Regresión cuantílica: Gradient Boosting Quantile Regression

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Marzo, 2020

Tabla de contenidos

Versión PDF: Github

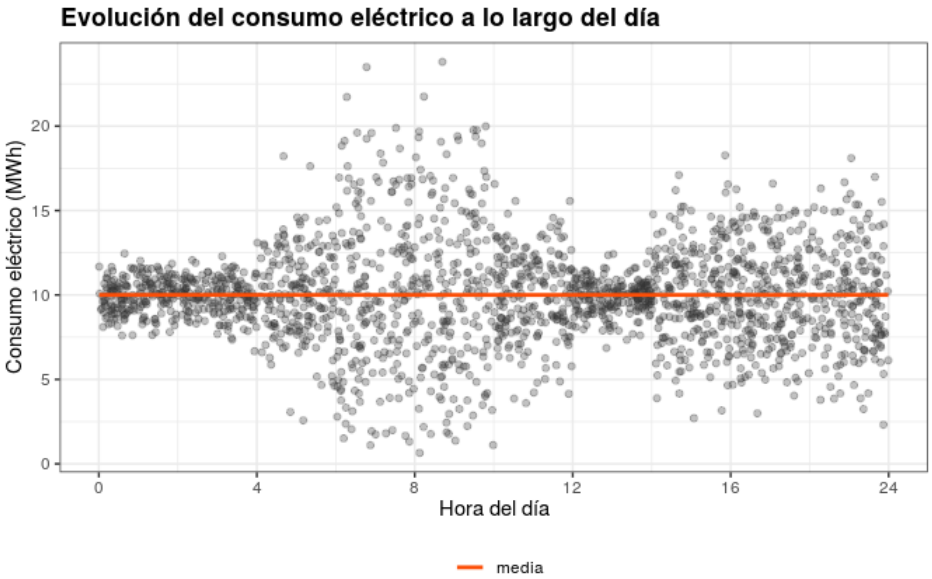
Más sobre ciencia de datos: cienciadedatos.net o joaquinamatrodrigo.github.io

Introducción

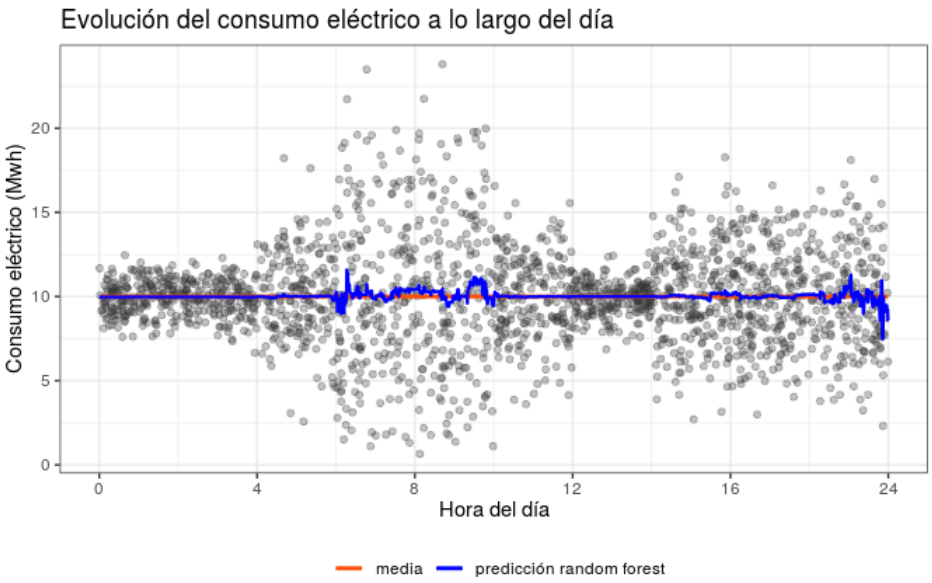
La predicción de una variable continua Y en función de uno o varios predictores X es un problema de aprendizaje supervisado que puede resolverse con múltiples métodos de Machine Learning y aprendizaje estadístico. Algunos de ellos consideran que la relación entre Y y X es únicamente lineal (regresión lineal, GLM), mientras que otros permiten incorporar relaciones no lineales o incluso interacciones entre predictores (SVM, Random Forest, Boosting). De una forma u otra, todos ellos tratan de inferir la relación entre X e Y.

El objetivo de la mayoría de estos algoritmos es predecir el valor promedio de Y en función del valor de X, $E(Y|X = x)$. Aunque conocer la media condicional es de utilidad, este resultado ignora otras características de la distribución de Y que pueden ser claves a la hora de tomar decisiones, por ejemplo, su dispersión.

Véase el siguiente ejemplo simulado (y muy simplificado) sobre la evolución del consumo eléctrico de todas las casas de una ciudad en función de la hora del día. Ver $Anexo^1$ con el código empleado para la simulación.



La media del consumo eléctrico es la misma durante todo el día, $\overline{consumo} = 10Mwh$, sin embargo, su dispersión no es constante (heterocedasticidad). Véase el resultado de predecir el consumo medio en función de la hora del día con un modelo Random Forest.

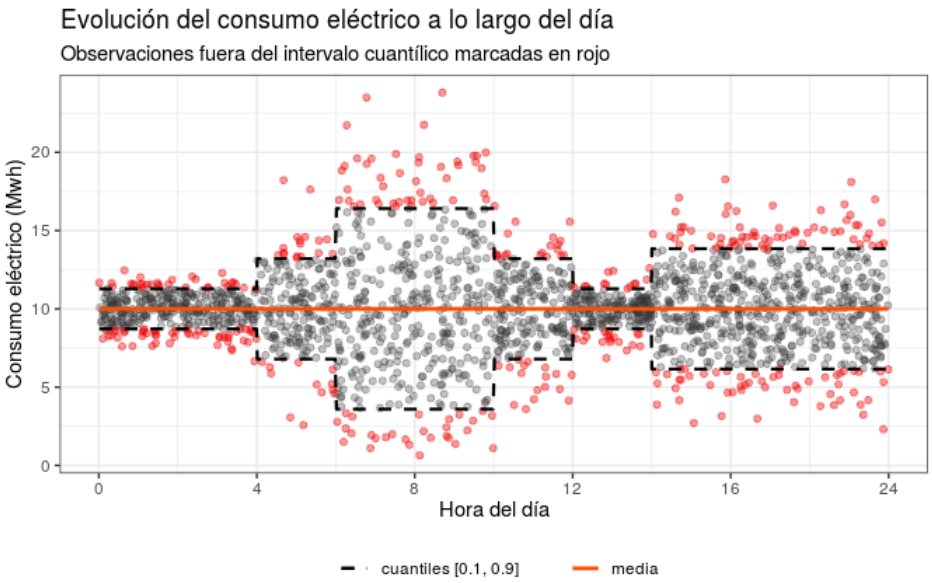


El valor predicho es muy próximo a la media real, es decir, el modelo es bueno prediciendo el consumo medio esperado. Ahora, imagínese que la compañía encargada de suministrar la electricidad debe de ser capaz de provisionar, en un momento dado, con hasta un 50% de electricidad extra respecto al promedio. Esto significa un máximo de 15 Mwh. Estar preparado para suministrar este extra de energía implica gastos de personal y maquinaría, por lo que la compañía se pregunta si es necesario estar preparado para producir tal cantidad durante todo el día, o si, por lo contrario, podría evitarse durante algunas horas, ahorrando así gastos.

Un modelo que predice únicamente el promedio no permite responder a esta pregunta, ya que tanto para las 2h de la mañana como para las 8h, el consumo promedio predicho es en torno a 10 Mwh, sin embargo, la probabilidad de que se alcancen consumos de 15 Mwh a las 2h es prácticamente nula mientras que esto ocurra a las 8h sí es razonable.

Una forma de describir la dispersión de una variable es el uso de cuantiles. El cuantil de orden τ $(0 < \tau < 1)$ de una distribución es el valor de la variable X que marca un corte tal que, una proporción τ de valores de la población, es menor o igual que dicho valor. Por ejemplo, el cuantil de orden 0.36 deja un 36% de valores por debajo y el cuantil de orden 0.50 el 50% (se corresponde con la mediana de la distribución).

Dado que los datos se han simulado empleando distribuciones normales, se conoce el valor de los cuantiles teóricos para cada X. Se muestra de nuevo el mismo gráfico pero esta vez añadiendo los cuantiles 0.1 y 0.9.



Si como resultado del modelo, además de la predicción de la media, se predice también el valor de los cuantiles, se dispone de una caracterización mayor de la distribución de la variable respuesta Y, y con ello se puede responder a más preguntas. Por ejemplo, en el caso de la energía, se tendría cierta seguridad al decir que, durante los intervalos de 0h a 4h y de 12h a 14h, es poco probable que se alcancen consumos de 15 Mwh.

Otros casos en los que conocer la distribución de cuantiles puede ser útil son:

- Identificación de regiones en las que la variable respuesta Y tiene mayor dispersión en torno a su media.
- Entrenar modelos que predicen la mediana (cuantil 0.5) en lugar de la media. Estos modelos son más robustos frente a *outliers*.
- Detectar anomalías, identificando aquellas observaciones que están fuera de un determinado intervalo cuantílico.

En los siguientes apartados se describe dos estrategias de cómo *Gradient Boosting* puede adaptarse para que aprenda la distribución de cuantiles.

Nota: Otras aproximaciones que tratan de resolver este mismo problema como son Quantile Regression Forest, Distributional Regression Forest: Random Forest probabilístico y GAMLSS.

Boosted Regression Splines

Introducción

Boosted Regression Splines es una adaptación del algoritmo de *Gradient Boosting* que emplea *Regression Splines* en lugar de árboles como *base learners*, y la función de coste propia de la regresión cuantílica. Como resultado de esta combinación se consiguen modelos lineales, en cuanto a la forma en que participan los predictores, pero permitiendo que la relación entre cada predictor y la variable respuesta, puede ser no lineal (*smooth*).

Se puede encontrar más detalles sobre el funcionamiento de cada uno de estos algoritmos en: Gradient Boosting, Splines y en el apartado de bibliografía.

Ejemplo

En **R** existen varios paquetes que incorporan la adaptación de *Gradient Boosting* para predecir cuantiles empleando *smooth splines* como *base learners*, dos de ellos son `mboost` y `GAMBoost`.

`mboost` permite ajustar todo un abanico de modelos (lineales, aditivos o con interacciones) empleando descenso de gradiente (*boosting*) con varios *base learners* y múltiples funciones de coste. Con la función `gamBoost()` se ajustan modelos que utilizan *smooth P-Splines* en cada uno de los predictores, en concreto *cubic P-splines* con 20 *knots* internos y 4 grados de libertad. Si además, se le indica el argumento `family = QuantReg()`, el ajuste es de tipo cuantílico.

Aunque este es el comportamiento que emplea por defecto `gamBoost()` para todos los predictores, con las funciones `bols()` y `bbs()` se puede controlar cómo participa cada predictor en el modelo: lineal o *splines*, así como la flexibilidad de los *splines*. Es altamente recomendable leer su documentación para conocer todas las posibilidades que ofrece este paquete.

Datos

Ver *Anexo*[1] para conocer más detalles de la simulación.

```
library(tidyverse)

# Simulación distribución no uniforme en el rango X
# ------------------------------------------------------------------------------
set.seed(12345)
n <- 2000
x <- runif(min = 0, max = 24, n = n)
y <- rnorm(
        n,
        mean = 10,
        sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
             1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
     )

# Cálculo del cuantil 0.1 y 0.9 para cada posición de x simulada.
cuantil_10 <- qnorm(
                p    = 0.1,
                mean = 10,
                sd   = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                       1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
              )

cuantil_90 <- qnorm(
                p    = 0.9,
                mean = 10,
                sd   = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                       1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
              )
datos <- data.frame(y, x, cuantil_10, cuantil_90)

# No puede haber consumos negativos
datos <- datos %>%
         filter(y >= 0)

datos <- datos %>%
         mutate(dentro_intervalo = ifelse(
                                     y > cuantil_10 & y < cuantil_90,
                                     TRUE,
                                     FALSE
                                   )
        )
```

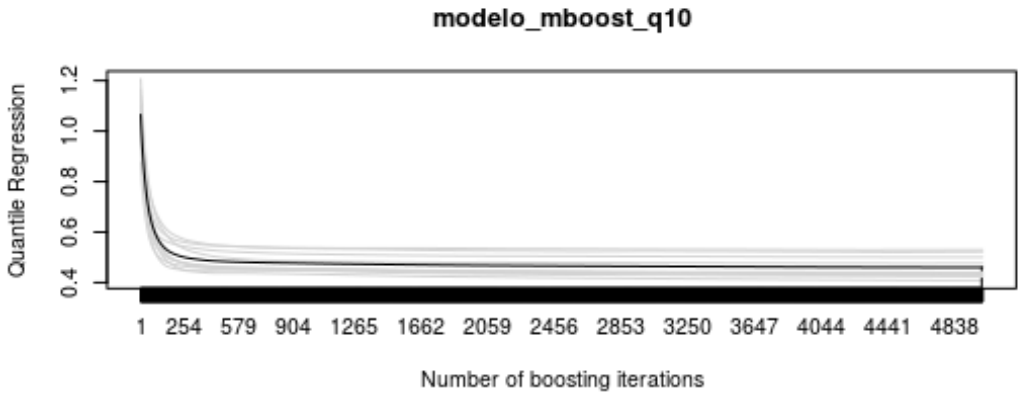
Modelo

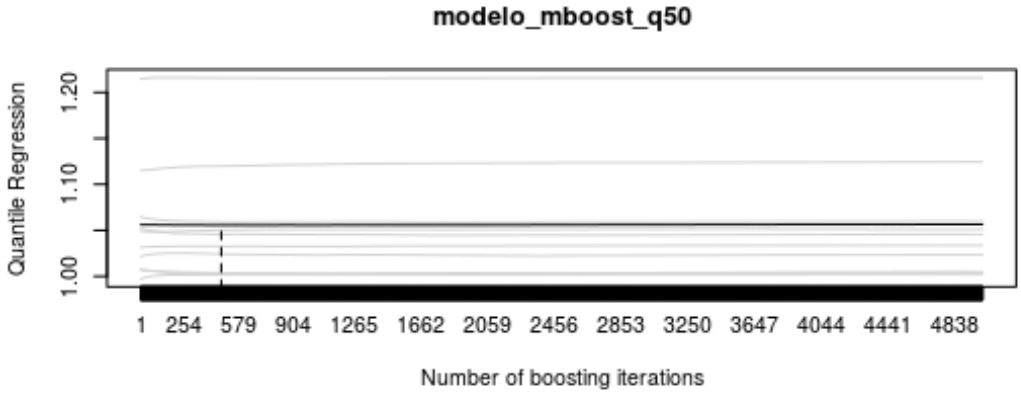
Se entrenan 3 modelos para los cuantiles 0.1, 0.5, 0.9, empleando *smooth P-spline* del predictor *x* como *base learners*.

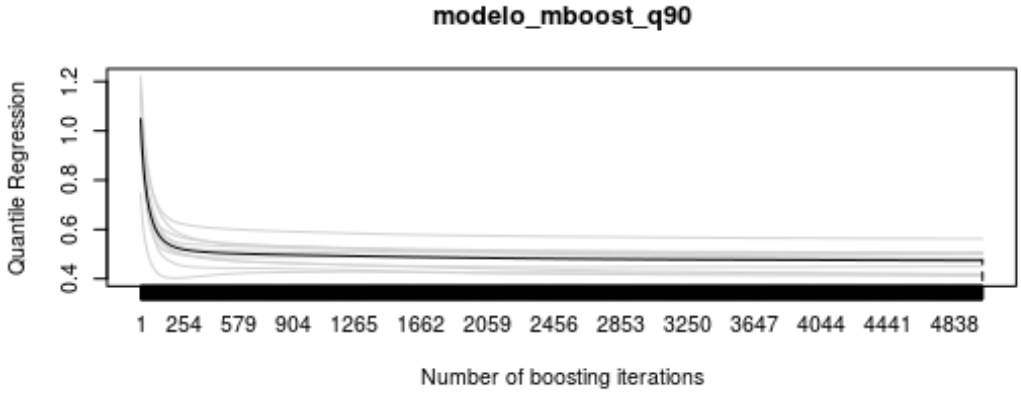
```
library(mboost)

modelo_mboost_q10 <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 5000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.1)
                     )

modelo_mboost_q50 <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 5000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.5)
                     )

modelo_mboost_q90 <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 5000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.9)
                     )
```

Para identificar el número óptimo de iteraciones y evitar *overfitting* se puede emplear métricas de *AIC*, *cross-validation*, *subsamplig* o *bootstrapping*. En el caso de regresión cuantílica, el *AIC* no está bien definido, por lo que no es un método aconsejable. Se procede a analizar, para cada modelo, la evolución del error en función del número de iteraciones empleando validación cruzada.

```r
cv_modelo_mboost_q10 <- cvrisk(
                          object = modelo_mboost_q10,
                          folds = cv(weights = model.weights(modelo_mboost_q10),
                                     type = "kfold",
                                     B = 10),
                          mc.cores = 4
                        )

cv_modelo_mboost_q50 <- cvrisk(
                          object = modelo_mboost_q50,
                          folds = cv(weights = model.weights(modelo_mboost_q50),
                                     type = "kfold",
                                     B = 10),
                          mc.cores = 4
                        )

cv_modelo_mboost_q90 <- cvrisk(
                          object = modelo_mboost_q90,
                          folds = cv(weights = model.weights(modelo_mboost_q90),
                                     type = "kfold",
                                     B = 10),
                          mc.cores = 4
                        )

par(mfrow=c(3,1))
plot(cv_modelo_mboost_q10, main = "modelo_mboost_q10")
plot(cv_modelo_mboost_q50, main = "modelo_mboost_q50")
plot(cv_modelo_mboost_q90, main = "modelo_mboost_q90")
```








```
par(mfrow=c(1,1))
```

```
# Valor óptimo de iteraciones (mstop)
mstop(cv_modelo_mboost_q10)
```

```
## [1] 4995
```

```
mstop(cv_modelo_mboost_q50)
```

```
## [1] 478
```

```
mstop(cv_modelo_mboost_q90)
```

```
## [1] 5000
```

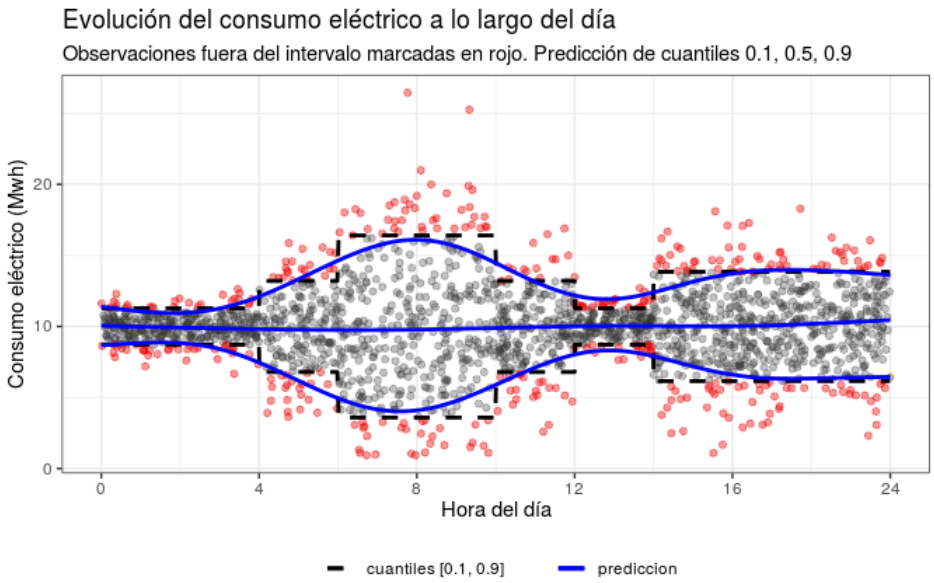
Para el modelo `cv_modelo_mboost_q50` el valor óptimo identificado por validación cruzada es de 478 iteraciones. Para los modelos `cv_modelo_mboost_q50` y `cv_modelo_mboost_q90` el error sigue disminuyendo alcanzados los 5000 árboles, por lo que, en la práctica, convendría incluir más iteraciones.

```
# Se limita cada modelo hasta el número de iteraciones optimo
modelo_mboost_q10 <- modelo_mboost_q10[mstop(cv_modelo_mboost_q10)]
modelo_mboost_q50 <- modelo_mboost_q50[mstop(cv_modelo_mboost_q50)]
modelo_mboost_q90 <- modelo_mboost_q90[mstop(cv_modelo_mboost_q90)]
```

Predicción

```
# Se predice todo el rango de X para representar los cuantiles
grid_predictor <- seq(0, 24, length.out = 2500)

predicciones_q10 <- predict(
                      modelo_mboost_q10,
                      newdata = data.frame(x = grid_predictor)
                    )
predicciones_q50 <- predict(
                      modelo_mboost_q50,
                      newdata = data.frame(x = grid_predictor)
                    )
predicciones_q90 <- predict(
                      modelo_mboost_q90,
                      newdata = data.frame(x = grid_predictor)
                    )

predicciones_cuantiles <- tibble(q10 = predicciones_q10,
                                 q50 = predicciones_q50,
                                 q90 = predicciones_q90)
predicciones_cuantiles <- bind_cols(data.frame(x = grid_predictor),
predicciones_cuantiles)
```

```r
p <- ggplot() +
      geom_point(
        data = datos %>% filter(dentro_intervalo == TRUE),
        aes(x = x, y = y),
        alpha = 0.3,
        color = "gray20") +
      geom_point(
        data = datos %>% filter(dentro_intervalo == FALSE),
        aes(x = x, y = y),
        alpha = 0.4,
        color = "red2") +
      geom_line(aes(x = x, y = cuantil_10, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(aes(x = x, y = cuantil_90, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q10, color = "prediccion"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q50, color = "prediccion"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q90, color = "prediccion"),
        size = 1) +
      scale_color_manual(
        name = "",
       breaks = c("prediccion"),
       values = c("prediccion" = "blue")) +
      scale_linetype_manual(
        name = "",
        breaks = c("cuantiles [0.1, 0.9]", "cuantiles [0.1, 0.9]"),
        values = c("cuantiles [0.1, 0.9]" = "dashed")) +
      labs(title = "Evolución del consumo eléctrico a lo largo del día",
           subtitle =  paste("Observaciones fuera del intervalo marcadas en rojo.",
                             "Predicción de cuantiles 0.1, 0.5, 0.9"),
           x = "Hora del día",
           y = "Consumo eléctrico (Mwh)") +
      theme_bw() +
      theme(legend.position = "bottom")

p <- p +
     scale_x_continuous(breaks = seq(0, 24, length.out = 6),
                        labels = c(0, 4, 8, 12, 16, 24))
p
```



Consideraciones prácticas

Ha diferencia de otros métodos como Quantile Regression Forest, Distributional Regression Forest: Random Forest probabilístico o rq, en este caso se tiene que ajustar un modelo por cada cuantil. Esto puede llevar a situaciones extrañas como por ejemplo que dos cuantiles se crucen.

Una ventaja de emplear *Splines* como *base learnes* es que el modelo tiene menos flexibilidad y por lo tanto es más robusto frente a *overfitting*. La desventaja es que el modelo no tiene en consideración interacciones entre predictores, a no ser que se le indique de forma explícita.

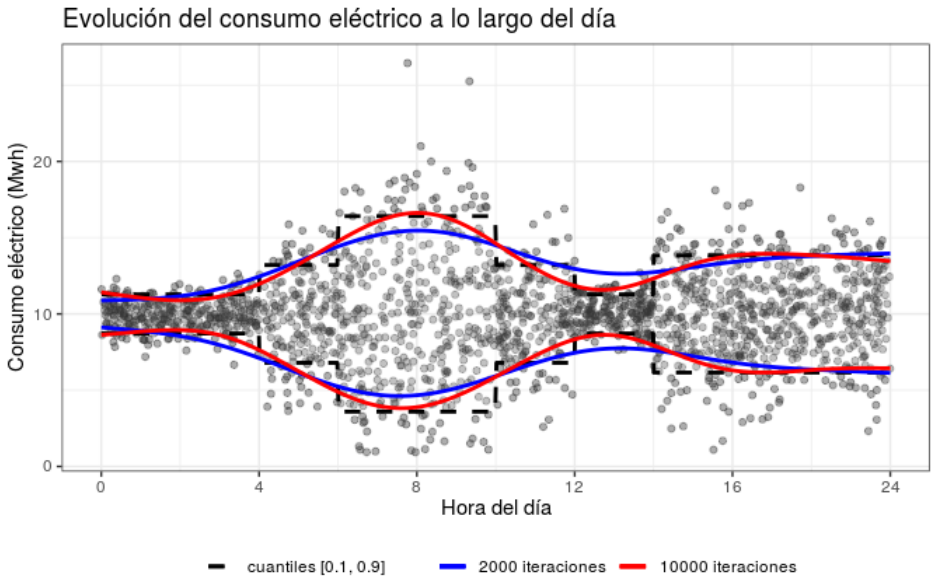
Véase cómo afecta pasar de 2000 a 10000 iteraciones de entrenamiento.

```
# Comparación de dos modelos: uno con 2000 y otro con 10000 iteraciones.
mboost_q10_2k <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 2000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.1)
                     )

mboost_q10_10k <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
```

```
                                  mstop = 10000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.1)
                     )

mboost_q90_2k <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 2000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.9)
                     )

mboost_q90_10k <- gamboost(
                      formula = y ~ bbs(x),
                      data    = datos,
                      control = boost_control(
                                  mstop = 10000,
                                  nu    = 0.1,
                                  stopintern = TRUE
                                ),
                      family  = QuantReg(tau = 0.9)
                     )

# Se predice todo el rango de X para representar los cuantiles
grid_predictor <- seq(0, 24, length.out = 2500)

pred_q10_2k <- predict(
                   mboost_q10_2k,
                   newdata = data.frame(x = grid_predictor)
                 )
pred_q10_10k <- predict(
                   mboost_q10_10k,
                   newdata = data.frame(x = grid_predictor)
                 )
pred_q90_2k <- predict(
                   mboost_q90_2k,
                   newdata = data.frame(x = grid_predictor)
                 )
pred_q90_10k <- predict(
                   mboost_q90_10k,
                   newdata = data.frame(x = grid_predictor)
                 )
```

```
predicciones_cuantiles <- tibble(
                            pred_q10_2k,
                            pred_q10_10k,
                            pred_q90_2k,
                            pred_q90_10k
                          )
predicciones_cuantiles <- bind_cols(
                            data.frame(x = grid_predictor),
                            predicciones_cuantiles
                          )

p <- ggplot() +
      geom_point(
        data = datos %>% filter(dentro_intervalo == TRUE),
        aes(x = x, y = y),
        alpha = 0.3,
        color = "gray20") +
      geom_point(
        data = datos %>% filter(dentro_intervalo == FALSE),
        aes(x = x, y = y),
        alpha = 0.4,
        color = "gray20") +
      geom_line(aes(x = x, y = cuantil_10, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(aes(x = x, y = cuantil_90, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q10_2k, color = "2000 iteraciones"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q90_2k, color = "2000 iteraciones"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q10_10k, color = "10000 iteraciones"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q90_10k, color = "10000 iteraciones"),
        size = 1) +
      scale_color_manual(
        name = "",
        breaks = c("2000 iteraciones", "10000 iteraciones"),
        values = c("2000 iteraciones" = "blue", "10000 iteraciones" = "red")) +
      scale_linetype_manual(
        name = "",
        breaks = c("cuantiles [0.1, 0.9]", "cuantiles [0.1, 0.9]"),
        values = c("cuantiles [0.1, 0.9]" = "dashed")) +
```

```
      labs(title = "Evolución del consumo eléctrico a lo largo del día",
           x = "Hora del día",
           y = "Consumo eléctrico (Mwh)") +
      theme_bw() +
      theme(legend.position = "bottom")

p <- p +
     scale_x_continuous(breaks = seq(0, 24, length.out = 6),
                        labels = c(0, 4, 8, 12, 16, 24))
p
```



Quantile Gradient Boosting

Introducción

La regresión de cuantiles también puede obtenerse empleando la aproximación más tradicional de *Gradient Boosting*, en la que se emplean árboles como *base learners*. La única diferencia es la distribución y función de coste objetivo. Con el paquete H2O se pueden ajustar este tipo de modelos de forma muy intuitiva indicando es sus argumentos `distribution = 'quantile'` y `quantile_alph`.

Ejemplo

Datos

Ver *Anexo*[1] para conocer más detalles de la simulación.

```
library(tidyverse)

# Simulación distribución no uniforme en el rango X
# ------------------------------------------------------------------------------
set.seed(12345)
n <- 2000
x <- runif(min = 0, max = 24, n = n)
y <- rnorm(
        n,
        mean = 10,
        sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
             1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
     )

# Cálculo del cuantil 0.1 y 0.9 para cada posición de x simulada.
cuantil_10 <- qnorm(
                p    = 0.1,
                mean = 10,
                sd   = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                       1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
              )
```

```
cuantil_90 <- qnorm(
                p    = 0.9,
                mean = 10,
                sd   = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                       1.5*(12 < x & x < 14.4) + 2*(x > 16.8)
              )
datos <- data.frame(y, x, cuantil_10, cuantil_90)

# No puede haber consumos negativos
datos <- datos %>%
         filter(y >= 0)

datos <- datos %>%
         mutate(dentro_intervalo = ifelse(
                                     y > cuantil_10 & y < cuantil_90,
                                     TRUE,
                                     FALSE
                                   )
        )
```

Modelo

Se entrenan 3 modelos para los cuantiles 0.1, 0.5, 0.9, empleando *smooth P-spline* del predictor *x* como *base learners*.

```
library(h2o)

# Creación de un cluster local con todos los cores disponibles.
h2o.init(ip = "localhost",
         # -1 indica que se empleen todos los cores disponibles.
         nthreads = -1,
         # Máxima memoria disponible para el cluster.
         max_mem_size = "4g")

# Se eliminan los datos del cluster por si ya había sido iniciado.
h2o.removeAll()
h2o.no_progress()

# Se tranfieren los datos al cluster de H2O
datos_h2o <- as.h2o(
              x = datos,
              destination_frame  = "datos_h2o"
             )
```

```
modelo_gbm_q10 <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.1,
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 2,
    ntrees      = 500,
    sample_rate = 0.9,
    # Parada temprana
    seed            = 123,
    nfolds          = 5,
    stopping_rounds = 4,
    stopping_metric = "MSE",
    stopping_tolerance  = 0.01,
    score_tree_interval = 100,
    model_id = "modelo_gbm_q10"
)

modelo_gbm_q50 <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.5,
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 2,
    ntrees      = 500,
    sample_rate = 0.9,
    # Parada temprana
    seed            = 123,
    nfolds          = 5,
    stopping_rounds = 4,
    stopping_metric = "MSE",
    stopping_tolerance  = 0.01,
    score_tree_interval = 100,
```

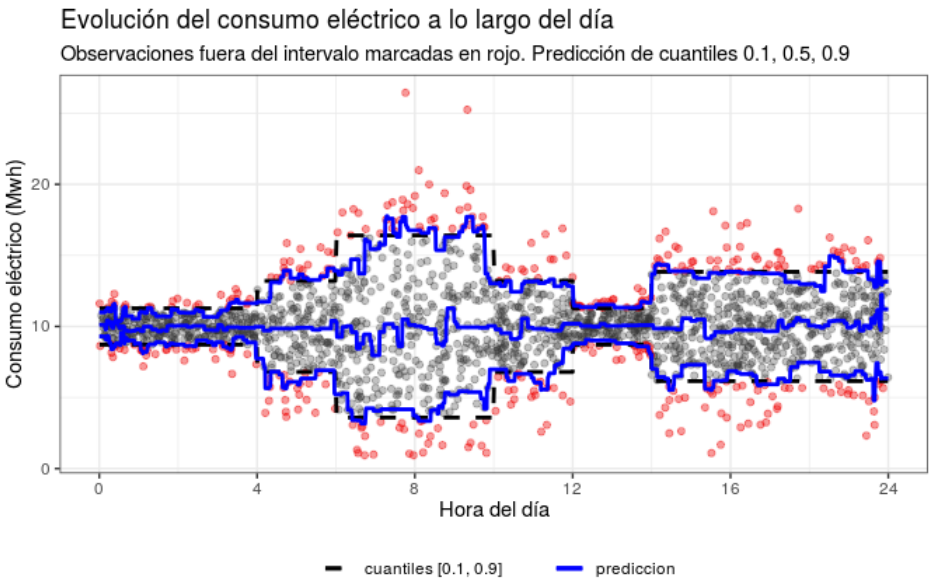
```
    model_id = "modelo_gbm_q50"
)

modelo_gbm_q90 <- h2o.gbm(
    # Variable respuesta y predictores
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.9,
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 2,
    ntrees      = 500,
    sample_rate = 0.9,
    # Parada temprana
    seed            = 123,
    nfolds          = 5,
    stopping_rounds = 4,
    stopping_metric = "MSE",
    stopping_tolerance  = 0.01,
    score_tree_interval = 100,
    model_id = "modelo_gbm_q90"
)
```

Predicción

```
# Se predice todo el rango de X para representar los cuantiles
grid_predictor <- seq(0, 24, length.out = 2500)

predicciones_q10 <- h2o.predict(
                      modelo_gbm_q10,
                      newdata = as.h2o(x = grid_predictor)
                    )
predicciones_q50 <- h2o.predict(
                      modelo_gbm_q50,
                      newdata = as.h2o(x = grid_predictor)
                    )
predicciones_q90 <- h2o.predict(
                      modelo_gbm_q90,
                      newdata = as.h2o(x = grid_predictor)
                    )
```

```
predicciones_cuantiles <- tibble(
                            q10 = as.vector(predicciones_q10$predict),
                            q50 = as.vector(predicciones_q50$predict),
                            q90 = as.vector(predicciones_q90$predict)
                          )
predicciones_cuantiles <- bind_cols(data.frame(x = grid_predictor),
predicciones_cuantiles)

p <- ggplot() +
      geom_point(
        data = datos %>% filter(dentro_intervalo == TRUE),
        aes(x = x, y = y),
        alpha = 0.3,
        color = "gray20") +
      geom_point(
        data = datos %>% filter(dentro_intervalo == FALSE),
        aes(x = x, y = y),
        alpha = 0.4,
        color = "red2") +
      geom_line(aes(x = x, y = cuantil_10, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(aes(x = x, y = cuantil_90, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q10, color = "prediccion"),
            size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q50, color = "prediccion"),
            size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = q90, color = "prediccion"),
            size = 1) +
      scale_color_manual(
        name = "",
        breaks = c("prediccion"),
        values = c("prediccion" = "blue")) +
      scale_linetype_manual(
        name = "",
        breaks = c("cuantiles [0.1, 0.9]", "cuantiles [0.1, 0.9]"),
        values = c("cuantiles [0.1, 0.9]" = "dashed")) +
      labs(title = "Evolución del consumo eléctrico a lo largo del día",
           subtitle =  paste("Observaciones fuera del intervalo marcadas en rojo.",
                             "Predicción de cuantiles 0.1, 0.5, 0.9"),
           x = "Hora del día",
           y = "Consumo eléctrico (Mwh)") +
      theme_bw() +
```

```
      theme(legend.position = "bottom")

p <- p +
     scale_x_continuous(breaks = seq(0, 24, length.out = 6),
                        labels = c(0, 4, 8, 12, 16, 24))
p
```



Consideraciones prácticas

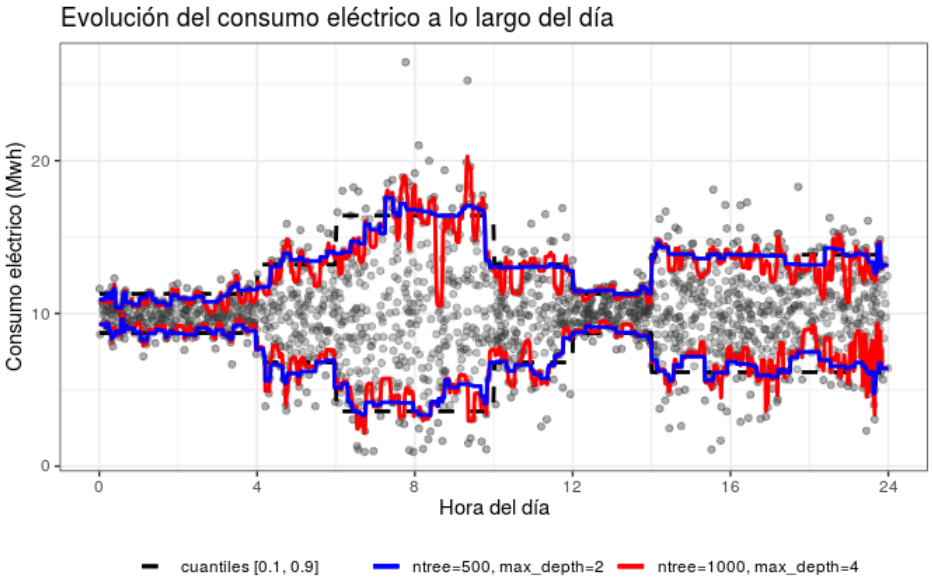
Ha diferencia de cuando se emplean *Splines* como *base learnes*, los árboles permíten mucha más flexibilidad, por lo que el modelo es altamente susceptible de sufrir *overfitting*. Los tres principales hiperparámetros a tener en cuenta son: el *learning rate* `learn_rate`, el número de árboles `ntrees` y su profundidad `max_depth`. Véase cómo afecta pasar de 500 a 1000 árboles, con una profundidad máxima de 2 a 4 respectivamente.

```
modelo_gbm_q10_05k <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.1,
```

```
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 2,
    ntrees      = 500,
    sample_rate = 0.9,
    model_id = "modelo_gbm_q10_05k"
)

modelo_gbm_q90_05k <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.9,
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 2,
    ntrees      = 500,
    sample_rate = 0.9,
    model_id = "modelo_gbm_q90_05k"
)

modelo_gbm_q10_1k <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.1,
    # Datos de entrenamiento
    training_frame = datos_h2o,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 4,
    ntrees      = 1000,
    sample_rate = 0.9,
    model_id = "modelo_gbm_q10_2k"
)
```

```
modelo_gbm_q90_1k <- h2o.gbm(
    # Variable respuesta y predictores.
    y = "y",
    x = "x",
    # Distribución para ajuste de cuantiles
    distribution   = 'quantile',
    quantile_alpha = 0.9,
    # Datos de entrenamiento.
    training_frame = datos_h2o,
    # Preprocesado
    ignore_const_cols = TRUE,
    # Hiperparámetros
    learn_rate  = 0.1,
    max_depth   = 4,
    ntrees      = 1000,
    sample_rate = 0.9,
    model_id = "modelo_gbm_q90_2k"
)

# Se predice todo el rango de X para representar los cuantiles
grid_predictor <- seq(0, 24, length.out = 2500)

predicciones_q10_05k <- h2o.predict(
                          modelo_gbm_q10_05k,
                          newdata = as.h2o(x = grid_predictor)
                        )
predicciones_q90_05k <- h2o.predict(
                          modelo_gbm_q90_05k,
                          newdata = as.h2o(x = grid_predictor)
                        )
predicciones_q10_1k <- h2o.predict(
                          modelo_gbm_q10_1k,
                          newdata = as.h2o(x = grid_predictor)
                       )

predicciones_q90_1k <- h2o.predict(
                         modelo_gbm_q90_1k,
                         newdata = as.h2o(x = grid_predictor)
                        )

predicciones_cuantiles <- tibble(
                            pred_q10_05k = as.vector(predicciones_q10_05k$predict),
                            pred_q90_05k = as.vector(predicciones_q90_05k$predict),
                            pred_q10_1k  = as.vector(predicciones_q10_1k$predict),
                            pred_q90_1k  = as.vector(predicciones_q90_1k$predict)
                          )
predicciones_cuantiles<- bind_cols(data.frame(x=grid_predictor),predicciones_cuantiles)

p <- ggplot() +
      geom_point(
```

```
        data = datos %>% filter(dentro_intervalo == TRUE),
        aes(x = x, y = y),
        alpha = 0.3,
        color = "gray20") +
      geom_point(
        data = datos %>% filter(dentro_intervalo == FALSE),
        aes(x = x, y = y),
        alpha = 0.4,
        color = "gray20") +
      geom_line(aes(x = x, y = cuantil_10, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(aes(x = x, y = cuantil_90, linetype = "cuantiles [0.1, 0.9]"),
                size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q10_1k, color = "ntree=1000, max_depth=4"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q90_1k, color = "ntree=1000, max_depth=4"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q10_05k, color = "ntree=500, max_depth=2"),
        size = 1) +
      geom_line(
        data = predicciones_cuantiles,
        aes(x = x, y = pred_q90_05k, color = "ntree=500, max_depth=2"),
        size = 1) +
      scale_color_manual(
        name = "",
        breaks = c("ntree=500, max_depth=2", "ntree=1000, max_depth=4"),
        values=c("ntree=500,max_depth=2"="blue","ntree=1000, max_depth=4"="red")) +
      scale_linetype_manual(
        name = "",
        breaks = c("cuantiles [0.1, 0.9]", "cuantiles [0.1, 0.9]"),
        values = c("cuantiles [0.1, 0.9]" = "dashed")) +
      labs(title = "Evolución del consumo eléctrico a lo largo del día",
           x = "Hora del día",
           y = "Consumo eléctrico (Mwh)") +
      theme_bw() +
      theme(legend.position = "bottom")

p <- p +
     scale_x_continuous(breaks = seq(0, 24, length.out = 6),
                        labels = c(0, 4, 8, 12, 16, 24))
p
```



Anexos

Anexo 1

Simulación ligeramente modificada del ejemplo publicado en *XGBoostLSS – An extension of XGBoost to probabilistic forecasting Alexander März*.

La ecuación empleada para generar los datos del ejemplo es:

$$y \sim \mathcal{N}(10, (1 + 1.5(4.8 < x < 7.2) + 4(7.2 < x < 12) + 1.5(12 < x < 14.4) + 2(x > 16.8))$$

Para el rango de valores $0 < x < 4.8$ los datos se distribuyen según una normal de media 10 y desviación típica 1. Para el rango de $4.8 < x < 7.2$ la desviación típica aumenta a 2.5. Para el rango $7.2 < x < 12$ pasa a 5, a continuación de $12 < x < 14.4$ desciende 2.5 y de $14.4 < x < 14.4$ a 1. Finalmente, para $x > 16.8$ la desviación aumenta a 3. El valor medio se mantiene constante (10).

```r
library(dplyr)

# Simulación distribución uniforme en el rango X
# ------------------------------------------------------------------------------
set.seed(123)
x <- rep(x = seq(0, 1, length.out = 96), each = 30)
y <- rnorm(
        length(x),
        mean = 10,
        sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
             1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
     )

# Cálculo del cuantil 0.1 y 0.9 para cada posición de x simulada.
cuantil_10 <- qnorm(
                p = 0.1,
                mean = 10,
                sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                     1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
              )
cuantil_90 <- qnorm(
                p = 0.9,
                mean = 10,
                sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                     1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
              )

datos    <- data.frame(y, x, cuantil_10, cuantil_90)
# No puede haber consumos negativos
datos <- datos %>%
         filter(y >=0)

# Simulación distribución no uniforme en el rango X
# ------------------------------------------------------------------------------
set.seed(12345)
n <- 2000
x <- runif(min = 0, max = 24, n = n)
y <- rnorm(
        n,
        mean = 10,
        sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
             1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
     )
# Cálculo del cuantil 0.1 y 0.9 para cada posición de x simulada.
cuantil_10 <- qnorm(
                p = 0.1,
                mean = 10,
                sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                     1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
```

```
              )

cuantil_90 <- qnorm(
                p = 0.9,
                mean = 10,
                sd = 1 + 1.5*(4.8 < x & x < 7.2) + 4*(7.2 < x & x < 12) +
                     1.5*(7.2 < x & x < 14.4) + 2*(x > 16.8)
              )
datos <- data.frame(y, x, cuantil_10, cuantil_90)

# No puede haber consumos negativos
datos <- datos %>%
         filter(y >=0)

datos <- datos %>%
         mutate(dentro_intervalo = ifelse(
                                     y > cuantil_10 & y < cuantil_90,
                                     TRUE,
                                     FALSE
                                   )
        )
```

Bibliografía

Benjamin Hofner, Andreas Mayr, Nikolay Robinzonov and Matthias Schmid (2014). Model-based Boosting in R: A Hands-on Tutorial Using the R Package mboost. Computational Statistics, 29, 3–35. http://dx.doi.org/10.1007/s00180-012-0382-5

Mayr, Andreas & Binder, Harald & Gefeller, Olaf & Schmid, Matthias. (2014). The Evolution of Boosting Algorithms From Machine Learning to Statistical Modelling. Methods of information in medicine. 53. 10.3414/ME13-01-0122. https://arxiv.org/abs/1403.1452v3

Nora Fenske, Thomas Kneib & Torsten Hothorn (2011) Identifying Risk Factors for Severe Childhood Malnutrition by Boosting Additive Quantile Regression, Journal of the American Statistical Association, 106:494, 494-510, DOI: 10.1198/jasa.2011.ap09272

Trabajo fin de máster - Tema: Regresión Cuantil Isabel Martínez Silva 30 de Junio de 2010

Métodos de suavizado eficientes con P-splines María Durbán Universidad Carlos III de Madrid

Cade, B.S. and Noon, B.R. (2003), A gentle introduction to quantile regression for ecologists. Frontiers in Ecology and the Environment, 1: 412-420. doi:10.1890/1540-9295(2003)001[0412:AGITQR]2.0.CO;2

März, Alexander. (2019). XGBoostLSS – An extension of XGBoost to probabilistic forecasting.