Gráficos ICE para interpretar modelos predictivos

Joaquín Amat Rodrigo

Diciembre, 2018

Tabla de contenidos

Versión PDF: Github

Introducción

Los gráficos *Individual Conditional Expectation (ICE)* muestran la variación de las predicciones de un modelo de *machine learning* en función del valor que toma alguno de sus predictores. Además de ser muy útiles para entender la relación entre la variable respuesta y los predictores aprendida por el modelo, permiten diferenciar cuándo, dicha relación, es aditiva o está afectada por interacciones con otros predictores. También permiten entender cómo se comporta un modelo cuando se extrapola a regiones para las que no se dispone de observaciones.

Los gráficos *ICE* pueden considerarse una extensión de los gráficos de dependencia parcial *Partial Dependence Plots* (*PDP*). La diferencia entre ambos reside en que, los *PDP*, muestran, con una única curva, cómo varía en promedio la predicción de la variable respuesta a medida que se modifica uno de los predictores.

Los *ICE*, muestran como varía la predicción para cada una de las observaciones (una curva distinta por cada observación).

A lo largo de este documento se muestran ejemplos de cómo se pueden obtener gráficos *ICE* y de qué información se puede extraer de ellos.

Paquete ICEbox

El paquete ICEbox contiene funciones que permiten calcular, explorar y representar gráficos *ICE* para cualquier tipo de modelo predictivo. A continuación, se muestra un ejemplo introductorio de cómo utilizarlo.

Curvas ICE

El set de datos `Boston` contiene información sobre la mediana del precio de las viviendas de la ciudad de Boston junto con variables relacionadas con las características de las casas y la zona donde se encuentran.

```
library(MASS)
datos <- Boston
head(datos)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12
##   lstat medv
## 1  4.98 24.0
## 2  9.14 21.6
## 3  4.03 34.7
## 4  2.94 33.4
## 5  5.33 36.2
## 6  5.21 28.7
```

Se entrena un modelo predictivo de tipo Random Forest con el objetivo de predecir el precio de la vivienda (*medv*) en función de todas las demás variables disponibles.

```
library(randomForest)
modelo_rf <- randomForest(formula = medv ~ ., data = datos, ntree = 500)
```

Una vez entrenado el modelo, con la función `ice()` se obtiene el gráfico *ICE* de cualquiera los predictores. Los principales argumentos de esta función son:

- `object`: modelo del cual se quieren obtener las curvas *ICE*.
- `X`: valor de los predictores con los que se ha entrenado el modelo.
- `y`: valor de la variable respuesta con la que se ha entrenado el modelo. Se emplea para identificar el rango del eje y.
- `predictor`: nombre o posición del predictor para el que se quiere obtener el gráfico *ICE*.

- `frac_to_build`: fracción de observaciones de entrenamiento que se incluyen en el gráfico *ICE*. Por defecto se emplean todas (`frac_to_build = 1`) pero, si el set de datos es muy grande, se recomienda reducirlo. La selección se hace de forma que se incluya aproximadamente todo el rango de valores observado en el entrenamiento.

- `indices_to_build`: índices de las observaciones que se incluyen en el gráfico *ICE*. Es una alternativa no aleatoria a `frac_to_build`. No pueden emplearse ambos argumentos a la vez.

- `num_grid_pts`: número de puntos dentro del rango del predictor empleados para construir la curva *ICE*. Por defecto, se utilizan todos los valores del predictor observados en los datos de entrenamiento del modelo.

- `predictfcn`: función opcional que acepta dos argumentos, un modelo (`object`) y un conjunto de datos (`newdata`), y devuelve un vector con las predicciones. Gracias a esta función se pueden obtener los gráficos *ICE* de cualquier modelo. Si este argumento no se especifica, se intenta encontrar automáticamente la función `predict()` correspondiente a la clase del modelo pasado a la función `ice()`.

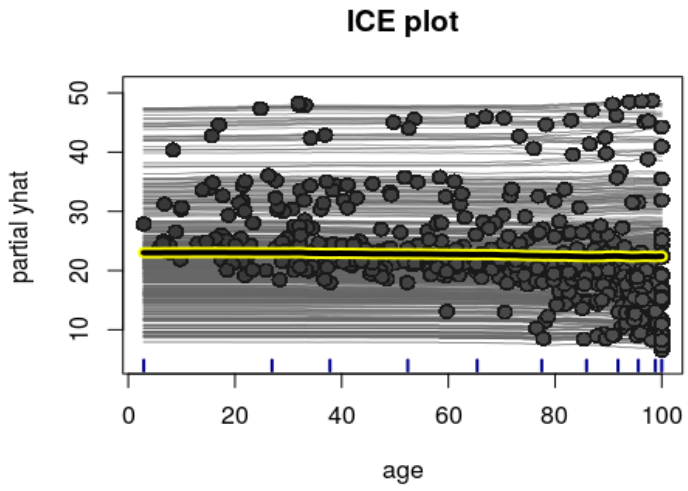
A continuación, se explora influencia que tiene la antigüedad de la vivienda (*age*) sobre el precio de la vivienda (*medv*).

```
library(ICEbox)

# Se separan los predictores de la variable respuesta.
datos_x      <- datos
datos_x$medv <- NULL
datos_y      <- datos$medv

ice_age <- ice(object = modelo_rf,
               X = datos_x,
               y = datos_y,
               predictor = "age",
               frac_to_build = 1,
               verbose = FALSE)
ice_age
```

```
## ice object generated on data with n = 506 for predictor "age"
## predictor considered continuous, logodds off
```

El objeto devuelto por `ice()` puede graficarse empleando la función `plot()`.

```
plot(ice_age,
     x_quantile = FALSE,
     plot_pdp = TRUE,
     plot_orig_pts_preds = TRUE,
     main = "ICE plot")
```



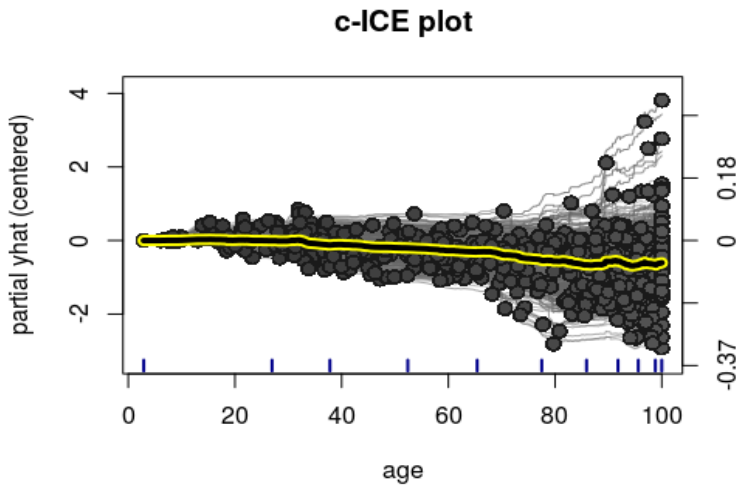
Cada curva del grafico anterior (curva *ICE*) muestra el valor predicho de la variable respuesta para cada observación con forme se va aumentando el valor de *age* y manteniendo constantes el resto de predictores en su valor observado. La curva resaltada en amarillo se corresponde con la curva *PDP*, que es la variación promedio de todas las observaciones. Además, el gráfico incluye puntos que representan el verdadero valor de *age* de cada observación.

La gran mayoría de las curvas son planas, lo que apunta a que, en la mayor parte de los casos, la antigüedad de la vivienda apenas influye en su precio. Sin embargo, puede apreciarse que, unas pocas observaciones, presentan una ligera tendencia de subida o bajada.

Curvas ICE centradas

Cuando los valores observados de la variable respuesta se acumulan en una región pequeña, el solapamiento de las curvas puede hacer difícil distinguir qué observaciones que se escapan de la tendencia general. Para evitar este problema, se puede recurrir a los gráficos *ICE* centrados (*c-ICE*). Los gráficos *c-ICE* se obtienen igual que los gráficos *ICE* con la única diferencia de que, a cada una de las curvas, se les resta un valor de referencia, normalmente el valor predicho para el mínimo observado del predictor. De esta forma, se consigue que todas las curvas tengan su origen en el 0.

```
plot(ice_age,
     x_quantile = FALSE,
     plot_pdp = TRUE,
     plot_orig_pts_preds = TRUE,
     centered = TRUE,
     main = "c-ICE plot")
```

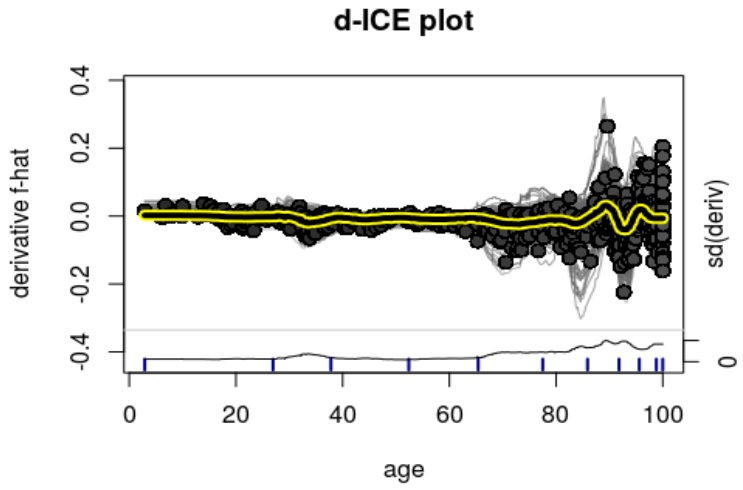


Con esta nueva representación puede observarse con más claridad que, aunque la mayoría de observaciones se mantienen constantes, algunas tienen un claro patrón divergente (para algunas el precio incrementa con la antigüedad y en otras disminuye). Tal y como se describe más adelante, esto suele ser un indicativo de que el predictor *age* interacciona con otros predictores. El eje vertical de la izquierda muestra el % de desviación respecto al rango de *y*.

Derivada de las curvas ICE

Si la relación existente entre la variable respuesta y el predictor estudiado es independiente del resto de predictores del modelo, entonces, las curvas del gráfico *ICE* comparten una misma forma y son paralelas las unas a las otras (la única diferencia es un desplazamiento en el eje vertical). Este comportamiento puede resultar complicado de validar visualmente cuando las curvas se superponen. Una forma de facilitar la identificación de interacciones entre predictores es representando las derivadas parciales de las curvas *ICE* (*d-ICE*). Si no existe ninguna interacción, todas las curvas son aproximadamente paralelas, sus derivadas aproximadamente iguales y, por lo tanto, el gráfico de derivadas muestra una única recta. Si existen interacciones, entonces, la representación de las derivadas parciales es heterogénea.

```
dice_age <- dice(ice_obj = ice_age)
plot(dice_age,
     plot_sd = TRUE,
     plot_orig_pts_deriv = TRUE,
     plot_dpdp = TRUE,
     main = "d-ICE plot")
```



El gráfico sugiere que, cuando la antigüedad de la vivienda es inferior a 60 años, las derivadas parciales son ≃ 0, por lo que no hay interacciones. Superados los 60 años, hay observaciones cuyas derivadas parciales se desvían sustancialmente de 0, indicando que, a partir de este valor, el predictor *age* interacciona con otros predictores.

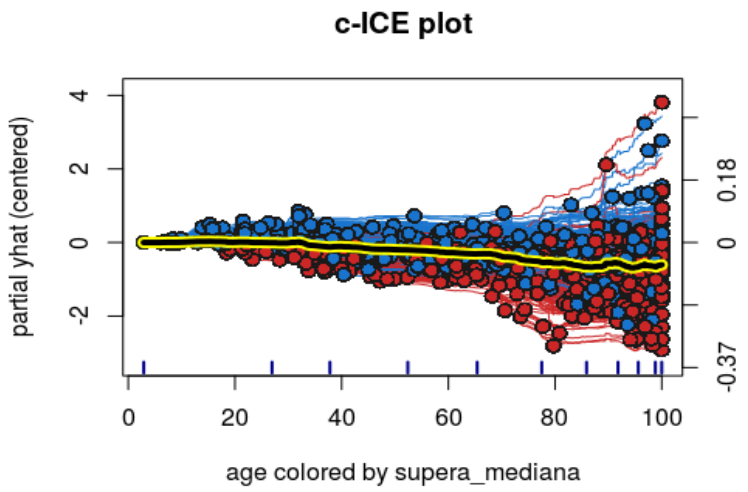
En la zona inferior del gráfico se muestra la desviación estándar de las derivadas parciales en cada punto, lo que facilita encontrar regiones de alta heterogeneidad (regiones de interacción).

Colorear curvas ICE

Como se ha visto en los apartados anteriores, algunas observaciones pueden alejarse de la tendencia general del modelo. Una forma de conseguir información extra que permita comprender las razones de estos patrones divergentes es colorear las curvas de cada observación en función de otro factor. Por ejemplo, en el modelo de predicción del valor medio, se crea una nueva variable binaria que indique si el número de habitaciones de la vivienda está por encima o por debajo de la mediana.

```
# Si la variable no es uno de los predictores originales con los que se entrenó
# el modelo, hay que añadirla en el objeto $Xice.
mediana_habitaciones <- median(x = ice_age$Xice$rm)
ice_age$Xice$supera_mediana <- ifelse(ice_age$Xice$rm > mediana_habitaciones,
                                      "si", "no")
plot(ice_age, x_quantile = FALSE, plot_pdp = TRUE, plot_orig_pts_preds = TRUE,
     centered = TRUE, color_by = "supera_mediana", main = "c-ICE plot")
```

```
## ICE Plot Color Legend
##  supera_mediana       color
##              no  firebrick3
##              si dodgerblue3
```



Gracias a los colores puede verse claramente que, para viviendas con un número de habitaciones por encima de la mediana (azul), la antigüedad de la vivienda está asociada positivamente con el precio. Para viviendas con un número de habitaciones inferior a la media, ocurre lo contrario.

Interacción entre predictores

En la introducción de documento, se menciona la diferencia entre los gráficos *PDP* y los *ICE*. La ventaja de los gráficos *ICE* queda patente cuando existe interacción entre predictores o cuando no todas las observaciones siguen una misma tendencia. Véase el siguiente ejemplo ilustrativo.

Se simula un set de datos siguiendo la siguiente ecuación:

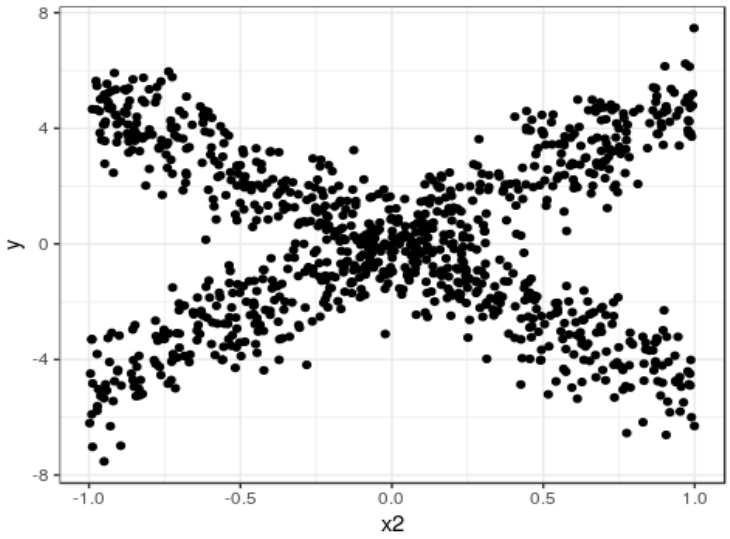
$$Y = 0.2X_1 - 5X_2 + 10X_2\mathbf{1}_{X_3 \geq 0} + \epsilon$$

o lo que es equivalente

$$Y = \begin{cases} 0.2X_1 - 5X_2 + 10X_2 + \epsilon & \text{si } X_3 \geq 0 \\ 0.2X_1 - 5X_2\epsilon & \text{si } X_3 < 0 \end{cases}$$

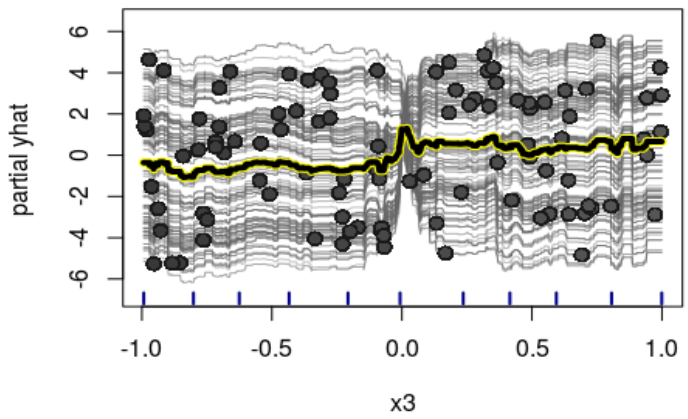
$$\epsilon \sim N(0,1) \quad X_1, X_2, X_3 \sim U(-1,1)$$

```
library(ggplot2)
set.seed(123)
x1 <- runif(n = 1000, min = -1, max = 1)
x2 <- runif(n = 1000, min = -1, max = 1)
x3 <- runif(n = 1000, min = -1, max = 1)
e  <- rnorm(n = 1000, mean = 0, sd = 1)
y <- 0.2*x1 - 5*x2 + 10*x2*I(x3 >= 0) + e
datos <- data.frame(x1, x2, x3, y)

ggplot(data = datos, aes(x = x2,y = y)) +
  geom_point() +
  theme_bw()
```



Se entrena un modelo GBM para predecir *y* en función de las 3 variables disponibles.

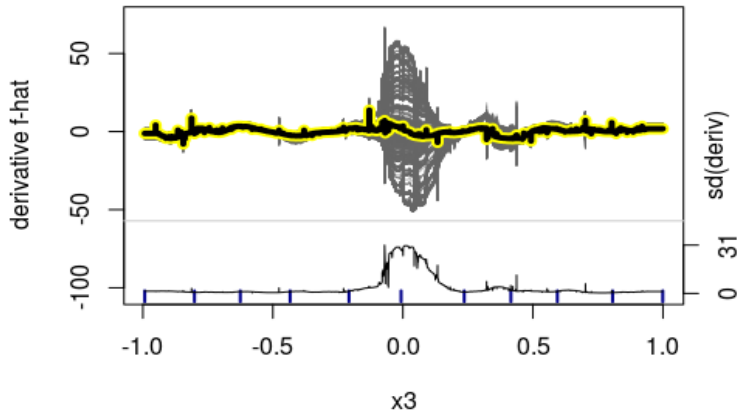
```
library(gbm)
set.seed(123)
modelo_gbm <- gbm(formula = y ~ .,
                  data = datos,
                  n.tree = 500,
                  interaction.depth = 3,
                  shrinkage = 0.1,
                  distribution = "gaussian",
                  cv.folds = 5,
                  verbose = FALSE)

# Se separan los predictores de la variable respuesta.
datos_x      <- datos
datos_x$medv <- NULL
datos_y      <- datos$medv

# Aunque existe una función predict.gbm(), a modo ilustrativo, se indica una
# función propia en el argumento predictfcn.
ice_gbm_x3 <- ice(object = modelo_gbm,
              X = datos_x,
              y = datos_y,
              predictor = "x3",
                    predictfcn = function(object, newdata){
                                   predict.gbm(object = object,
                                               newdata = newdata,
                                               n.trees = 435)
                                 },
                    frac_to_build = 1,
                    verbose = FALSE)

# Se grafican únicamente el 1% de las curvas.
plot(ice_gbm_x3, x_quantile = FALSE, plot_pdp = TRUE, frac_to_plot = 0.1)
```



Viendo únicamente la curva *PDP*, podría asumirse que, la variable respuesta Y, apenas varía en función del valor que tome el predictor X_3. Esta interpretación puede llevar a conclusiones erróneas ya que el promedio está ocultando el verdadero comportaminto individual de las observaciones.

Si se representa el valor de las derivadas de las curvas *ICE*, puede verse claramente que, la relación entre la variable respuesta Y y el predictor X_3 está de alguna forma modulada por interacciones con los predictores X_2 y X_3.

```
# Gráfico de las derivadas.
dice_gbm_x3 <- dice(ice_obj = ice_gbm_x3)
plot(dice_gbm_x3, plot_orig_pts_deriv = FALSE)
```



Además, este tipo de representación permite identificar en qué rango del predictor ocurre la interacción (región de interacción *ROI*). En este ejemplo, tal como cabe esperar dada la ecuación empleada para generar los datos, la interacción ocurre en torno al valor $X_3 = 0$.

Comportamiento en regiones extrapoladas

Una característica frecuente en muchos de los modelos de *machine learning* actuales es el alto número de predictores que incorporan. Cuanto mayor es la cantidad de predictores (dimensiones del espacio muestral), más dispersas se encuentran las observaciones. Esto se traduce en que, muchas predicciones, se hacen en regiones del espacio para las que no se dispone de observaciones, en otras palabras, se extrapola.

En los modelos predictivos la extrapolación supone un riesgo, ya que implica asumir que, la relación entre las variables aprendida por el modelo a partir de las regiones observadas, también se cumple en las regiones no observadas. Los gráficos *ICE* permiten ganar cierta intuición sobre cómo se está comportando el modelo en las regiones extrapoladas. Véase a continuación un ejemplo.

Se simula un set de datos en dos dimensiones (X_1, X_2), cada una en el rango $[-1,1]$, pero donde la región ($X_1 > 0, X_2 > 0$) no contiene ninguna observación.

$$Y = 10X_1^2 + \mathbf{1}_{X_2 \geq 0} + \epsilon$$

o lo que es equivalente

$$Y = \begin{cases} 10X_1^2 + X_2 & \text{si } X_2 \geq 0 \\ 10X_1^2 & \text{si } X_3 < 0 \end{cases}$$

donde

$$\epsilon \sim N(0,0.1)$$

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim \begin{Bmatrix} U(-1,0) & U(-1,0) & \text{probabilidad } \frac{1}{3} \\ U(0,1) & U(-1,0) & \text{probabilidad } \frac{1}{3} \\ U(-1,0) & U(0,1) & \text{probabilidad } \frac{1}{3} \\ U(0,1) & U(0,1) & \text{probabilidad } 0 \end{Bmatrix}$$

```
simular_observaciones <- function(n){
  datos <- matrix(data = NA, nrow = n, ncol = 2)
  for (i in 1:n) {
    caso <- sample(x = 1:3, size = 1)
    if(caso == 1){
      x1 <- runif(1, min = -1, max = 0)
      x2 <- runif(1, min = -1, max = 0)
    }else if(caso == 2){
      x1 <- runif(1, min = 0, max = 1)
      x2 <- runif(1, min = -1, max = 0)
    }else{
      x1 <- runif(1, min = -1, max = 0)
      x2 <- runif(1, min = 0 , max = 1)
    }
    datos[i,] <- cbind(x1, x2)
  }
  datos <- as.data.frame(datos)
  colnames(datos) <- c("x1", "x2")
  return(datos)
}

datos       <- simular_observaciones(n = 500)
e           <- rnorm(n = 500, mean = 0, sd = 0.1)
datos$y     <- 10*datos$x1^2 + I(datos$x2 >= 0) + e
```

Se asignan las observaciones a dos grupos dependiendo de si $X_2 \geq 0$.

```
datos$grupo <- ifelse(datos$x2 >= 0, "grupo1", "grupo2")
head(datos)
```

```
##           x1         x2            y  grupo
## 1 -0.1271645 -0.8806371 0.055302.... grupo2
## 2 -0.8239177 -0.5572727 6.832984.... grupo2
## 3 -0.9257164  0.4709457 9.613035.... grupo1
## 4  0.3053958 -0.3355885 0.853887.... grupo2
## 5 -0.7572411  0.2290509 6.657954.... grupo1
## 6 -0.4201603  0.8457528 2.773269.... grupo1
```

Se ajusta un modelo *random forest* a los datos.

```
modelo_rf <- randomForest(formula = y ~ x1 + x2,
                          data = datos,
                          ntree = 500)
```

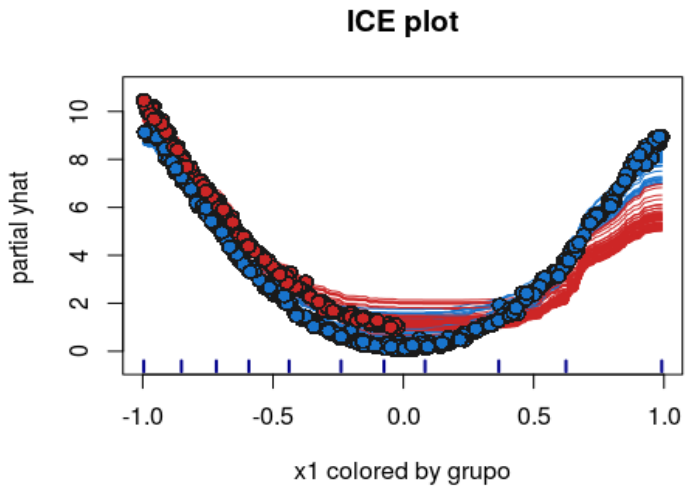
Se calculan las curvas *ICE*.

```
datos_x      <- datos
datos_x$y    <- NULL
datos_y      <- datos$y

# Se crea el objeto ice.
ice_x1 <- ice(object = modelo_rf,
              X = datos_x,
              y = datos_y,
              predictor = "x1",
              frac_to_build = 1,
              verbose = FALSE)
```

Nota: La función `plot.ice()` del paquete `ICEbox` parece generar un error cuando se especifica que no se muestre la curva PDP (`plot_pdp = FALSE`). En el apartado Anexos se propone una ligera modificación de la función original para que no genere el error.

```
# Se representan las curvas ICE coloreadas por grupo.
plot(ice_x1,
     x_quantile = FALSE,
     plot_pdp = FALSE,
     plot_orig_pts_preds = TRUE,
     color_by = "grupo",
     main = "ICE plot"
     )
```

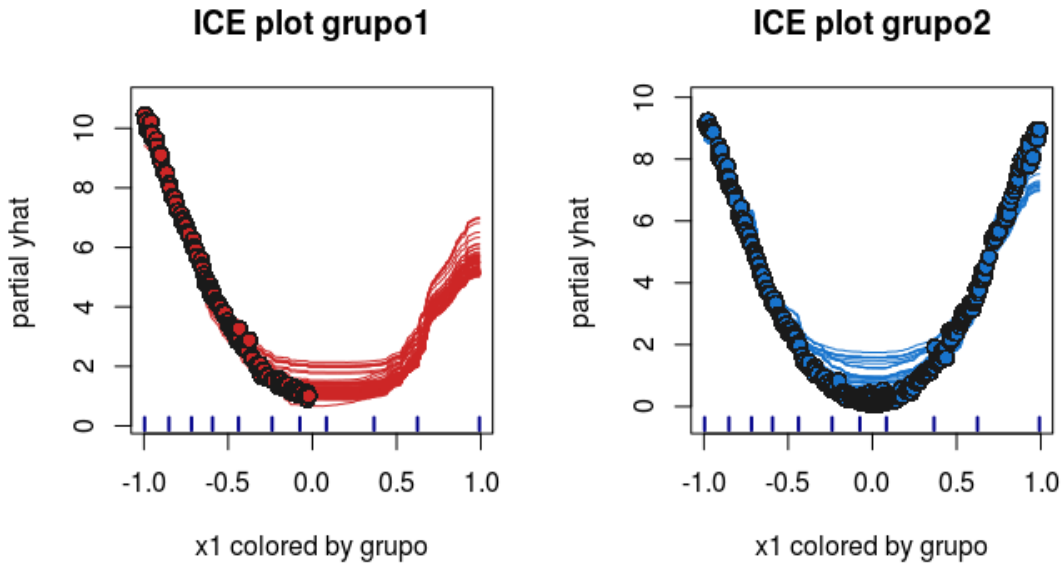
```
## ICE Plot Color Legend
##   grupo       color
##  grupo1  firebrick3
##  grupo2 dodgerblue3
```



```
par(mfrow=c(1,2))
# Se representan las curvas ICE solo del grupo 1.
plot(ice_x1,
     x_quantile = FALSE,
     plot_pdp = FALSE,
     plot_orig_pts_preds = TRUE,
     plot_points_indices = which(ice_x1$Xice$grupo == "grupo1"),
     color_by = "grupo",
     main = "ICE plot grupo1"
     )
```

```
## ICE Plot Color Legend
##   grupo       color
##  grupo1  firebrick3
##  grupo2 dodgerblue3
```

```
# Se representan las curvas ICE solo del grupo 2.
plot(ice_x1,
     x_quantile = FALSE,
     plot_pdp = FALSE,
     plot_orig_pts_preds = TRUE,
     plot_points_indices = which(ice_x1$Xice$grupo == "grupo2"),
     color_by = "grupo",
     main = "ICE plot grupo2"
     )
```



Puede observarse que, en el caso del grupo 1, el modelo mantiene la tendencia aprendida con las observaciones del grupo 2, aunque realmente no existe ninguna observación del grupo 1 en esa región. Que esta extrapolación sea aceptable o no, depende del caso de uso en cuestión.

Versión ggplot2

Toda la información presente necesaria para crear el gráfico está contenida en los objetos `ice` y `dice`, por lo que puede reproducirse la misma representación con `ggplot2`.

```
library(MASS)
library(randomForest)
library(ICEbox)
datos <- Boston
head(datos)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12
##   lstat medv
## 1  4.98 24.0
## 2  9.14 21.6
## 3  4.03 34.7
## 4  2.94 33.4
## 5  5.33 36.2
## 6  5.21 28.7
```

```
modelo_rf <- randomForest(formula = medv ~ .,
                          data = datos,
                          ntree = 500)
datos_x      <- datos
datos_x$medv <- NULL
datos_y      <- datos$medv

ice_age <- ice(object = modelo_rf,
               X = datos_x,
               y = datos_y,
               predictor = "age",
               frac_to_build = 1,
               verbose = FALSE)

dice_age <- dice(ice_obj = ice_age)
```

Gráfico ICE

```
library(tidyverse)

plot_ice <- function(objeto, pdp = TRUE, color_by = NULL){
  # Esta función devuelve el gráfico de las curvas ICE
  # Argumentos
  #   objeto:   un objeto devuelto por la función ICEbox::ice
  #   pdp:      si se muestra o no la curva promedio pdp.
  #   color_by: predictor por el cual que quiere colorear las curvas.
  # Output
  #   Gráfico ggplot

  predictor <- objeto$predictor

  datos_ice <- objeto$ice_curves %>%
               as.data.frame() %>%
               mutate(observacion_id = rownames(objeto$Xice)) %>%
               gather(key = !!predictor, value = "y", -observacion_id) %>%
               mutate(!!sym(predictor) := as.numeric(!!sym(predictor)))

  if(!is.null(color_by)){
    datos_color <- objeto$Xice %>%
                   rownames_to_column(var = "observacion_id") %>%
                   select(observacion_id, !!sym(color_by))

    datos_ice <- datos_ice %>%
                 left_join(datos_color, by = "observacion_id")
  }

  datos_observaciones <- objeto$xj %>%
                         as.data.frame() %>%
                         rename(!!predictor := !!sym(".")) %>%
                         mutate(y = objeto$actual_prediction,
                                observacion_id = rownames(objeto$Xice))

  datos_pdp <- objeto$pdp %>%
               as.data.frame() %>%
               rownames_to_column(var = predictor) %>%
               mutate(!!sym(predictor) := as.numeric(!!sym(predictor))) %>%
               rename(y := !!sym("."))

  if(pdp & is.null(color_by)){

    plot <- ggplot(data = datos_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group = observacion_id), color = "gray30", alpha=0.5) +
            geom_point(data=datos_observaciones, aes(x= !!sym(predictor), y=y),
```

```
                      colour = "black", pch = 21, fill = "gray20") +
            geom_line(data = datos_pdp, aes(x = !!sym(predictor), y = y),
                      colour = "red", size = 1.2)  +
            labs(title = paste("Curvas ICE:", predictor)) +
            theme_bw() +
            theme(legend.position = "bottom")

  }else if(pdp & !is.null(color_by)){
    plot <- ggplot(data = datos_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                      alpha = 0.5) +
            geom_point(data=datos_observaciones, aes(x = !!sym(predictor), y=y),
                       colour = "black", pch = 21, fill = "gray20") +
            geom_line(data = datos_pdp, aes(x = !!sym(predictor), y = y),
                      colour = "red", size = 1.2) +
            labs(title = paste("Curvas ICE:", predictor)) +
            theme_bw() +
            theme(legend.position = "bottom")

  }else if(!pdp & is.null(color_by)){
    plot <- ggplot(data = datos_ice, aes(x = !!sym(predictor), y = y)) +
              geom_path(aes(group=observacion_id), color="gray30", alpha=0.5) +
              geom_point(data=datos_observaciones, aes(x= !!sym(predictor), y=y),
                         colour = "black", pch = 21, fill = "gray20") +
            labs(title = paste("Curvas ICE:", predictor)) +
            theme_bw() +
            theme(legend.position = "bottom")

  }else{
    plot <- ggplot(data = datos_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                      alpha = 0.5) +
            geom_point(data=datos_observaciones, aes(x= !!sym(predictor), y=y),
                       colour = "black", pch = 21, fill = "gray20") +
            labs(title = paste("Curvas ICE:", predictor)) +
            theme_bw() +
            theme(legend.position = "bottom")

  }
  plot
  return(plot)
}

plot_ice(objeto = ice_age, pdp = TRUE)
```

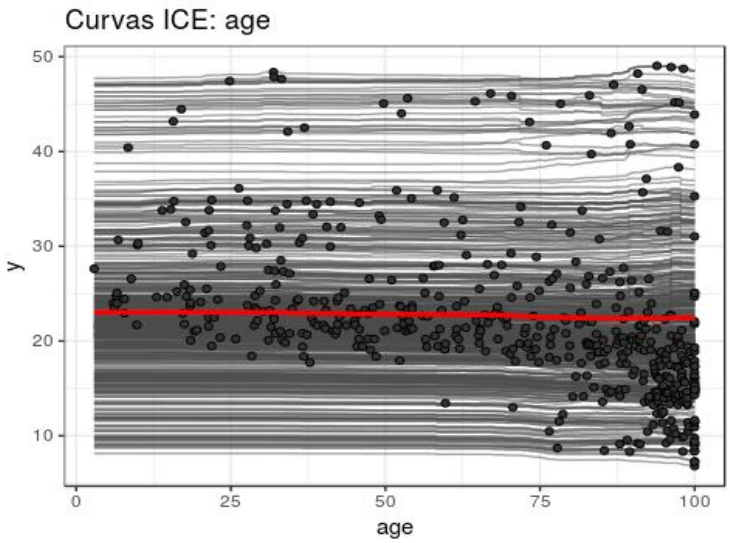
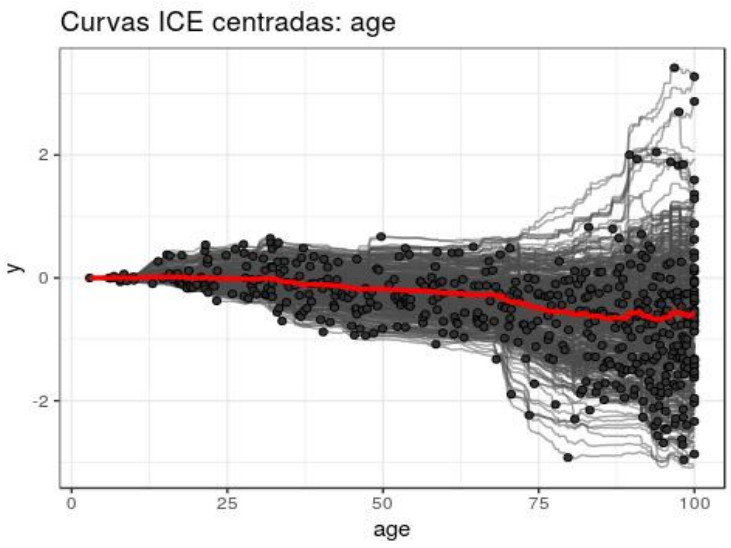


Gráfico c-ICE

```
plot_c_ice <- function(objeto, pdp = TRUE, color_by = NULL){
  # Esta función devuelve el gráfico de las curvas ICE centrados
  # Argumentos
  #   objeto:   un objeto devuelto por la función ICEbox::ice
  #   pdp:      si se muestra o no la curva promedio pdp
  #   color_by: predictor por el cual que quiere colorear las curvas.
  # Output
  #   Gráfico ggplot



  predictor <- objeto$predictor

  datos_ice <- objeto$ice_curves %>%
               as.data.frame() %>%
               mutate(observacion_id = rownames(objeto$Xice)) %>%
               gather(key = !!predictor, value = "y", -observacion_id) %>%
               mutate(!!sym(predictor) := as.numeric(!!sym(predictor)))

  minimo_por_curva <- datos_ice %>%
                      group_by(observacion_id) %>%
                      summarize(minimo = y[which.min(!! sym(predictor))])

  datos_c_ice <- datos_ice %>%
                 left_join(minimo_por_curva, by = "observacion_id") %>%
                  mutate(y = y - minimo)
```

```
if(!is.null(color_by)){
  datos_color <- objeto$Xice %>%
                 rownames_to_column(var = "observacion_id") %>%
                 select(observacion_id, !!sym(color_by))

  datos_c_ice <- datos_c_ice %>%
                 left_join(datos_color, by = "observacion_id")
}

datos_observaciones <- objeto$xj %>%
                       as.data.frame() %>%
                       rename(!!predictor := !!sym(".")) %>%
                       mutate(y = objeto$actual_prediction,
                              observacion_id = rownames(objeto$Xice))
datos_c_observaciones <- datos_observaciones %>%
                         left_join(minimo_por_curva, by = "observacion_id") %>%
                         mutate(y = y - minimo)

datos_c_pdp <- objeto$pdp %>%
               as.data.frame() %>%
               rownames_to_column(var = predictor) %>%
               mutate(!!sym(predictor) := as.numeric(!!sym(predictor))) %>%
               rename(y := !!sym(".")) %>%
               mutate(y = y - y[which.min(!! sym(predictor))])


if(pdp & is.null(color_by)){

  plot <- ggplot(data = datos_c_ice, aes(x = !!sym(predictor), y = y)) +
          geom_path(aes(group = observacion_id), color = "gray30", alpha = 0.5) +
          geom_point(data=datos_c_observaciones, aes(x= !!sym(predictor), y=y),
                     colour = "black", pch = 21, fill = "gray20") +
          geom_line(data = datos_c_pdp, aes(x = !!sym(predictor), y = y),
                    colour = "red", size = 1.2) +
          labs(title = paste("Curvas ICE centradas:", predictor)) +
          theme_bw()  +
          theme(legend.position = "bottom")

}else if(pdp & !is.null(color_by)){
  plot <- ggplot(data = datos_c_ice, aes(x = !!sym(predictor), y = y)) +
          geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                    alpha = 0.5) +
          geom_point(data=datos_c_observaciones, aes(x= !!sym(predictor), y=y),
                     colour = "black", pch = 21, fill = "gray20") +
          geom_line(data = datos_c_pdp, aes(x = !!sym(predictor), y = y),
                    colour = "red", size = 1.2) +
          labs(title = paste("Curvas ICE centradas:", predictor)) +
          theme_bw() +
          theme(legend.position = "bottom")
```

```
  }else if(!pdp & is.null(color_by)){
    plot <- ggplot(data = datos_c_ice, aes(x = !!sym(predictor), y = y)) +
              geom_path(aes(group = observacion_id), color = "gray30", alpha=0.5) +
              geom_point(data=datos_c_observaciones, aes(x= !!sym(predictor), y=y),
                         colour = "black", pch = 21, fill = "gray20") +
              labs(title = paste("Curvas ICE centradas:", predictor)) +
              theme_bw() +
              theme(legend.position = "bottom")
  }else{

    plot <- ggplot(data = datos_c_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                      alpha = 0.5) +
            geom_point(data=datos_c_observaciones, aes(x = !!sym(predictor), y=y),
                       colour = "black", pch = 21, fill = "gray20") +
            labs(title = paste("Curvas ICE centradas:", predictor)) +
            theme_bw() +
            theme(legend.position = "bottom")
  }
  plot
  return(plot)
}

plot_c_ice(objeto = ice_age, pdp = TRUE)
```



Colorear curvas

Mismo gráfico, pero coloreando por una variable.

```
# Si la variable no es uno de los predictores originales con los que se entrenó
# el modelo, hay que añadirla en el objeto $Xice.
mediana_habitaciones <- median(x = ice_age$Xice$rm)
ice_age$Xice$supera_mediana <- ifelse(ice_age$Xice$rm > mediana_habitaciones,
                                      "si", "no")
plot_c_ice(objeto = ice_age, pdp = TRUE, color_by = "supera_mediana")
```

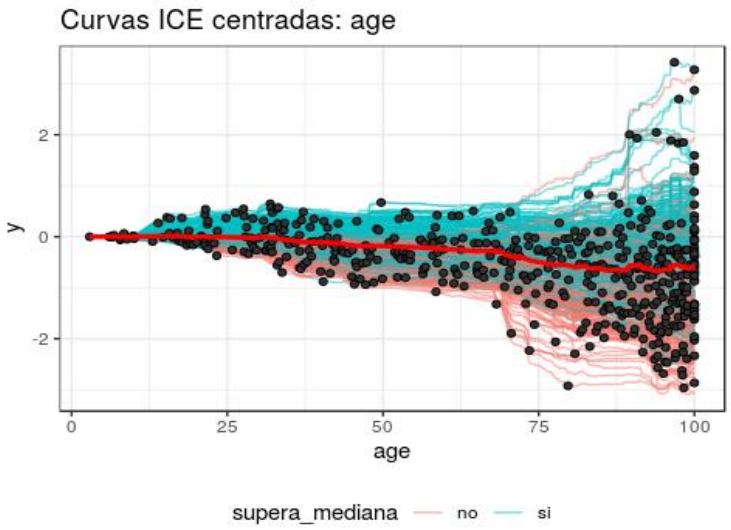
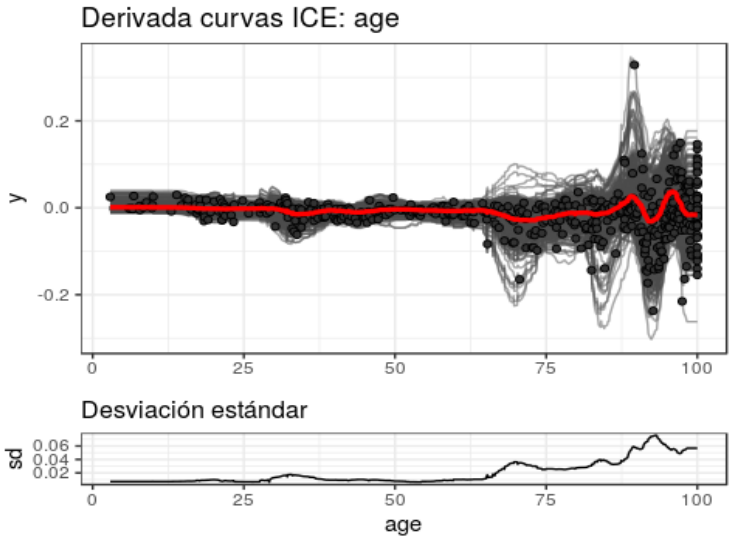


Gráfico d-ICE

```
library(ggpubr)

plot_d_ice <- function(objeto, pdp = TRUE, color_by = NULL){
  # Esta función devuelve el gráfico de las curvas d-ICE
  # Argumentos
  #   objeto:   un objeto devuelto por la función ICEbox::dice
  #   pdp:      si se muestra o no la curva promedio pdp
  #   color_by: predictor por el cual que quiere colorear las curvas.
  # Output
  #   Gráfico ggplot

  predictor <- objeto$predictor

  datos_d_ice <- objeto$d_ice_curves %>%
                 as.data.frame() %>%
                 setNames(as.character(objeto$gridpts)) %>%
                 mutate(observacion_id = rownames(objeto$Xice)) %>%
                 gather(key = !!predictor, value = "y", -observacion_id) %>%
                 mutate(!!sym(predictor) := as.numeric(!!sym(predictor)))

  if(!is.null(color_by)){
    datos_color <- objeto$Xice %>%
                   rownames_to_column(var = "observacion_id") %>%
                   select(observacion_id, !!sym(color_by))

    datos_d_ice <- datos_d_ice %>%
                   left_join(datos_color, by = "observacion_id")
  }

  datos_d_observaciones <- objeto$actual_deriv %>%
                           as.data.frame() %>%
                           rename(y = !!sym(".")) %>%
                           mutate(!!sym(predictor) := objeto$xj,
                                  observacion_id = rownames(objeto$Xice))

  datos_d_pdp <- data.frame(y = objeto$dpdp) %>%
                 mutate(!!sym(predictor) := objeto$gridpts)

  if(pdp & is.null(color_by)){

    plot <- ggplot(data = datos_d_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group = observacion_id), color = "gray30", alpha=0.5) +
            geom_point(data=datos_d_observaciones, aes(x= !!sym(predictor), y=y),
                       colour = "black", pch = 21, fill = "gray20") +
            geom_line(data = datos_d_pdp, aes(x = !!sym(predictor), y = y),
```

```
                    colour = "red", size = 1.2)  +
          labs(title = paste("Derivada curvas ICE:", predictor)) +
          theme_bw() +
          theme(legend.position = "bottom",
                axis.title.x = element_blank())

}else if(pdp & !is.null(color_by)){
  plot <- ggplot(data = datos_d_ice, aes(x = !!sym(predictor), y = y)) +
          geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                    alpha = 0.5) +
          geom_point(data=datos_d_observaciones, aes(x = !!sym(predictor), y=y),
                     colour = "black", pch = 21, fill = "gray20") +
          geom_line(data = datos_d_pdp, aes(x = !!sym(predictor), y = y),
                    colour = "red", size = 1.2) +
          labs(title = paste("Derivada curvas ICE:", predictor)) +
          theme_bw() +
          theme(legend.position = "bottom",
                axis.title.x = element_blank())

}else if(!pdp & is.null(color_by)){
  plot <- ggplot(data = datos_d_ice, aes(x = !!sym(predictor), y = y)) +
            geom_path(aes(group=observacion_id), color="gray30", alpha=0.5) +
            geom_point(data=datos_d_observaciones, aes(x= !!sym(predictor), y=y),
                       colour = "black", pch = 21, fill = "gray20") +
          labs(title = paste("Derivada curvas ICE:", predictor)) +
          theme_bw() +
          theme(legend.position = "bottom",
                axis.title.x = element_blank())

}else{
  plot <- ggplot(data = datos_d_ice, aes(x = !!sym(predictor), y = y)) +
          geom_path(aes(group = observacion_id, color = !!sym(color_by)),
                    alpha = 0.5) +
          geom_point(data=datos_d_observaciones, aes(x= !!sym(predictor), y=y),
                     colour = "black", pch = 21, fill = "gray20") +
          labs(title = paste("Derivada curvas ICE:", predictor)) +
          theme_bw() +
          theme(legend.position = "bottom",
                axis.title.x = element_blank())

}

datos_sd_derivada <- data.frame(sd = objeto$sd_deriv) %>%
                     mutate(!!sym(predictor) := objeto$gridpts)

plot_sd <- ggplot(data = datos_sd_derivada, aes(x = !!sym(predictor), y = sd)) +
           geom_line() +
           labs(title = "Desviación estándar") +
           theme_bw() +
```

```
            theme(legend.position = "bottom",
                  plot.title = element_text(size = 12))

  plot_combi <- ggpubr::ggarrange(plotlist = list(plot, plot_sd),
                                  nrow = 2,
                                  align = "v",
                                  heights = c(2.5,1))
  plot_combi

  return(plot_combi)
}

plot_d_ice(objeto = dice_age, pdp = TRUE)
```



Grid múltiples predictores

Las siguiente funciones permiten calcular y graficar las curvas *ICE*, *c-ICE* y *d-ICE* de uno, varios o todos los predictores de un modelo. Por defecto (`predictores = NULL`), obtienen los gráficos para todos lo predictores numéricos del modelo.

ICE

Función

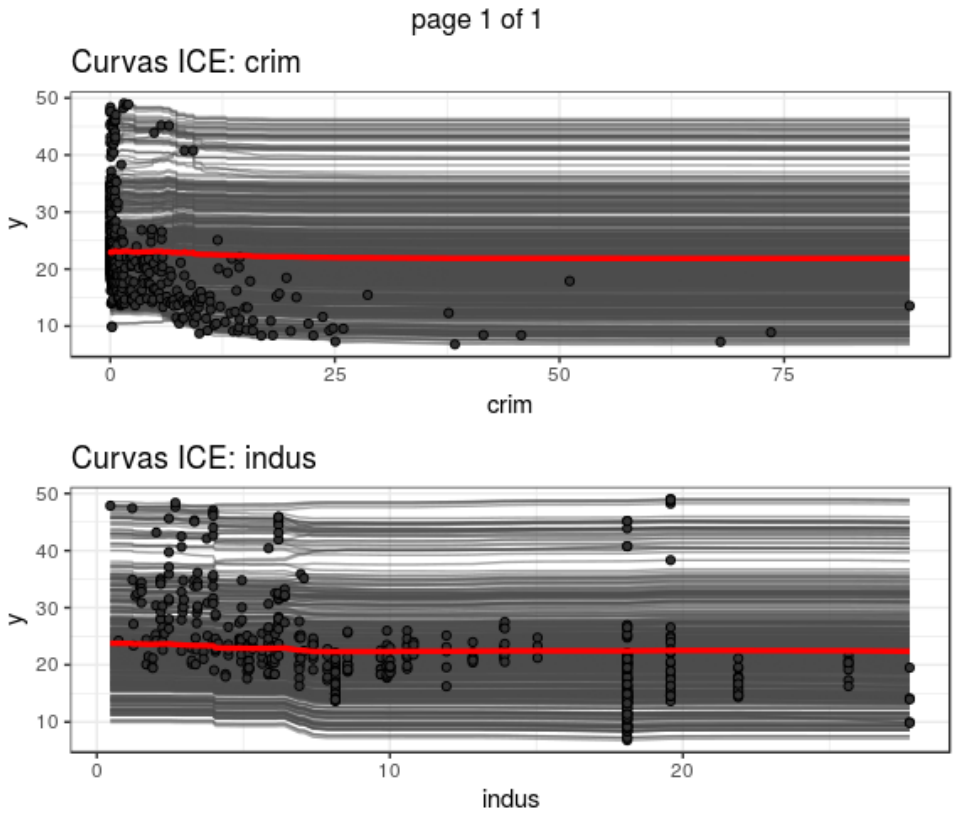
```
calcular_graficar_ice <- function(modelo, X, y, predictores = NULL,
                                  predictfcn = NULL, frac_to_build = 1,
                                  pdp = TRUE, color_by = NULL,
                                  parallel = TRUE){
  library(purrr)
  library(furrr)
  library(future)
  library(ggplot2)
  library(gridExtra)
  library(ICEbox)

  if(is.null(predictores)){
    predictores <- colnames(X)
  }
  if(any(!purrr::map_lgl(.x = X[, predictores], .f = is.numeric))){
    print("Solo pueden calcularse curvas ICE de predictores numéricos")
    predictores <- predictores[purrr::map_lgl(.x = X[, predictores],
                                              .f = is.numeric)]
  }
  # Si se paraleliza se emplea furrr::future_map
  if(parallel){
    # Paralelización en múltiples cores.
    future::plan(strategy = future::multiprocess,
                 workers = future::availableCores(constraints = "multicore") - 1)
    # Cálculo de curvas ice para cada uno de los predictores.
    if(is.null(predictfcn)){
      curvas_ice_predictores <- furrr::future_map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
    }else{
      curvas_ice_predictores <- furrr::future_map(
```

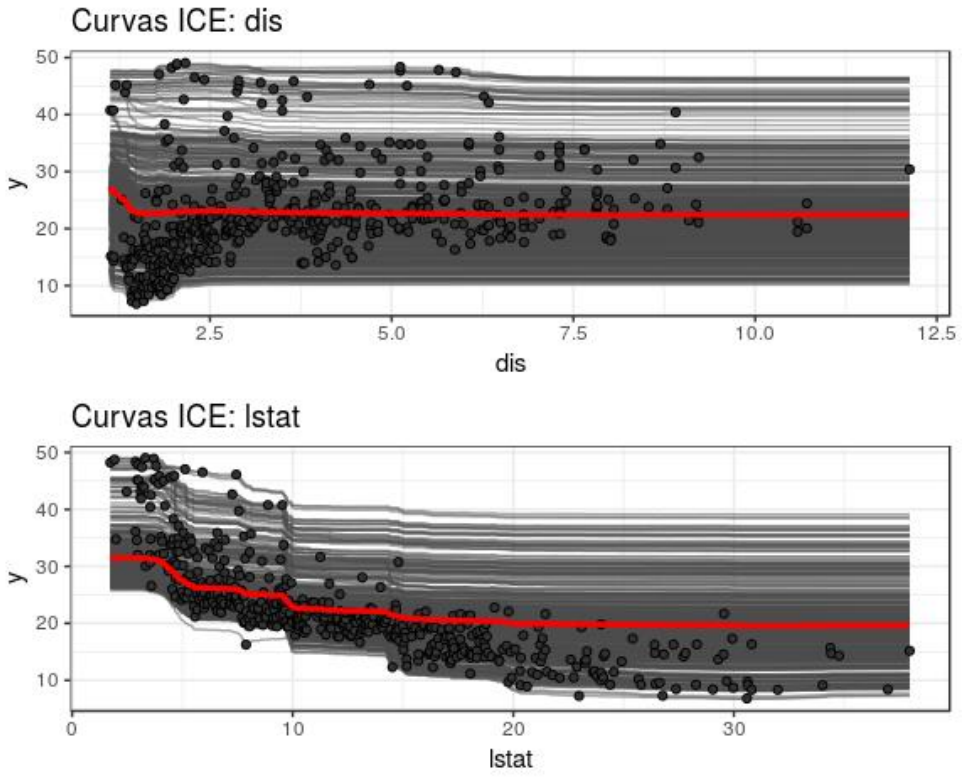
```
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  predictfcn = predictfcn,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }

  names(curvas_ice_predictores) <- predictores
  # Creación gráficos ice
  graficos <- furrr::future_map(
                .x = curvas_ice_predictores,
                .f = plot_ice,
                pdp = pdp,
                color_by = color_by
              )
}

# Si no se paraleliza se emplea purrr:map
if(!parallel){
  # Cálculo de curvas ice para cada uno de los predictores.
  if(is.null(predictfcn)){
    curvas_ice_predictores <- purrr::map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }else{
    curvas_ice_predictores <- purrr::map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  predictfcn = predictfcn,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }
  names(curvas_ice_predictores) <- predictores
  # Creación gráficos ice
  graficos <- purrr::map(
              .x = curvas_ice_predictores,
```

```
                  .f = plot_ice,
                  pdp = pdp,
                  color_by = color_by
                )
  }

 gridExtra::marrangeGrob(graficos, ncol = 1, nrow = length(predictores))
}
```

Ejemplo

```
calcular_graficar_ice(modelo = modelo_rf,
                      X = datos_x,
                      y = datos_y,
                      predictores = c("crim","indus", "dis", "lstat"),
                      frac_to_build = 1,
                      pdp = TRUE,
                      parallel = TRUE
                      )
```





c-ICE

Función

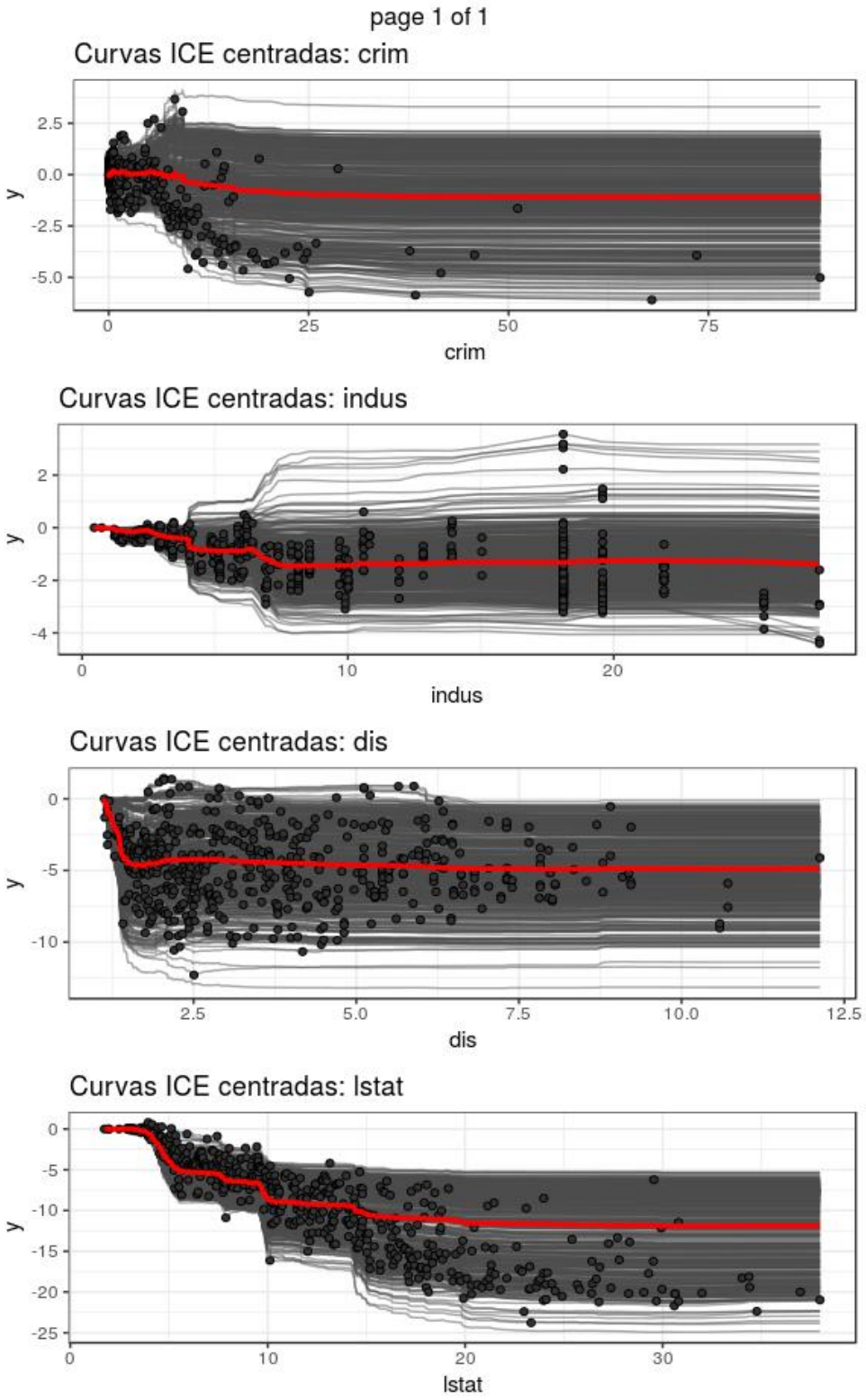
```
calcular_graficar_c_ice <- function(modelo, X, y, predictores = NULL,
                                    predictfcn = NULL, frac_to_build = 1,
                                    pdp = TRUE, color_by = NULL,
                                    parallel = TRUE){
  library(purrr)
  library(furrr)
  library(future)
  library(ggplot2)
  library(gridExtra)
  library(ICEbox)

  if(is.null(predictores)){
    predictores <- colnames(X)
  }

  if(any(!purrr::map_lgl(.x = X[, predictores], .f = is.numeric))){
    print("Solo pueden calcularse curvas ICE de predictores numéricos")
    predictores <- predictores[purrr::map_lgl(.x = X[, predictores],
                                              .f = is.numeric)]
  }
  # Si se paraleliza se emplea furrr::future_map
```

```
  if(parallel){
    # Paralelización en múltiples cores.
    future::plan(strategy = future::multiprocess,
                 workers = future::availableCores(constraints = "multicore") - 1)

    # Cálculo de curvas ice para cada uno de los predictores.
    if(is.null(predictfcn)){
      curvas_ice_predictores <- furrr::future_map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
    }else{
      curvas_ice_predictores <- furrr::future_map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  predictfcn = predictfcn,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
    }
    names(curvas_ice_predictores) <- predictores

    # Creación gráficos c-ice
    graficos <- furrr::future_map(
                  .x = curvas_ice_predictores,
                  .f = plot_c_ice,
                  pdp = pdp,
                  color_by = color_by
                )
  }
  # Si no se paraleliza se emplea purrr::map
  if(!parallel){
    # Cálculo de curvas ice para cada uno de los predictores.
    if(is.null(predictfcn)){
      curvas_ice_predictores <- purrr::map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
```

```
                                )
    }else{
      curvas_ice_predictores <- purrr::map(
                                .x = predictores,
                                .f = ICEbox::ice,
                                object = modelo,
                                predictfcn = predictfcn,
                                X = X,
                                y = y,
                                frac_to_build = frac_to_build,
                                verbose = FALSE
                                )
    }
    names(curvas_ice_predictores) <- predictores

    # Creación gráficos c-ice
    graficos <- purrr::map(
                  .x = curvas_ice_predictores,
                  .f = plot_c_ice,
                  pdp = pdp,
                  color_by = color_by
                )
  }

   gridExtra::marrangeGrob(graficos, ncol = 1, nrow = length(predictores))
}
```

Ejemplo

```
calcular_graficar_c_ice(modelo = modelo_rf,
                        X = datos_x,
                        y = datos_y,
                        predictores = c("crim","indus", "dis", "lstat"),
                        frac_to_build = 1,
                        pdp = TRUE,
                        parallel = TRUE
                       )
```


page 1 of 1
Curvas ICE centradas: crim
y
crim
Curvas ICE centradas: indus
y
indus
Curvas ICE centradas: dis
y
dis
Curvas ICE centradas: lstat
y
lstat

d-ICE

Función

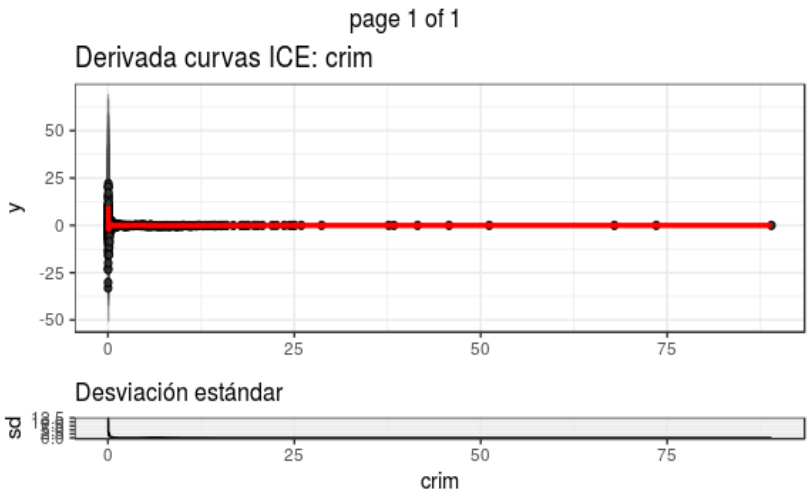
```
calcular_graficar_d_ice <- function(modelo, X, y, predictores = NULL,
                                    predictfcn = NULL, frac_to_build = 1,
                                    pdp = TRUE, color_by = NULL,
                                    parallel = TRUE){

  library(purrr)
  library(furrr)
  library(future)
  library(ggplot2)
  library(gridExtra)
  library(ICEbox)

  if(is.null(predictores)){
    predictores <- colnames(X)
  }

  if(any(!purrr::map_lgl(.x = X[, predictores], .f = is.numeric))){
    print("Solo pueden calcularse curvas ICE de predictores numéricos")
    predictores <- predictores[purrr::map_lgl(.x = X[, predictores],
                                              .f = is.numeric)]
  }

  # Si se paraleliza se emplea furrr::future_map
  if(parallel){
    # Paralelización en múltiples cores.
    future::plan(strategy = future::multiprocess,
                 workers = future::availableCores(constraints = "multicore") - 1)

    # Cálculo de curvas ice para cada uno de los predictores.
    if(is.null(predictfcn)){
      curvas_ice_predictores <- furrr::future_map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
    }else{
      curvas_ice_predictores <- furrr::future_map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  predictfcn = predictfcn,
```

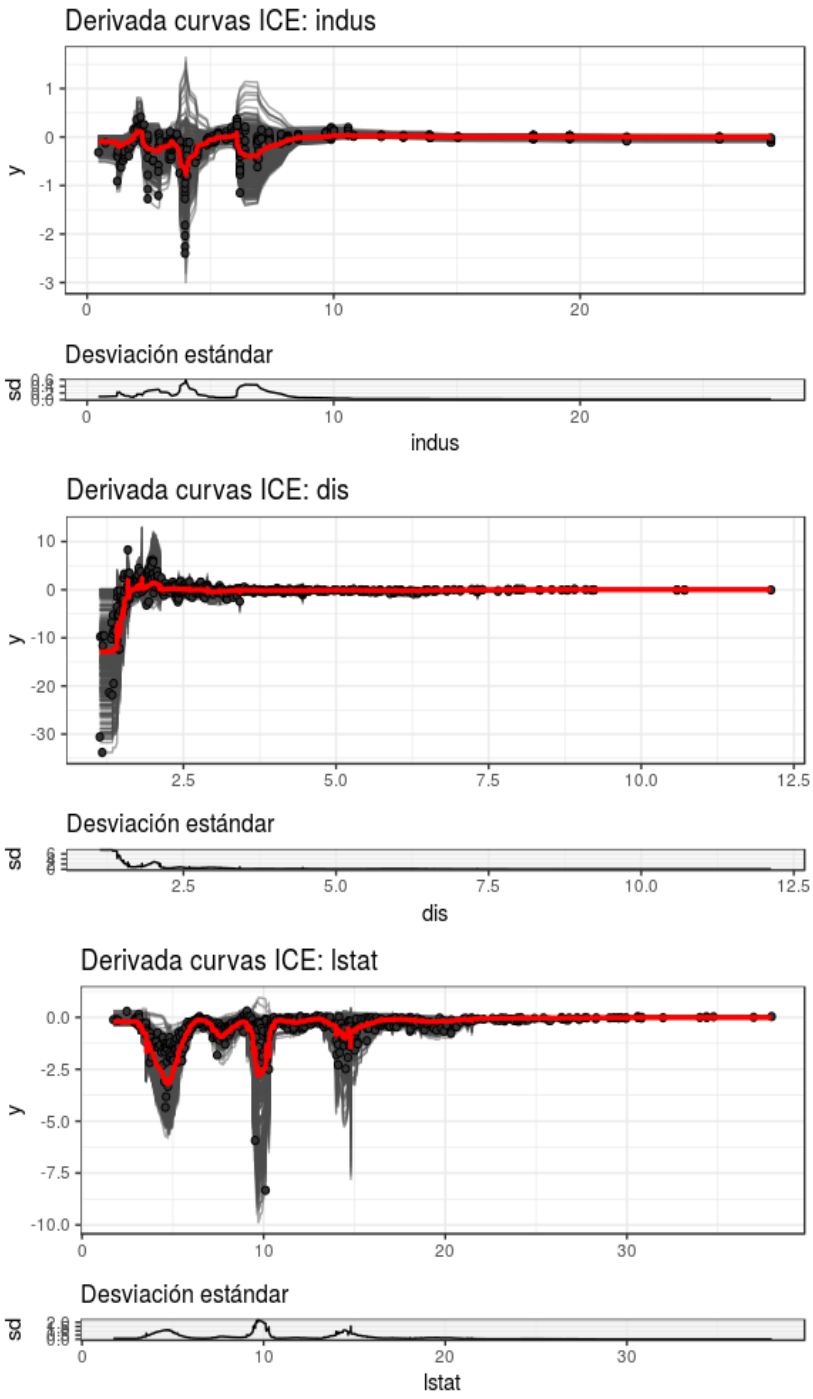
```
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }
  names(curvas_ice_predictores) <- predictores
  # Cálculo de derivadas ice para cada uno de los predictores.
  derivadas_ice_predictores <- furrr::future_map(
                                  .x = curvas_ice_predictores,
                                  .f = ICEbox::dice
                                  )
  # Creación gráficos d-ice
  graficos <- furrr::future_map(
                .x = derivadas_ice_predictores,
                .f = plot_d_ice,
                pdp = pdp,
                color_by = color_by
              )
}

# Si no se paraleliza se emplea purrr::map
if(parallel){
  # Paralelización en múltiples cores.
  future::plan(strategy = future::multiprocess,
               workers = future::availableCores(constraints = "multicore") - 1)

  # Cálculo de curvas ice para cada uno de los predictores.
  if(is.null(predictfcn)){
    curvas_ice_predictores <- purrr::map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }else{
    curvas_ice_predictores <- purrr::map(
                                  .x = predictores,
                                  .f = ICEbox::ice,
                                  object = modelo,
                                  predictfcn = predictfcn,
                                  X = X,
                                  y = y,
                                  frac_to_build = frac_to_build,
                                  verbose = FALSE
                                )
  }
```

```
    names(curvas_ice_predictores) <- predictores
    # Cálculo de derivadas ice para cada uno de los predictores.
    derivadas_ice_predictores <- purrr::map(
                                    .x = curvas_ice_predictores,
                                    .f = ICEbox::dice
                                   )
    # Creación gráficos d-ice
    graficos <- purrr::map(
                    .x = derivadas_ice_predictores,
                    .f = plot_d_ice,
                    pdp = pdp,
                    color_by = color_by
                )
  }
 gridExtra::marrangeGrob(graficos, ncol = 1, nrow = length(predictores))
}
```

Ejemplo

```
calcular_graficar_d_ice(modelo = modelo_rf,
                        X = datos_x,
                        y = datos_y,
                        predictores = c("crim","indus", "dis", "lstat"),
                        frac_to_build = 1,
                        pdp = TRUE,
                        parallel = TRUE
                       )
```




Derivada curvas ICE: indus
y
Desviación estándar
sd
indus
Derivada curvas ICE: dis
y
Desviación estándar
sd
dis
Derivada curvas ICE: lstat
y
Desviación estándar
sd
lstat

ICE de modelos H2O

La librería H2O se ha convertido en un referente para modelos de *machine learning*. Como los modelos de **H2O** no se crean en el entorno de **R**, sino en un *cluster* de *Java*, para obtener gráficos *ICE* son necesarios algunas modificaciones.

- Definir una función `predict` que envíe los nuevos datos al *cluster* H2O en el formato adecuado, y que devuelva un vector con un valor numérico.
- Pasar los datos en formato `h2o` de nuevo a `data.frame`.
- Desactivar la paralelización. Por alguna razón, la función `furture_map` genera un error al interactuar con H2O.

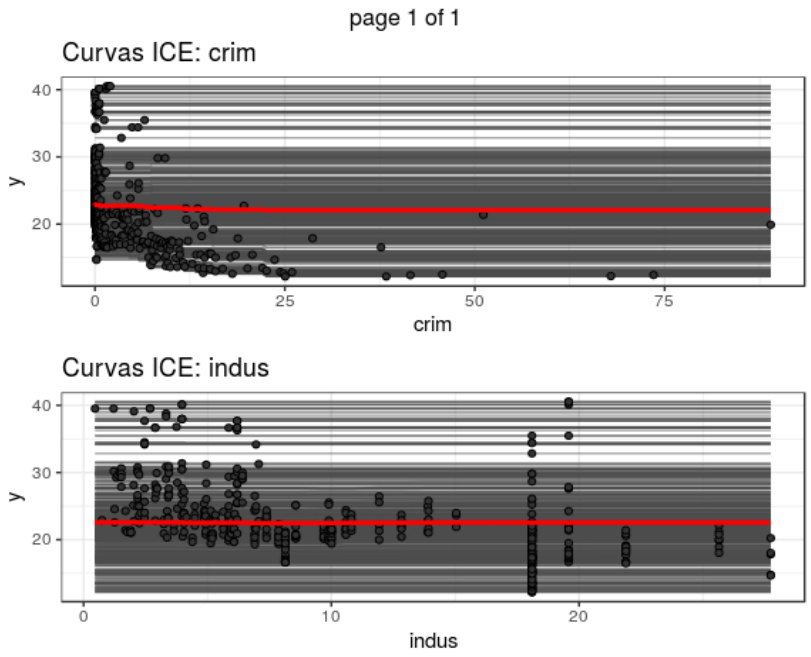
Debido a que los datos se tienen que transferir continuamente entre el cluster H2O y el entorno de R, la creación de los gráficos ICE aquí propuesta puede resultar muy lenta. Para una implementación más eficiente, conviene utilizar el paquete `pdp` tal como se describe en el documento Machine Learning con H2O y R.

Se inicia un *cluster* H2O y se ajusta un modelo *GBM* empleando la función h2o.gbm. Se puede encontrar información más detallada sobre el uso de esta libraría en el documento Machine Learning con H2O y R.

```
library(h2o)

datos <- MASS::Boston

h2o.init(ip = "localhost",
         nthreads = -1,
         max_mem_size = "4g")

h2o.removeAll()
h2o.no_progress()
```

```
# Carga de datos en el cluster H2O.
datos_h2o <- as.h2o(x = datos)

# Una vez cargados los datos en H2O, se eliminan del entorno R.
rm("datos")

# Definición de los predictores y de la variable respuesta.
predictores <- c("crim", "zn", "indus", "chas", "nox", "rm", "age", "dis",
                 "rad", "tax", "ptratio", "black", "lstat")
y <- "medv"
```

```
# Ajuste del modelo.
modelo_h2o <- h2o.gbm(
                  x = predictores,
                  y = y,
                  training_frame = datos_h2o,
                  max_depth =  10,
                  learn_rate =  0.01,
                  ntrees = 5000,
                  min_rows = 10,
                  min_split_improvement = 0,
                  nbins = 20,
                  sample_rate = 1,
                  distribution = "gaussian",
                  stopping_rounds = 4,
                  stopping_metric = "MSE",
                  stopping_tolerance = 0.05,
                  model_id = "modelo_gbm"
          )

# Función predict especial para un modelo H2O.
predict_custom <- function(object, newdata){
  as.vector(predict(object, newdata = as.h2o(newdata)))
}

calcular_graficar_ice(
    modelo = modelo_h2o,
    X = as.data.frame(datos_h2o)[, predictores],
    y = as.data.frame(datos_h2o)[, y],
    predictfcn = predict_custom,
    predictores = c("crim","indus"),
    frac_to_build = 1,
    pdp = TRUE,
    parallel = FALSE
  )
```




```
# Se apaga el cluster H2O.
h2o.shutdown(prompt = FALSE)
```

```
## [1] TRUE
```

Anexos

Función plot.dice corregida

La función `plot.ice()` del paquete `ICEbox` parece generar un error cuando se especifica que no se muestre la curva PDP (`plot_pdp = FALSE`). A continuación, se muestra una ligera modificación de la función que incluye las siguientes líneas de código y que soluciona el error.

```
else{
    pdp = NULL
  }
```

```
plot.ice <- function (x, plot_margin = 0.05, frac_to_plot = 1,
                      plot_points_indices = NULL,
                      plot_orig_pts_preds = TRUE, pts_preds_size = 1.5, colorvec,
                      color_by = NULL, x_quantile = FALSE, plot_pdp = TRUE,
                      centered = FALSE, prop_range_y = TRUE,
                      rug_quantile = seq(from = 0, to = 1, by = 0.1),
                      centered_percentile = 0, point_labels = NULL,
                      point_labels_size = NULL, prop_type = "sd", ...)
{
  DEFAULT_COLORVEC = c("firebrick3", "dodgerblue3", "gold1",
                       "darkorchid4", "orange4", "forestgreen", "grey", "black")
  if (class(x) != "ice") {
    stop("object is not of class \"ice\"")
  }
  if (frac_to_plot <= 0 || frac_to_plot > 1) {
    stop("frac_to_plot must be in (0,1]")
  }
  if (!(prop_type %in% c("sd", "range"))) {
    stop("prop_type must be either 'sd' or 'range'")
  }
  grid = x$gridpts
  n_grid = length(grid)
  ecdf_fcn = NULL
  if (x_quantile) {
    ecdf_fcn = ecdf(grid)
    grid = ecdf_fcn(grid)
  }
  ice_curves = x$ice_curves
  N = nrow(ice_curves)
  if (!is.null(point_labels)) {
    if (length(point_labels) != N) {
```

```
      stop("point_labels must be same length as number of ICE curves: ",
           N)
    }
  }
  legend_text = NULL
  if (missing(colorvec) && missing(color_by)) {
    colorvec = sort(rgb(rep(0.4, N), rep(0.4, N), rep(0.4,
                                                      N), runif(N, 0.4, 0.8)))
  }
  if (!missing(colorvec) && !missing(color_by)) {
    if (!missing(colorvec) && length(colorvec) < N) {
      stop("color vector has length ", length(colorvec),
           " but there are ", N, " lines to plot")
    }
  }
  if (!missing(color_by) && missing(colorvec)) {
    arg_type = class(color_by)
    if (!(arg_type %in% c("character", "numeric", "factor"))) {
      stop("color_by must be a column name in X or a column index")
    }
    if (class(color_by) == "character") {
      if (!(color_by %in% names(x$Xice))) {
        stop("The predictor name given by color_by was not found in the X matrix")
      }
      x_color_by = x$Xice[, color_by]
    }
    else if (length(color_by) > N) {
      stop("The color_by_data vector you passed in has ",
           length(color_by), " entries but the ICEbox object only has ",
           N, " curves.")
    }
    else if (length(color_by) == N) {
      x_color_by = color_by
    }
    else {
      if (color_by < 1 || color_by > ncol(x$Xice) || (color_by%%1 !=
                                                      0)) {
        stop("color_by must be a column name in X or a column index")
      }
      x_color_by = x$Xice[, color_by]
    }
    x_unique = sort(unique(x_color_by))
    num_x_color_by = length(x_unique)
    if (num_x_color_by <= 10) {
      if (missing(colorvec)) {
        which_category = match(x_color_by, x_unique)
        colorvec = DEFAULT_COLORVEC[which_category]
      }
      legend_text = as.data.frame(cbind(x_unique,
DEFAULT_COLORVEC[1:num_x_color_by]))
```

```
      x_column_name = ifelse(length(color_by) == N, "data vector level",
                             ifelse(is.character(color_by),
                                    color_by,
                                    paste("x_", color_by, sep = "")))
      names(legend_text) = c(x_column_name, "color")
      cat("ICE Plot Color Legend\n")
      print(legend_text, row.names = FALSE)
    }
    else {
      if (is.factor(x_color_by)) {
        warning("color_by is a factor with greater than 10 levels: coercing to
numeric.")
        x_color_by = as.numeric(x_color_by)
      }
      alpha_blend_colors = matrix(0, nrow = N, ncol = 3)
      alpha_blend_colors[, 1] = seq(from = 1, to = 0, length.out = N)
      alpha_blend_colors[, 2] = seq(from = 0, to = 1, length.out = N)
      alpha_blend_colors[, 3] = 0
      rgbs = array(NA, N)
      for (i in 1:N) {
        rgbs[i] = rgb(alpha_blend_colors[i, 1],
                      alpha_blend_colors[i, 2],
                      alpha_blend_colors[i, 3])
      }
      colorvec = rgbs[sort(x_color_by, index.return = T)$ix]
      cat("ICE Plot Color Legend: red = low values of the color_by variable and
green = high values\n")
    }
  }
  if (is.null(plot_points_indices)) {
    plot_points_indices = sample(1:N, round(frac_to_plot *
                                              N))
  }
  else {
    if (frac_to_plot < 1) {
      stop("frac_to_plot has to be 1 when plot_points_indices is passed to the plot
function.")
    }
  }
  ice_curves = ice_curves[plot_points_indices, ]
  if (nrow(ice_curves) == 0) {
    stop("no rows selected: frac_to_plot too small.")
  }
  if (centered) {
    centering_vector = ice_curves[, max(ncol(ice_curves) *
                                          centered_percentile, 1)]
    ice_curves = ice_curves - centering_vector
  }
  colorvec = colorvec[plot_points_indices]
  min_ice_curves = min(ice_curves)
```

```
  max_ice_curves = max(ice_curves)
  range_ice_curves = max_ice_curves - min_ice_curves
  min_ice_curves = min_ice_curves - plot_margin * range_ice_curves
  max_ice_curves = max_ice_curves + plot_margin * range_ice_curves
  arg_list = list(...)
  arg_list = modifyList(arg_list, list(x = grid, y = ice_curves[1,
                                                                ]))
  if (is.null(arg_list$xlab)) {
    xlab = x$xlab
    arg_list = modifyList(arg_list, list(xlab = xlab))
    if (x_quantile) {
      xlab = paste("quantile(", xlab, ")", sep = "")
      arg_list = modifyList(arg_list, list(xlab = xlab))
    }
    if (!missing(color_by)) {
      xlab = paste(xlab, "colored by", ifelse(length(color_by) ==
                                                N, "a provided data vector",
color_by))
      arg_list = modifyList(arg_list, list(xlab = xlab))
    }
  }
  if (is.null(arg_list$ylab)) {
    if (x$logodds) {
      ylab = "partial log-odds"
      arg_list = modifyList(arg_list, list(ylab = ylab))
    }
    else if (x$probit) {
      ylab = "partial probit"
      arg_list = modifyList(arg_list, list(ylab = ylab))
    }
    else {
      ylab = paste("partial yhat", ifelse(centered, "(centered)",
                                          ""))
      arg_list = modifyList(arg_list, list(ylab = ylab))
    }
  }
  if (is.null(arg_list$xaxt)) {
    xaxt = ifelse(x$nominal_axis, "n", "s")
    arg_list = modifyList(arg_list, list(xaxt = xaxt))
  }
  if (is.null(arg_list$ylim)) {
    ylim = c(min_ice_curves, max_ice_curves)
    arg_list = modifyList(arg_list, list(ylim = ylim))
  }
  if (is.null(arg_list$type)) {
    type = "n"
    arg_list = modifyList(arg_list, list(type = type))
  }
  do.call("plot", arg_list)
  if (x$nominal_axis) {
```

```
    axis(1, at = sort(x$xj), labels = sort(x$xj), cex.axis = arg_list$cex.axis)
  }
  if (centered && prop_range_y && !x$logodds && !x$probit) {
    at = seq(min(ice_curves), max(ice_curves), length.out = 5)
    at = at - min(abs(at))
    if (prop_type == "range") {
      labels = round(at/x$range_y, 2)
    }
    else {
      labels = round(at/x$sd_y, 2)
    }
    axis(4, at = at, labels = labels, cex.axis = arg_list$cex.axis)
  }
  for (i in 1:nrow(ice_curves)) {
    points(grid, ice_curves[i, ], col = colorvec[i], type = "l")
  }
  if (plot_orig_pts_preds) {
    yhat_actual = x$actual_prediction[plot_points_indices]
    if (centered) {
      yhat_actual = yhat_actual - centering_vector
    }
    if (x_quantile) {
      xj = ecdf_fcn(x$xj)[plot_points_indices]
    }
    else {
      xj = x$xj[plot_points_indices]
    }
    for (i in 1:length(xj)) {
      points(xj[i], yhat_actual[i], col = rgb(0.1, 0.1,
                                              0.1), pch = 16, cex = pts_preds_size)
      points(xj[i], yhat_actual[i], col = colorvec[i],
             pch = 16, cex = round(pts_preds_size * 0.7))
    }
  }
  if (!is.null(point_labels)) {
    text(xj, yhat_actual, pos = 4, labels = point_labels[plot_points_indices],
         cex = ifelse(is.null(point_labels_size), pts_preds_size,
                      point_labels_size))
  }
  if (!is.null(rug_quantile) && !x_quantile) {
    axis(side = 1, line = -0.1, at = quantile(x$xj, rug_quantile),
         lwd = 0, tick = T, tcl = 0.4, lwd.ticks = 2, col.ticks = "blue4",
         labels = FALSE, cex.axis = arg_list$cex.axis)
  }
  if (plot_pdp) {
    pdp = apply(ice_curves, 2, mean)
    if (centered) {
      pdp = pdp - pdp[ceiling(length(pdp) * centered_percentile +
                                1e-05)]
    }
```

```
    num_lines = length(plot_points_indices)
    points(grid, pdp, col = "yellow", type = "l", lwd = min(5.5 +
                                                              (num_lines/100) *
0.75, 8))
    points(grid, pdp, col = "BLACK", type = "l", lwd = 4)
  }else{
    pdp = NULL
  }
  invisible(list(ice_curves = ice_curves, range_ice_curves = range_ice_curves,
                 plot_points_indices = plot_points_indices, legend_text =
legend_text,
                 pdp = pdp))
}
```

Bibliografía

Goldstein, Alex & Kapelner, Adam & Bleich, Justin & Pitkin, Emil. (2013). Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. Journal of Computational and Graphical Statistics.