnostico de los residuos del modelo.

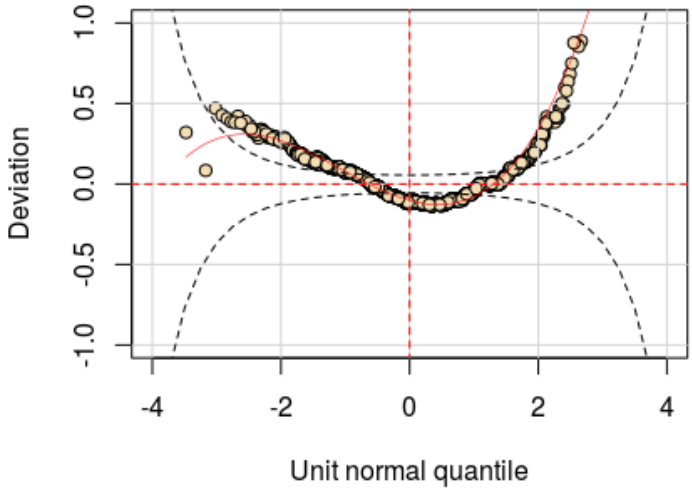
```
plot(modelo_lm)
```




```
## ****************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  4.959554e-13
##                         variance   =  1.000508
##                 coef. of skewness  =  0.7470097
##                 coef. of kurtosis  =  4.844416
## Filliben correlation coefficient   =  0.9859819
## ****************************************************************
```

Los gráficos *worm* son otra forma de evaluar, visualmente, la calidad de un modelo a través de sus residuos. Esta representación es similar a la de un gráfico *Q-Q* pero sustrayendo la línea teórica, por lo que, idealmente, los valores resultantes deberían de ser cero (línea horizontal discontinua). Las dos curvas elípticas discontinuas muestran el intervalo de confianza del 95%. Si el modelo es correcto, sólo entorno al 5% de las observaciones deberían quedar fuera. Por defecto, la función `wp()` muestra también un ajuste cúbico (curva continua roja) que ayuda a identificar la tendencia de los residuos.

```
# Worm plot de los residuos
wp(modelo_lm, ylim.all = 1)
```



El estudio de los residuos muestra que:

- Los errores no se distribuyen de forma aleatoria en torno a cero, sino que aumentan a medida que aumenta el valor predicho. Esto es indicativo y consecuencia de la heterogeneidad de la varianza, en concreto, la varianza aumenta con la media. (Gráfico superior-izquierda)
- La condición de normalidad no se cumple, el gráfico *Q-Q* y el gráfico *worm* muestran una clara desviación entre los cuantiles observados y los teóricos.

La violación de la condición de distribución normal conlleva que el modelo *LM* no sea adecuado para captar la relación entre el precio y los predictores.

Lineal Generalizado (GLM)

La distribución de la variable `precio` tiene dos características: no puede ser negativa y tiene una cola positiva (asimetría). Esto hace de la distribución gamma una buena candidata. Se puede encontrar un listado con todas las distribuciones disponibles en `gamlss` en este documento.

La distribución *gamma* tiene dos parámetros que tienen que ser estimados: media (μ) y escala (σ). Por defecto, el paquete `gamlss` emplea para ambos la función *log* como *link*, ya que los dos parámetros tienen un dominio (0, ∞).

$$\eta_1 = g_1(\mu) = \log(\mu)$$

$$\eta_2 = g_1(\sigma) = \log(\sigma)$$

Es muy importante tener en cuenta este aspecto a la hora de utilizar `gamlss`, ya que, dependiendo de qué función *link* que se emplee, una misma distribución puede dar lugar a modelos distintos. Por defecto, este paquete emplea funciones *link* que reflejan el dominio del parámetro: "identity" para $(-\infty, \infty)$, "log" para $(0, \infty)$, "logit" para $(0, 1)$, etc.

```
# Mostrar las funciones link por defecto de la distribución gamma.
GA()
```

```
##
## GAMLSS Family: GA Gamma
## Link function for mu    : log
## Link function for sigma: log
```

```
# Mostrar las funciones link disponibles para la distribución gamma.
show.link(family = GA)
```

```
## $mu
## c("inverse", "log", "identity", "own")
##
## $sigma
## c("inverse", "log", "identity", "own")
```

```
modelo_glm <- gamlss(
                formula = precio ~ metros + anyo + calefaccion + situacion,
                family  = GA,
                data    = datos,
                trace   = FALSE
              )

summary(modelo_glm)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:  gamlss(formula = precio ~ metros + anyo + calefaccion +
##     situacion, family = GA, data = datos, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.8649770  0.5681265   5.043 5.01e-07 ***
## metros        0.0106232  0.0004128  25.735  < 2e-16 ***
## anyo          0.0015100  0.0002886   5.232 1.86e-07 ***
## calefaccion1 -0.3000745  0.0231141 -12.982  < 2e-16 ***
## situacion2    0.1907641  0.0305197   6.251 5.00e-10 ***
## situacion3    0.2640828  0.0329216   8.022 1.78e-15 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.98220    0.01558  -63.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  7
##       Residual Deg. of Freedom:  1962
##                       at cycle:  2
##
## Global Deviance:     27764.59
##             AIC:     27778.59
##             SBC:     27817.69
## ******************************************************************
```

El modelo resultante es:

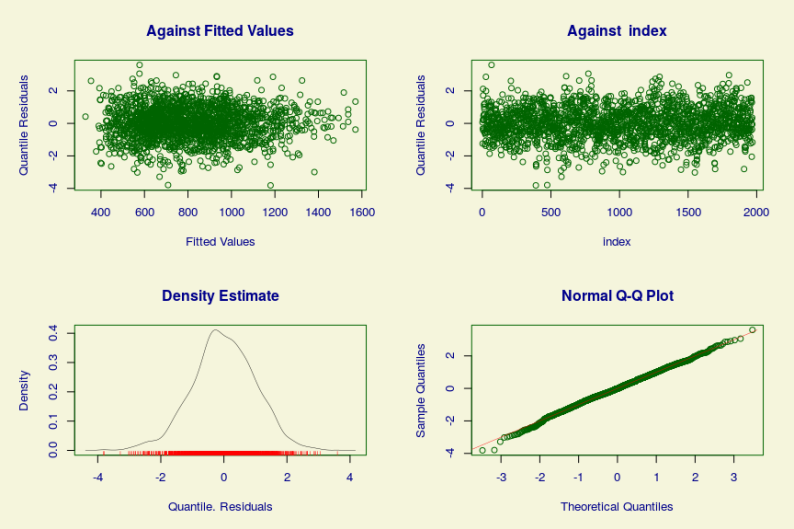
$$\text{precio} \sim GA(\hat{\mu}, \hat{\sigma})$$

donde:

$log(\hat{\mu}) = 2.865 + 0.0106\text{metros} + 0.00151\text{año} - \\0.3 \ (\text{si calefaccion=1}) + 0.191 \ (\text{si localicacion=2}) + 0.264 \ (\text{si localicacion=3})$

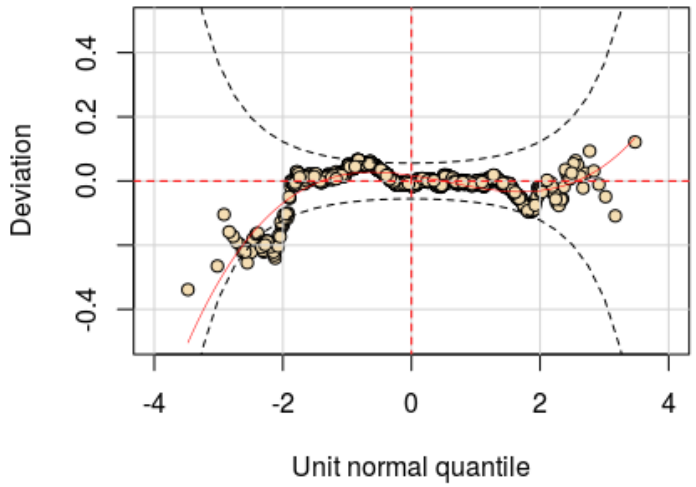
$log(\hat{\sigma}) = -0.9822$

```
plot(modelo_glm)
```




```
## ****************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  0.0004795675
##                         variance   =  1.000657
##                 coef. of skewness  =  -0.1079453
##                 coef. of kurtosis  =  3.255464
## Filliben correlation coefficient   =  0.9990857
## ****************************************************************
```

```
# Worm plot de los residuos
wp(modelo_glm, ylim.all = 0.5)
```



El análisis de los residuos muestra un diagnóstico más favorable que en el caso del modelo lineal *LM*. El gráfico de los residuos *vs* el valor real (esquina superior izquierda) tiene menor heterogeneidad, y la curvatura es las colas del gráfico *Q-Q* (esquina inferior derecha) se ha

reducido notablemente. Esto apunta a que el modelo *GLM* con una distribución *gamma* es más adecuado para modelar el precio de las viviendas. ¿Cómo corroborarlo? Una forma de hacerlo es comparando la métrica *GAIC* de ambos modelos.

```
# Por defecto, GAIC emplea una penalización k = 2.5
GAIC(modelo_lm, modelo_glm)
```

```
##             df      AIC
## modelo_glm   7 27778.59
## modelo_lm    7 28173.00
```

El modelo *GLM* consigue reducir el valor *GAIC*. Teniendo en cuenta esto, junto con el diagnóstico de los residuos, puede considerarse mejor candidato que el modelo *LM*.

Generalizado Aditivo (GAM)

En los modelos *GAM* se pueden aplicar funciones (lineales o no lineales) a cada uno de los predictores. El paquete `gamlss` implementa varias funciones *smooth* que suelen dar buenos resultados. En este caso se emplea *P-splines* (`pb()`).

```
# Se emplea P-splines para los predictores continuos
# Distribución gamma para la variable respuesta
modelo_gam <- gamlss(
                formula = precio ~ pb(metros) + pb(anyo) + calefaccion + situacion,
                family  = GA,
                data    = datos,
                trace   = FALSE
              )

summary(modelo_gam)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:  gamlss(formula = precio ~ pb(metros) + pb(anyo) + calefaccion +
##     situacion, family = GA, data = datos, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.0851197  0.5691903   5.420 6.69e-08 ***
## pb(metros)    0.0103084  0.0004031  25.574  < 2e-16 ***
```

```
## pb(anyo)      0.0014062  0.0002892   4.862 1.26e-06 ***
## calefaccion1 -0.3008111  0.0225874 -13.318  < 2e-16 ***
## situacion2    0.1886692  0.0299282   6.304 3.57e-10 ***
## situacion3    0.2719856  0.0322843   8.425  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.00196    0.01559  -64.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  11.21547
##       Residual Deg. of Freedom:  1957.785
##                       at cycle:  3
##
## Global Deviance:     27683.22
##             AIC:     27705.65
##             SBC:     27768.29
## ******************************************************************
```

Debido a la incorporación de las funciones no lineales (*smoothers*), hay que ser cauto a la hora de interpretar los coeficientes y los errores mostrados en el `summary`. Los coeficientes y errores de los predictores con funciones *smooth* (`metros` y `anyo`) contemplan únicamente la parte lineal, ignorando la no lineal. En el caso de los predictores lineales (`calefacción` y `situacion`) sus errores se estiman asumiendo que los términos con *smoother* son fijos, y no contemplan la incertidumbre introducida al ajustar cada una de las funciones *smooth*.

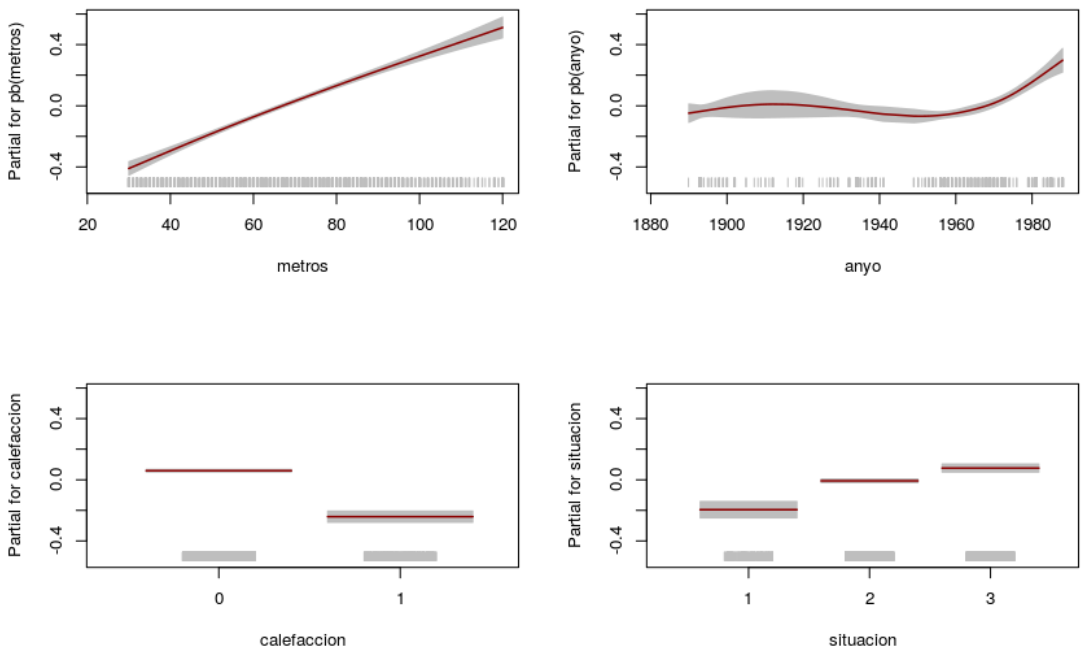
Para conocer la contribución total (lineal y no lineal) de los predictores transformados por funciones *smooth* se puede emplear la función `drop1()`. Esta función calcula el impacto que tiene en el modelo (en términos de grados de libertad totales, *AIC* y significancia estadística) el eliminar cada uno de los predictores de forma secuencial.

```
# Esta función puede tardar si hay muchos predictores no lineales o muchos datos.
drop1(modelo_gam, parallel = "multicore", ncpus = 4)
```

```
## Single term deletions for
## mu
##
## Model:
## precio ~ pb(metros) + pb(anyo) + calefaccion + situacion
##                 Df   AIC    LRT   Pr(Chi)
## <none>              27706
## pb(metros)  1.4680 28261 558.59 < 2.2e-16 ***
## pb(anyo)    4.3149 27798 101.14 < 2.2e-16 ***
## calefaccion 1.8445 27862 160.39 < 2.2e-16 ***
## situacion   2.0346 27770  68.02 1.825e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Otra consecuencia de añadir términos no lineales en un modelo es que su descripción matemática no es fácilmente interpretable, en su lugar, es más ilustrativo representar la relación de cada predictor con la variable respuesta. Esto puede hacerse con la función `term.plot()`.

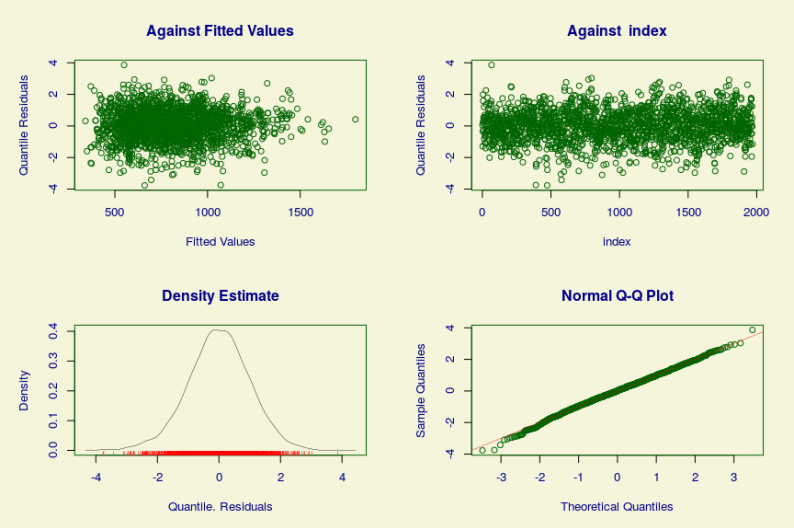
```
term.plot(modelo_gam, pages = 1, ask = FALSE, rug = TRUE)
```



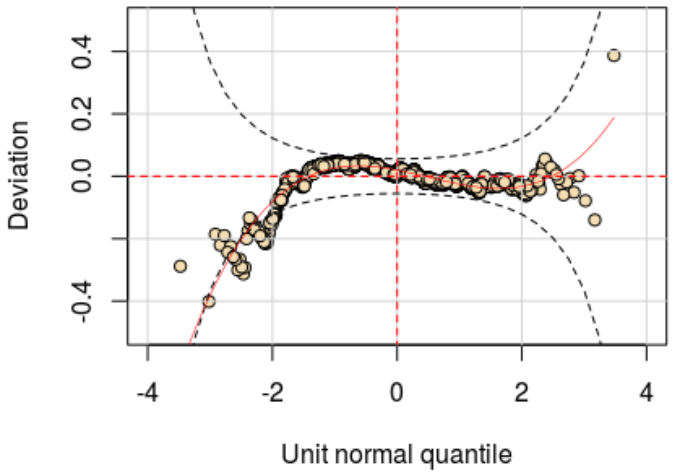
Los gráficos muestran la relación entre $\eta = log(\mu)$ (el logaritmo de la media del precio de las viviendas) con cada uno de los predictores:

- El valor aumenta linealmente a medida que aumentan los `metros` de la vivienda.
- La relación con el `anyo` de construcción no es lineal: se mantiene constante hasta aproximadamente 1960 y luego aumenta.
- La interpretación de los dos otros predictores (`calefaccion` y `situación`) es la misma que en los modelos anteriores, ya que se han incorporado al modelo sin sufrir ninguna transformación.

```
plot(modelo_gam)
```




```
# Worm plot de los residuos
wp(modelo_gam, ylim.all = 0.5)
```



Se compara el modelo *GAM* con los dos anteriores:

```
GAIC(modelo_lm, modelo_glm, modelo_gam)
```

```
##                   df      AIC
## modelo_gam 11.21547 27705.65
## modelo_glm  7.00000 27778.59
## modelo_lm   7.00000 28173.00
```

El modelo *GAM* ha conseguido reducir todavía más el valor *AIC* y la distribución de sus residuos ha mejorado ligeramente.

Generalizado Aditivo Localización, Escala y Forma (GAMLSS)

Existen 3 formas distintas de parametrizar la distribución gamma, una de ellas es mediante la media μ y un parámetro de escala θ (llamado σ en el paquete `gamlss`). Hasta el momento, se ha modelado únicamente μ en función de los predictores, asumiendo que la escala es constante y puede ser estimada a partir de la media.

Una forma de mejorar el modelo es modelando también su parámetro de escala en función de los predictores. Esto resulta particularmente útil cuando los datos muestran heterocedasticidad (su varianza no es constante). Se repite el ajuste, pero esta vez modelando sus dos parámetros en función de los predictores.

$$Y \sim GA(\mu, \sigma)$$

$$\eta_1 = g_1(\mu) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

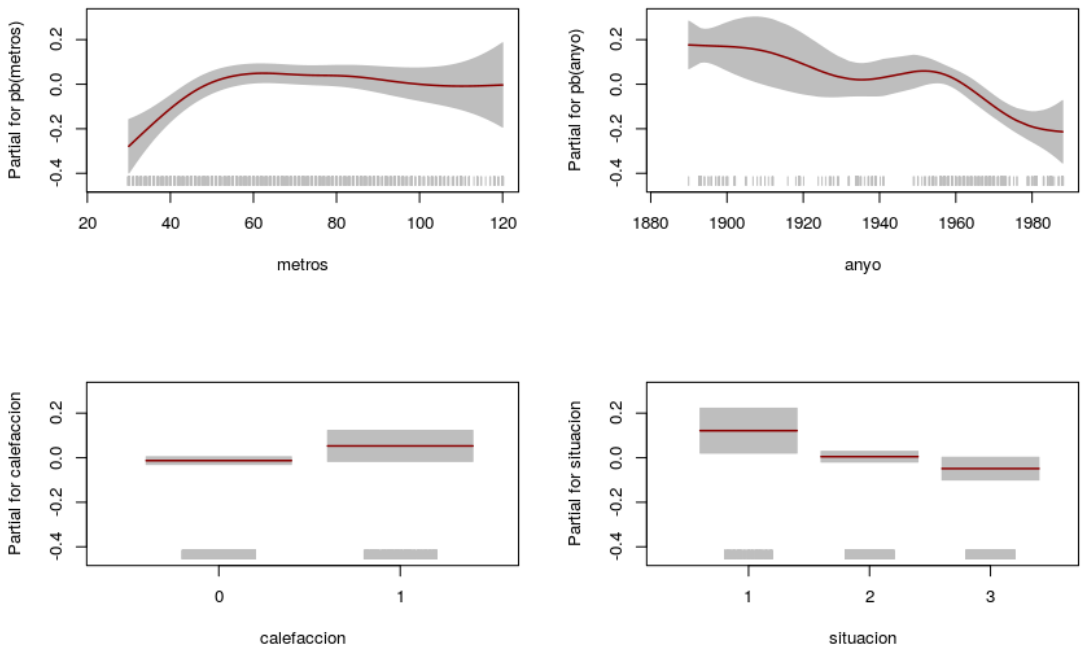
$$\eta_2 = g_2(\sigma) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p)$$

```
# Modelo GAMLSS para distribución gamma con parámetros de media y escala
# Se emplea P-splines para los predictores continuos
modelo_gamlss <- gamlss(
                  formula = precio ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                  sigma.formula = ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                  family  = GA,
                  data    = datos,
                  trace   = FALSE
                )

summary(modelo_gamlss)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:  gamlss(formula = precio ~ pb(metros) + pb(anyo) + calefaccion +
##     situacion, sigma.formula = ~pb(metros) + pb(anyo) +
##     calefaccion + situacion, family = GA, data = datos,      trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.8844571  0.5843467   4.936 8.64e-07 ***
## pb(metros)    0.0105402  0.0003620  29.116  < 2e-16 ***
## pb(anyo)      0.0014977  0.0002966   5.050 4.83e-07 ***
## calefaccion1 -0.2918923  0.0242854 -12.019  < 2e-16 ***
## situacion2    0.1938688  0.0323332   5.996 2.40e-09 ***
## situacion3    0.2734326  0.0339197   8.061 1.31e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.9225569  0.8645448   6.850 9.84e-12 ***
## pb(metros)    0.0019227  0.0007734   2.486  0.01300 *
## pb(anyo)     -0.0035795  0.0004372  -8.187 4.79e-16 ***
## calefaccion1  0.0659289  0.0415210   1.588  0.11248
## situacion2   -0.1166325  0.0565969  -2.061  0.03946 *
## situacion3   -0.1702147  0.0608990  -2.795  0.00524 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  22.25035
##       Residual Deg. of Freedom:  1946.75
##                       at cycle:  4
##
## Global Deviance:     27570.28
##             AIC:     27614.78
##             SBC:     27739.06
## ******************************************************************
```

Al igual que se hizo anteriormente con μ, se puede evaluar la influencia de cada predictor sobre el parámetro σ.

```
term.plot(modelo_gamlss, parameter = "sigma", pages = 1, ask = FALSE, rug = TRUE)
```




```
drop1(modelo_gamlss, parameter = "sigma", parallel = "multicore", ncpus = 4)
```

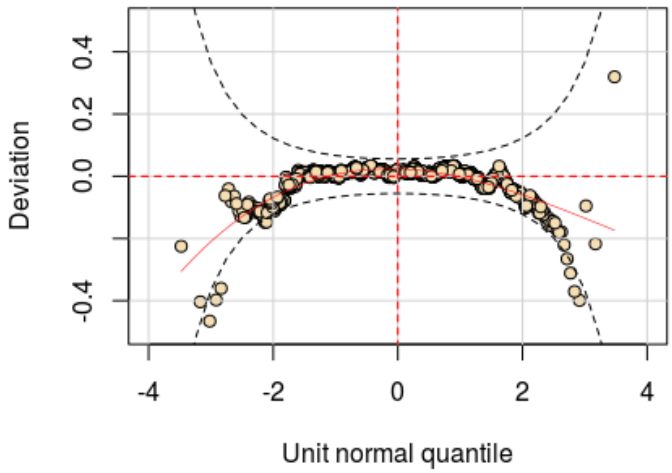
```
## Single term deletions for
## sigma
##
## Model:
## ~pb(metros) + pb(anyo) + calefaccion + situacion
##                   Df   AIC    LRT   Pr(Chi)
## <none>                27615
## pb(metros)  4.02694 27631 24.683 5.997e-05 ***
## pb(anyo)    3.87807 27659 52.167 1.067e-10 ***
## calefaccion 0.88335 27615  1.866   0.14788
## situacion   2.03694 27619  8.036   0.01872 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Viendo el gráfico y el impacto que tiene eliminar el predictor `calefaccion` del modelo, puede decirse que, en presencia de los otros, no aporta información para modelar el parámetro σ.

```
modelo_gamlss <- gamlss(
                  formula = precio ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                  sigma.formula = ~ pb(metros)+pb(anyo)+situacion,
                  family  = GA,
                  data    = datos,
                  trace   = FALSE
                )
```

Véanse ahora los residuos.

```
# Worm plot de los residuos
wp(modelo_gamlss, ylim.all = 0.5)
```



Un estudio visual del gráfico *worm* indica que, con este modelo, el número de observaciones cuyos residuos están fuera del intervalo de confianza del 95% es el que cabría esperar. Aun así, la clara tendencia en *U*, refleja que el modelo no es lo suficientemente flexible para capturar la asimetría (cola) de la distribución.

Por último, se comparan los 3 modelos ajustados:

```
GAIC(
  modelo_lm,
  modelo_glm,
  modelo_gam,
  modelo_gamlss
)
```

```
##                       df      AIC
## modelo_gamlss 21.36701 27614.88
## modelo_gam    11.21547 27705.65
## modelo_glm     7.00000 27778.59
## modelo_lm      7.00000 28173.00
```

Acorde al criterio *GAIC*, el modelo *GAMLSS* es el que mejor captura la relación entre el precio de la vivienda y los predictores empleados.

La distribución *gamma* empleada tiene únicamente dos parámetros, uno de posición μ y otro de escala σ. La limitación de las distribuciones con solo dos parámetros es que asumen que la asimetría (*skewness* y *kurtosis*) es constante para un valor fijo de μ y σ. Cuando **se dispone de una cantidad elevada de datos** se puede tratar de emplear distribuciones en las que sus parámetros de asimetría, llamados en este paquete η y ν, pueden ser modelados también en función de los predictores. Es aquí donde reside el potencial de estos modelos.

Véase el resultado de emplear una distribución con 3 parámetros Box-Cox Cole-Green o Box-Cox normal, que resulta útil para modelar datos con asimetría positiva.

```
BCCGo()
```

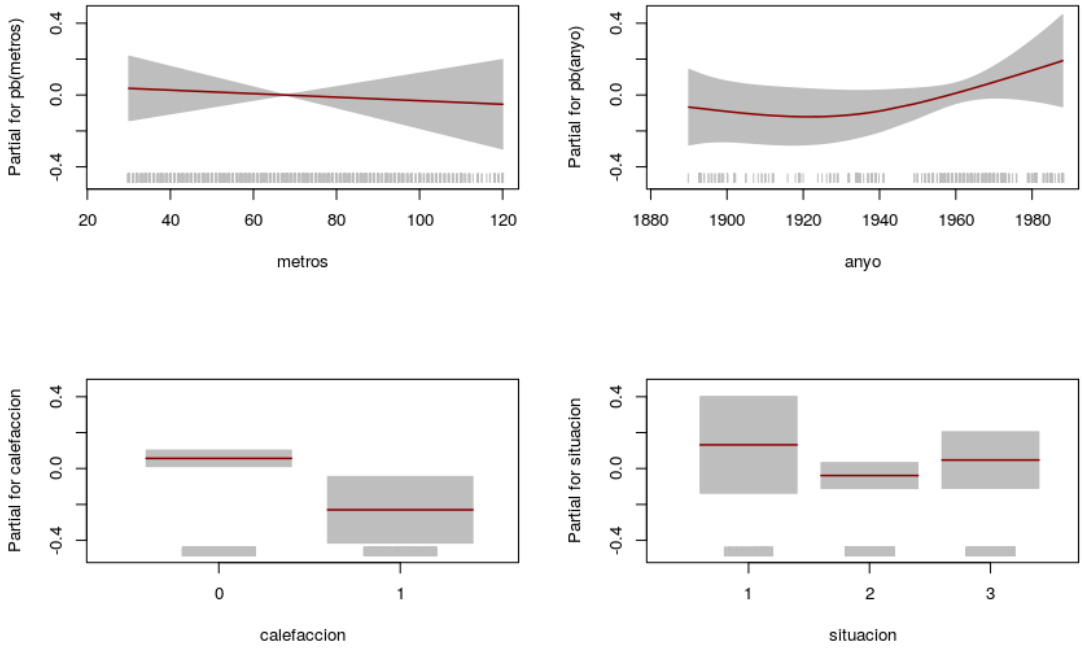
```
##
## GAMLSS Family: BCCGo Box-Cox-Cole-Green-orig.
## Link function for mu    : log
## Link function for sigma: log
## Link function for nu    : identity
```

```
# Modelo GAMLSS para distribución Box-Cox Cole-Green con parámetros de media,
# escala y forma. Se emplea P-splines para los predictores continuos
modelo_gamlss_2 <- gamlss(
                    formula = precio ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                    sigma.formula = ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                    nu.formula    = ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                    family  = BCCGo,
                    data    = datos,
                    trace   = FALSE
                   )

summary(modelo_gamlss_2)
```

```
## ******************************************************************
## Family:  c("BCCGo", "Box-Cox-Cole-Green-orig.")
##
## Call:  gamlss(formula = precio ~ pb(metros) + pb(anyo) + calefaccion +
##     situacion, sigma.formula = ~pb(metros) + pb(anyo) +
##     calefaccion + situacion, nu.formula = ~pb(metros) +
##     pb(anyo) + calefaccion + situacion, family = BCCGo,
##     data = datos, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.0285797  0.5869375   3.456  0.00056 ***
## pb(metros)    0.0104079  0.0003755  27.721  < 2e-16 ***
## pb(anyo)      0.0019277  0.0002973   6.485 1.12e-10 ***
## calefaccion1 -0.3213906  0.0268845 -11.955  < 2e-16 ***
## situacion2    0.1853274  0.0347087   5.340 1.04e-07 ***
## situacion3    0.2742129  0.0364029   7.533 7.57e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.6534017  0.8661933   7.681 2.48e-14 ***
## pb(metros)    0.0018721  0.0008059   2.323   0.0203 *
## pb(anyo)     -0.0039673  0.0004385  -9.047  < 2e-16 ***
## calefaccion1  0.0819273  0.0430147   1.905   0.0570 .
## situacion2   -0.0851622  0.0627391  -1.357   0.1748
## situacion3   -0.1410790  0.0675848  -2.087   0.0370 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -------------------------------------------------------------------
## Nu link function:  identity
## Nu Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.160132   3.203110  -0.987    0.324
## pb(metros)   -0.000983   0.002295  -0.428    0.669
## pb(anyo)      0.001989   0.001619   1.229    0.219
## calefaccion1 -0.286663   0.122130  -2.347    0.019 *
## situacion2   -0.171143   0.183091  -0.935    0.350
## situacion3   -0.084593   0.201476  -0.420    0.675
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## -------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  28.41391
##       Residual Deg. of Freedom:  1940.586
##                       at cycle:  7
##
## Global Deviance:     27551.32
##             AIC:     27608.15
##             SBC:     27766.85
## ******************************************************************
```

```
term.plot(modelo_gamlss_2, parameter = "nu", pages = 1, ask = FALSE, rug = TRUE)
```

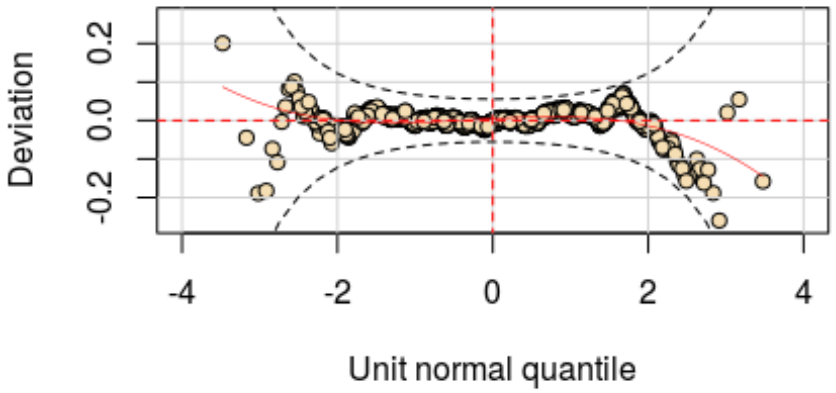



```
drop1(modelo_gamlss_2, parameter = "nu", parallel = "multicore", ncpus = 4)
```

```
## Single term deletions for
## nu
##
## Model:
## ~pb(metros) + pb(anyo) + calefaccion + situacion
##                  Df   AIC    LRT Pr(Chi)
## <none>              27608
## pb(metros)  1.0234 27606 0.2085 0.65746
## pb(anyo)    1.7773 27608 3.6102 0.13685
## calefaccion 1.1919 27612 5.7613 0.02197 *
## situacion   1.8542 27605 0.6695 0.68075
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

El estudio de los coeficientes apunta a que, únicamente el predictor `calefacción`, contribuye a modelar el parámetro ν.

```
modelo_gamlss_2 <- gamlss(
                     formula = precio ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                     sigma.formula = ~ pb(metros)+pb(anyo)+calefaccion+situacion,
                     nu.formula    = ~ calefaccion,
                     family  = BCCGo,
                     data    = datos,
                     trace   = FALSE
                   )

wp(modelo_gamlss_2)
```



La distribución de los residuos ha mejorado respecto a la del modelo *gamma*. Por último, se comprueba si la métrica *AIC* también es mejor.

```
GAIC(
  modelo_lm,
  modelo_glm,
  modelo_gam,
  modelo_gamlss,
  modelo_gamlss_2
)
```

```
##                       df      AIC
## modelo_gamlss_2 23.77443 27603.60
## modelo_gamlss   21.36701 27614.88
## modelo_gam      11.21547 27705.65
## modelo_glm       7.00000 27778.59
## modelo_lm        7.00000 28173.00
```

A lo largo de este ejemplo se han mostrado las ventajas derivadas de la flexibilidad que ofrecen los modelos *GAMLSS*. En los siguientes apartados, se profundiza en más aspectos teóricos y prácticos de este tipo de modelos.

Distribuciones

Los modelos *GAMLSS* permiten ajustar prácticamente cualquier distribución paramétrica. Esto supone una gran libertad para crear modelos de regresión, pero también más responsabilidad a la hora de decidir qué distribución emplear. Afortunadamente, el paquete `gamlss` y sus extensiones incorporan algunas estrategias para, junto con el conocimiento del analista, ayudar a identificar la mejor distribución.

Comparación y selección

La función `fitDist()` ajusta toda las distribuciones paramétricas disponibles de una determinada familia, y las compara acorde al *GAIC* (*generalized Akaike information criterion*). La familia de distribuciones se especifica con el argumento `type` y puede ser: `"realAll"`, `"realline"`, `"realplus"`, `"real0to1"`, `"counts"` y `"binom"`.

- `realAll`: distribuciones de la familia `realline` + `realplus`.
- `realline`: distribuciones continuas en el dominio $(-\infty, \infty)$: `NO`, `GU`, `RG` ,`LO`, `NET`, `TF`, `TF2`, `PE`,`PE2`, `SN1`, `SN2`, `exGAUS`, `SHASH`, `SHASHo`, `SHASHo2`, `EGB2`, `JSU`, `JSUo`, `SEP1`, `SEP2`, `SEP3`, `SEP4`, `ST1`, `ST2`, `ST3`, `ST4`, `ST5`, `SST`, `GT`.
- `realplus`: distribuciones continuas en el dominio $(0, \infty]$: `EXP`, `GA`,`IG`,`LOGNO`, `LOGNO2`,`WEI`, `WEI2`, `WEI3`, `IGAMMA`,`PARETO2`, `PARETO2o`, `GP`, `BCCG`, `BCCGo`, `exGAUS`, `GG`, `GIG`, `LNO`,`BCTo`, `BCT`, `BCPEo`, `BCPE`, `GB2`.
- `real0to1`: distribuciones continuas en el dominio [0,1]: `BE`, `BEo`, `BEINF0`, `BEINF1`, `BEOI`, `BEZI`, `BEINF`, `GB1`.
- `counts`: distribuciones para cuentas: `PO`, `GEOM`, `GEOMo`,`LG`, `YULE`, `ZIPF`, `WARING`, `GPO`, `DPO`, `BNB`, `NBF`,`NBI`, `NBII`, `PIG`, `ZIP`,`ZIP2`, `ZAP`, `ZALG`, `DEL`, `ZAZIPF`, `SI`, `SICHEL`,`ZANBI`, `ZAPIG`, `ZINBI`, `ZIPIG`, `ZINBF`, `ZABNB`, `ZASICHEL`, `ZINBF`, `ZIBNB`, `ZISICHEL`.
- `binom`: distribuciones para datos binomiales: `BI`, `BB`, `DB`, `ZIBI`, `ZIBB`, `ZABI`, `ZABB`.

Otra alternativa disponible es la función `fitDistPred()`, que ajusta las distribuciones igual que `fitDist()` pero, en lugar de compararlas por el *GAIC*, emplea un conjunto de test para calcular la bondad de ajuste (*global deviance*).

```
distribuciones <- fitDist(
                    datos$precio,
                    k = 2, # esta penalización equivale al AIC
                    type = "realplus",
                    trace = FALSE,
                    try.gamlss = TRUE
                  )

distribuciones$fits %>%
  enframe(name = "distribucion", value = "GAIC") %>%
  arrange(GAIC)
```

```
## # A tibble: 23 x 2
##    distribucion   GAIC
##    <chr>         <dbl>
##  1 BCCG         28614.
##  2 BCCGo        28614.
##  3 GG           28614.
##  4 BCPEo        28615.
##  5 BCPE         28615.
##  6 GIG          28616.
##  7 BCTo         28616.
##  8 GA           28616.
##  9 BCT          28616.
## 10 GB2          28616.
## # … with 13 more rows
```

El objeto devuelto por `fitDist()` almacena la mejor de entre todas las distribuciones probadas.

```
summary(distribuciones)
```

```
## ******************************************************************
## Family:  c("BCCG", "Box-Cox-Cole-Green")
##
## Call:  gamlssML(formula = y, family = DIST[i], data = sys.parent())
##
## Fitting method: "nlminb"
## Coefficient(s):
##              Estimate  Std. Error   t value   Pr(>|t|)
## eta.mu    749.2743664   8.5740894  87.38822 < 2.22e-16 ***
## eta.sigma  -0.7520409   0.0163055 -46.12189 < 2.22e-16 ***
## eta.nu      0.2530936   0.0379940   6.66141 2.7122e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Degrees of Freedom for the fit: 3 Residual Deg. of Freedom   1966
## Global Deviance:     28607.7
##             AIC:     28613.7
##             SBC:     28630.4
```

Simulación

Todas las distribuciones de los paquetes `gamlss.*` disponen de funciones `d`, `p`, `q` y `r` para calcular probabilidad, densidad, cuantiles y generar valores aleatorios.

```
BCCG()
```

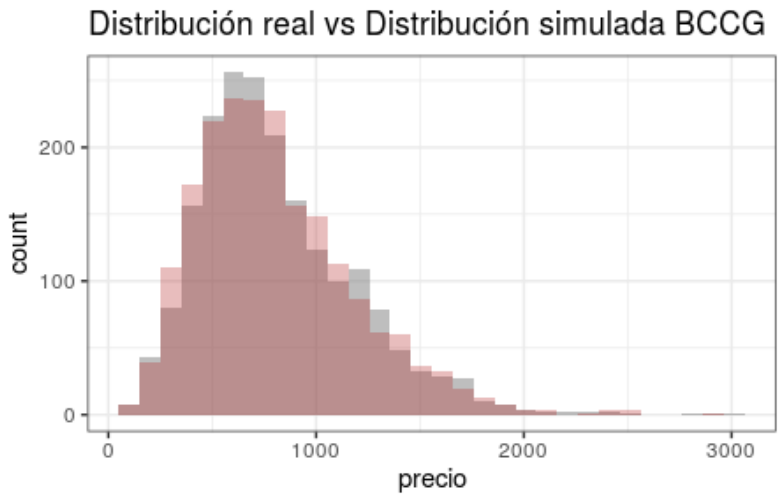
```
##
## GAMLSS Family: BCCG Box-Cox-Cole-Green
## Link function for mu    : identity
## Link function for sigma: log
## Link function for nu    : identity
```

```
# La función link de BCCG() es el log, por lo que el valor devuelto por el
# ajuste está en logaritmo
rBCCG(n = 1, mu = 749.2743664, sigma = exp(-0.7520409), nu = 0.2530936)
```

```
## [1] 1425.154
```

Cuando se selecciona una distribución, conviene simular nuevos valores para ver si estos se parecen a los datos observados. Empleando la distribución *Box-Cox-Cole-Green* (*BCCG*) con los parámetros estimados por `fitDist()`, se simulan 2000 observaciones y se compara su distribución con la de los precios observados.

```
# Simulación de datos
simulacion <- rBCCG(
                n  = 2000,
                mu = 749.2743664,
                sigma = exp(-0.7520409),
                nu = 0.2530936
              )

ggplot() +
  geom_histogram(
    data = datos,
    aes(x = precio),
    alpha = 0.5, fill = "gray50") +
  geom_histogram(
    data = data.frame(simulacion),
    aes(x = simulacion),
    alpha = 0.3, fill = "firebrick") +
  labs(title = "Distribución real vs Distribución simulada BCCG") +
  theme_bw()
```



Distribuciones truncadas

Una distribución truncada es aquella en la que, el rango de posibles valores para las variables Y, es un subconjunto del rango de otra distribución. No hay que confundir una distribución truncada con una censurada. En la primera, valores por encima o por debajo de un determinado valor no existen, mientras que en las segundas los valores sí existen pero no pueden ser observados.

La versión truncada de cualquier distribución presente en `gamlss.family` puede obtenerse gracias a la función `gen.trun()` del paquete `gamlss.tr`. La idea es sencilla: se parte una distribución ya existente en `gamlss`, se determinan los límites de truncado (izquierda, derecha o ambos) y automáticamente se generan nuevas funciones `d`, `p`, `q` y `r` con esas características.

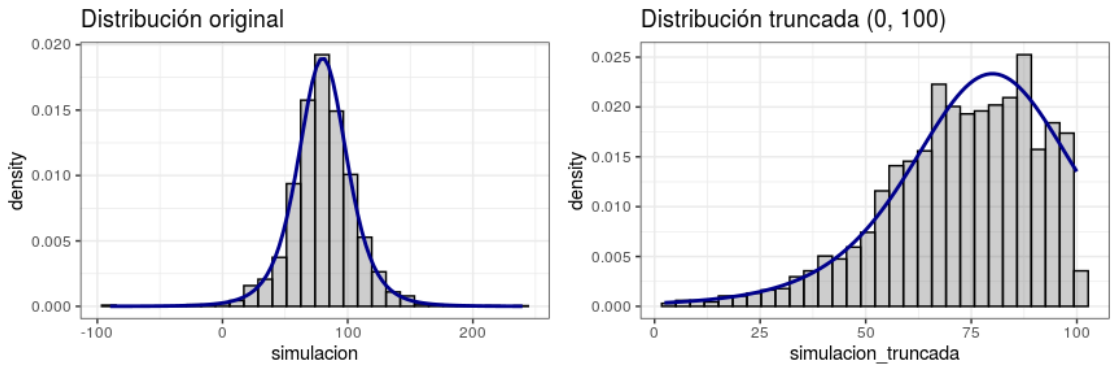
En el siguiente ejemplo se parte de una distribución t-student (`TF()`) y se trunca en ambas colas con límites 0 y 100.

```
# Distribución T con parámetros:media: mu = 80, varianza: sigma = 20
# grados de libertad: nu = 5
simulacion <- rTF(n = 2000, mu = 80, sigma = 20, nu = 5)
```

```
library(gamlss.tr)

# Misma distribución pero truncada entre 0 y 100.
gen.trun(
  par  = c(0,100),
  family = "TF",
  name = "0_100",
  type = "both"
)
```

```
## A truncated family of distributions from TF has been generated
##  and saved under the names:
##  dTF0_100 pTF0_100 qTF0_100 rTF0_100 TF0_100
## The type of truncation is both
##  and the truncation parameter is 0 100
```

```
simulacion_truncada <- rTF0_100(n = 2000, mu = 80, sigma = 20, nu = 5)

p1 <- ggplot(data = data.frame(simulacion)) +
      geom_histogram(
        aes(x = simulacion, y = after_stat(density)),
        color = "black",
        alpha = 0.3) +
      stat_function(
        fun   = function(.x){dTF(x = .x, mu = 80, sigma = 20, nu = 5)},
        color = "darkblue",
        size  = 1) +
      labs(title = "Distribución original") +
      theme_bw()

p2 <- ggplot(data = data.frame(simulacion_truncada)) +
      geom_histogram(
        aes(x = simulacion_truncada, y = after_stat(density)),
        color = "black",
        alpha = 0.3) +
      stat_function(
        fun   = function(.x){dTF0_100(x = .x, mu = 80, sigma = 20, nu = 5)},
        color = "darkblue",
        size  = 1) +
      labs(title = "Distribución truncada (0, 100)") +
      theme_bw()

ggarrange(p1, p2)
```



Distribuciones censuradas

Una distribución censurada es aquella en la que, el valor de las variables *Y* se desconoce cuando es superior o inferior a un determinado límite. La diferencia con las distribuciones truncadas es que, en estas últimas, los valores por debajo o arriba de un determinado valor no existen, mientras que en las distribuciones censuradas, los valores sí existen pero se desconocen.

La versión censurada de cualquier distribución presente en `gamlss.family` puede obtenerse gracias a la función `gen.cens()` del paquete `gamlss.tr`. La idea es sencilla: se parte una distribución ya existente en `gamlss`, se determinan los límites de censura (izquierda, derecha o intervalo) y automáticamente se generan nuevas funciones `d`, `p`, `q` y `r` con esas características.

Distribuciones mixtas

Las distribuciones mixtas son aquellas que se crean como resultado de combinar varias distribuciones. Este tipo de ajustes son útiles cuando la distribución de la variable respuesta es de tipo multimodal o para crear modelos con variables latentes.

Cualquier combinación de distribuciones `gamlss.family` puede ser ajustada empleando dos funciones del paquete `gamlss.mx`:

- `gamlssMX()`: cuando las distribuciones no tienen parámetros en común, pudiendo incluso ser distribuciones de distintas familias.
- `gamlssNP()`: cuando las distribuciones tienen parámetros en común.

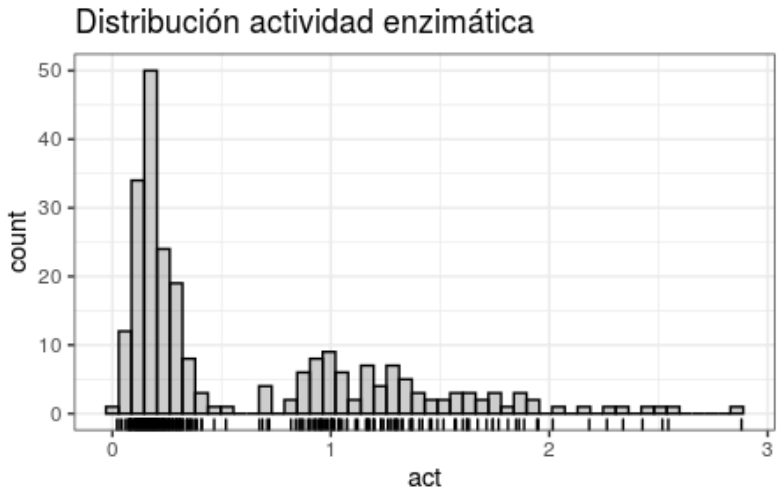
En ambos casos, el modelo se ajusta empleando el algoritmo *EM*.

Distribución multimodal

El set de datos `enzyme` del paquete `gamlss.mx` contiene información sobre la actividad en sangre de una enzima en 245 pacientes.

```
library(gamlss.mx)
data(enzyme)

ggplot(data = enzyme, aes(x = act)) +
  geom_histogram(color = "black", alpha = 0.3, bins = 50) +
```

```
  geom_rug() +
  labs(title = "Distribución actividad enzimática") +
  theme_bw()
```



La distribución es bimodal, lo que apunta a que es resultado de combinar dos distribuciones distintas. Se ajusta un modelo mixto con dos componentes (`k=2`), en el que cada componente es una distribución *Reverse Gumbel* (`family = RG`).

```
modelo_mx <- gamlssMX(
                formula = act ~ 1,
                data    = enzyme,
                family  = RG,
                K       = 2,
                control = MX.control(plot = FALSE)
             )
modelo_mx
```

```
##
## Mixing Family:  c("RG", "RG")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = act ~ 1, family = RG, K = 2, data = enzyme,
##     control = MX.control(plot = FALSE))
##
## Mu Coefficients for model: 1
## (Intercept)
##      0.1557
## Sigma Coefficients for model: 1
## (Intercept)
##      -2.641
## Mu Coefficients for model: 2
## (Intercept)
##       1.127
```

```
## Sigma Coefficients for model: 2
## (Intercept)
##      -1.091
##
## Estimated probabilities: 0.6239809 0.3760191
##
## Degrees of Freedom for the fit: 5 Residual Deg. of Freedom   240
## Global Deviance:     86.2916
##             AIC:     96.2916
##             SBC:     113.798
```

Como el modelo resultante es una combinación de distribuciones, para obtener la densidad estimada de una nueva observación, hay que calcular primero la densidad de cada distribución por separado. Con la función `getpdfMX()` se pueden crear funciones auxiliares que automatizan el proceso.

```
d_modelo_mx <- getpdfMX(modelo_mx)
d_modelo_mx(y = 2)
```

```
## [1] 0.07720277
```

```
ggplot(data = enzyme) +
geom_histogram(
  aes(x = act, y = after_stat(density)),
  color = "black",
  alpha = 0.3) +
stat_function(
  fun   = function(.x){d_modelo_mx(y = .x)},
  color = "darkblue",
  size  = 1) +
labs(title = "Distribución estimada por modelo mixto de dos componentes") +
theme_bw()
```

El ajuste por EM se caracteriza por correr el riesgo de caer en mínimos locales y no llegar a la solución óptima. Para evitar este problema, se recomienda ajustar el modelo varias veces, en cada una partiendo de unos valores iniciales distintos. La función `gamlssMXfits()` automatiza este proceso ajustando el modelo `n` veces y devolviendo al final el que mejor resultado ha conseguido (menor *global deviance*).

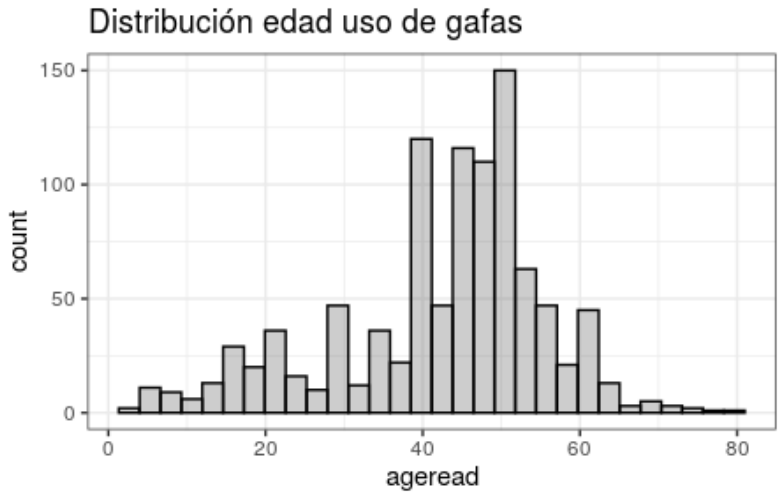
```
modelo_mx <- gamlssMXfits(
                n       = 10,
                formula = act ~ 1,
                data    = enzyme,
                family  = RG,
                K       = 2,
                control = MX.control(plot = FALSE)
             )
```

Modelo con variables latentes

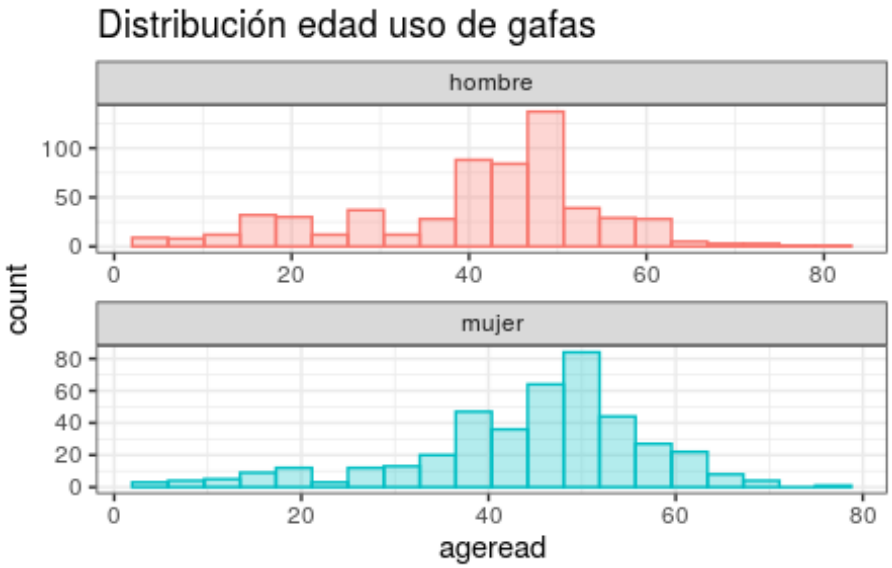
Los modelos de variables latentes se emplean en escenarios en los que se asume que la población objetivo está dividida en varias clases, pero se desconocen cuáles son. El objetivo de estos modelos es estimar la probabilidad que tiene cada observación de pertenecer a cada una de las clases. En la mayoría de casos reales, el número total de las clases se desconoce, por lo que tiene que ser estimado a partir de los datos.

El set de datos `glasses` del paquete `gamlss.data` contiene información sobre la edad a la que empezaron a utilizar gafas 1016 personas. Se dispone también del sexo de cada persona.

```
data(glasses)
glasses <- glasses %>%
           mutate(
             sex = recode_factor(sex, "1" = "hombre", "2" = "mujer")
           )

ggplot(data = glasses, aes(x = ageread)) +
  geom_histogram(alpha = 0.3, color = "black") +
  labs(title = "Distribución edad uso de gafas") +
  theme_bw() +
  theme(legend.position = "none")
```




```
ggplot(data = glasses, aes(x = ageread, fill = sex, color = sex)) +
  geom_histogram(alpha = 0.3, bins = 20) +
  labs(title = "Distribución edad uso de gafas") +
  facet_wrap(facets = vars(sex), nrow = 2, scales = "free") +
  theme_bw() +
  theme(legend.position = "none")
```



Se ajustan dos modelos mixtos, uno con componentes normales y otro con componentes *gamma*, en los que la variable respuesta es `ageread` y se incorpora la variable `sex` como predictor.

```
set.seed(123)
modelo_mx_no <- gamlssMX(
                  formula = ageread ~ sex,
                  data    = glasses,
                  family  = NO,
                  K       = 2,
                  control = MX.control(plot = FALSE)
                )

modelo_mx_ga <- gamlssMX(
                  formula = ageread ~ sex,
                  data    = glasses,
                  family  = GA,
                  K       = 2,
                  control = MX.control(plot = FALSE)
                )
```

Se comparan los modelos por el *GAIC*.

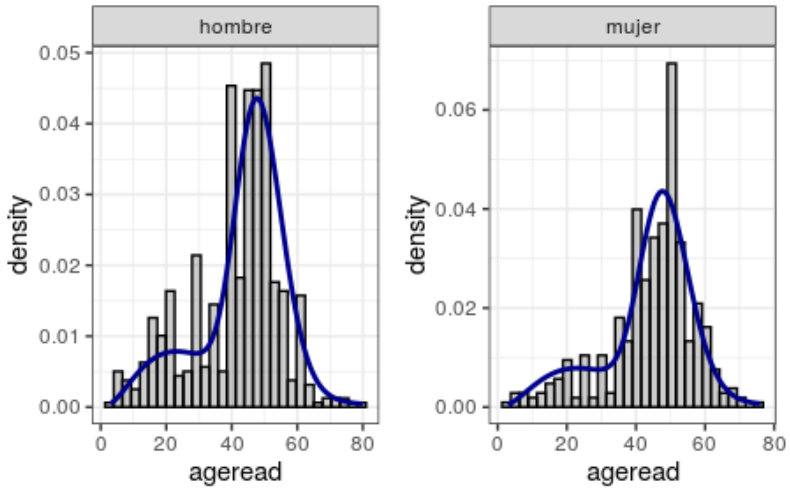
```
GAIC(
  modelo_mx_no,
  modelo_mx_ga
)
```

```
##              df      AIC
## modelo_mx_ga  7 7922.612
## modelo_mx_no  7 7945.059
```

En base al criterio *GAIC*, el modelo con formado por distribuciones *gamma* es mejor.

```
d_modelo_mx <- getpdfMX(modelo_mx_ga)

p1 <- ggplot(data = glasses %>% filter(sex == "hombre")) +
      geom_histogram(
        aes(x = ageread, y = after_stat(density)),
        color = "black",
        alpha = 0.3) +
      stat_function(
        fun   = function(.x){d_modelo_mx(y = .x)},
        color = "darkblue",
        size  = 1) +
      facet_wrap(vars(sex)) +
      theme_bw()

p2 <- ggplot(data = glasses %>% filter(sex == "mujer")) +
      geom_histogram(
        aes(x = ageread, y = after_stat(density)),
        color = "black",
        alpha = 0.3) +
      stat_function(
```

```
        fun   = function(.x){d_modelo_mx(y = .x)},
        color = "darkblue",
        size  = 1) +
      facet_wrap(vars(sex)) +
      theme_bw()

ggarrange(p1, p2)
```

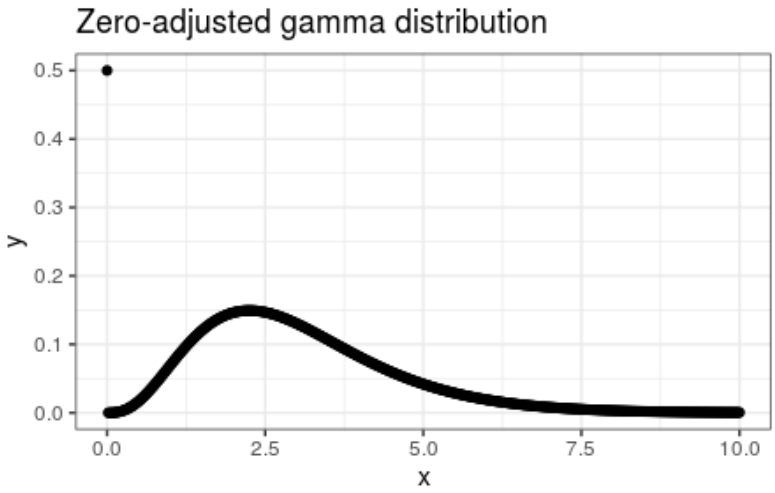


Dentro del objeto `gamlssMX` se almacenan información del modelo final, destacan: los modelos individuales que forman parte del modelo mixto (`$models`) y la probabilidad de cada observación de pertenecer a cada uno de las componentes (`$post.prob`).

Zero-inflated y zero-adjusted

Las distribuciones *zero-inflated* y *zero-adjusted* son casos especiales de distribuciones mixtas en las que uno de los componentes es exactamente 0 con probabilidad 1 y el otro componente es una distribución continua. Un ejemplo de variable que tiene este comportamiento es la precipitación. Para los días que no llueve, la cantidad de precipitación es exactamente 0, sin embargo, cuando llueve, la precipitación acumulada sigue una distribución continua. El paquete `gamlss` ya tiene predefinidas algunas funciones de este tipo como `ZAGA()` y `ZAIG()`

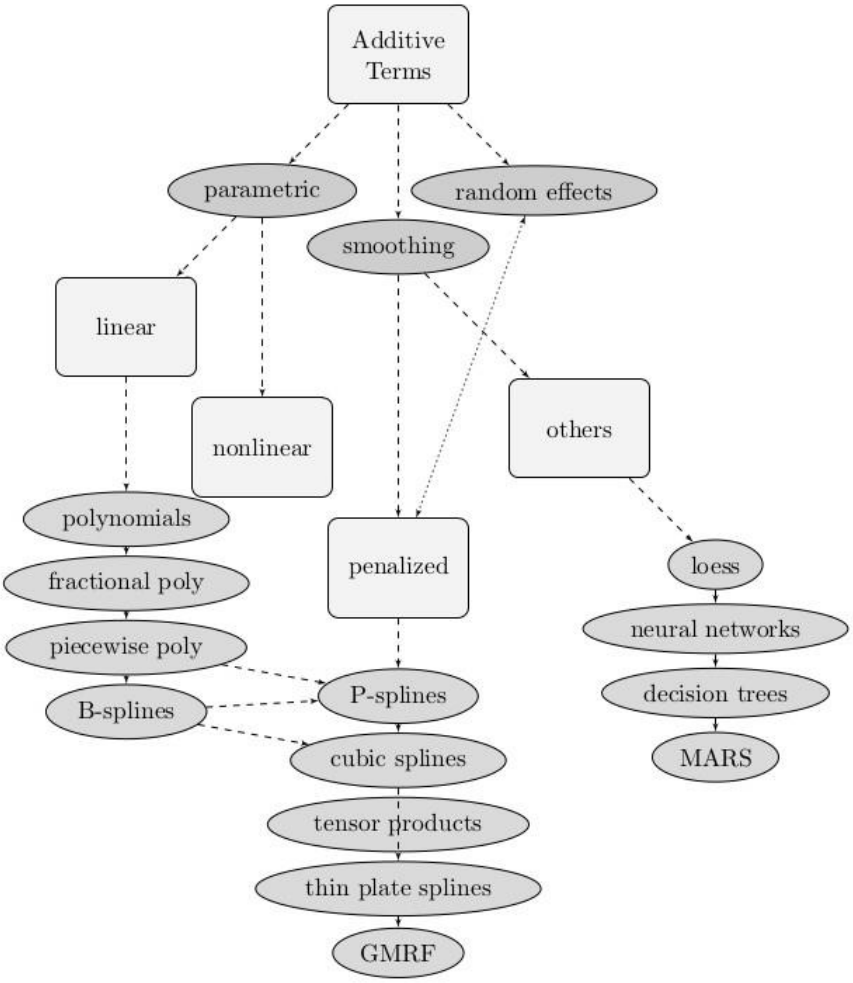
```
# plotZAGA(mu = 3, sigma = 0.5, nu = 0.5, from = 0, to = 10 n = 101,  main=NULL)

x <- seq(0, 10, length.out = 500)
y <- dZAGA(x = x, mu = 3, sigma = 0.5, nu = 0.5)
ggplot(data = data.frame(x, y), aes(x,y)) +
  geom_point() +
  labs(title = "Zero-adjusted gamma distribution")+
  theme_bw()
```



Funciones no lineales

Son muchas las funciones no lineales (*smoothers*) que se pueden utilizar para transformar predictores e incorporar así relaciones no lineales en los modelos *GAMLSS*.

En términos generales, los *smoothers* univariantes $s(x)$ se pueden entender como la forma de recoger el principal efecto no lineal de los predictores sobre la variable respuesta, y los multivariante $s(x_1, x_2)$ la parte no lineal de las interacciones. En ambos casos, estas transformaciones no lineales se pueden incorporar al modelo como si de otro término aditivo se tratara.



Términos aditivos disponibles en gamlss. Imagen obtenida de: Stasinopoulos, Dm & Rigby, Robert & Heller, Gillian & Voudouris, Vlasios & De Bastiani, Fernanda. (2017). Flexible regression and smoothing: Using GAMLSS in R

Cada tipo de *somoother* tiene unos parámetros propios, pero todos ellos comparten al menos uno en común, el *smoothing parameter* (λ), que determina la flexibilidad de la curva ajustada. Uno de los aspectos más importantes relacionado con las funciones *smooth* es cómo elegir el valor apropiado de λ. Las tres estrategias disponibles en `gamlss` son:

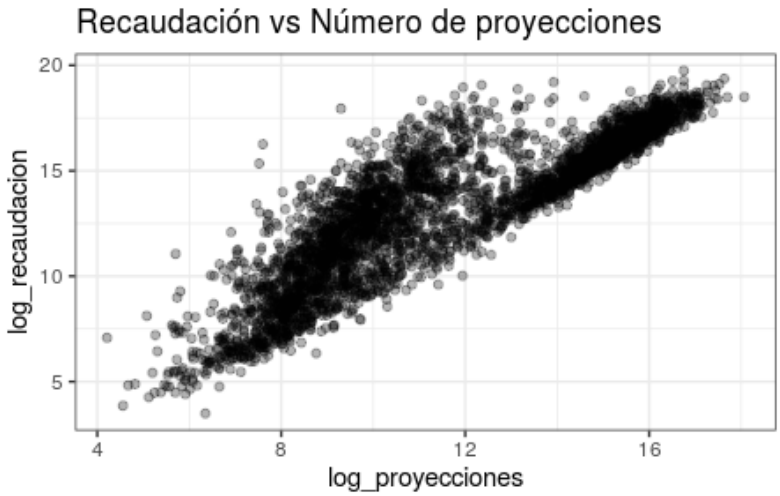
- Minimizar el *generalized cross validation (GCV)*.
- Minimizar el *GAIC*.
- Maximizar la verosimilitud (*maximum local likelihood method*).

En el siguiente ejemplo se muestran algunos de los *smoothers* más empleadas.

Datos

El set de datos `film90` del paquete `gamlss.data` contiene información sobre el número de proyecciones y la recaudación conseguida en su primera semana de estreno (ambas en escala logarítmica) para 4015 películas.

```
data("film90")
datos <- film90
datos <- film90 %>%
         dplyr::select(log_recaudacion = lborev1, log_proyecciones =  lboopen)

ggplot(data = datos, aes(x = log_proyecciones, y = log_recaudacion)) +
  geom_point(alpha = 0.3)  +
  labs(title = "Recaudación vs Número de proyecciones") +
  theme_bw()
```



A partir del gráfico se evidencia que:

- La relación entre la variable respuesta y el predictor no es lineal.
- La dispersión y forma de la distribución varía en los diferentes niveles de la variable respuesta.
-

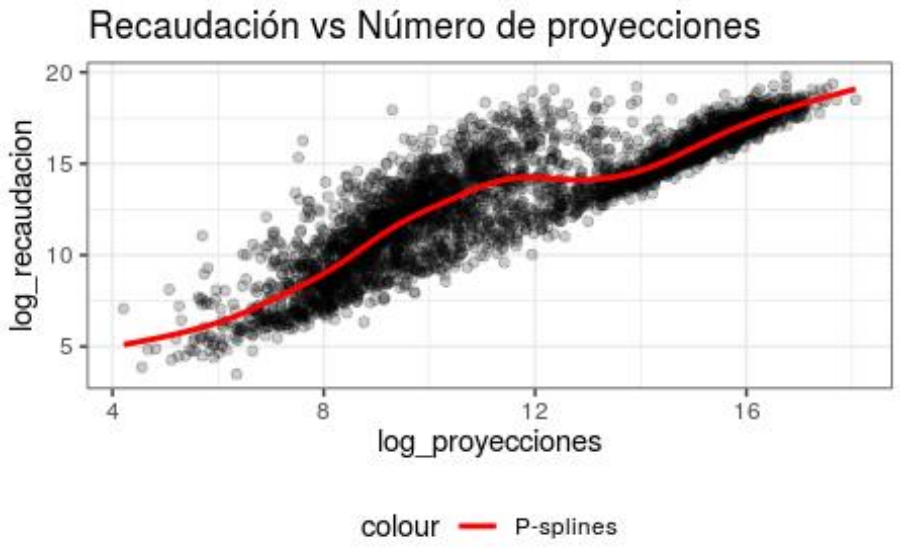
P-Splines

La función `pb()` permite incorporar *P-Splines* (*Penalized Smoothing Splines*) a los predictores continuos del modelo. Esta función emplea el método de *local maximum likelihood* para seleccionar automáticamente los grados de libertad efectivos óptimos (flexibilidad).

```r
modelo_ps <- gamlss(
              formula = log_recaudacion ~ pb(log_proyecciones),
              data    = datos,
              family  = NO,
              trace   = FALSE
            )

summary(modelo_ps)
```

```
## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:  gamlss(formula = log_recaudacion ~ pb(log_proyecciones),
##     family = NO, data = datos, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           2.347147   0.087053   26.96   <2e-16 ***
## pb(log_proyecciones) 0.928889   0.007149  129.93   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.33120    0.01114   29.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## ------------------------------------------------------------------
## No. of observations in the fit:  4031
## Degrees of Freedom for the fit:  12.73672
##       Residual Deg. of Freedom:  4018.263
##                       at cycle:  2
##
## Global Deviance:     14109.58
##             AIC:     14135.05
##             SBC:     14215.32
## ******************************************************************
```

```
datos <- datos %>%
         mutate(prediccion_ps = fitted(modelo_ps))

ggplot(
  data = datos,
  aes(x = log_proyecciones, y = log_recaudacion)) +
geom_point(alpha = 0.2)  +
geom_line(
  aes(x = log_proyecciones, y = prediccion_ps, color = "P-splines"),
  size = 1) +
scale_color_manual(
  breaks = c("P-splines"),
  values = c("P-splines" = "red")) +
labs(title = "Recaudación vs Número de proyecciones") +
theme_bw() +
theme(legend.position = "bottom")
```



Cuando se incorpora en algún predictor de un modelo aditivo (*GAM* o *GAMLSS*) una función *smooth*, el coeficiente de regresión y su error estimado no se pueden interpretar de la forma convencional, ya que estos solo contemplan la aportación lineal, ignorando la no lineal. La mejor forma de interpretar el impacto de una función *smooth* no lineal es:

- Mediante gráficos de dependencia parcial, que muestran el impacto en las predicciones del modelo a medida que varía el valor de un predictor y se mantienen constantes los otros. `getPEF()`
- Reajustar el modelo excluyendo el predictor de interés y evaluar el impacto que tiene sobre el modelo. `drop1()`

```
getPEF(
  modelo_ps,
  term      = "log_proyecciones",
  parameter = "mu",
  plot      = TRUE
)

drop1(object = modelo_ps, parallel = "multicore", ncpus = 4)
```

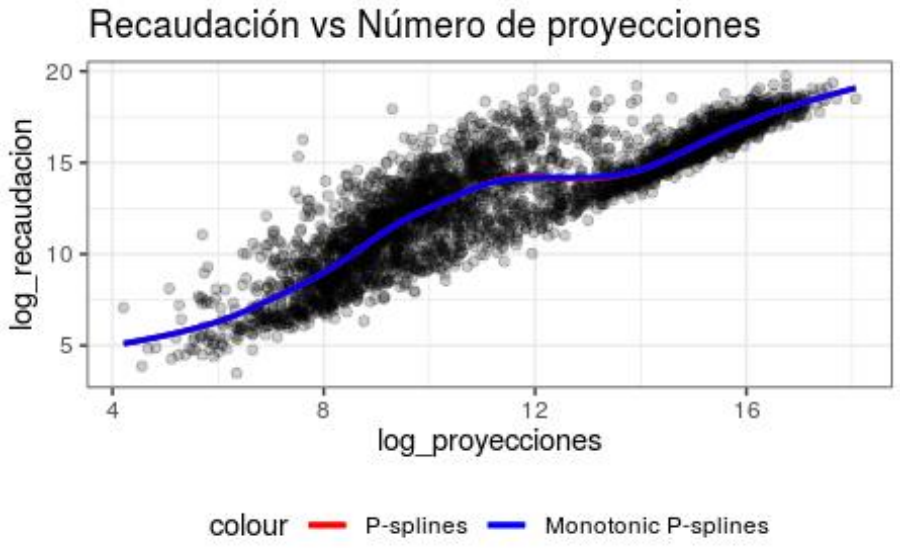
```
## Single term deletions for
## mu
##
## Model:
## log_recaudacion ~ pb(log_proyecciones)
##                           Df   AIC    LRT   Pr(Chi)
## <none>                       14135
## pb(log_proyecciones) 10.737 20956 6842.4 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Monotonic P-Splines

La función `pbm()` incorpora una modificación de las *P-splines* que fuerza a la que curva resultante sea monótona (siempre ascendente o siempre descendente). Los argumentos son los mismo que los de `pb()` pero con el añadido de `mono`, que indica la dirección que debe tener la curva ("up" o "down").

```
modelo_psm <- gamlss(
                    formula = log_recaudacion ~ pbm(log_proyecciones, mono = "up"),
                    data    = datos,
                    family  = NO,
                    trace   = FALSE
                  )

datos <- datos %>%
         mutate(prediccion_psm = fitted(modelo_psm))

ggplot(
  data = datos,
  aes(x = log_proyecciones, y = log_recaudacion)) +
geom_point(alpha = 0.2)  +
geom_line(
  aes(x = log_proyecciones, y = prediccion_ps, color = "P-splines"),
  size = 1) +
geom_line(
  aes(x = log_proyecciones, y = prediccion_psm, color = "Monotonic P-splines"),
```

```
  size = 1) +
scale_color_manual(
  breaks = c("P-splines", "Monotonic P-splines"),
  values = c("P-splines" = "red", "Monotonic P-splines" = "blue")) +
labs(title = "Recaudación vs Número de proyecciones") +
theme_bw() +
theme(legend.position = "bottom")
```



Cubic Splines

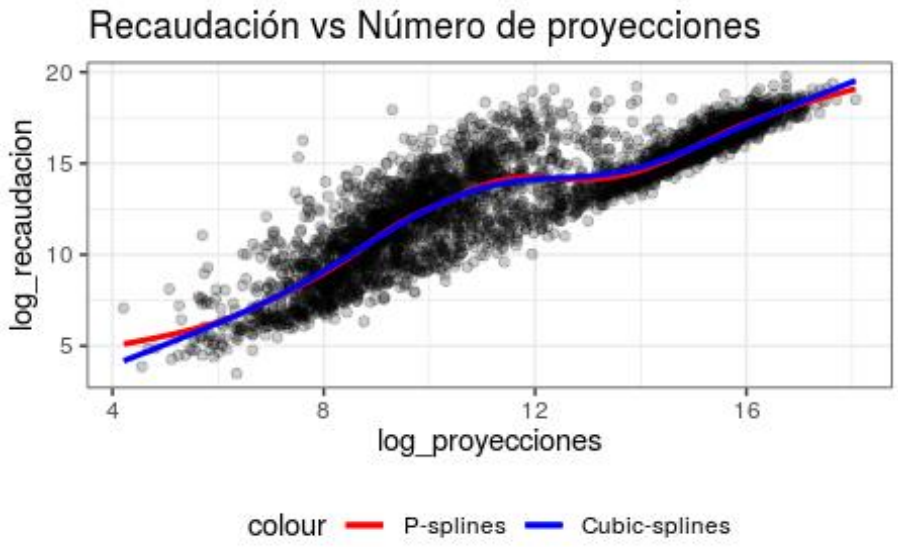
La función `cs()` permite incorporar *Cubic Splines* a los predictores continuos del modelo. A diferencia de `pb()`, en este caso sí es necesario especificar los grados de libertad.

```
# Modelo con cubic splines de 5 grados de libertad.
modelo_cs <- gamlss(
               formula = log_recaudacion ~ cs(log_proyecciones, df = 5),
               data    = datos,
               family  = NO,
               trace   = FALSE
             )

summary(modelo_cs)
```

```
## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:  gamlss(formula = log_recaudacion ~ cs(log_proyecciones,
##     df = 5), family = NO, data = datos, trace = FALSE)
##
## Fitting method: RS()
```

```
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   2.347158   0.087369   26.86   <2e-16 ***
## cs(log_proyecciones, df = 5) 0.928888   0.007175  129.46   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.33481    0.01114   30.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## ------------------------------------------------------------------
## No. of observations in the fit:  4031
## Degrees of Freedom for the fit:  8.000946
##       Residual Deg. of Freedom:  4022.999
##                       at cycle:  2
##
## Global Deviance:     14138.75
##             AIC:     14154.75
##             SBC:     14205.17
```

```r
datos <- datos %>%
         mutate(prediccion_cs = fitted(modelo_cs))

ggplot(
  data = datos,
  aes(x = log_proyecciones, y = log_recaudacion)) +
geom_point(alpha = 0.2)  +
geom_line(
  aes(x = log_proyecciones, y = prediccion_ps, color = "P-splines"),
  size = 1) +
geom_line(
  aes(x = log_proyecciones, y = prediccion_cs, color = "Cubic-splines"),
  size = 1) +
scale_color_manual(
  breaks = c("P-splines", "Cubic-splines"),
  values = c("P-splines" = "red", "Cubic-splines" = "blue")) +
labs(title = "Recaudación vs Número de proyecciones") +
theme_bw() +
theme(legend.position = "bottom")
```

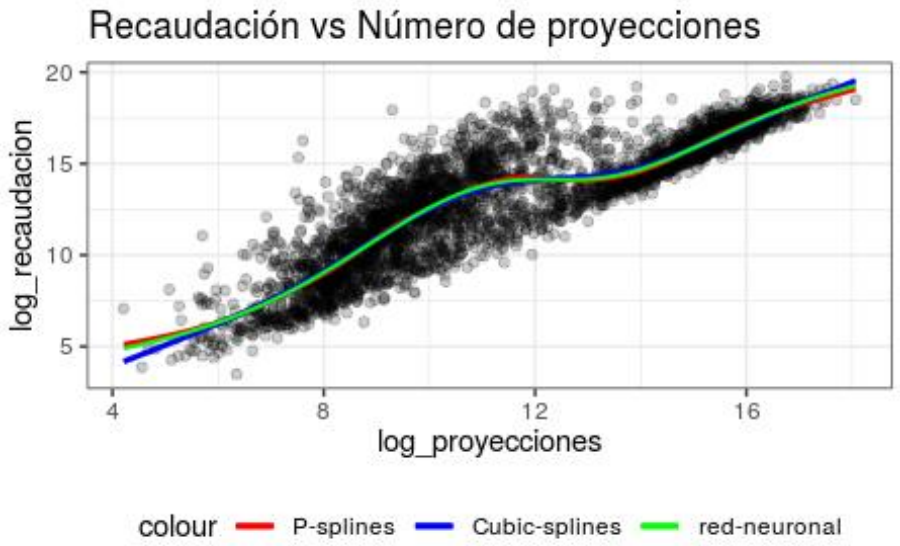


Redes neuronales

Las redes neuronales (multiperceptrón) también pueden emplearse como funciones *smooth* (requiere del paquete `gamlss.add`).

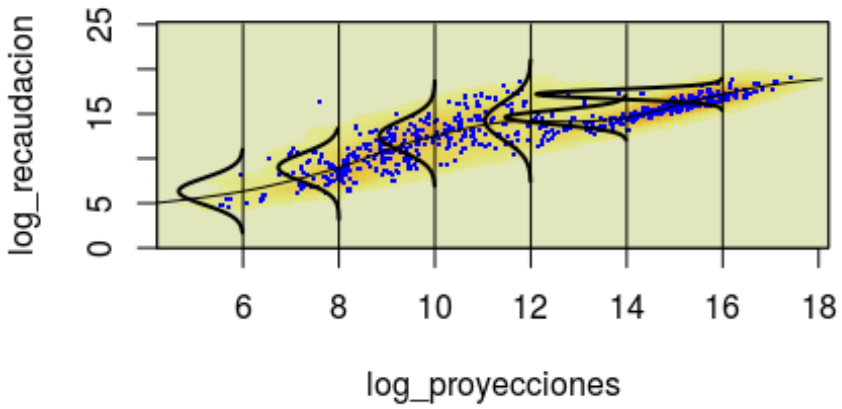
```
library(gamlss.add)
modelo_nn <- gamlss(
               formula = log_recaudacion ~ nn(~log_proyecciones, size = 20, decay =
0.1),
               data    = datos,
               family  = NO,
               trace   = FALSE
             )

datos <- datos %>%
         mutate(prediccion_nn = fitted(modelo_nn))

ggplot(
  data = datos,
  aes(x = log_proyecciones, y = log_recaudacion)) +
geom_point(alpha = 0.2)  +
geom_line(
  aes(x = log_proyecciones, y = prediccion_ps, color = "P-splines"),
  size = 1) +
geom_line(
  aes(x = log_proyecciones, y = prediccion_cs, color = "Cubic-splines"),
  size = 1) +
geom_line(
  aes(x = log_proyecciones, y = prediccion_nn, color = "red-neuronal"),
  size = 0.7) +
```

```
scale_color_manual(
  breaks = c("P-splines", "Cubic-splines", "red-neuronal"),
  values = c("P-splines"="red", "Cubic-splines"="blue", "red-neuronal"="green")) +
labs(title = "Recaudación vs Número de proyecciones") +
theme_bw() +
theme(legend.position = "bottom")
```



Cando el modelo tiene un único predictor, resulta útil para su interpretación representar cómo varía la distribución estimada para la variable respuesta en función del predictor. La función `plotSimpleGamlss()` del paquete `gamlss.util` genera este tipo de gráficos.

```
modelo_gamlss <- gamlss(
                  formula = log_recaudacion ~ pb(log_proyecciones),
                  sigma.formula = log_recaudacion ~ pb(log_proyecciones),
                  data    = datos,
                  family  = NO,
                  trace   = FALSE
                )

library(gamlss.util)
library(colorspace)
plotSimpleGamlss(
  y = log_recaudacion,
  x = log_proyecciones,
  model = modelo_gamlss,
  data  = datos,
  x.val = seq(6, 16, 2),
  val   = 5,
  N     =  1000,
  ylim  = c(0,25),
  cols  = heat_hcl(100)
)
```



Inferencia y predicción

El objetivo final de un modelo estadístico es entender la relación que existe entre la variable respuesta y los predictores, o para la predicción de nuevos valores.

Error de los coeficientes

El paquete `gamlss` incorpora dos funciones para calcular los errores estándar de los parámetros estimados: `vcov()` y `rvcov()`. La primera es la utilizada por defecto en el `summary()` del modelo, y la segunda es una estimación robusta (*robust or sandwich standard errors*). El orden en el que aparecen los resultados de cada parámetro es (μ, σ, ν, τ), aunque no todos los modelos tienen que tener los 4.

```
modelo_gamlss <- gamlss(
                   formula = log_recaudacion ~ log_proyecciones,
                   sigma.formula = log_recaudacion ~ log_proyecciones,
                   data    = datos,
                   family  = NO,
                   trace   = FALSE
                 )

vcov(modelo_gamlss, type = "se")
```

```
##      (Intercept) log_proyecciones      (Intercept) log_proyecciones
##      0.106965570      0.007378298      0.054075872      0.004490221
```

```
rvcov(modelo_gamlss, type = "se")
```

```
##      (Intercept) log_proyecciones      (Intercept) log_proyecciones
##      0.106793092      0.007090207      0.047121586      0.004005457
```

En cualquiera de los dos casos, para predictores que incluyan funciones *smooth*, su coeficiente solo contempla la contribución lineal, por lo tanto, no deben considerarse o interpretarse con mucha cautela.

Significancia

El *p-value* asociado a cada uno de los términos del modelo mostrado en el `summary` se calcula mediante el *Wald test*. Los propios autores del paquete recomiendan que, si este valor es de importancia para el analista, se emplee la función `drop1()` en su lugar. Esta función calcula, mediante un *generalized likelihood ratio test (GLRT)*, la significancia del impacto que tiene excluir del modelo cada término. En ambos casos, las estimaciones solo aplican si el modelo no incluye términos con funciones *smooth*. En el caso de que sí los incluya, el resultado obtenido solo sirve como una aproximación.

```
modelo_gamlss <- gamlss(
                   formula       = log_recaudacion ~ log_proyecciones,
                   sigma.formula = log_recaudacion ~ log_proyecciones,
                   data    = datos,
                   family  = NO,
                   trace   = FALSE
                 )

drop1(object = modelo_gamlss, parallel = "multicore", ncpus = 4)
```

```
## Single term deletions for
## mu
##
## Model:
## log_recaudacion ~ log_proyecciones
##                   Df   AIC    LRT   Pr(Chi)
## <none>               13965
## log_proyecciones  1 19904 5940.8 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Intervalos de confianza

EL paquete `gamlss` incorpora tres funciones distintas para calcular el intervalo de confianza de los parámetros estimados. Los resultados mostrados en el `summary` se corresponden con los calculados por la función `confint()`, que emplea intervalos de confianza basados en el *Wald standard error*. Las dos otras alternativas son las funciones `prof.dev()` y `prof.term()` que calculan intervalos basados en el *profile likelihood*.

```
modelo_gamlss <- gamlss(
                   formula       = log_recaudacion ~ log_proyecciones,
                   sigma.formula = log_recaudacion ~ log_proyecciones,
                   data    = datos,
                   family  = NO,
                   trace   = FALSE
                 )
```

La función `confint()` requiere que se especifique sobre qué parámetro se quiere calcular el intervalo de confianza.

```
confint(
  modelo_gamlss,
  what   = c("mu"),
  level  = 0.95,
  robust = TRUE
)
```

```
##                       2.5 %    97.5 %
## (Intercept)      2.9031354 3.3217566
## log_proyecciones 0.8548881 0.8826812
```

```
confint(
  modelo_gamlss,
  what   = c("sigma"),
  level  = 0.95,
  robust = TRUE
)
```

```
##                        2.5 %     97.5 %
## (Intercept)       1.9732621  2.3918833
## log_proyecciones -0.1726126 -0.1448195
```

Predicción

Una vez que el modelo `gamlss` ha sido entrenado, se puede predecir el valor de cada uno de los parámetros para nuevos valores de los predictores. Dos funciones pueden emplearse con este fin:

- `predict()`: genera las predicciones para uno de los parámetros del modelo, especificado por el usuario.
- `predictAll()`: genera las predicciones para todos parámetros del modelo.

Ambas funciones permiten calcular tres tipos distintos de predicciones. Su comportamiento se controla con el argumento `type`:

- `type="link"`: devuelve la predicción del parámetro en la escala de la función *link*. Por ejemplo, si la función *link* de la varianza del modelo es log, el valor devuelto es el logaritmo de la varianza.
- `type="response"`: devuelve la predicción del parámetro en la misma escala de la variable respuesta.
- `type="terms"`: muestra la contribución de cada término del modelo a la predicción.

```
modelo_gamlss <- gamlss(
                   formula = log_recaudacion ~ log_proyecciones,
                   sigma.formula = log_recaudacion ~ log_proyecciones,
                   data    = datos,
                   family  = NO,
                   trace   = FALSE
                 )
```

Para la media μ de la distribución normal, como su función *link* es la identidad, los resultados obtenidos empleando `type="link"` y `type="response"` son iguales.

```
predict(
  object  = modelo_gamlss,
  what    = "mu",
  type    = "link",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
## [1] 12.66908
```

```
predict(
  object  = modelo_gamlss,
  what    = "mu",
  type    = "response",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
## [1] 12.66908
```

```
predict(
  object  = modelo_gamlss,
  what    = "mu",
  type    = "terms",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
##   log_proyecciones
## 1       -0.6808943
## attr(,"constant")
## [1] 13.34997
```

En el caso de la varianza, la función *link* empleada es log, por lo que el resultado si se indica `type="response"` equivale al exponente del resultado obtenido con `type="link"`.

```
predict(
  object  = modelo_gamlss,
  what    = "sigma",
  type    = "link",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
## [1] 0.4366962
```

```
predict(
  object  = modelo_gamlss,
  what    = "sigma",
  type    = "response",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
## [1] 1.547586
```

La función `predictAll()` funciona del mismo modo, pero devuelve la predicción de todos los parámetros de la distribución a la vez.

```
predictAll(
  object  = modelo_gamlss,
  type    = "response",
  newdata = data.frame(log_proyecciones = 11)
)
```

```
## $mu
## [1] 12.66908
##
## $sigma
## [1] 1.547586
##
## attr(,"family")
## [1] "NO"     "Normal"
```

A la hora de generar predicciones, es muy importante tener en cuenta si el valor de los predictores está dentro del rango observado durante el entrenamiento del modelo. De no ser así, hay que ser muy cauto, ya que la relación aprendida entre la variable respuesta y los predictores podría no cumplirse fuera de las regiones observadas. Es buena práctica evitar las predicciones en regiones extrapoladas.

Selección de modelos

En términos generales, la utilidad de un modelo estadístico suele cuantificarse por su capacidad para explicar o predecir la variable respuesta en observaciones que el modelo no ha visto (observaciones de test). Encontrar el mejor modelo *GAMLSS*, requiere evaluar y comparar diferentes candidatos, cada uno con una combinación de: distribución para la variable respuesta, función *link* para cada uno de los parámetros, predictores e hiperparámetros.

Dos elementos son necesarios para lograrlo: una forma de generar los modelos candidatos y una forma de cuantificar cómo de capaz es cada modelo de aprender la distribución de la variable respuesta.

Métricas de ajuste

Tres métodos están disponibles en `gamlss` para cuantificar la capacidad predictiva de los modelos: *Generalized Akaike information criterion* (*GAIC*), validación simple y *cross-validation*.

GAIC

En modelos estadísticos paramétricos que han sido ajustados siguiendo una estrategia basada en el *likelihood*, una forma de cuantificar la bondad de ajuste del modelo es empleando el *fitted global deviance (GDEV)*, que no es más que el *log likelihood* del modelo multiplicado por -2.

$$\text{GDEV} = -2\log(likelihood)$$

El problema de emplear el *GDEV* para comparar modelos es que no tiene en cuenta los grados de libertad de cada uno (su flexibilidad). En términos generales, cuantos más parámetros tenga un modelo, con más facilidad se ajusta a los datos y menor es su *log likelihood*, pero a su vez, mayor es su riesgo de *overfitting*. Una forma de evitar este problema es mediante el uso del *GAIC* (*generalized Akaike information criterion*), que incorpora una penalización k por cada grado de libertad (*df*) que tenga el modelo:

$$\text{GAIC} = \text{GDEV} + k \times df = -2\log(likelihood) + k \times df$$

Dependiendo del grado de penalización, se favorece más o menos la sencillez del modelo. Dos estándares de esta métrica son el *AIC* (Criterio de información de Akaike) y *BIC* (*Bayesian information criterion*) también conocida como *SBC*. El primero utiliza como valor de penalización $k = 2$ y el segundo $k = \log(\text{nº observaciones})$.

$$\text{AIC} = -2\log(\text{likelihood}) + 2 \times \text{nº parametros}$$

$$\text{BIC} = -2\log(\text{likelihood}) + log(\text{nº observaciones}) \times \text{nº parametros}$$

En la práctica, *AIC* suele seleccionar modelos con un exceso de grados de libertad (*overfitting*), mientras *BIC/SBC* tiende al *underfitting*. Los autores del paquete `gamlss` recomiendan el uso de valores de k en el rango $2.5 \leq k \leq 4$ o de $k = \sqrt{\log(n)}$ cuando $n \geq 1000$.

Para todas estas métricas, cuanto menor sea el valor, mejor el ajuste. Es importante tener en cuenta que ninguna sirve para cuantificar la calidad del modelo en un sentido absoluto, sino para comparar la calidad relativa entre modelos. Si todos los ajustes candidatos son malos, no proporcionan ningún aviso de ello.

K-fold cross validation

El método de *K-fold cross validation* funciona de la siguiente forma. Las observaciones de entrenamiento se reparten en k conjuntos (*folds*) del mismo tamaño. El modelo se ajusta con todas las observaciones excepto las del primer *fold* y se evalúa con las observaciones del *fold* que ha quedado excluido, obteniendo así la primera métrica. El proceso se repite k veces, excluyendo un *fold* distinto en cada iteración. Al final del proceso, se generan k valores de la métrica que se agregan (normalmente con la media y la desviación típica) generando la estimación final de validación.

Validación simple

Las observaciones de entrenamiento se reparten en dos conjuntos, uno de entrenamiento y otro de validación/test. El modelo se ajusta con todas las observaciones del conjunto de entrenamiento y se evalúa las observaciones del conjunto de validación, obteniendo así la métrica.

Ejemplo

En el siguiente ejemplo se muestran varias funciones disponibles en `gamlss` para cuantificar la bondad de ajuste de modelos y compararlos entre ellos.

Empleando los datos de `rent` explicados en el primer de este documento, se entrenan 3 modelos distintos con el objetivo de identificar cuál de ellos predice mejor el precio de alquiler de las viviendas.

- `modelo_1`: predice el precio del alquiler en función de los metros cuadrados de la vivienda, asumiendo que la variable respuesta sigue una distribución normal.
- `modelo_2`: predice el precio del alquiler en función de los metros cuadrados de la vivienda y de su año de construcción, asumiendo que la variable respuesta sigue una distribución normal.
- `modelo_3`: predice el precio del alquiler en función de los metros cuadrados de la vivienda, asumiendo que la variable respuesta sigue una distribución gamma.
- `modelo_4`: predice el precio del alquiler en función de los metros cuadrados de la vivienda y de su año de construcción, asumiendo que la variable respuesta sigue una distribución gamma.

Datos

```
data("rent")
datos <- rent
datos <- datos %>% select(R, Fl, A, H, loc)
# Se renombran las variables para que sean más explicativas
colnames(datos) <- c("precio", "metros", "anyo", "calefaccion", "situacion")
```

Modelos

```
# Modelos candidatos
modelo_1 <- gamlss(formula = precio ~ metros,
                   family = NO, data = datos, trace = FALSE)
modelo_2 <- gamlss(formula = precio ~ metros + anyo,
                   family = NO, data = datos, trace = FALSE)
modelo_3 <- gamlss(formula = precio ~ metros,
                   family = GA, data = datos, trace = FALSE)
modelo_4 <- gamlss(formula = precio ~ metros + anyo,
                   family = GA, data = datos, trace = FALSE)
```

GAIC

```
GAIC(modelo_1, modelo_2, modelo_3, modelo_4, k = 3)
```

```
##          df      AIC
## modelo_4  4 28000.20
## modelo_3  3 28107.43
## modelo_2  4 28355.14
## modelo_1  3 28472.35
```

Cross-validation

```
cv_modelo_1 <- gamlssCV(formula = precio ~ metros, family = NO, data = datos,
                        K.fold = 10, parallel = "multicore", ncpus = 4,
                        set.seed = 1)
cv_modelo_2 <- gamlssCV(formula = precio ~ metros + anyo, family = NO, data = datos,
                        K.fold = 10, parallel = "multicore", ncpus = 4,
                        set.seed = 1)
cv_modelo_3 <- gamlssCV(formula = precio ~ metros, family = GA, data = datos,
                        K.fold = 10, parallel = "multicore", ncpus = 4,
                        set.seed = 1)
cv_modelo_4 <- gamlssCV(formula = precio ~ metros + anyo, family = GA, data = datos,
                        K.fold = 10, parallel = "multicore", ncpus = 4,
                        set.seed = 1)

CV(cv_modelo_1, cv_modelo_2, cv_modelo_3, cv_modelo_4)
```

```
##             val[o.val]
## cv_modelo_2   28358.70
## cv_modelo_1   28472.52
## cv_modelo_4   71465.24
## cv_modelo_3   71702.74
```

Validación simple

```
# Partición de los datos
set.seed(123)
id_train <- sample(x = 1:nrow(datos), size = nrow(datos)*0.8, replace = FALSE)
id_test  <- (1:nrow(datos))[-id_train]

# Modelos candidatos entrenados con la partición de entrenamiento
modelo_1 <- gamlss(formula = precio ~ metros,
                   family = NO, data = datos[id_train, ], trace = FALSE)
modelo_2 <- gamlss(formula = precio ~ metros + anyo,
                   family = NO, data = datos[id_train, ], trace = FALSE)
modelo_3 <- gamlss(formula = precio ~ metros,
                   family = GA, data = datos[id_train, ], trace = FALSE)
modelo_4 <- gamlss(formula = precio ~ metros + anyo,
                   family = GA, data = datos[id_train, ], trace = FALSE)

# Calculo métrica de test
test_modelo_1 <- getTGD(modelo_1, newdata = datos[id_test, ])
test_modelo_2 <- getTGD(modelo_2, newdata = datos[id_test, ])
test_modelo_3 <- getTGD(modelo_3, newdata = datos[id_test, ])
test_modelo_4 <- getTGD(modelo_4, newdata = datos[id_test, ])
TGD(test_modelo_1, test_modelo_2, test_modelo_3, test_modelo_4)
```

```
##                Pred.GD
## test_modelo_4 5569.789
## test_modelo_3 5598.020
## test_modelo_2 5623.987
## test_modelo_1 5656.076
```

Método de búsqueda

En el apartado anterior, se ha mostrado cómo comparar modelos asumiendo que se conoce cuáles son los candidatos a comparar. Cuando no se sabe qué predictores pueden ser los adecuados, conviene seguir una estrategia de búsqueda.

Una complejidad añadida de los modelos *GAMLSS* es que, cada parámetro del modelo, puede modelarse empleando distintos predictores. Por lo tanto, la búsqueda debe realizarse una vez por cada parámetro que tenga la distribución seleccionada. Las funciones `stepGAICAll.A()`, `stepGAICAll.B()` y `stepTGDAll.A()` ayudan en este proceso.

stepGAICAll.A()

```
modelo_nulo <- gamlss(precio~1, data = datos, family = GA, trace = FALSE)
mejor_modelo <- stepGAICAll.A(
                  object = modelo_nulo,
                  scope = list(lower=~1, upper=~metros+anyo+calefaccion+situacion),
                  k      = log(nrow(datos)),
                  parallel = "multicore",
                  ncpus  = 4
                )
```

```
## ----------------------------------------------------
## Distribution parameter:  mu
## Start:  AIC= 28626.75
##  precio ~ 1
##
##               Df   AIC
## + metros       1 28121
## + calefaccion  1 28422
## + situacion    2 28567
## + anyo         1 28593
## <none>           28627
##
## Step:  AIC= 28121.18
##  precio ~ metros
##
##               Df   AIC
## + calefaccion  1 27881
## + anyo         1 28018
## + situacion    2 28054
## <none>           28121
##
## Step:  AIC= 27881.43
##  precio ~ metros + calefaccion
##
##             Df   AIC
```

```
## + situacion  2 27833
## + anyo       1 27864
## <none>         27881
##
## Step:  AIC= 27832.68
##  precio ~ metros + calefaccion + situacion
##
##        Df   AIC
## + anyo  1 27818
## <none>    27833
##
## Step:  AIC= 27817.69
##  precio ~ metros + calefaccion + situacion + anyo
##
## ------------------------------------------------------
## Distribution parameter:  sigma
## Start:  AIC= 27817.69
##  ~1
##
##               Df   AIC
## + anyo         1 27782
## + calefaccion  1 27808
## <none>           27818
## + metros       1 27821
## + situacion    2 27824
##
## Step:  AIC= 27781.76
##  ~anyo
##
##               Df   AIC
## <none>           27782
## + calefaccion  1 27786
## + metros       1 27788
## + situacion    2 27789
## ------------------------------------------------------
## Distribution parameter:  mu
## Start:  AIC= 27781.76
##  precio ~ metros + calefaccion + situacion + anyo
##
##               Df   AIC
## <none>           27782
## - anyo         1 27807
## - situacion    2 27834
## - calefaccion  1 27902
## - metros       1 28383
## ------------------------------------------------------
```

```
mejor_modelo
```

```
##
## Family:  c("GA", "Gamma")
## Fitting method: RS()
##
## Call:  gamlss(formula = precio ~ metros + calefaccion + situacion +
##     anyo, sigma.formula = ~anyo, family = GA, data = datos,      trace = FALSE)
##
##
## Mu Coefficients:
##  (Intercept)        metros  calefaccion1    situacion2    situacion3
##      1.92187       0.01082      -0.28960       0.20406       0.27559
##         anyo
##      0.00198
## Sigma Coefficients:
## (Intercept)         anyo
##    5.971699    -0.003575
##
##  Degrees of Freedom for the fit: 8 Residual Deg. of Freedom   1961
## Global Deviance:     27721.1
##             AIC:     27737.1
##             SBC:     27781.8
```

El mejor modelo encontrado con `stepGAICAll.A()` emplea los predictores `metros`, `calefaccion`, `situacion` y `anyo` para modelar el parámetro μ, y únicamente el predictor `anyo` para modelar el parámetro σ.

stepGAICAll.B()

La función `stepGAICAll.B()` sigue la misma estrategia que `stepGAICAll.A()` pero obligando a que todos los parámetros de la distribución sean modelados con son los mismos predictores.

```
modelo_nulo <- gamlss(precio~1, data = datos, family = GA, trace = FALSE)
mejor_modelo <- stepGAICAll.B(
                  object = modelo_nulo,
                  scope = list(lower=~1, upper=~metros+anyo+calefaccion+situacion),
                  k      = log(nrow(datos)),
                  parallel = "multicore",
                  ncpus  = 4
                )
```

```
## Start:  AIC= 28626.75
##  precio ~ 1
##
##               Df   AIC
## + metros       2 28121
## + calefaccion  2 28424
## + anyo         2 28576
```

```
## + situacion    4 28580
## <none>           28627
##
## Step:  AIC= 28121.2
##  precio ~ metros
##
##               Df   AIC
## + calefaccion  2 27871
## + anyo         2 27973
## + situacion    4 28060
## <none>           28121
## - metros       2 28627
##
## Step:  AIC= 27870.78
##  precio ~ metros + calefaccion
##
##               Df   AIC
## + situacion    4 27835
## + anyo         2 27841
## <none>           27871
## - calefaccion  2 28121
## - metros       2 28424
##
## Step:  AIC= 27835.07
##  precio ~ metros + calefaccion + situacion
##
##               Df   AIC
## + anyo         2 27800
## <none>           27835
## - situacion    4 27871
## - calefaccion  2 28060
## - metros       2 28397
##
## Step:  AIC= 27799.49
##  precio ~ metros + calefaccion + situacion + anyo
##
##               Df   AIC
## <none>           27800
## - anyo         2 27835
## - situacion    4 27841
## - calefaccion  2 27914
## - metros       2 28401
```

```
mejor_modelo
```

```
##
## Family:  c("GA", "Gamma")
## Fitting method: RS()
##
## Call:  gamlss(formula = precio ~ metros + calefaccion + situacion +
```

```
##     anyo, sigma.formula = ~metros + calefaccion + situacion +
##     anyo, family = GA, data = datos, trace = FALSE)
##
## Mu Coefficients:
##  (Intercept)        metros  calefaccion1    situacion2    situacion3
##     1.911247      0.010903     -0.287947      0.201223      0.270280
##         anyo
##     0.001984
## Sigma Coefficients:
##  (Intercept)        metros  calefaccion1    situacion2    situacion3
##     4.934544      0.001269      0.075659     -0.106901     -0.157943
##         anyo
##    -0.003038
##
##  Degrees of Freedom for the fit: 12 Residual Deg. of Freedom   1957
## Global Deviance:     27708.5
##             AIC:     27732.5
##             SBC:     27799.5
```

GamboostLSS

La estrategia inicial propuesta por los autores para crear un modelo *GAMLSS*, es ajustar el modelo mediante el algoritmo de *backfitting Gaussian-Newton*, que tiene en cuenta el *log-likelihood del modelo*, y seleccionar los predictores e hiperparámetros en base a la métrica *Generalized Akaike information criterion* (*GAIC*). Uno de los problemas prácticos que puede tener el guiar la selección del mejor modelo en base al (*GAIC*) es que esta métrica se vuelve inestable a medida que aumenta el número de predictores candidatos y parámetro de la distribución (dimensionalidad).

Una alternativa es ajustar el modelo empleando el método de *component-wise gradient boosting* con el valor negativo de *log likelihood* como función de coste. De esta forma, se puede conseguir un ajuste incluso cuando el número de predictores supera al de observaciones, a la vez que se realiza la selección de los más relevantes. El paquete `gamboostLSS` combina las funcionalidades del paquete `gamlss` y `mboost` para entrenar modelos *GAMLSS* mediante *gradient boosting*.

Ejemplo

El ajuste de modelos con `gamboostLSS` es muy similar al realizado con `gamlss`. Para mostrarlo, se emplea un set de datos `india` con el que se crea un modelo predictivo del índice de malnutrición infantil.

El índice *stunted growth* se define como la diferencia estandarizada entre el crecimiento de un niño y el crecimiento medio (mediana) de todos los niños de la población a la que pertenece. Está considerado como una de las principales consecuencias de la malnutrición de las madres durante el embarazo o de los niños durante los primeros meses de vida. Valores negativos indican un retraso en el crecimiento del niño.

Se crea un modelo para predecir el *stunted growth* en función de la edad e índice de masa corporal de la madre y de los niños durante los primeros meses de vida.

```
library(gamboostLSS)
data(india)
datos <- india

head(datos)
```

```
##       stunting     cbmi cage  mbmi mage mcdist              mcdist_lab
## 7448      0.98 16.06220   10 21.26   26    367                   Jalna
## 23043     0.95 13.79388    4 19.43   26    418                North Goa
## 8237     -2.79 17.64873   16 23.35   24    381 Mumbai (Greater Mumbai)
## 9206     -3.13 16.93727   35 22.43   22    215            Jaintia Hills
## 4551     -1.32 16.20755   28 17.05   27    458               Malappuram
## 21928    -3.51 19.77523    8 18.64   20    223                    Patna
```

El paquete `gamboostLSS` tiene dos funciones para el ajuste de modelos: `glmboostLSS()` para modelos lineales y `gamboostLSS()` para todos los demás. En ambos casos, los argumentos principales son:

- `formula`: formula que define la variable respuesta y los predictores. Si solo se indica una fórmula, se emplea esa misma para estimar todos los parámetros de la distribución. También es posible pasar una lista en la que se especifica el modelo para cada uno de los parámetros.
- `families`: una lista con la familia de distribución para cada uno de los parámetros.
- `boost-control()`: parámetros para el algoritmo de boosting. De entre ellos destaca `mstop` que controla el ńumero de iteraciones de entrenamiento para cada parámetro, y `nu` que indica el ratio de aprendizaje (*learning rate*). Si solo se indica un valor de `mstops` se emplea ese número de iteraciones para ajustar todos los parámetros del modelo.

Aunque el paquete de `gamboostLSS` incorpora las principales distribuciones, se puede convertir cualquier distribución del paquete `gamlss.dist` con la función `as.families()`. Por ejemplo, para utilizar la función *gamma* se obtiene con `as.families("GA")`.

En cuanto a los *base learners* disponibles para el ajuste por *boosting*, `gamboostLSS` emplea los disponibles en el paquete `mboost`:

- `bbs()`: para incorporar relaciones no lineales mediante *penalized regression splines*.
- `bols()`: para relaciones lineales.

Se procede a ajustar un modelo *GAMLSS* que prediga el índice `stunting` en función de la edad e índice de masa corporal del niño y de la madre. Se emplea una distribución normal para la variable respuesta, se modelan sus dos parámetros (μ y σ) en función de los cuatro predictores, y se emplean *penalized regression splines* en todos los predictores para incorporar relaciones no lineales.

```
modelo <- gamboostLSS(
            formula = list(
                       mu = stunting ~ bbs(mage) + bbs(mbmi) + bbs(cage) + bbs(cbmi),
                       sigma = stunting ~ bbs(mage) + bbs(mbmi) + bbs(cage) + bbs(cbmi)
                      ),
            families = GaussianLSS(stabilization = "MAD"),
            data = india,
            control = boost_control(
                       trace = FALSE,
                       mstop = c(mu = 200, sigma = 100)
                      )
          )
```

Con el argumento `mstop = c(mu = 200, sigma = 100)` se está indicando que durante el ajuste, se vaya actualizando cada parámetro de forma alterna hasta alcanzar las 100 iteraciones A partir de ahí, el valor de *sigma* se mantiene constante y solo se sigue actualizando el de *mu* hasta alcanzar las 200 iteraciones.

```
summary(modelo)
```

```
##
##   LSS Models fitted via Model-based Boosting
##
## Call:
## gamboostLSS(formula = list(mu = stunting ~ bbs(mage) + bbs(mbmi) +     bbs(cage)
+ bbs(cbmi), sigma = stunting ~ bbs(mage) + bbs(mbmi) +     bbs(cage) + bbs(cbmi)),
data = india, families = GaussianLSS(stabilization = "MAD"),     control = boost_co
ntrol(trace = FALSE, mstop = c(mu = 200,         sigma = 100)))
##
## Number of boosting iterations (mstop):  mu = 200, sigma = 100
## Step size:  mu = 0.1, sigma = 0.1
##
## Families:
##
##   Normal distribution: mu(id link)
##
## Loss function: -dnorm(x = y, mean = f, sd = sigma, log = TRUE)
##
##
##   Normal distribution: sigma (log link)
##
## Loss function: -dnorm(x = y, mean = mu, sd = exp(f), log = TRUE)
##
## Selection frequencies:
## Parameter mu:
## bbs(cage) bbs(cbmi) bbs(mbmi) bbs(mage)
##     0.460     0.235     0.160     0.145
## Parameter sigma:
## bbs(mage) bbs(cage) bbs(cbmi) bbs(mbmi)
##      0.40      0.21      0.21      0.18
```

Como en todo modelo ajustado empleando *gradient boosting*, el principal hiperparámetro es el número de iteraciones de entrenamiento `mstop`, ya que esto impacta directamente en el balance entre generalización y *overfitting*. En este tipo de modelos, no hay una forma clara de definir los grados de libertad, por lo que no puede emplearse como métrica de validación el *GAIC*. En su lugar, se recurre a validación cruzada (*cross-validation*) o *bootstrapping* como forma de cuantificar la bondad de ajuste de un modelo *gamboostlss*.

La optimización del hiperparámetro `mstop` en los modelos *gamboostlss* tiene una complejidad añadida comparado a los modelos de *gradient boosting* tradicionales. En este caso existe un valor de `mstop` distinto para cada parámetro de la distribución del modelo, por lo que es necesario una búsqueda combinada de todos ellos (grid multidimensional).

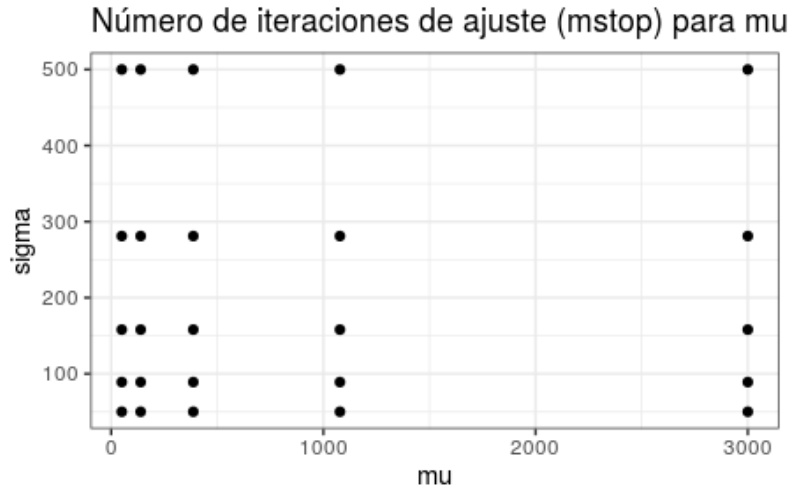
Todo este proceso puede hacerse con las funciones `cvrisk()` y `make.grid()`. En esta búsqueda es importante tener en cuenta la jerarquía de importancia en los parámetros del modelo. En términos generales, el parámetro μ de cualquier distribución suele ser más crítico que σ, por lo que vale la pena ser más exhaustivo en la búsqueda del valor de `mstop` asociado a este parámetro. Con este objetivo, la función `make.grid()` acepta el argumento `dense_mu_grid` para dar más granularidad en el grid de este parámetro.

Se realiza la búsqueda del mejor valor de `mstop` para cada parámetro del modelo anterior.

```
# Reducir length.out en caso de tiempo de ejecución excesivo
grid <- make.grid(
          max = c(mu = 3000, sigma = 500),
          min = 50,
          length.out = 5,
          dense_mu_grid = FALSE
        )

head(grid)
```

```
##     mu sigma
## 1   50    50
## 2  139    50
## 3  387    50
## 4 1078    50
## 5 3000    50
## 6   50    89
```

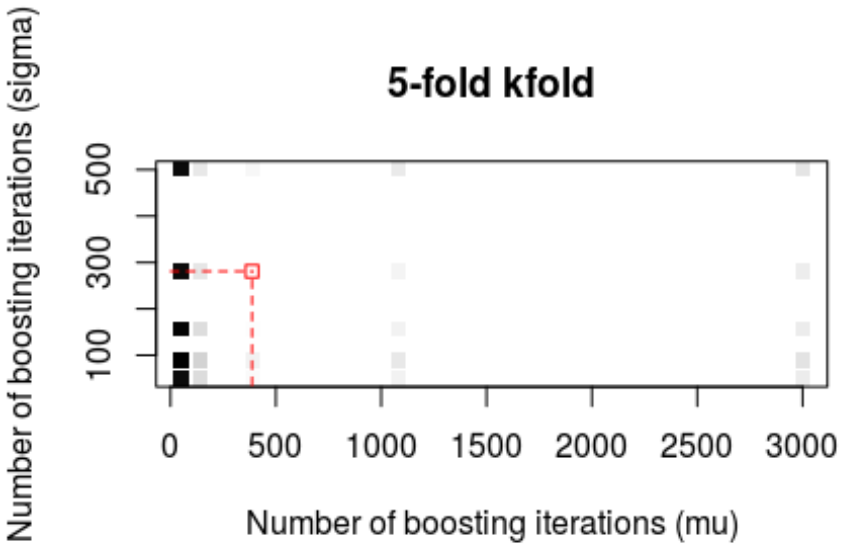
```
ggplot(data = grid, aes(x = mu, y = sigma)) +
  geom_point() +
  labs(title = "Número de iteraciones de ajuste (mstop) para mu y sigma") +
  theme_bw()
```




```
result_grid <- cvrisk(
                object = modelo,
                folds = cv(weights = model.weights(modelo), type = "kfold", B = 5),
                grid = grid,
                mc.cores = 4
              )

result_grid
```

```
##
##   Cross-validated risk
##   gamboostLSS(formula = list(mu = stunting ~ bbs(mage) + bbs(mbmi) +      bbs(ca
ge) + bbs(cbmi), sigma = stunting ~ bbs(mage) + bbs(mbmi) +      bbs(cage) + bbs(cb
mi)), data = india, families = GaussianLSS(stabilization = "MAD"),      control = b
oost_control(trace = FALSE, mstop = c(mu = 200,          sigma = 100)))
##
##    50,50   139,50   387,50  1078,50  3000,50    50,89   139,89   387,89
## 1.817459 1.811000 1.810080 1.810340 1.810504 1.817789 1.811279 1.810322
##  1078,89  3000,89   50,158  139,158  387,158 1078,158 3000,158   50,281
## 1.810628 1.810804 1.817695 1.810981 1.809992 1.810315 1.810507 1.817456
##  139,281  387,281 1078,281 3000,281   50,500  139,500  387,500 1078,500
## 1.810700 1.809929 1.810268 1.810471 1.817464 1.810625 1.810187 1.810572
## 3000,500
## 1.810748
##
##   Optimal number of boosting iterations: 387 281
```

```
plot(result_grid)
```



Con la función `mstop()` se extrae el valor óptimo de `mstop` encontrado en la búsqueda.

```
mejores_hiperparametros <- mstop(result_grid)
mejores_hiperparametros
```

```
##    mu sigma
##   387   281
```

Una vez identificados, se reestrena el modelo empleando los valores óptimos con la función `mstop()` aplicada a un modelo previamente definido.

```
mstop(modelo) <- mstop(result_grid)
modelo
```

```
##
##   LSS Models fitted via Model-based Boosting
##
## Call:
## gamboostLSS(formula = list(mu = stunting ~ bbs(mage) + bbs(mbmi) +     bbs(cage)
+ bbs(cbmi), sigma = stunting ~ bbs(mage) + bbs(mbmi) +     bbs(cage) + bbs(cbmi)),
data = india, families = GaussianLSS(stabilization = "MAD"),     control = boost_co
ntrol(trace = FALSE, mstop = c(mu = 200,         sigma = 100)))
##
## Number of boosting iterations (mstop):  mu = 387, sigma = 281
## Step size:  mu = 0.1, sigma = 0.1
##
## Families:
##
##   Normal distribution: mu(id link)
##
```

```
## Loss function: -dnorm(x = y, mean = f, sd = sigma, log = TRUE)
##
##
##   Normal distribution: sigma (log link)
##
## Loss function: -dnorm(x = y, mean = mu, sd = exp(f), log = TRUE)
##
```

Bibliografía

Stasinopoulos, Dm & Rigby, Robert & Heller, Gillian & Voudouris, Vlasios & De Bastiani, Fernanda. (2017). Flexible regression and smoothing: Using GAMLSS in R. 10.1201/b21973.

Stasinopoulos, D., & Rigby, R. (2007). Generalized Additive Models for Location Scale and Shape (GAMLSS) in R. Journal of Statistical Software, 23(7), 1 - 46. doi:http://dx.doi.org/10.18637/jss.v023.i07

Rigby, R.A. and Stasinopoulos, D.M. (2005), Generalized additive models for location, scale and shape. Journal of the Royal Statistical Society: Series C (Applied Statistics), 54: 507-554. doi:10.1111/j.1467-9876.2005.00510.x

Stasinopoulos, Mikis & Rigby, Bob & Akantziliotou, Calliope. (2008). Instructions on how to use the gamlss package in R Second Edition.

GAMLSS Practicals for the Bilbao short course October 2019 Mikis Stasinopoulos September 27, 2019

Hofner, B., Mayr, A., & Schmid, M. (2016). gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework. Journal of Statistical Software, 74(1), 1 - 31. doi:http://dx.doi.org/10.18637/jss.v074.i01