Ajuste de distribuciones con R

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

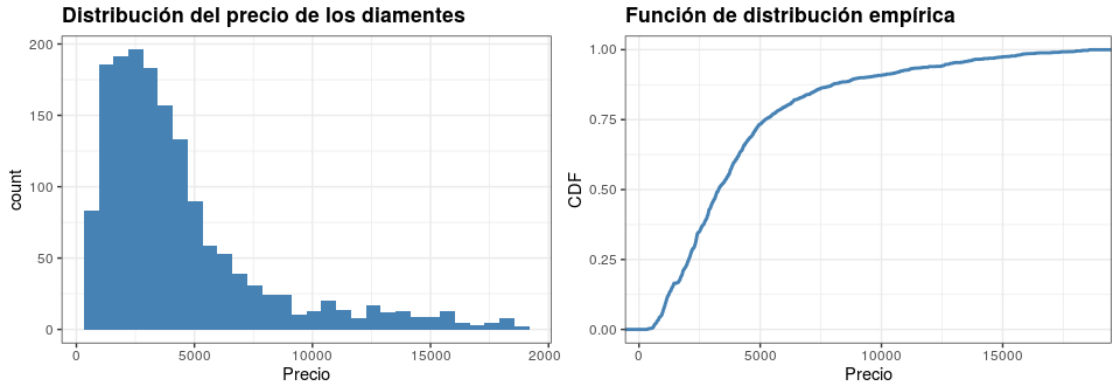
Identificar el tipo de distribución que tiene a una variable es un paso fundamental en prácticamente todos los estudios que implican datos, desde los contrastes de hipótesis hasta la creación de modelos por aprendizaje estadístico y machine learning.

En **R** existen varios paquetes que permiten ajustar distribuciones. En este documento se muestran las funcionalidades de los paquetes `univariateML`, `fitdistrplus` y `gamlss`, haciendo hincapié en cómo comparar múltiples distribuciones con el objetivo de identificar a cuál de ellas se ajustan mejor los datos.

Datos

Para esta demostración se emplean como datos el precio de los diamantes disponible en data set `diamonds`.

```
library(tidyverse)
datos <- diamonds %>% filter(cut == "Fair")

p1<- ggplot(data = datos, aes(x = price)) +
     geom_histogram(fill = "steelblue") +
     labs(title = "Distribución del precio de los diamentes",
          x = "Precio") +
     theme_bw() +
     theme(plot.title = element_text(face = "bold"))

p2 <- ggplot(data = datos, aes(x = price)) +
      stat_ecdf(geom = "step", color = "steelblue", size = 1) +
      labs(title = "Función de distribución empírica",
           x = "Precio",
           y = "CDF") +
      theme_bw() +
      theme(plot.title = element_text(face = "bold"))

ggpubr::ggarrange(plotlist = list(p1, p2), ncol = 2)
```



Métricas de ajuste

Ajustar una distribución paramétrica a partir de un conjunto de datos consiste en encontrar el valor de los parámetros con los que, con mayor probabilidad, dicha distribución puede haber generado los datos observados. Por ejemplo, la distribución normal tiene dos parámetros (media y varianza), una vez conocidos estos dos parámetros, se conoce toda la distribución.

Existen múltiples de métricas que permiten cuantificar cómo de bien se ajusta una distribución a los datos observados. Dos de las más empleadas son *AIC* (Criterio de información de Akaike) y *BIC* (*Bayesian information criterion*) también conocida como *SBC*. Ambas tienen en cuenta el *log-likelihood* y añaden una penalización proporcional el número de parámetros de la distribución (grados de libertad). Esto último hace posible comparar el ajuste entre distribuciones con diferente número de parámetro, ya que, en términos generales, cuantos más parámetros tenga una distribución, con más facilidad se ajusta a los datos y menor es su *log likelihood*.

La diferencia entre *AIC* y *BIC* es la severidad con la que penalizan el número de parámetros de la distribución.

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

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

Una generalización de estas métricas es el *GAIC* (*generalized Akaike information criterion*), en el que la penalización puede ser cualquier valor k:

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

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

UnivariateML

Introducción

El paquete UnivariateML permite ajustar distribuciones paramétricas mediante el método de *Maximum Likelihood*. La versión `1.0.0` contiene más de 20 distribuciones. En este listado puede encontrase información sobre el nombre, los parámetros que las caracterizan y el rango de valores (soporte) que puede tomar la distribución. Las funciones implementadas en `UnivariateML` permiten:

- Ajustar el modelo (distribución) a los datos.
- Estimar los parámetros de la distribución por *Maximum Likelihood*.
- Estimar la incertidumbre e intervalos de confianza de los parámetros mediante *bootstrap*.
- Comparar distintos modelos (ajustes) empleando las métricas *AIC* y *BIC*.
- Crear gráficos tipo *QQ*, *PP* y curvas de densidad para diagnóstico.
- Dada una determinada distribución, estimar probabilidades, cuantiles y simular nuevos datos.

Comparación de distribuciones

Se comparan varias distribuciones con el objetivo de identificar la mejor de ellas.

```
library(univariateML)

# Se comparan únicamente las distribuciones con un dominio [0, +inf)
comparacion_aic <- AIC(
                    mlbetapr(datos$price),
                    mlexp(datos$price),
                    mlinvgamma(datos$price),
                    mlgamma(datos$price),
                    mllnorm(datos$price),
                    mlrayleigh(datos$price),
                    mlinvgauss(datos$price),
                    mlweibull(datos$price),
                    mlinvweibull(datos$price),
                    mllgamma(datos$price)
                   )
comparacion_aic %>% rownames_to_column(var = "distribucion") %>% arrange(AIC)
```

```
##                 distribucion df      AIC
## 1       mllnorm(datos$price)  2 29775.43
## 2    mlinvgauss(datos$price)  2 29781.92
## 3      mllgamma(datos$price)  2 29787.68
## 4       mlgamma(datos$price)  2 29883.30
## 5      mlbetapr(datos$price)  2 29920.27
## 6    mlinvgamma(datos$price)  2 29920.51
## 7     mlweibull(datos$price)  2 29973.84
## 8  mlinvweibull(datos$price)  2 30021.99
## 9         mlexp(datos$price)  1 30205.41
## 10   mlrayleigh(datos$price)  1 30541.14
```

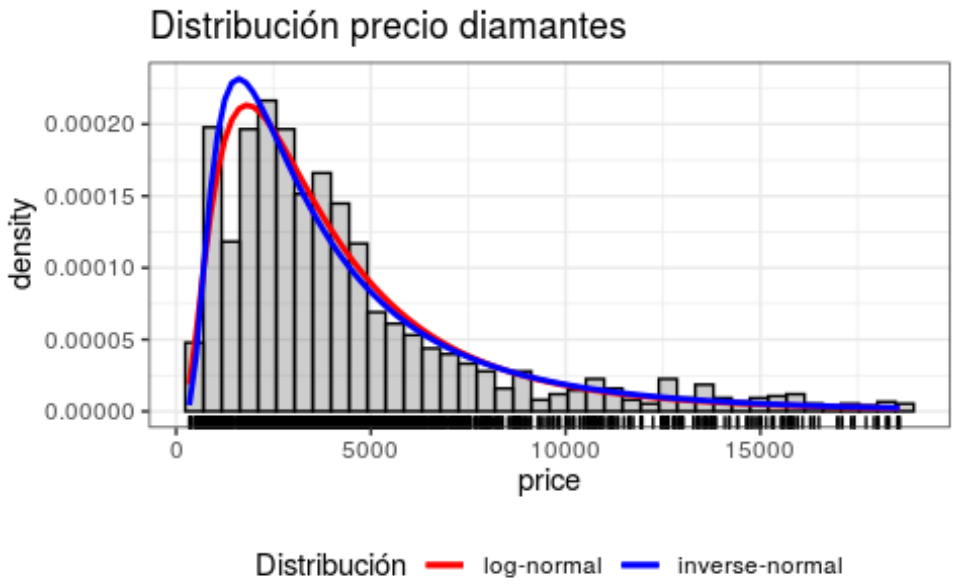
```
# Se comparan únicamente las distribuciones con un dominio [0, +inf)
comparacion_bic <- BIC(
                    mlbetapr(datos$price),
                    mlexp(datos$price),
                    mlinvgamma(datos$price),
                    mlgamma(datos$price),
                    mllnorm(datos$price),
                    mlrayleigh(datos$price),
                    mlinvgauss(datos$price),
                    mlweibull(datos$price),
                    mlinvweibull(datos$price),
                    mllgamma(datos$price)
                    )
comparacion_bic %>% rownames_to_column(var = "distribucion") %>% arrange(BIC)
```

```
##                 distribucion df      BIC
## 1       mllnorm(datos$price)  2 29786.20
## 2    mlinvgauss(datos$price)  2 29792.69
## 3      mllgamma(datos$price)  2 29798.45
## 4       mlgamma(datos$price)  2 29894.06
## 5      mlbetapr(datos$price)  2 29931.04
## 6    mlinvgamma(datos$price)  2 29931.27
## 7     mlweibull(datos$price)  2 29984.61
## 8  mlinvweibull(datos$price)  2 30032.76
## 9         mlexp(datos$price)  1 30210.80
## 10   mlrayleigh(datos$price)  1 30546.52
```

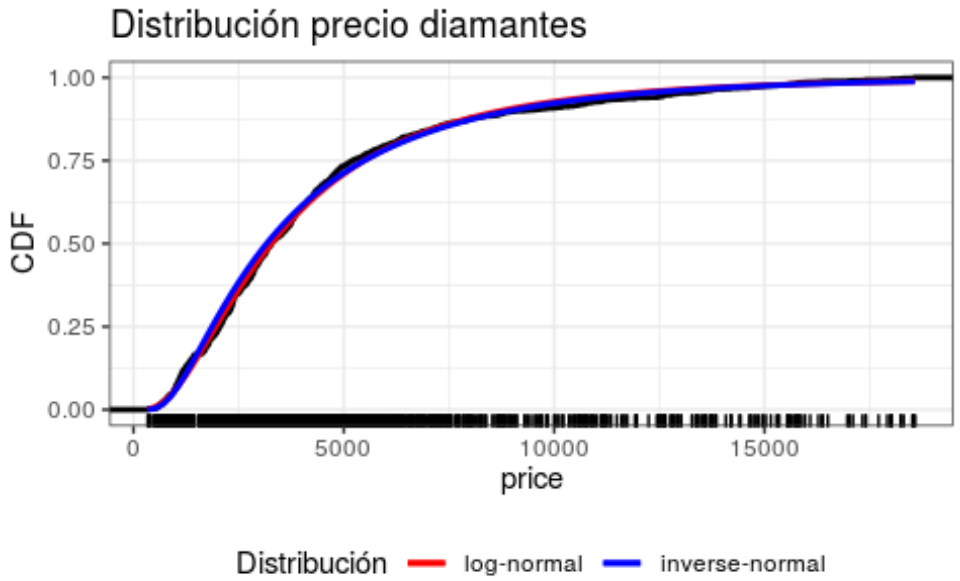
Acorde a la comparación por *AIC* y por *BIC*, las dos distribuciones que mejor se ajustan a los datos son la Log-normal y la *Inverse Gaussian*.

Representación gráfica de distribuciones

```
hist(datos$price,
     main = "Distribución precio diamantes",
     freq = FALSE,
     ylim = c(0, 0.00025))
lines(mllnorm(datos$price), lwd = 2, lty = 1, col = "blue")
lines(mlinvgauss(datos$price), lwd = 2, lty = 2, col = "red")
legend(x = 15000, y = 0.0001, legend = c("lnorm", "invgauss"),
       col = c("blue", "red"), lty = 1:2)
rug(datos$price)

ggplot(data = datos) +
  geom_histogram(aes(x = price, y =  after_stat(density)),
                 bins = 40,
                 alpha = 0.3, color = "black") +
  geom_rug(aes(x = price)) +
  stat_function(fun = function(.x){dml(x = .x, obj = mllnorm(datos$price))},
                aes(color = "log-normal"),
                size = 1) +
  stat_function(fun = function(.x){dml(x = .x, obj = mlinvgauss(datos$price))},
                aes(color = "inverse-normal"),
                size = 1) +
  scale_color_manual(breaks = c("log-normal", "inverse-normal"),
                     values = c("log-normal" = "red", "inverse-normal" = "blue")) +
  labs(title = "Distribución precio diamantes",
       color = "Distribución") +
  theme_bw() +
  theme(legend.position = "bottom")
```



```
ggplot(data = datos) +
  stat_ecdf(aes(x = price), geom = "step", color = "black", size = 1) +
  geom_rug(aes(x = price)) +
  stat_function(fun = function(.x){pml(q = .x, obj = mllnorm(datos$price))},
                aes(color = "log-normal"),
                size = 1) +
  stat_function(fun = function(.x){pml(q = .x, obj = mlinvgauss(datos$price))},
                aes(color = "inverse-normal"),
                size = 1) +
  scale_color_manual(breaks = c("log-normal", "inverse-normal"),
                     values = c("log-normal" = "red", "inverse-normal" = "blue")) +
  labs(title = "Distribución precio diamantes",
       color = "Distribución",
       y = "CDF") +
  theme_bw() +
  theme(legend.position = "bottom")
```



Ajuste de una distribución

Se ajusta una distribución log-normal a los datos de precio.

```
# Ajuste de una distribución log-normal
distribucion <- mllnorm(x = datos$price)
```

El objeto `univariateML` almacena el valor estimado para cada uno de los parámetros.

```
summary(distribucion)
```

```
##
## Maximum likelihood for the Lognormal model
##
## Call:  mllnorm(x = datos$price)
##
## Estimates:
##   meanlog      sdlog
## 8.0934414  0.7659848
##
## Data:            datos$price (1610 obs.)
## Support:         (0, Inf)
## Density:         stats::dlnorm
## Log-likelihood:  -14885.72
```

Intervalos de confianza por bootstrapping

Además del valor de máxima verosimilitud de cada parámetro, es útil conocer la incertidumbre de esta estimación. Una forma de hacerlo es mediante intervalos de confianza *bootstrap*.

```
# Intervalo de confianza del 95% estimados por bootstrapping
bootstrapml(distribucion, probs = c(0.05, 0.95), reps = 1000)
```

```
##                5%       95%
## meanlog 8.0631897 8.1238516
## sdlog   0.7434969 0.7883493
```

Función de densidad, cuantil y muestreo

Con las funciones `dml()`, `pml()`, `qml()` y `rml()` se puede calcular la densidad, probabilidad de acumulada, cuantiles, y muestreo de nuevos valores de cualquiera de las distribuciones disponibles en el paquete. Por ejemplo, se pueden simular 5 nuevos valores de diamantes acorde a la distribución ajustada.

```
set.seed(123)
rml(n = 5, obj = distribucion)
```

```
## [1]  2130.529  2743.882 10800.937  3454.558  3613.651
```

fitdistrplus

Introducción

Acorde a sus autores, el paquete fitdistrplus está orientado a facilitar las tareas de:

- Ajustar distribuciones por los métodos: *maximum likelihood estimation (MLE)*, *moment matching estimation (MME)*, *quantile matching estimation (QME)* y *maximum goodness-of-fit estimation (MGE)*.
- Incorporar distribuciones censuradas.
- Evaluar y diagnosticar de los ajustes.

Las distribuciones disponibles son: *norm*, *lnorm*, *exp*, *pois*, *cauchy*, *gamma*, *logis*, *nbinom*, *geom*, *beta* y *weibull* del paquete `stats`; *invgamma*, *llogis*, *invweibull*, *pareto1* y *pareto* del paquete `actuar`.

Descripción de la distribución

La función `descdist` genera los estadísticos principales que describen una distribución: *mínimo*, *máximo*, *media*, *mediana*, *desviación estándar*, *skewness* y *kurtosis*. Un valor de *skewness* distinto de cero indica una falta de simetría en la distribución, mientras que la *kurtosis* cuantifica el peso de las colas en comparación a una distribución normal con una *kurtosis* de 3. Cabe resaltar que los estadísticos de *skewness* y *kurtosis* son poco robustos (estimaciones por *bootstrapping* son más adecuadas).

```
library(fitdistrplus)

descdist(data = datos$price, graph = FALSE)
```

```
## summary statistics
## ------
## min:  337   max:  18574
## median:  3282
## mean:  4358.758
## estimated sd:  3560.387
## estimated skewness:  1.783535
## estimated kurtosis:  6.088025
```

Ajuste de una distribución

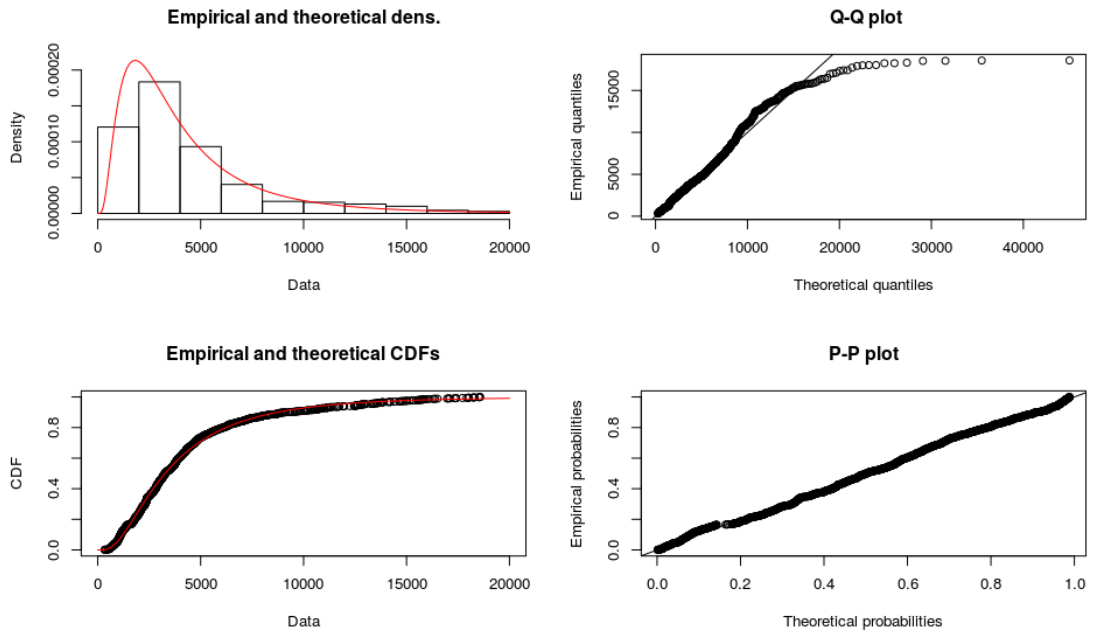
Se ajusta una distribución log-normal a los datos de precio por *maximum likelihood*.

```
distribucion <- fitdist(datos$price, distr = "lnorm")
summary(distribucion)
```

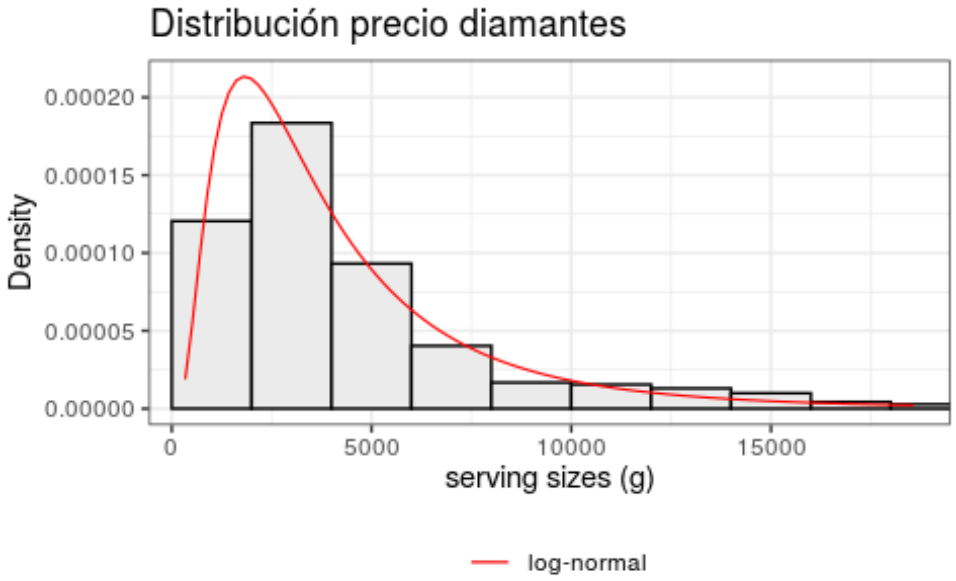
```
## Fitting of the distribution ' lnorm ' by maximum likelihood
## Parameters :
##          estimate Std. Error
## meanlog 8.0934414 0.01909006
## sdlog   0.7659848 0.01349860
## Loglikelihood:  -14885.72   AIC:  29775.43   BIC:  29786.2
## Correlation matrix:
##         meanlog sdlog
## meanlog       1     0
## sdlog         0     1
```

Representación gráfica de distribuciones

```
plot(distribucion)
```

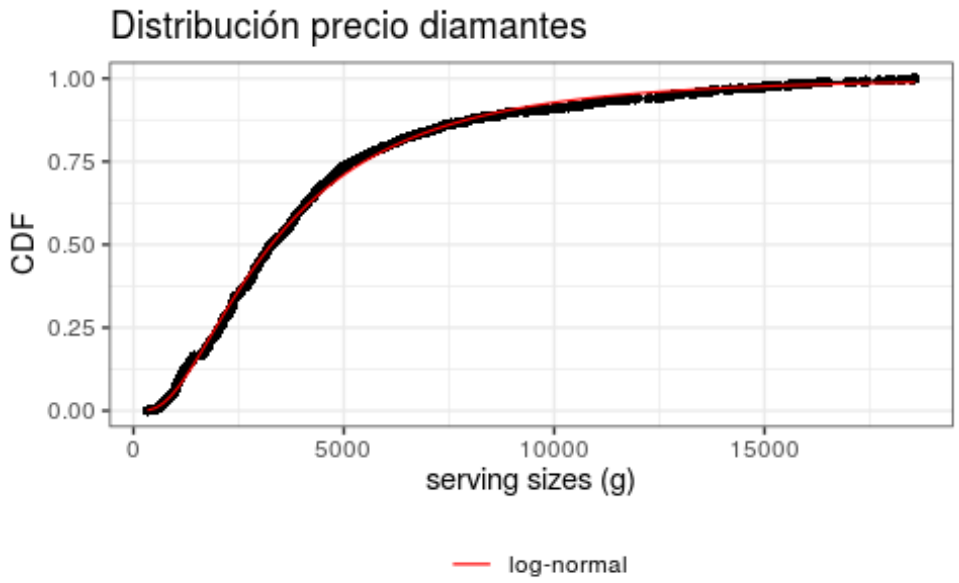


```
p <- denscomp(
       list(distribucion),
       legendtext = c("log-normal"),
       xlab = "serving sizes (g)",
       #xlim = c(0, 250),
       fitcol = c("red"),
       fitlty = 1,
       xlegend = "topright",
       plotstyle = "ggplot",
       addlegend = FALSE
      )

p <- p +
      ggplot2::ggtitle("Distribución precio diamantes") +
      theme_bw() +
      theme(legend.position = "bottom")

p
```




```
p <- cdfcomp(
       list(distribucion),
       legendtext = c("log-normal"),
       xlab = "serving sizes (g)",
       #xlim = c(0, 250),
       fitcol = c("red"),
       fitlty = 1,
       xlegend = "topright",
       plotstyle = "ggplot",
       addlegend = FALSE
      )
```

```
p <- p +
     ggplot2::ggtitle("Distribución precio diamantes") +
     theme_bw() +
     theme(legend.position = "bottom")

p
```



Comparación de distribuciones

Para comparar distribuciones con el paquete `descdist`, primero hay que ajustar cada una de las distribuciones candidata por separado. Una vez ajustadas, se pasan a la función `gofstat()` para obtener las métricas de cada una de ellas.

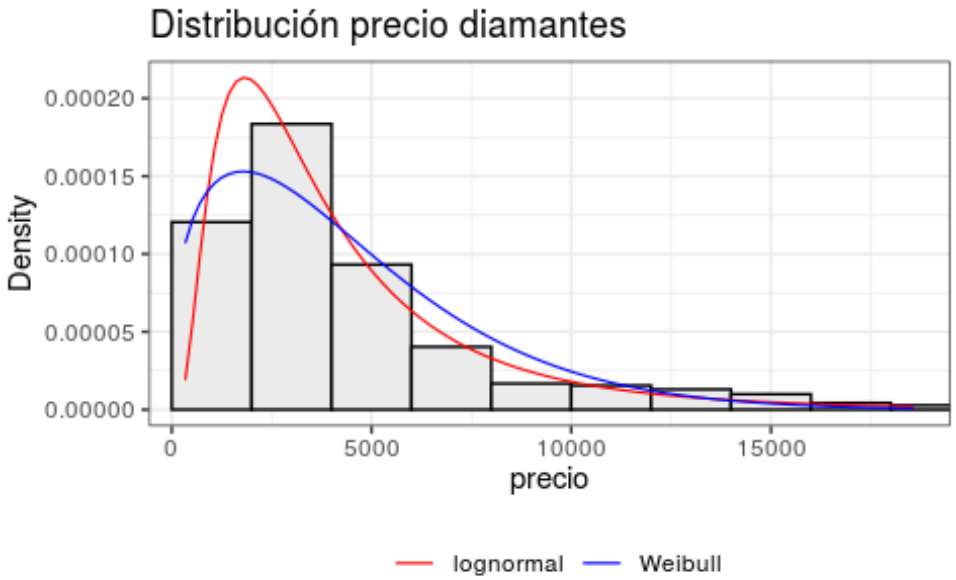
```
# Distribución log-nomral
dist_lnorm <- fitdist(datos$price, distr = "lnorm")
# Distribución weibull
dist_weibull <- fitdist(datos$price, distr = "weibull")

comparacion <- gofstat(f = list(dist_lnorm, dist_weibull))
comparacion
```

```
## Goodness-of-fit statistics
##                               1-mle-lnorm 2-mle-weibull
## Kolmogorov-Smirnov statistic  0.02728292    0.08497167
## Cramer-von Mises statistic    0.31557777    3.11751571
## Anderson-Darling statistic    2.47232876   20.47556604
##
```

```
## Goodness-of-fit criteria
##                                1-mle-lnorm 2-mle-weibull
## Akaike's Information Criterion    29775.43      29973.84
## Bayesian Information Criterion    29786.20      29984.61
```

Además de los estadísticos *AIC* y *BIC*, la función `gofstat()` devuelve 3 estadísticos de bondad de ajuste, (Kolmogorov-Smirnov, *Cramer-von Mises* y *Anderson-Darling*). Estos estadísticos, también conocidos como *goodness-of-fit*, contrastan la similitud entre la distribución empírica obtenida y la distribución teórica con los parámetros estimados. Ninguno de estos 3 últimos tiene en consideración el número de parámetros, por lo que no deben emplearse para comparar distribuciones con distintos grados de libertad.

```
p <- denscomp(
      list(dist_lnorm, dist_weibull),
      legendtext = c("lognormal", "Weibull"),
      xlab = "precio",
      fitcol = c("red", "blue"),
      fitlty = 1,
      xlegend = "topright",
      plotstyle = "ggplot",
      addlegend = FALSE)
p <- p +
     ggplot2::ggtitle("Distribución precio diamantes") +
     theme_bw() +
     theme(legend.position = "bottom")

p
```



gamlss

Introducción

El paquete `gamlss` está orientado a la creación de modelos aditivos generalizados para posición, escala y forma **GAMLSS** (*Generalized Additive Model for Location, Scale and Shape*). Entre las muchas funciones que contiene, `fitDist()` permite ajustar un amplio abanico de distribuciones paramétricas.

Comparación de distribuciones

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*).

```
library(gamlss)
library(gamlss.dist)

distribuciones <- fitDist(
                    datos$price,
                    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 GIG          29771.
##  2 LOGNO        29775.
##  3 LNO          29775.
##  4 LOGNO2       29775.
##  5 GG           29777.
##  6 BCCG         29777.
##  7 BCCGo        29777.
##  8 BCPE         29779.
##  9 BCPEo        29779.
## 10 BCT          29779.
## # … with 13 more rows
```

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

```
summary(distribuciones)
```

```
## ******************************************************************
## Family:  c("GIG", "Generalised Inverse Gaussian")
##
## Call:  gamlssML(formula = y, family = DIST[i], data = sys.parent())
##
## Fitting method: "nlminb"
##
## Coefficient(s):
##             Estimate  Std. Error   t value   Pr(>|t|)
## eta.mu     8.3799443   0.0205003 408.77140 < 2.22e-16 ***
## eta.sigma -0.1345106   0.0201044  -6.69059 2.2227e-11 ***
## eta.nu     0.1672232   0.1834827   0.91138    0.36209
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Degrees of Freedom for the fit: 3 Residual Deg. of Freedom   1607
## Global Deviance:     29764.7
##             AIC:     29770.7
##             SBC:     29786.8
```

Ajuste de una distribución

Se ajusta una distribución log-normal a los datos de precio.

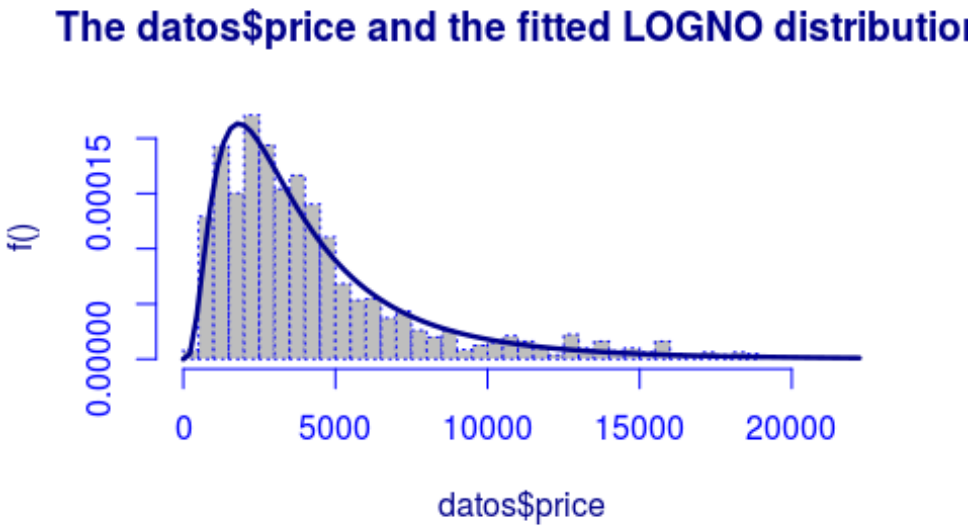
```
dist_gig <- gamlss(
              formula = datos$price~1,
              family  = LOGNO,
              data    = datos,
              trace   = FALSE
            )
summary(dist_gig)
```

```
## ******************************************************************
## Family:  c("LOGNO", "Log Normal")
##
## Call:  gamlss(formula = datos$price ~ 1, family = LOGNO, data = datos,
##     trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.09344    0.01909     424   <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) -0.26659    0.01762  -15.13   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  1610
```

```
## Degrees of Freedom for the fit:  2
##       Residual Deg. of Freedom:  1608
##                       at cycle:  2
##
## Global Deviance:     29771.43
##             AIC:     29775.43
##             SBC:     29786.2
## ******************************************************************
```

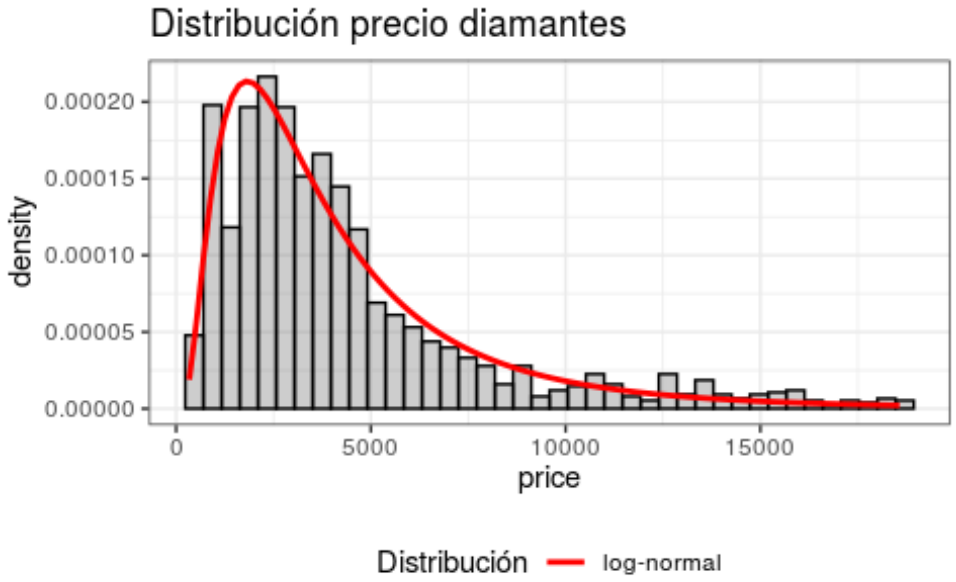
Representación gráfica de distribuciones

```
histDist(datos$price, family=LOGNO, nbins=30, line.col="darkblue", line.wd=2.5)
```




```
## Family:  c("LOGNO", "Log Normal")
## Fitting method: "nlminb"
##
## Call:  gamlssML(formula = datos$price, family = "LOGNO", data = sys.parent())
##
## Mu Coefficients:
## [1]  8.093
## Sigma Coefficients:
## [1]  -0.2666
##
##  Degrees of Freedom for the fit: 2 Residual Deg. of Freedom   1608
## Global Deviance:     29771.4
##             AIC:     29775.4
##             SBC:     29786.2
```

```
ggplot(data = datos) +
  geom_histogram(aes(x = price, y =  after_stat(density)),
                 bins = 40,
                 alpha = 0.3, color = "black") +
  stat_function(fun = function(.x){dLOGNO(x = .x, mu = 8.093, sigma = exp(-
0.2666))},
                aes(color = "log-normal"),
                size = 1) +
  scale_color_manual(breaks = c("log-normal"),
                     values = c("log-normal" = "red")) +
  labs(title = "Distribución precio diamantes",
       color = "Distribución") +
  theme_bw() +
  theme(legend.position = "bottom")
```



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.

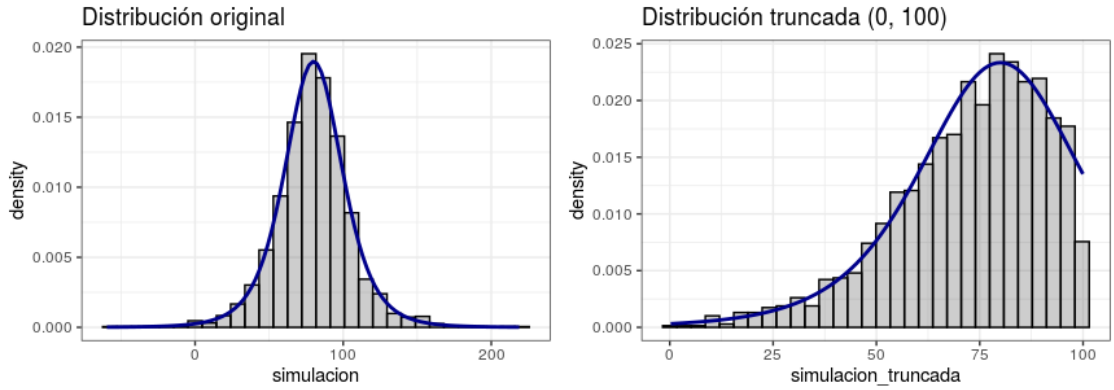
La versión truncada de cualquier distribución presente en `gamlss.family` puede obtenerse gracias a la función `gen.trun()` del paquete `gamlss.tr`. La idea es sencilla: se parte una distribución ya existente en `gamlss`, se determinan los límites de truncado (izquierda, derecha o ambos) y automáticamente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

ggpubr::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.

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

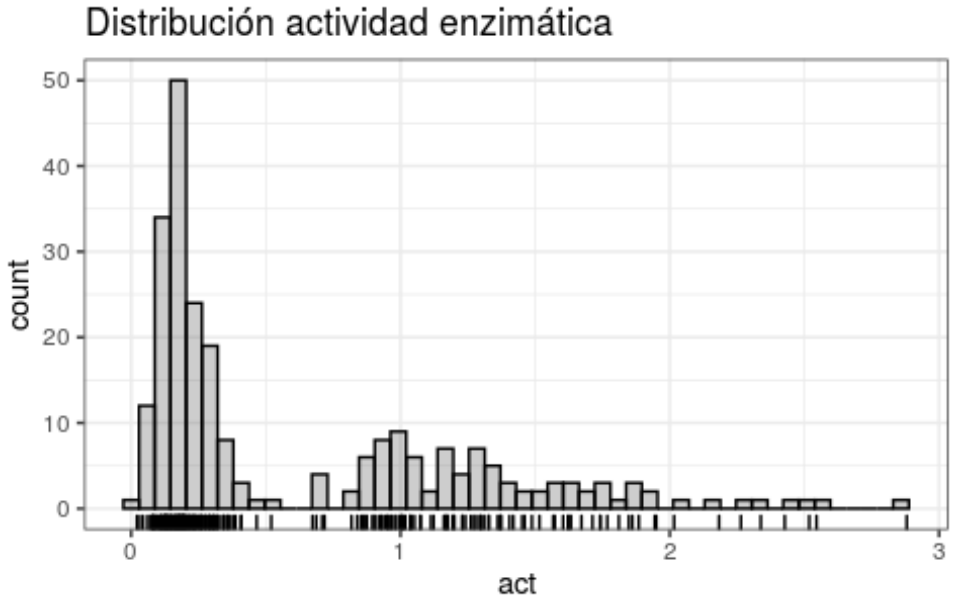
- `gamlssMX()`: cuando las distribuciones no tienen parámetros en común, pudiendo incluso ser distribuciones de distintas familias.
- `gamlssNP()`: cuando las distribuciones tienen parámetros en común.

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
```

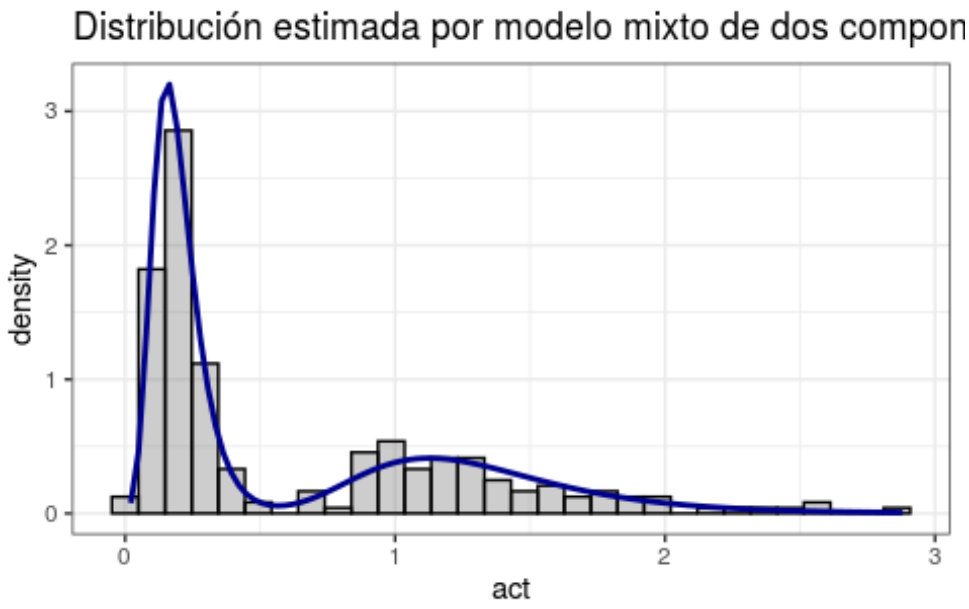
```
##
## Mixing Family:  c("RG", "RG")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = act ~ 1, family = RG, K = 2, data = enzyme,
##     control = MX.control(plot = FALSE))
##
## Mu Coefficients for model: 1
## (Intercept)
##      0.1557
## Sigma Coefficients for model: 1
## (Intercept)
##      -2.641
## Mu Coefficients for model: 2
## (Intercept)
##       1.127
## Sigma Coefficients for model: 2
## (Intercept)
##      -1.091
##
## Estimated probabilities: 0.6239827 0.3760173
##
## 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] 0.07719847
```

```
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*).

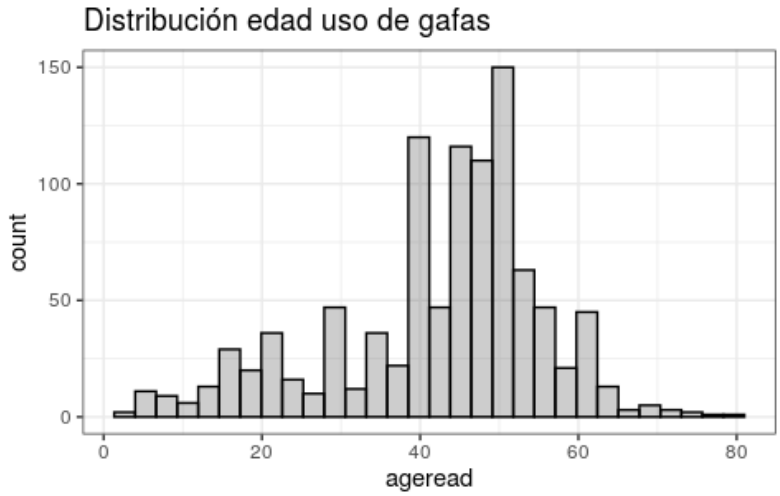
```
modelo_mx <- gamlssMXfits(
                n       = 10,
                formula = act ~ 1,
                data    = enzyme,
                family  = RG,
                K       = 2,
                control = MX.control(plot = FALSE)
             )
```

Modelo con variables latentes

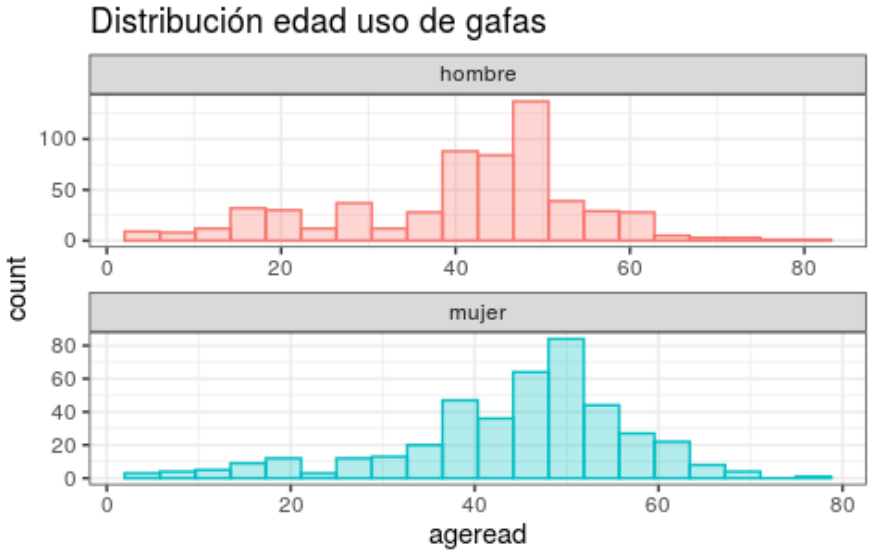
Los modelos de variables latentes se emplean en escenarios en los que se asume que la población objetivo está dividida en varias clases, pero se desconocen cuáles son. El objetivo de estos modelos es estimar la probabilidad que tiene cada observación de pertenecer a cada una de las clases. En la mayoría 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*.

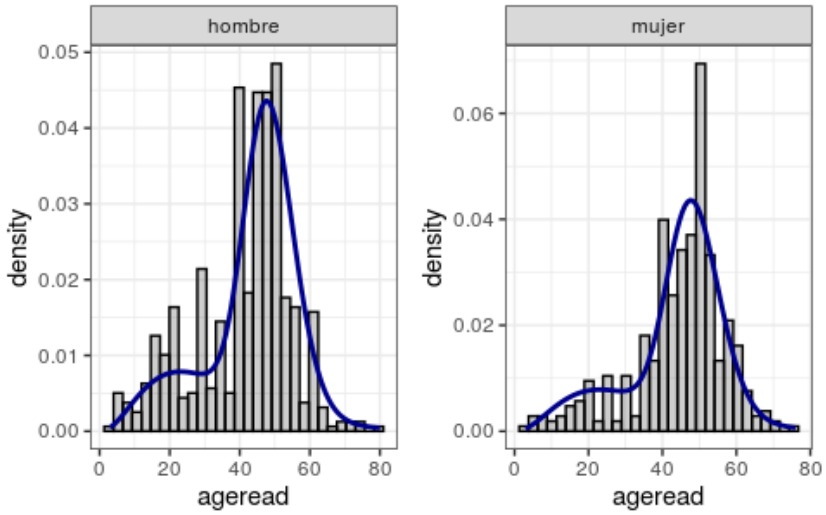
```
GAIC(
  modelo_mx_no,
  modelo_mx_ga
)
```

```
##              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()

ggpubr::ggarrange(p1, p2)
```

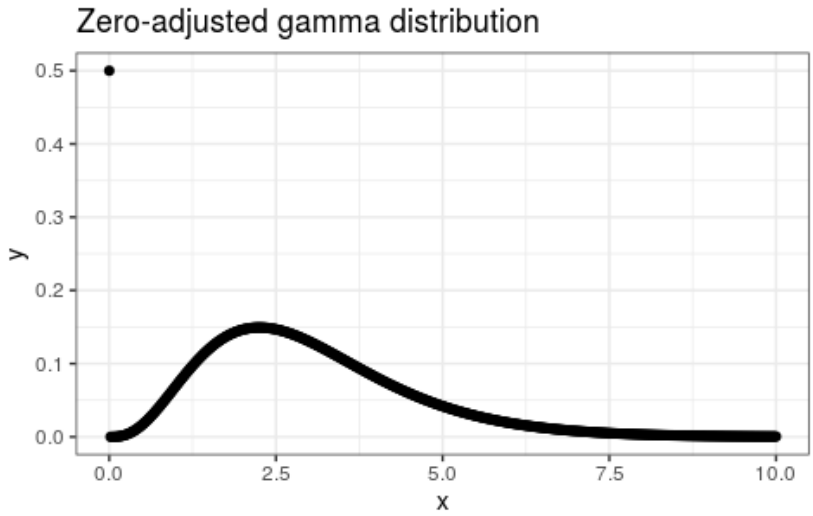


Dentro del objeto `gamlssMX` se almacenan información del modelo final, destacan: los modelos individuales que forman parte del modelo mixto (`$models`) y la probabilidad de cada observación de pertenecer a cada uno de las componentes (`$post.prob`).

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()`

```r
# 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()
```



Bibliografía

https://univariateml.netlify.com/index.html

Delignette-Muller, M., & Dutang, C. (2015). fitdistrplus: An R Package for Fitting Distributions. doi:http://dx.doi.org/10.18637/jss.v064.i04

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

Instructions on how to use the gamlss package in R Second Edition Mikis Stasinopoulos, Bob Rigby and Calliope Akantziliotou January 11, 2008

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