Algoritmo genético para selección de predictores

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Noviembre, 2018

Tabla de contenidos

Introducción 2
Algoritmo 3
Población inicial 4
Fitness de un individuo 6
Fitness de todos los individuos de una población 14
Seleccionar individuos 18
Cruzar dos individuos (*crossover*, recombinación) 23
Mutar individuo 25
Algoritmo completo 27
Ejemplo regresión 32
Simulación de datos 32
Selección de predictores 34
Mejor individuo 40
Evolución del error 41
Frecuencia selección variables 41
Ejemplo clasificación 45
Datos 45
Selección de predictores 45
Mejor individuo 46
Evolución del error 46
Frecuencia selección variables 47
Paralelización 49
Versión paralelizada 49
Comparación 57
Bibliografía 59

Versión PDF: Github

Introducción

Los algoritmos genéticos son métodos de optimización heurística que, entre otras aplicaciones, pueden emplearse para encontrar la combinación de variables que consigue maximizar la capacidad predictiva de un modelo. Su funcionamiento está inspirado en la teoría evolutiva de selección natural propuesta por Darwin y Alfred Russel: los individuos de una población se reproducen generando nuevos descendientes, cuyas características, son combinación de las características de los progenitores (más ciertas mutaciones). De todos ellos, únicamente los mejores individuos sobreviven y pueden reproducirse de nuevo, transmitiendo así sus características a las siguientes generaciones.

Los algoritmos genéticos son solo una de las muchas estrategias que existen para seleccionar los predictores más relevantes, y no tiene por qué ser la más adecuada en todos los escenarios. Por ejemplo, existen estrategias iterativas Stepwise selection, modelos como Random Forest, Boosting y Lasso capaces de excluir automáticamente predictores, y técnicas de reducción de dimensión como PCA y t-SNE.

La implementación de algoritmo genético que se muestra en este documento pretende ser lo más explicativa posible aunque para ello no sea la más eficiente.

El código de las funciones desarrolladas a lo largo del documento puede descargarse en el siguiente link.

Algoritmo

Aunque existen variaciones, algunas de las cuales se describen a lo largo de este documento, en términos generales, la estructura de un algoritmo genético para la selección de predictores sigue los siguientes pasos:

---

1. Crear una población inicial aleatoria de *P* individuos. En este caso, cada individuo representa una combinación de predictores.
2. Calcular la fortaleza (*fitness*) de cada individuo de la población.
3. Crear una nueva población vacía y repetir los siguientes pasos hasta que se hayan creado *P* nuevos individuos.

   3.1 Seleccionar dos individuos de la población existente, donde la probabilidad de selección es proporcional al *fitness* de los individuos.

   3.2 Cruzar los dos individuos seleccionados para generar un nuevo descendiente (*crossover*).

   3.3 Aplicar un proceso de mutación aleatorio sobre el nuevo individuo.

   3.4 Añadir el nuevo individuo a la nueva población.
4. Reemplazar la antigua población por la nueva.
5. Si no se cumple un criterio de parada, volver al paso 2.

En los siguientes apartados se describe cada una de las etapas del proceso para, finalmente, combinarlas todas en una única función.

Población inicial

En el contexto de algoritmos genéticos, el término individuo hace referencia a cada una de las posibles soluciones del problema que se quiere optimizar. En el caso particular de la selección de mejores predictores, cada individuo representa una posible combinación de variables. Para representar dichas combinaciones, se pueden emplear vectores binarios, cuya longitud es igual al número total de predictores disponibles, y cada posición toma el valor TRUE/FALSE dependiendo de si el predictor que ocupa esa posición se incluye o excluye. También es común encontrar la representación en términos 0/1.

Por ejemplo, supóngase que las variables disponibles son: X_1, X_2, X_3, X_4 y X_5. El individuo $TRUE, FALSE, TRUE, TRUE, FALSE$ o su equivalente codificación 1,0,1,1,0 representa la selección de los predictores X_1, X_3, y X_4.

El primer paso del algoritmo genético para la selección de predictores consiste en crear una población inicial aleatoria de individuos. La siguiente función crea una matriz binaria en la que, cada fila, está formada una combinación aleatoria de valores TRUE y FALSE. Además, el número máximo y mínimo de TRUEs por fila puede estar acotado. Esta acotación resulta útil cuando se quiere limitar, en cierta medida, el número de predictores que pueden incluir los individuos. Es en cierta medida porque, debido a los cruces y mutaciones a lo largo de las generaciones, se pueden crear individuos que excedan los límites iniciales.

```
crear_poblacion <- function(n_poblacion, n_variables, n_max = NULL, n_min = NULL,
                            verbose = TRUE) {
  # ARGUMENTOS
  # ============================================================================
  # n_poblacion: número total de individuos de la población.
  # n_variables: longitud de los individuos.
  # n_max:       número máximo de TRUEs que puede contener un individuo.
  # n_min:       número mínimo de TRUEs que puede contener un individuo.
  # verbose:     mostrar información del proceso por pantalla.

  # RETORNO
  # ============================================================================
  # Una matriz de tamaño n_poblacion x n_variables que representa una población.

  # Comprobaciones.
  if (isTRUE(n_max > n_variables)) {
    stop("n_max no puede ser mayor que n_variables.")
  }
```

```r
  # Si no se especifica n_max, el número máximo de predictores (TRUEs) que puede
  # contener un individuo es igual al número total de variables disponibles.
  if (is.null(n_max)) {
    n_max <- n_variables
  }

  # Si no se especifica n_min, el número mínimo de predictores (TRUEs) que puede
  # contener un individuo es 1.
  if (is.null(n_min)) {
    n_min <- 1
  }

  # Matriz donde almacenar los individuos generados.
  poblacion <- matrix(data = NA, nrow = n_poblacion, ncol = n_variables)

  # Bucle para crear cada individuo.
  for (i in 1:n_poblacion) {
    # Se selecciona (con igual probabilidad) el número de valores = TRUE que puede
    # tener el individuo, dentro del rango acotado por n_min y n_max.
    n_true <- sample(x = n_min:n_max, size = 1)

    # Se crea un vector con todo FALSE que representa el individuo.
    individuo <- rep(FALSE, times = n_variables)

    # Se sustituyen (n_true) posiciones aleatorias por valores TRUE.
    individuo[sample(x = 1:n_variables, size = n_true)] <- TRUE

    # Se añade el nuevo individuo a la población.
    poblacion[i, ] <- individuo
  }

  if (verbose) {
    print("Población inicial creada")
    print(paste("Número de individuos =", n_poblacion))
    print(paste("Número de predictores mínimo por individuo =", n_min))
    print(paste("Número de predictores máximo por individuo =", n_max))
    cat("\n")
    print(poblacion)
    cat("\n")
  }

  return(poblacion)
}
```

Ejemplo

Se crea una población de 10 individuos de longitud 8, con un número de valores TRUE acotado entre 1 y 5.

```
población <- crear_poblacion(
                n_poblacion = 10,
                n_variables = 8,
                n_max = 5,
                n_min = 1,
                verbose = TRUE
             )
```

```
## [1] "Población inicial creada"
## [1] "Número de individuos = 10"
## [1] "Número de predictores mínimo por individuo = 1"
## [1] "Número de predictores máximo por individuo = 5"
##
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
##  [1,]  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE
##  [2,] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
##  [3,] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [4,]  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [5,] FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
##  [6,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  [7,] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [8,]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
##  [9,] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
## [10,] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

Fitness de un individuo

Cada individuo de la población debe ser evaluado para cuantificar su fortaleza (*fitness*). Dado que, en este caso, el objetivo es encontrar la combinación de predictores que da lugar al mejor modelo, el *fitness* de un individuo se calcula con una métrica de calidad del modelo. Dependiendo de la métrica, la relación con el *fitness* puede ser:

- Error del modelo: si se emplea una métrica de error, el individuo tiene mayor *fitness* cuanto menor sea el error.
- *Accuracy*, *precision*, *recall*, *F1*...: con este tipo de métricas, el individuo tiene mayor *fitness* cuanto mayor sea la métrica.

Para conseguir que, independientemente de la métrica, cuanto mayor sea su valor, mayor el *fitness* del individuo, se puede utilizar el $-(error)$. De esta forma, la estrategia de selección es

la misma. Es importante recordar que, en el caso del $-(error)$, el intervalo de valores posibles es $-\infty$ a 0. Además, para que las estimaciones sean robustas y evitar el *overfitting*, es muy importante recurrir a estrategias de validación cruzada o *bootstrapping* en la cuantificación del *fitness*.

En este ejemplo, se presentan tres opciones de modelos de evaluación: un modelo lineal por mínimos cuadrados, un modelo de regresión logística y un *random forest*.

- El modelo lineal solo puede utilizarse para problemas de regresión y emplea la métrica -*MSE* como indicativo de *fitness*.
- El modelo de regresión logística solo puede utilizarse para clasificación binaria y emplea las métricas *Accuracy*, índice *Kappa* o índice *F1*.
- El *random forest* puede utilizarse para problemas de regresión, en cuyo caso emplea la métrica *-(MSE)*, y clasificación, en cuyo caso emplea el *Accuracy*, índice *Kappa* o índice *F1*.

Versión con random forest

```
calcular_fitness_individuo_rf <- function(x, y, cv, metrica=NULL,
                                          nivel_referencia=NULL,
                                          seed=123, verbose=TRUE, ...) {
  library(MLmetrics)
  library(caret)
  # ARGUMENTOS
  # ============================================================================
  # x:       matriz de predictores.
  # y:       variable respuesta.
  # cv:      número de particiones de validación cruzada.
  # seed:    semilla para garantizar reproducibilidad en el proceso de CV.
  # metrica: métrica utilizad apara calcular el fitness. Por defecto es el -MSE
  #          para regresión y Accuracy para clasificación. En el caso de
  #          clasificación binaria, se acepta también f1 y kappa.
  # nivel_referencia: valor de la variable respuesta considerado como referencia.
  #                   Necesario cuando la métrica es f1 o kappa.
  # verbose: mostrar información del proceso por pantalla.

  # RETORNO
  # ============================================================================
  # fitness del individuo obtenido por validación cruzada empleando
  # random forest como modelo.

  # Repartición de las observaciones para la validación cruzada (CV).
  set.seed(seed)
  #indices_cv <- sample(x = 1:cv, size = nrow(x), replace = TRUE)
  indices_cv <- caret::createFolds(y = y, k = cv, list = FALSE)
  # Vector para almacenar el fitness de cada iteración CV.
```

```
  fitness_cv <- rep(NA, times = cv)

  for (i in 1:cv) {
    set.seed(seed)
    modelo <- randomForest::randomForest(
                              x = x[indices_cv != i, , drop = FALSE],
                              y = y[indices_cv != i],
                              ntree = 100
                              )
    predicciones <- predict(modelo, newdata = x[indices_cv == i, , drop = FALSE])

    if (is.numeric(y)) {
      # Si y es numérico (regresión), el fitness es igual al -MSE.
      metrica  <- "-(mse)"
      residuos <- predicciones - y[indices_cv == i]
      fitness_cv[i] <- -(mean(residuos^2))
    } else {
      if (is.null(metrica) | metrica == "accuracy") {
        # Si y es factor (clasificación), el fitness por defecto es el accuracy.
        metrica <- "accuracy"
        accuracy <- MLmetrics::Accuracy(
                                    y_pred = predicciones,
                                    y_true = y[indices_cv == i]
                                    )
        fitness_cv[i] <- accuracy
      } else if (metrica == "kappa") {
        if (is.null(nivel_referencia)) {
          stop("Para la métrica kappa es necesario indicar el nivel de referencia.")
        } else {
            mat_confusion<- caret::confusionMatrix(table(predicciones,
                                                         y[indices_cv == i]),
                                                   positive = nivel_referencia
                                                   )
          kappa <- mat_confusion$overall["Kappa"]
          fitness_cv[i] <- kappa
        }
      } else if (metrica == "f1") {
        if (is.null(nivel_referencia)) {
          stop("Para la métrica f1 es necesario indicar el nivel de referencia.")
        } else if (sum(predicciones == nivel_referencia) < 1) {
          print(paste(
            "Ningún valor predicho ha sido el nivel de referencia,",
            "por lo tanto, el valor de precisión no puede ser calculado",
            "y tampoco la métrica F1."))
        } else {
          f1 <- MLmetrics::F1_Score(
                              y_true = y[indices_cv == i],
                              y_pred = predicciones,
                              positive = nivel_referencia
                              )
```

```
          fitness_cv[i] <- f1
        }
      }
    }

    if (is.na(fitness_cv[i])) {
      print(paste("En la particion", i, "de CV, fitness del individuo no",
                  "ha podido ser calculado."))
      print("Valores reales:")
      print(y[indices_cv == i])
      print("Valores predichos:")
      print(predicciones)
    }
  }

  if (any(is.na(fitness_cv))) {
    warning(paste(
      "En una o más particiones de CV, el fitness del individuo no",
      "ha podido ser calculado."))
  }

  if (is.na(mean(fitness_cv, na.rm = TRUE))) {
    stop("El fitness del individuo no ha podido ser calculado.")
  }

  if (verbose) {
    print(paste(
      "El fitness calculado por CV =", cv, "empleando el",
      metrica, "es de:", mean(fitness_cv, na.rm = TRUE)))
    cat("\n")
  }
  return(mean(fitness_cv, na.rm = TRUE))
}
```

Versión con modelo lineal

```
calcular_fitness_individuo_lm <- function(x, y, cv, seed=123, verbose=TRUE, ...) {
  library(MLmetrics)
  library(caret)

  # ARGUMENTOS
  # ============================================================================
  # x:       matriz de predictores.
  # y:       variable respuesta.
  # cv:      número de particiones de validación cruzada.
  # seed:    semilla para garantizar reproducibilidad en el proceso de CV.
  # verbose: mostrar información del proceso por pantalla.
```

```
  # RETORNO
  # ============================================================================
  # Fitness del individuo obtenido por validación cruzada empleando MSE como
  # métrica y regresión lineal como tipo de modelo.

  # Repartición de las observaciones para la validación cruzada (CV).
  set.seed(seed)
  #indices_cv <- sample(x = 1:cv, size = nrow(x), replace = TRUE)
  indices_cv <- caret::createFolds(y = y, k = cv, list = FALSE)

  # Vector para almacenar el fitness de cada iteración CV.
  fitness_cv <- rep(NA, times = cv)

  for (i in 1:cv) {
    datos_modelo <- as.data.frame(cbind(x[indices_cv != i, , drop = FALSE],
                                        y = y[indices_cv != i]))
    modelo <- lm(formula = y ~ .,
                 data = datos_modelo
                )

    predicciones <- predict(modelo,
                            newdata = as.data.frame(
                                        x[indices_cv == i, , drop = FALSE]
                                      )
                           )
    residuos <- predicciones - y[indices_cv == i]
    metrica  <- "-(mse)"
    fitness_cv[i] <- -(mean(residuos^2))
  }

  if (any(is.na(fitness_cv))) {
    warning(paste("En una o más particiones de CV, fitness del individuo no",
                  "ha podido ser calculado."))
  }
  if (is.na(mean(fitness_cv, na.rm = TRUE))) {
    stop("El fitness del individuo no ha podido ser calculado.")
  }
  if (verbose) {
    print(paste(
      "El fitness calculado por CV =", cv, "empleando el",
      metrica, "es de:", mean(fitness_cv, na.rm = TRUE)
    ))
    cat("\n")
  }
  return(mean(fitness_cv, na.rm = TRUE))
}
```

Versión con modelo de regresión logística

```
calcular_fitness_individuo_glm <- function(x, y, cv, metrica = NULL,
                                           nivel_referencia = NULL,
                                           seed = 123, verbose = TRUE, ...) {
  library(MLmetrics)
  library(caret)

  # ARGUMENTOS
  # ============================================================================
  # x:       matriz de predictores.
  # y:       variable respuesta.
  # cv:      número de particiones de validación cruzada.
  # seed:    semilla para garantizar reproducibilidad en el proceso de CV.
  # metrica: métrica utilizad apara calcular el fitness. Por defecto es el -MSE
  #          para regresión y Accuracy para clasificación. En el caso de
  #          clasificación binaria, se acepta también f1 y kappa.
  # nivel_referencia: valor de la variable respuesta considerado como referencia.
  #                   Necesario cuando la métrica es f1 o kappa.
  # verbose: mostrar información del proceso por pantalla.
  #
  # RETORNO
  # ============================================================================
  # Fitness del individuo obtenido por validación cruzada empleando glm como
  # modelo.

  # Comprobaciones
  if (is.numeric(y)) {
    stop("El modelo glm solo puede emplearse para clasificación binaria.")
  }

  if (is.character(y)) {
    y <- as.factor(y)
  }

  if (length(levels(y)) != 2) {
    stop("El modelo glm solo puede emplearse para clasificación binaria.")
  }


  # Repartición de las observaciones para la validación cruzada (CV).
  set.seed(seed)
  indices_cv <- caret::createFolds(y = y, k = cv, list = FALSE)

  # Vector para almacenar el fitness de cada iteración CV.
  fitness_cv <- rep(NA, times = cv)
```

```
  for (i in 1:cv) {
    datos_modelo <- as.data.frame(cbind(x[indices_cv != i, , drop = FALSE],
                                        y = y[indices_cv != i]))
    modelo <- glm(
            formula = y ~ .,
            family = "binomial",
            data = datos_modelo
            )

    predicciones <- predict(
                        modelo,
                        newdata = as.data.frame(
                                    x[indices_cv == i, , drop = FALSE]
                                  ),
                        type = "response"
                    )
    predicciones <- unname(predicciones)
    predicciones <- ifelse(predicciones < 0.5, levels(y)[1], levels(y)[2])
    predicciones <- factor(x = predicciones, levels = levels(y))

    if (is.null(metrica) | metrica == "accuracy") {
      # Si y es factor (clasificación), el fitness por defecto es el accuracy.
      metrica <- "accuracy"
      accuracy <- MLmetrics::Accuracy(
                              y_pred = predicciones,
                              y_true = y[indices_cv == i]
                              )
      fitness_cv[i] <- accuracy
    } else if (metrica == "kappa") {
      if (is.null(nivel_referencia)) {
        stop("Para la métrica kappa es necesario indicar el nivel de referencia.")
      } else {
        mat_confusion <- caret::confusionMatrix(table(predicciones,
                                                      y[indices_cv == i]),
                                                positive = nivel_referencia
                                                )
        kappa <- mat_confusion$overall["Kappa"]
        fitness_cv[i] <- kappa
      }
    } else if (metrica == "f1") {
      if (is.null(nivel_referencia)) {
        stop("Para la métrica f1 es necesario indicar el nivel de referencia.")
      } else if (sum(predicciones == nivel_referencia) < 1) {
        print(paste(
          "Ningún valor predicho ha sido el nivel de referencia,",
          "por lo tanto, el valor de precisión no puede ser calculado",
          "y tampoco la métrica F1."))
      } else {
        f1 <- MLmetrics::F1_Score(
          y_true = y[indices_cv == i],
```

```
          y_pred = predicciones,
          positive = nivel_referencia
        )
        fitness_cv[i] <- f1
      }
    }

    if (is.na(fitness_cv[i])) {
      print(paste(
        "En la particion", i, "de CV, fitness del individuo no",
        "ha podido ser calculado."))
      print("Valores reales:")
      print(y[indices_cv == i])
      print("Valores predichos:")
      print(predicciones)
      cat("\n")
    }
  }

  if (any(is.na(fitness_cv))) {
    warning(paste(
      "En una o más particiones de CV, el fitness del individuo no",
      "ha podido ser calculado."))
  }

  if (is.na(mean(fitness_cv, na.rm = TRUE))) {
    stop("El fitness del individuo no ha podido ser calculado.")
  }

  if (verbose) {
    print(paste(
      "El fitness calculado por CV =", cv, "empleando el",
      metrica, "es de:", mean(fitness_cv, na.rm = TRUE)))
    cat("\n")
  }
  return(mean(fitness_cv, na.rm = TRUE))
}
```

Ejemplo

Se calcula el *fitness* de un individuo que incluye 10 predictores simulados.

```
set.seed(123)
predictores   <- matrix(data = rnorm(n = 1000), nrow = 100, ncol = 10)
var_respuesta <- rnorm(n = 100)

calcular_fitness_individuo_rf(x = predictores,
                              y = var_respuesta,
                              metrica = "mse",
                              cv = 5,
                              seed = 123,
                              verbose = TRUE)
```

```
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -
1.1596684352833"

## [1] -1.159668
```

```
calcular_fitness_individuo_lm(x = predictores,
                              y = var_respuesta,
                              cv = 5,
                              seed = 123,
                              verbose = TRUE)
```

```
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.218267668276"

## [1] -1.218268
```

Fitness de todos los individuos de una población

Esta función recibe como argumentos una población de individuos, una matriz de predictores, un vector con la variable respuesta, y devuelve el *fitness* de cada individuo.

```
calcular_fitness_poblacion <- function(poblacion, x, y, cv, seed = 123,
                                       modelo = "rf", metrica = NULL,
                                       nivel_referencia = NULL, verbose = TRUE) {
  # ARGUMENTOS
  # ============================================================================
  # poblacion: matriz que representa la población de individuos.
  # x:         matriz de predictores.
  # y:         variable respuesta.
  # cv:        número de particiones de validación cruzada.
  # seed:      semilla para garantizar reproducibilidad en el proceso de CV.
  # verbose:   mostrar información del proceso por pantalla.
```

```
  # modelo:    tipo de modelo empleado para calcular el fitness. Puede ser
  #            lm, glm o rf.
  # metrica: métrica utilizad apara calcular el fitness. Por defecto es el -MSE
  #          para regresión y accuracy para clasificación. En el caso de
  #          clasificación binaria, se acepta también f1.
  # nivel_referencia: valor de la variable respuesta considerado como referencia.
  #                   Necesario cuando la métrica es f1 o kappa.

  # RETORNO
  # ============================================================================
  # Vector con el fitness de todos los individuos de la población, obtenido por
  # validación cruzada. El orden de los valores se corresponde con el orden de
  # las filas de la matriz población.

  # Vector donde almacenar el fitness de cada individuo.
  fitness_poblacion <- rep(NA, times = nrow(poblacion))

  # Tipo de modelo utilizado para calcular el fitness.
  if (modelo == "lm") {
    calcular_fitness_individuo <- calcular_fitness_individuo_lm
  } else if (modelo == "rf") {
    calcular_fitness_individuo <- calcular_fitness_individuo_rf
  } else if (modelo == "glm") {
    calcular_fitness_individuo <- calcular_fitness_individuo_glm
  } else {
    stop(paste(
      "El modelo empleado para calcular el fitness debe ser",
      "lm (linear model)",
      "glm (logistic regresion)",
      "o rf (randomforest)."
    ))
  }

  for (i in 1:nrow(poblacion)) {
    individuo <- poblacion[i, ]

    if (verbose) {
      print(paste("Individuo", i, ":", paste(individuo, collapse = " ")))
    }

    fitness_individuo <- calcular_fitness_individuo(
      x = x[, individuo, drop = FALSE],
      y = y,
      cv = cv,
      metrica = metrica,
      nivel_referencia = nivel_referencia,
      seed = seed,
      verbose = verbose
    )
```

```
    fitness_poblacion[i] <- fitness_individuo
  }

  if (verbose) {
    print(paste(
      "Fitness calculado para los",
      nrow(poblacion),
      "individuos de la población."
    ))
    print(paste("Modelo empleado para el cálculo del fitness:", modelo))
    cat("\n")
  }
  return(fitness_poblacion)
}
```

Ejemplo

Se calcula el *fitness* de todos los individuos de una población formada por 10 individuos de longitud 8, con un número de valores TRUE acotado entre 1 y 5.

```
# Población simulada
poblacion <- crear_poblacion(n_poblacion = 10,
                             n_variables = 8,
                             n_max = 5,
                             n_min = 1,
                             verbose = TRUE)
```

```
## [1] "Población inicial creada"
## [1] "Número de individuos = 10"
## [1] "Número de predictores mínimo por individuo = 1"
## [1] "Número de predictores máximo por individuo = 5"
##
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
##  [1,] FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
##  [2,] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
##  [3,] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE
##  [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [5,] FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
##  [6,] FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [7,] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
##  [8,]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [9,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [10,] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
```

```
# Datos simulados
predictores   <- matrix(data = rnorm(n = 1000), nrow = 100, ncol = 10)
var_respuesta <- rnorm(n = 100)
# Cálculo del fitness de todos los individuos
fitness_poblacion <- calcular_fitness_poblacion(
                        poblacion = poblacion, x = predictores, y = var_respuesta,
                        modelo = "lm",cv = 5, seed = 123, verbose = TRUE
                       )
```

```
## [1] "Individuo 1 : FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.26398981912453"
##
## [1] "Individuo 2 : FALSE FALSE FALSE TRUE FALSE FALSE TRUE TRUE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.25786253846106"
##
## [1] "Individuo 3 : FALSE TRUE FALSE FALSE FALSE FALSE TRUE TRUE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.29032489766018"
##
## [1] "Individuo 4 : FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.23406328778667"
##
## [1] "Individuo 5 : FALSE TRUE TRUE TRUE FALSE TRUE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.32992782534365"
##
## [1] "Individuo 6 : FALSE TRUE TRUE FALSE FALSE TRUE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.31457138117646"
##
## [1] "Individuo 7 : FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.23757810735827"
##
## [1] "Individuo 8 : TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.21537355550888"
##
## [1] "Individuo 9 : FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.23340115773431"
##
## [1] "Individuo 10 : FALSE FALSE TRUE FALSE TRUE FALSE TRUE TRUE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -1.26828050462259"
##
## [1] "Fitness calculado para los 10 individuos de la población."
## [1] "Modelo empleado para el cálculo del fitness: lm"
```

```
fitness_poblacion
```

```
##  [1] -1.263990 -1.257863 -1.290325 -1.234063 -1.329928 -1.314571 -1.237578
##  [8] -1.215374 -1.233401 -1.268281
```

El vector devuelto contiene el *fitness* de cada uno de los individuos en el mismo orden que se encuentran en la matriz de la población.

Seleccionar individuos

La forma en que se seleccionan los individuos que participan en cada cruce difiere en las distintas implementaciones de los algoritmos genéticos. Por lo general, todas ellas tienden a favorecer la selección de aquellos individuos con mayor *fitness*. Algunas de las estrategias más comunes son:

- Método de ruleta: la probabilidad de que un individuo sea seleccionado es proporcional a su *fitness* relativo, es decir, a su *fitness* dividido por la suma del *fitness* de todos los individuos de la población. Si el *fitness* de un individuo es el doble que el de otro, también lo será la probabilidad de que sea seleccionado. Este método presenta problemas si el *fitness* de unos pocos individuos es muy superior (varios órdenes de magnitud) al resto, ya que estos serán seleccionados de forma repetida y casi todos los individuos de la siguiente generación serán "hijos" de los mismos "padres" (poca variación).

- Método *rank*: la probabilidad de selección de un individuo es inversamente proporcional a la posición que ocupa tras ordenar todos los individuos de mayor a menor *fitness*. Este método es menos agresivo que el método ruleta cuando la diferencia entre los mayores *fitness* es varios órdenes de magnitud superior al resto.

- Selección competitiva (*tournament*): se seleccionan aleatoriamente dos parejas de individuos de la población (todos con la misma probabilidad). De cada pareja se selecciona el que tenga mayor *fitness*. Finalmente, se comparan los dos finalistas y se selecciona el de mayor *fitness*. Este método tiende a generar una distribución de la probabilidad de selección más equilibrada que las dos anteriores.

- Selección truncada (*truncated selection*): se realizan selecciones aleatorias de individuos, habiendo descartado primero los *n* individuos con menor *fitness* de la población.

La siguiente función recibe como argumento un vector con el *fitness* de cada individuo y selecciona una de las posiciones, donde la probabilidad de selección es proporcional al *fitness* relativo.

La conversión de *fitness* a probabilidad es distinta dependiendo de la métrica utilizada para calcular el *fitness*.

- Si el *fitness* es el valor negativo del error (-MSE), cuanto menor sea el *fitness* (menor la magnitud del valor negativo), mayor debe ser la probabilidad de ser seleccionado. Para lograr la conversión se emplea $\frac{-1}{fitness}$.

- Si el *fitness* es el *accuracy*, cuanto mayor sea el *fitness*, mayor debe ser la probabilidad de ser seleccionado.

```r
seleccionar_individuo <- function(vector_fitness, metrica,
                                  metodo_seleccion = "tournament",
                                  verbose = TRUE) {
  # ARGUMENTOS
  # ============================================================================
  # vector_fitness:   un vector con el fitness de cada individuo.
  # metrica:          métrica empleada para calcular el fitness.
  # metodo_seleccion: método para establecer la probabilidad de selección.
  #                   Puede ser "ruleta", "rank", o "tournament".
  #
  # RETORNO
  # ============================================================================
  # El índice que ocupa el individuo seleccionado.

  # Selección del individuo
  if (metodo_seleccion == "ruleta") {
    if (metrica == "mse") {
      probabilidad_seleccion <- (-1 / vector_fitness) / sum(-1 / vector_fitness)
    } else if (metrica == "accuracy") {
      probabilidad_seleccion <- (vector_fitness) / sum(vector_fitness)
    }

    ind_seleccionado <- sample(
      x = 1:length(vector_fitness),
      size = 1,
      prob = probabilidad_seleccion
    )
  } else if (metodo_seleccion == "rank") {
    probabilidad_seleccion <- 1 / rank(-1*vector_fitness)

    ind_seleccionado <- sample(
      x = 1:length(vector_fitness),
      size = 1,
      prob = probabilidad_seleccion
    )
  } else if (metodo_seleccion == "tournament") {
    # Se seleccionan aleatoriamente dos parejas de individuos.
    ind_candidatos_a <- sample(x = 1:length(vector_fitness), size = 2)
    ind_candidatos_b <- sample(x = 1:length(vector_fitness), size = 2)

    # De cada pareja se selecciona el de mayor fitness.
    ind_ganador_a <- ifelse(
      vector_fitness[ind_candidatos_a[1]] > vector_fitness[ind_candidatos_a[2]],
      ind_candidatos_a[1],
      ind_candidatos_a[2]
    )
    ind_ganador_b <- ifelse(
      vector_fitness[ind_candidatos_b[1]] > vector_fitness[ind_candidatos_b[2]],
      ind_candidatos_b[1],
```

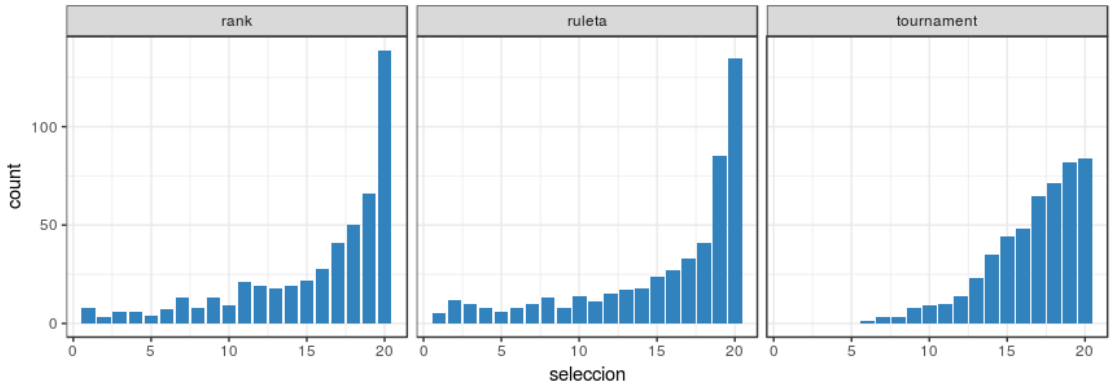
```
      ind_candidatos_b[2]
    )

    # Se comparan los dos ganadores de cada pareja.
    ind_seleccionado <- ifelse(
      vector_fitness[ind_ganador_a] > vector_fitness[ind_ganador_b],
      ind_ganador_a,
      ind_ganador_b
    )
  } else {
    stop("El argumento metodo_seleccion debe ser: ruleta, rank o tournament")
  }

  if (verbose) {
    print(paste("Métrica de selección empleada:", metrica))
    print(paste("Método de selección empleado:", metodo_seleccion))
    print(paste("Individuo seleccionado:", ind_seleccionado))
    cat("\n")
  }
  return(ind_seleccionado)
}
```

Ejemplo

Se compara el patrón de selección del mejor individuo entre los métodos ruleta, *rank* y *tournament*. En primer lugar, se muestra un caso en el que la diferencia entre el mayor y el menor de los *fitness* no es muy acusada, y un segundo caso en el que sí lo es (dos órdenes de magnitud). En este ejemplo, el *fitness* se corresponde con el valor negativo del MSE, por lo que, cuanto más próximo a cero, mayor debe de ser la probabilidad de selección.

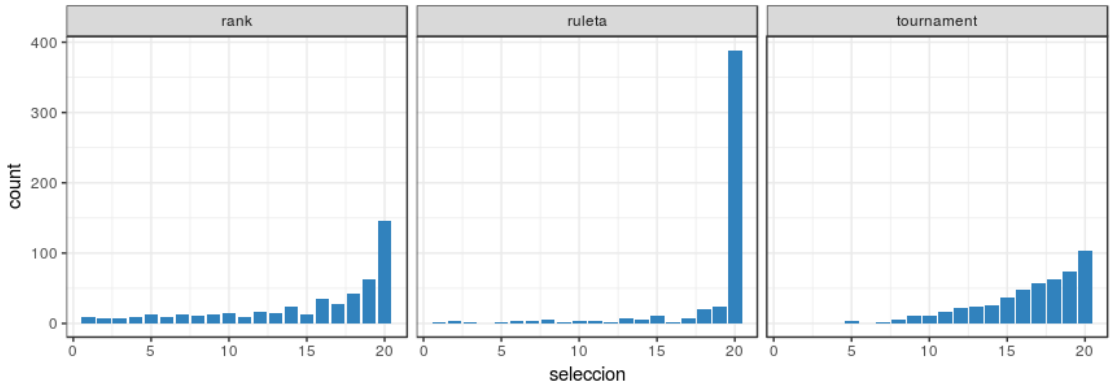
```
library(tidyverse)

fitness_poblacion <- c(-20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9,
                       -8, -7, -6, -5, -4, -3, -2, -1)

selecciones_ruleta <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_ruleta[i] <- seleccionar_individuo(vector_fitness= fitness_poblacion,
                                                 metodo_seleccion = "ruleta",
                                                 metrica = "mse",
                                                 verbose = FALSE)
}
selecciones_ruleta <- data.frame(seleccion = selecciones_ruleta) %>%
                      mutate(metodo_seleccion = "ruleta")
```

```
selecciones_rank <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_rank[i] <- seleccionar_individuo(vector_fitness = fitness_poblacion,
                                               metodo_seleccion = "rank",
                                               metrica = "mse",
                                               verbose = FALSE)
}
selecciones_rank <- data.frame(seleccion = selecciones_rank) %>%
                      mutate(metodo_seleccion = "rank")


selecciones_tournament <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_tournament[i] <- seleccionar_individuo(
                                 vector_fitness= fitness_poblacion,
                                 metodo_seleccion = "tournament",
                                 metrica = "mse",
                                 verbose = FALSE
                                 )
}
selecciones_tournament <- data.frame(seleccion = selecciones_tournament) %>%
                          mutate(metodo_seleccion = "tournament")

bind_rows(selecciones_ruleta, selecciones_rank, selecciones_tournament) %>%
  ggplot(aes(x = seleccion)) +
  geom_bar(fill = "#3182bd") +
  facet_grid(.~metodo_seleccion) +
  theme_bw()
```



Cuando no existe una gran diferencia entre el individuo de mayor *fitness* y el resto, con el método *rank*, el individuo con mayor *fitness* se selecciona con mucha más frecuencia que el resto. Con los otros dos métodos, la probabilidad de selección decae de forma más gradual.

```
fitness_poblacion <- c(-200, -190, -180, -170, -160, -150, -140, -130, -120, -110,
                       -100, -90, -80, -70, -60, -50, -40, -30, -20, -1)
selecciones_ruleta <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_ruleta[i] <- seleccionar_individuo(vector_fitness=fitness_poblacion,
                                                 metodo_seleccion = "ruleta",
                                                 metrica = "mse",
                                                 verbose = FALSE)
}
selecciones_ruleta <- data.frame(seleccion = selecciones_ruleta) %>%
                      mutate(metodo_seleccion = "ruleta")
selecciones_rank <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_rank[i] <- seleccionar_individuo(vector_fitness = fitness_poblacion,
                                               metodo_seleccion = "rank",
                                               metrica = "mse",
                                               verbose = FALSE)
}
selecciones_rank <- data.frame(seleccion = selecciones_rank) %>%
                    mutate(metodo_seleccion = "rank")
selecciones_tournament <- rep(NA, times=500)
for (i in 1:500) {
  selecciones_tournament[i] <- seleccionar_individuo(
                                 vector_fitness = fitness_poblacion,
                                 metodo_seleccion = "tournament",
                                 metrica = "mse",
                                 verbose = FALSE
                               )
}
selecciones_tournament <- data.frame(seleccion = selecciones_tournament) %>%
                          mutate(metodo_seleccion = "tournament")

bind_rows(selecciones_ruleta, selecciones_rank, selecciones_tournament) %>%
  ggplot(aes(x = seleccion)) +
  geom_bar(fill = "#3182bd") +
  facet_grid(.~metodo_seleccion) +
  theme_bw()
```



Cuando existe una gran diferencia entre el individuo de mayor *fitness* y el resto (uno o varios órdenes de magnitud), con el método *ruleta*, el individuo con mayor *fitness* se selecciona con mucha más frecuencia que el resto. A diferencia del caso anterior, en esta situación, la probabilidad de selección decae de forma más gradual con los métodos *rank* y *tournament*.

Teniendo en cuenta los comportamientos de selección de cada método, el método *tournament* parece ser la opción más equilibrada.

Cruzar dos individuos (*crossover*, recombinación)

El objetivo de esta etapa es generar, a partir de individuos ya existentes (parentales), nuevos individuos (descendencia) que combinen las características de los anteriores. Este es otro de los puntos del algoritmo en los que se puede seguir varias estrategias. Tres de las más empleadas son:

- Cruzamiento a partir de uno solo punto: se selecciona aleatoriamente una posición que actúa como punto de corte. Cada individuo parental se divide en dos partes y se intercambian las mitades. Como resultado de este proceso, por cada cruce, se generan dos nuevos individuos.
- Cruzamiento a partir múltiples puntos: se seleccionan aleatoriamente varias posiciones que actúan como puntos de corte. Cada individuo parental se divide por los puntos de corte y se intercambian las partes. Como resultado de este proceso, por cada cruce, se generan dos nuevos individuos.
- Cruzamiento uniforme: el valor que toma cada posición del nuevo individuo se obtiene de uno de los dos parentales. Por lo general, la probabilidad de que el valor proceda de cada parental es la misma, aunque podría, por ejemplo, estar condicionada al *fitness* de cada uno. A diferencia de las anteriores estrategias, con esta, de cada cruce se genera un único descendiente.

```
cruzar_individuos <- function(parental_1, parental_2, metodo_cruce = "uniforme",
                              verbose = TRUE) {
  # Esta función devuelve un individuo resultado de cruzar dos individuos
  # parentales con el método de cruzamiento uniforme.
  #
  # ARGUMENTOS
  # ============================================================================
  # parental_1: vector que representa a un individuo.
  # parental_2: vector que representa a un individuo.
  # metodo_cruce: estrategia de cruzamiento.
```

```
  # RETORNO
  # ============================================================================
  # Un vector que representa a un nuevo individuo.

  # Para crear el nuevo individuo, se emplea el método de cruzamiento uniforme,
  # con la misma probabilidad de que el valor proceda de cada parental.

  if (length(parental_1) != length(parental_2)) {
    stop(paste0(
      "La longitud de los dos vectores que representan a los ",
      "individuos debe ser la misma."
    ))
  }
  if (!(metodo_cruce %in% c("uniforme", "punto_simple"))) {
    stop("El método de cruzamiento debe ser: uniforme o punto_simple.")
  }

  # Se crea el vector que representa el nuevo individuo
  descendencia <- rep(NA, times = length(parental_1))

  if (metodo_cruce == "uniforme") {
    # Se seleccionan aleatoriamente las posiciones que se heredan del parental_1.
    herencia_parent_1 <- sample(
                           x = c(TRUE, FALSE),
                           size = length(parental_1),
                           replace = TRUE
                         )
    # El resto de posiciones se heredan del parental_2.
    herencia_parent_2 <- !herencia_parent_1

    descendencia[herencia_parent_1] <- parental_1[herencia_parent_1]
    descendencia[herencia_parent_2] <- parental_2[herencia_parent_2]
  } else {
    punto_cruce <- sample(
                     x    = 2:length(parental_1),
                     size = 1
                   )
    descendencia <- c(
                     parental_1[1:(punto_cruce-1)],
                     parental_2[punto_cruce: length(parental_1)]
                    )
  }

  if(verbose){
    print(paste("Método de cruzamiento:", metodo_cruce))
    print("--------------------------------")
    print("Parental 1")
    print(parental_1)
    print("Parental 2")
```

```
    print(parental_2)
    print("Descendencia")
    print(descendencia)
    cat("\n")
  }
  return(descendencia)
}
```

Ejemplo

Se obtiene un nuevo individuo a partir del cruce de los individuos `c(T, T, T, T, T)` y `c(F, F, F, F, F)`.

```
cruzar_individuos(parental_1 = c(T, T, T, T, T),
                  parental_2 =  c(F, F, F, F, F),
                  metodo     = "uniforme",
                  verbose    = TRUE)
```

```
## [1] "Método de cruzamiento: uniforme"
## [1] "------------------------------"
## [1] "Parental 1"
## [1] TRUE TRUE TRUE TRUE TRUE
## [1] "Parental 2"
## [1] FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
## [1]  TRUE  TRUE FALSE  TRUE FALSE

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

Mutar individuo

Tras generar cada nuevo individuo de la descendencia, este se somete a un proceso de mutación en el que, cada una de sus posiciones, puede verse modificada con una probabilidad *p*. Este paso es importante para añadir diversidad al proceso y evitar que el algoritmo caiga en mínimos locales por que todos los individuos sean demasiado parecidos de una generación a otra.

```
mutar_individuo <- function(individuo, prob_mut = 0.01) {

  # ARGUMENTOS
  # ============================================================================
  # individuo: vector que representa a un individuo.
  # prob_mut:  probabilidad que tiene cada posición del vector de mutar.

  # RETORNO
  # ============================================================================
  # Un vector que representa al individuo tras someterse a las mutaciones.

  # Selección de posiciones a mutar.
  posiciones_mutadas <- runif(n = length(individuo), min = 0, max = 1) < prob_mut

  # Se modifica el valor de aquellas posiciones que hayan sido seleccionadas para
  # mutar. Si el valor de prob_mut es muy bajo, las mutaciones serán muy poco
  # frecuentes y el individuo devuelto será casi siempre igual al original.
  individuo[posiciones_mutadas] <- !(individuo[posiciones_mutadas])
  return(individuo)
}
```

Ejemplo

Se somete a un individuo al proceso de mutación, con una probabilidad de mutación de 0.2.

```
mutar_individuo(individuo = c(T,T,T,T,T,T,T,T,T,T), prob_mut = 0.2)
```

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

Algoritmo completo

En cada uno de los apartados anteriores se ha definido una de las etapas del algoritmo genético. A continuación, se combinan todas ellas dentro de una única función.

```
selecionar_predictores <- function(x,
                                   y,
                                   n_poblacion = 20,
                                   n_generaciones = 10,
                                   n_max_predictores = NULL,
                                   n_min_predictores = NULL,
                                   modelo = "lm",
                                   cv = 5,
                                   metrica = NULL,
                                   nivel_referencia = NULL,
                                   elitismo = 0.1,
                                   prob_mut = 0.01,
                                   metodo_seleccion = "tournament",
                                   metodo_cruce = "uniforme",
                                   parada_temprana = FALSE,
                                   rondas_parada = NULL,
                                   tolerancia_parada = NULL,
                                   verbose = TRUE,
                                   ...) {
  library(randomForest)
  library(caret)
  library(MLmetrics)



  # COMPROBACIONES INICIALES
  # ============================================================================
  # Comprobación de que la variable respuesta es numérica si el modelo es lm.
  if (!is.numeric(y) & modelo == "lm") {
    stop(paste(
      "El modelo lm solo puede aplicarse a problemas de regresión,",
      "(variable respuesta numérica)."
    ))
  }

  # El número máximo de predictores no puede superar el número de columnas de x.
  if (n_max_predictores > ncol(x)) {
    stop(paste(
      "El número máximo de predictores no puede superar al número de",
      "variables disponibles en x."
    ))
  }
```

```
# Si se activa la parada temprana, hay que especificar los argumentos
# rondas_parada y tolerancia_parada.
if(isTRUE(parada_temprana)&(is.null(rondas_parada)|is.null(tolerancia_parada))) {
  stop(paste(
    "Para activar la parada temprana es necesario indicar un valor",
    "de rondas_parada y de tolerancia_parada."
  ))
}

# ALMACENAMIENTO DE RESULTADOS
# ==============================================================================
# Por cada generación se almacena el mejor individuo, su fitness, y el porcentaje
# de mejora respecto a la última generación.
resultados_fitness   <- vector(mode = "list", length = n_generaciones)
resultados_individuo <- vector(mode = "list", length = n_generaciones)
porcentaje_mejora    <- vector(mode = "list", length = n_generaciones)

# CREACIÓN DE LA POBLACIÓN INICIAL
# ==============================================================================
poblacion <- crear_poblacion(
  n_poblacion = n_poblacion,
  n_variables = ncol(x),
  n_max = n_max_predictores,
  n_min = n_min_predictores,
  verbose = verbose
)
# ITERACIÓN DE POBLACIONES
# ==============================================================================
for (i in 1:n_generaciones) {
  if (verbose) {
    print("----------------------")
    print(paste("Generación:", i))
    print("----------------------")
  }

  # CALCULAR FITNESS DE LOS INDIVIDUOS DE LA POBLACIÓN
  # ============================================================================
    fitness_ind_poblacion <- calcular_fitness_poblacion(
      poblacion = poblacion,
      x = x,
      y = y,
      modelo = modelo,
      cv = cv,
      metrica = metrica,
      nivel_referencia = nivel_referencia,
      verbose = verbose
    )
```

```
    # SE ALMACENA EL MEJOR INDIVIDUO DE LA POBLACIÓN ACTUAL
    # ============================================================================
    fitness_mejor_individuo   <- max(fitness_ind_poblacion)
    mejor_individuo           <- poblacion[which.max(fitness_ind_poblacion), ]
    resultados_fitness[[i]]   <- fitness_mejor_individuo
    resultados_individuo[[i]] <- colnames(x)[mejor_individuo]

    # SE CALCULA LA MEJORA RESPECTO A LA GENERACIÓN ANTERIOR
    # ============================================================================
    # El porcentaje de mejora solo puede calcularse a partir de la segunda
    # generación.
    if (i > 1) {
      porcentaje_mejora[[i]] <- 1 - (resultados_fitness[[i]] /
                                     resultados_fitness[[i - 1]])
    }

    # NUEVA POBLACIÓN
    # ============================================================================
    nueva_poblacion <- matrix(
      data = NA,
      nrow = nrow(poblacion),
      ncol = ncol(poblacion)
    )

    # ELITISMO
    # ============================================================================
    # El elitismo indica el porcentaje de mejores individuos de la población
    # actual que pasan directamente a la siguiente población. De esta forma, se
    # asegura que, la siguiente generación, no sea nunca inferior.
    if (elitismo > 0) {
      n_elitismo <- ceiling(nrow(poblacion) * elitismo)
      posicion_n_mejores <- order(fitness_ind_poblacion, decreasing = TRUE)
      posicion_n_mejores <- posicion_n_mejores[1:n_elitismo]
      nueva_poblacion[1:n_elitismo, ] <- poblacion[posicion_n_mejores, ]
    } else {
      n_elitismo <- 0
    }

    # CREACIÓN DE NUEVOS INDIVIDUOS POR CRUCES
    # ============================================================================
    for (j in (n_elitismo + 1):nrow(nueva_poblacion)) {
      # Seleccionar parentales
      metrica <- ifelse(test = is.numeric(y), "mse", "accuracy")
      indice_parental_1 <- seleccionar_individuo(
        vector_fitness = fitness_ind_poblacion,
        metrica = metrica,
        metodo_seleccion = metodo_seleccion,
        verbose = verbose
      )
```

```
      indice_parental_2 <- seleccionar_individuo(
        vector_fitness = fitness_ind_poblacion,
        metrica = metrica,
        metodo_seleccion = metodo_seleccion,
        verbose = verbose
      )
      parental_1 <- poblacion[indice_parental_1, ]
      parental_2 <- poblacion[indice_parental_2, ]

      # Cruzar parentales para obtener la descendencia
      descendencia <- cruzar_individuos(
        parental_1 = parental_1,
        parental_2 = parental_2,
        metodo_cruce = metodo_cruce,
        verbose = verbose
      )
      # Mutar la descendencia
      descendencia <- mutar_individuo(
        individuo = descendencia,
        prob_mut = prob_mut
      )
      # Almacenar el nuevo descendiente en la nueva población: puede ocurrir que
      # el individuo resultante del cruce y posterior mutación no contenga ningún
      # predictor (todas sus posiciones son FALSE). Si esto ocurre, y para evitar
      # que se produzca un error al ajustar el modelo, se introduce aleatoriamente
      # un valor TRUE.
      if (all(descendencia == FALSE)) {
        descendencia[sample(x = 1:length(descendencia), size = 1)] <- TRUE
      }
      nueva_poblacion[j, ] <- descendencia
    }
    poblacion <- nueva_poblacion

    # CRITERIO DE PARADA
    # ==============================================================================
    # Si durante las últimas n generaciones el mejor individuo no ha mejorado por
    # una cantidad superior a tolerancia_parada, se detiene el algoritmo y no se
    # crean nuevas generaciones.

    if (parada_temprana && (i > rondas_parada)) {
      ultimos_n <- tail(unlist(porcentaje_mejora), n = rondas_parada)
      if (all(ultimos_n < tolerancia_parada)) {
        print(paste(
          "Algoritmo detenido en la generacion", i,
          "por falta de mejora mínima de", tolerancia_parada,
          "durante", rondas_parada,
          "generaciones consecutivas."
        ))
        break()
```

```
      }
    }
  }

  # IDENTIFICACIÓN DEL MEJOR INDIVIDUO DE TODO EL PROCESO
  # ==============================================================================
  mejor_individuo_global<-resultados_individuo[[which.max(unlist(resultados_fitness))]]

  # RESULTADOS
  # ==============================================================================
  # La función devuelve una lista con 4 elementos:
  #   fitness:           una lista con el fitness del mejor individuo de cada
  #                      generación.
  #   mejor_individuo:   una lista con la combinación de predictores  del mejor
  #                      individuo de cada generación.
  #   porcentaje_mejora: una lista con el porcentaje de mejora entre el mejor
  #                      individuo de cada generación.
  #   df_resultados:     un dataframe con todos los resultados anteriores.
  #   mejor_individuo_global: la combinación de predictores  del mejor individuo
  #                           encontrado en todo el proceso.

  # Para crear el dataframe se convierten las listas a vectores del mismo tamaño.
  fitness <- unlist(resultados_fitness)
  predictores <- resultados_individuo[!sapply(resultados_individuo, is.null)]
  predictores <- sapply(predictores, function(x) {
                        paste(x, collapse = ", ")
                      })
  mejora <- c(NA, unlist(porcentaje_mejora))

  df_resultados <- data.frame(
    generacion = seq_along(fitness),
    fitness = fitness,
    predictores = predictores,
    mejora = mejora
  )

  return(list(
    fitness = resultados_fitness,
    mejor_individuo = resultados_individuo,
    porcentaje_mejora = porcentaje_mejora,
    df_resultados = df_resultados,
    mejor_individuo_global = mejor_individuo_global
  ))
}
```

Ejemplo regresión

Simulación de datos

En el paquete `mlbench` puede encontrarse, entre muchos otros data sets, el problema de regresión propuesto por Friedman (1991) y Breiman (1996). Con la función `mlbench.friedman1()` se puede generar un conjunto de datos simulados que siguen la ecuación:

$$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x5 + \epsilon$$

Además de las primeras 5 columnas $(x_1, \ldots, x_5)$, que están relacionadas con la variable respuesta, se añaden automáticamente 5 columnas adicionales $(x_6, \ldots, x_{10})$ que siguen una distribución uniforme [0,1] y que, por lo tanto, no guardan relación alguna con la variable respuesta.

```
library(mlbench)
library(dplyr)
set.seed(123)
simulacion <- mlbench.friedman1(n = 500, sd = 1)
# El objeto simulación es una lista que contiene una matriz con los 10 predictores
# y un vector con la variable respuesta. Se unen todos los datos en un único
# dataframe.
datos <- as.data.frame(simulacion$x)
datos$y <- simulacion$y
colnames(datos) <- c(paste("x", 1:10, sep = ""), "y")
datos <- datos %>% select(y, everything())
head(datos)
```

```
##           y        x1         x2        x3        x4        x5        x6
## 1 13.665374 0.2875775 0.35360608 0.2736227 0.9380283 0.1596740 0.8535317
## 2 19.232842 0.7883051 0.36644144 0.5938669 0.9880033 0.1445159 0.6660066
## 3 10.519756 0.4089769 0.28710013 0.1601848 0.4563196 0.1491804 0.7333267
## 4  9.042955 0.8830174 0.07997291 0.8534302 0.2306149 0.5144343 0.3147403
## 5 21.430304 0.9404673 0.36545427 0.8477392 0.6954893 0.4928273 0.6677179
## 6  8.436942 0.0455565 0.17801381 0.4778868 0.5566323 0.6163428 0.4638867
##          x7        x8        x9        x10
## 1 0.2058269 0.2414939 0.3044642 0.10504174
## 2 0.9425390 0.4107782 0.8328188 0.40250560
## 3 0.3793238 0.8111750 0.5936475 0.45567659
## 4 0.6262401 0.3791412 0.8071966 0.62464525
## 5 0.1835024 0.4469502 0.2940508 0.07885528
## 6 0.6592076 0.5705078 0.1410852 0.34198272
```

Se añaden además 20 columnas adicionales con valores aleatorios distribuidos de forma normal.

```
n <- 500
p <- 20
ruido <- matrix(rnorm(n * p), nrow = n)
colnames(ruido) <- paste("x", 11:(10+p), sep = "")
ruido <- as.data.frame(ruido)
datos <- bind_cols(datos, ruido)
head(datos)
```

```
##           y        x1         x2        x3        x4        x5        x6
## 1 13.665374 0.2875775 0.35360608 0.2736227 0.9380283 0.1596740 0.8535317
## 2 19.232842 0.7883051 0.36644144 0.5938669 0.9880033 0.1445159 0.6660066
## 3 10.519756 0.4089769 0.28710013 0.1601848 0.4563196 0.1491804 0.7333267
## 4  9.042955 0.8830174 0.07997291 0.8534302 0.2306149 0.5144343 0.3147403
## 5 21.430304 0.9404673 0.36545427 0.8477392 0.6954893 0.4928273 0.6677179
## 6  8.436942 0.0455565 0.17801381 0.4778868 0.5566323 0.6163428 0.4638867
##           x7        x8        x9        x10         x11        x12
## 1 0.2058269 0.2414939 0.3044642 0.10504174 -0.15030748  1.4783345
## 2 0.9425390 0.4107782 0.8328188 0.40250560 -0.32775713 -1.4067867
## 3 0.3793238 0.8111750 0.5936475 0.45567659 -1.44816529 -1.8839721
## 4 0.6262401 0.3791412 0.8071966 0.62464525 -0.69728458 -0.2773662
## 5 0.1835024 0.4469502 0.2940508 0.07885528  2.59849023  0.4304278
## 6 0.6592076 0.5705078 0.1410852 0.34198272 -0.03741501 -0.1287867
##           x13        x14        x15        x16        x17        x18
## 1  0.1965498  0.8343715 -0.4941739 -1.3572306 -0.6992281  0.5844478
## 2  0.6501132 -0.6984039  1.1275935 -1.2926978  0.9964515 -0.1975278
## 3  0.6710042  1.3092405 -1.1469495 -1.5172073 -0.6927454  1.7974935
## 4 -1.2841578 -0.9801776  1.4810186  0.8591760 -0.1034830  1.5628134
## 5 -2.0261096  0.7479851  0.9161912 -1.2146175  0.6038661 -0.7512625
## 6  2.2053261  1.2577966  0.3351310  0.6190554 -0.6080450  1.3784108
##           x19        x20        x21          x22        x23        x24
## 1 -1.6180367  0.3500025  0.5110004  0.272647072  1.9315759 -0.3378621
## 2  0.3791812  0.8144417  1.8079928  0.041452123 -0.6164747 -0.1471321
## 3  1.9022505 -0.5166661 -1.7026150 -0.004881852 -0.5625675 -0.6736354
## 4  0.6018743 -2.6922644  0.2874488 -0.976219229 -0.9899631 -0.5686278
## 5  1.7323497 -1.0969546 -0.2691142  0.144050084  2.7312276  0.4708565
## 6 -0.1468696 -1.2554751 -0.3795247  0.456619219 -0.7216662  1.3013367
##           x25        x26        x27        x28        x29         x30
## 1  2.3707252  0.9863167  0.2677904 -0.5582810 -0.1506300  0.01014789
## 2 -0.1668120 -0.6629674 -0.3993609  0.1254263  0.8009406  1.56213812
## 3  0.9269614  0.5767211  0.2768838  1.8151103 -1.1867178  0.41284605
## 4 -0.5681517 -0.1378378 -1.2336734  0.1630368  0.4306364 -1.18886219
## 5  0.2250901 -0.1681767  0.7535749 -0.7234664  0.2167471  0.71454993
## 6  1.1319859  0.5761930  0.2892610  0.5809549  0.9612923  0.58507617
```

Como resultado se ha generado un conjunto de datos con 30 predictores, de los cuales, solo 5 están realmente relacionados con la variable respuesta.

Selección de predictores

En primer lugar se emplean pocos individuos y generaciones para mostrar la información que se imprime por pantalla cuando `verbose = TRUE`.

```
seleccion <- selecionar_predictores(x = datos[, -1],
                                    y = datos$y,
                                    n_poblacion = 5,
                                    n_generaciones = 2,
                                    n_max_predictores = 5,
                                    n_min_predictores = 1,
                                    elitismo = 0.01,
                                    prob_mut = 0.1,
                                    metodo_seleccion = "tournament",
                                    metodo_cruce = "uniforme",
                                    modelo = "rf",
                                    cv = 5,
                                    parada_temprana = TRUE,
                                    rondas_parada = 3,
                                    tolerancia_parada = 0.01,
                                    verbose = TRUE
                                    )
```

```
## [1] "Población inicial creada"
## [1] "Número de individuos = 5"
## [1] "Número de predictores mínimo por individuo = 1"
## [1] "Número de predictores máximo por individuo = 5"
##
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11]
## [1,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [5,]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##      [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30]
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [3,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "----------------------"
## [1] "Generación: 1"
## [1] "----------------------"
```

```
## [1] "Individuo 1 : FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -32.3302686058316"
##
## [1] "Individuo 2 : FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE TRUE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -31.9290151449685"
##
## [1] "Individuo 3 : FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
TRUE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -25.0581343360507"
##
## [1] "Individuo 4 : FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE TRUE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -34.930121392734"
##
## [1] "Individuo 5 : TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -24.2684135976247"
##
## [1] "Fitness calculado para los 5 individuos de la población."
## [1] "Modelo empleado para el cálculo del fitness: rf"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
```

```
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 3"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
## [23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "---------------------"
## [1] "Generación: 2"
## [1] "---------------------"
## [1] "Individuo 1 : TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -24.2684135976247"
##
## [1] "Individuo 2 : TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE FALSE TRUE FALSE FALSE FALSE
FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -19.7634802955449"
##
## [1] "Individuo 3 : TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -20.5924915106893"
##
## [1] "Individuo 4 : TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -19.9745884923466"
##
## [1] "Individuo 5 : TRUE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE
FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE
FALSE FALSE FALSE"
## [1] "El fitness calculado por CV = 5 empleando el -(mse) es de: -17.7222521195156"
##
## [1] "Fitness calculado para los 5 individuos de la población."
## [1] "Modelo empleado para el cálculo del fitness: rf"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 2"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [1] "Parental 2"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 2"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 5"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "--------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##
```

```
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 2"
##
## [1] "Métrica de selección empleada: mse"
## [1] "Método de selección empleado: tournament"
## [1] "Individuo seleccionado: 4"
##
## [1] "Método de cruzamiento: uniforme"
## [1] "------------------------------"
## [1] "Parental 1"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Parental 2"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [1] "Descendencia"
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# Mejor fitness de cada generación.
unlist(seleccion$fitness)
```

```
## [1] -24.26841 -17.72225
```

```
# Mejor individuo de cada generación.
seleccion$mejor_individuo
```

```
## [[1]]
## [1] "x1"
##
## [[2]]
## [1] "x1"  "x3"  "x7"  "x11" "x15" "x19" "x24"
```

```
# Porcentaje de mejora entre generaciones.
unlist(seleccion$porcentaje_mejora)
```

```
## [1] 0.26974
```

```
# Resultados en forma de dataframe
seleccion$df_resultados
```

```
##   generacion   fitness                    predictores  mejora
## 1          1 -24.26841                             x1      NA
## 2          2 -17.72225 x1, x3, x7, x11, x15, x19, x24 0.26974
```

Se repite el proceso pero, esta vez, con 50 individuos por generación y 20 generaciones. Se especifica que `verbose = FALSE` para no saturar la consola.

```
seleccion <- selecionar_predictores(x = datos[, -1],
                                    y = datos$y,
                                    n_poblacion = 50,
                                    n_generaciones = 20,
                                    n_max_predictores = 10,
                                    n_min_predictores = 1,
                                    elitismo = 0.01,
                                    prob_mut = 0.1,
                                    metodo_seleccion = "tournament",
                                    metodo_cruce = "uniforme",
                                    modelo = "rf",
                                    cv = 5,
                                    parada_temprana = TRUE,
                                    rondas_parada = 5,
                                    tolerancia_parada = 0.01,
                                    verbose = FALSE
                                    )
```

```
## [1] "Algoritmo detenido en la generacion 14 por falta de mejora mínima de 0.01
durante 5 generaciones consecutivas."
```

Mejor individuo

El objeto devuelto por la función seleccionar_predictores almacena el mejor individuo de cada generación, la evolución del *fitness* a lo largo de las generaciones y el mejor individuo encontrado a lo largo de todo el proceso.

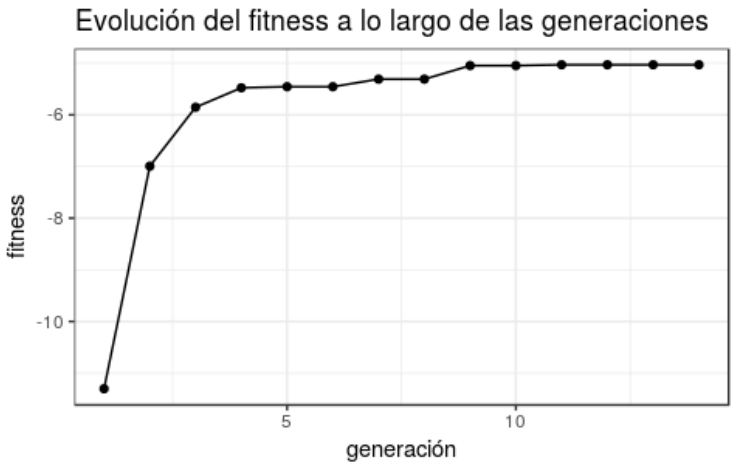
```
seleccion$mejor_individuo_global
```

```
## [1] "x1"  "x2"  "x3"  "x4"  "x5"  "x7"  "x22"
```

Evolución del error

En el siguiente gráfico se puede ver cómo evoluciona el *fitness* del mejor individuo a medida que avanzan las generaciones.

```
library(ggplot2)
ggplot(data = seleccion$df_resultados,
       aes(x = generacion, y = fitness)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  labs(title = "Evolución del fitness a lo largo de las generaciones",
       x = "generación") + theme_bw()
```



Frecuencia selección variables

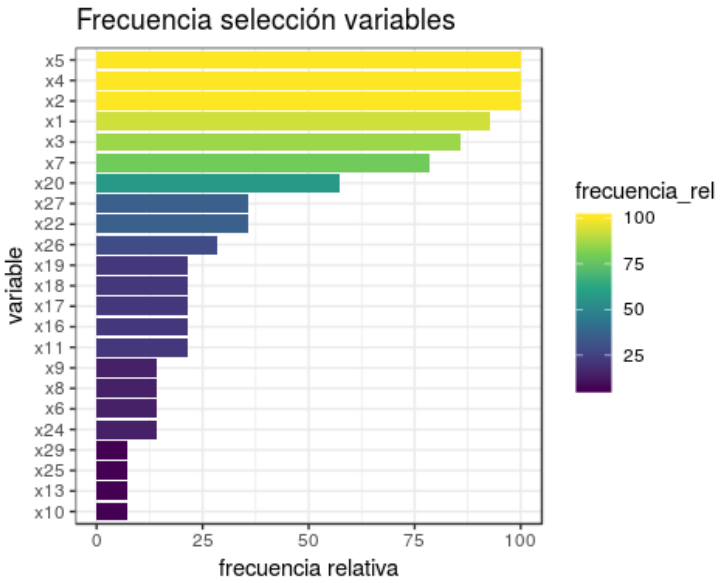
Tal y como ocurre en este caso (sobre todo si se emplea *ranfom forest* con muchos árboles), el mejor individuo puede incorporar algunos predictores no relevantes o dejarse fuera alguno de los importantes. Una forma de ser críticos con el resultado obtenido al aplicar el algoritmo genético, es seguir la siguiente idea: si realmente existen variables útiles, es de esperar que estas tiendan a aparecer en el mejor individuo de cada generación, mientras que, las variables no relevantes, cambiarán con más frecuencia.

Como se han almacenado las variables que incluye el mejor individuo de cada generación, se puede calcular la frecuencia con la que ha aparecido cada una.

```
frecuencia_abs <- table(unlist(seleccion$mejor_individuo))
n_generaciones <- sum(!sapply(seleccion$mejor_individuo, is.null))
frecuencia_rel <- (frecuencia_abs / n_generaciones) * 100

as.data.frame(frecuencia_rel) %>%
  rename(variable = Var1, frecuencia_rel = Freq) %>%
  arrange(desc(frecuencia_rel))
```

```
##    variable frecuencia_rel
## 1        x2     100.000000
## 2        x4     100.000000
## 3        x5     100.000000
## 4        x1      92.857143
## 5        x3      85.714286
## 6        x7      78.571429
## 7       x20      57.142857
## 8       x22      35.714286
## 9       x27      35.714286
## 10      x26      28.571429
## 11      x11      21.428571
## 12      x16      21.428571
## 13      x17      21.428571
## 14      x18      21.428571
## 15      x19      21.428571
## 16      x24      14.285714
## 17       x6      14.285714
## 18       x8      14.285714
## 19       x9      14.285714
## 20      x10       7.142857
## 21      x13       7.142857
## 22      x25       7.142857
## 23      x29       7.142857
```

```
as.data.frame(frecuencia_rel) %>%
  rename(variable = Var1, frecuencia_rel = Freq) %>%
  arrange(desc(frecuencia_rel)) %>%
  ggplot(aes(x = reorder(variable, frecuencia_rel), y = frecuencia_rel,
         fill = frecuencia_rel)) +
    geom_col() +
    scale_fill_viridis_c() +
    labs(title = "Frecuencia selección variables",
         x = "variable",
         y = "frecuencia relativa") +
    coord_flip() +
    theme_bw()
```



Una forma de facilitar la valoración de estas frecuencias es compararlas frente a la frecuencia esperada solo por azar. La probabilidad de que una variable i perteneciente un conjunto de variables de tamaño M esté incluida en un subconjunto de tamaño N obtenido por muestreo aleatorio sin reposición es:

$$\mathrm{P(i\ en\ N)} = N/M$$

Como el tamaño de cada individuo varía, hay que tenerlo en cuenta en el cálculo. Supóngase 5 individuos de tamaños $K = \{5,5,3,2,1\}$ y que el total de variables disponibles es M. El número de individuos en los que se espera que una determinada variable haya sido seleccionada por azar es:

$$\sum_{i \in K} \frac{i}{M}$$

La siguiente función realiza el cálculo de la frecuencia esperada.

```
calcular_frecuencia_esperada <- function(n_variables, individuos){
  # Argumentos:
  #   n_variables: número total de variables disponibles.
  #   individuos:  lista con el mejor individuo de cada generación.

  # Retorno:
  # Porcentaje de individuos en los que se espera que una variable aparezca
  # solo por azar.

  n_individuos <- sum(!sapply(X = individuos, FUN = is.null))
  tamayos      <- sapply(X = individuos, FUN = length)
  tamayos      <- tamayos[tamayos != 0]
```

```
  frecuencia_esperada <- 0
  for (i in tamayos) {
    frecuencia_esperada <- frecuencia_esperada + i/n_variables
  }
  return(frecuencia_esperada/n_individuos)
}

calcular_frecuencia_esperada(n_variables = ncol(datos)-1,
                             individuos  = seleccion$mejor_individuo)
```

```
## [1] 0.302381
```

Puede observarse que, las variables que aparecen con más frecuencia en los mejores individuos de cada generación, son las que están realmente relacionadas con la variable respuesta. Además, todas ellas superan la frecuencia esperada por azar. Combinando el resultado obtenido en el individuo final del algoritmo genético con la información de las frecuencias, se pueden obtener buenos resultados de selección.

Si se utiliza elitismo, el mejor individuo de una generación pasa por las siguientes n generaciones de forma directa. De ser así, las variables no relevantes que arrastre este individuo aparecerán con una frecuencia superior a la esperada por azar aun cuando no son predictores útiles.

Ejemplo clasificación

Datos

El set de datos `titanic_train` del paquete titanic contiene información sobre los pasajeros del transatlántico titanic así como si sobrevivieron o no al naufragio. Se pretende identificar las variables con las que mejor se puede predecir si un pasajero vivió o falleció.

```
library(titanic)

# Se eliminan para este ejemplo 3 variables.
datos <- titanic_train %>% select(-Name, -Ticket, -Cabin)

# Se eliminan observaciones incompletas.
datos <- na.omit(datos)

# Se convierten a factor las variables de tipo character.
datos <- datos %>% mutate_if(is.character, as.factor)
datos <- datos %>% mutate(Survived = as.factor(Survived))
```

Selección de predictores

```
seleccion <- selecionar_predictores(x = datos[, -2],
                                    y = datos[, 2],
                                    n_poblacion = 50,
                                    n_generaciones = 20,
                                    n_max_predictores = 8,
                                    n_min_predictores = 1,
                                    elitismo = 0.01,
                                    metodo_seleccion = "tournament",
                                    metodo_cruce = "uniforme",
                                    modelo = "rf",
                                    metrica = "f1",
                                    nivel_referencia = "1",
                                    cv = 5,
                                    parada_temprana = TRUE,
                                    rondas_parada = 5,
                                    tolerancia_parada = 0.01,
                                    verbose = FALSE
                                    )
```

```
## [1] "Algoritmo detenido en la generacion 6 por falta de mejora mínima de 0.01
durante 5 generaciones consecutivas."
```

```
seleccion$df_resultados
```

```
##   generacion   fitness                                  predictores
## 1          1 0.7661365 PassengerId, Pclass, Sex, Age, SibSp, Fare
## 2          2 0.8249483       Pclass, Sex, Age, SibSp, Parch, Fare
## 3          3 0.8361470              Pclass, Sex, Age, SibSp, Fare
## 4          4 0.8361470              Pclass, Sex, Age, SibSp, Fare
## 5          5 0.8361470              Pclass, Sex, Age, SibSp, Fare
## 6          6 0.8361470              Pclass, Sex, Age, SibSp, Fare
##        mejora
## 1          NA
## 2 -0.07676418
## 3 -0.01357498
## 4  0.00000000
## 5  0.00000000
## 6  0.00000000
```

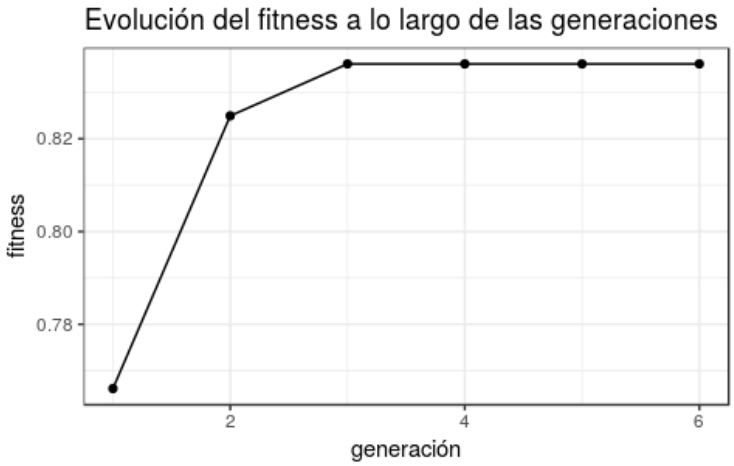
Mejor individuo

```
seleccion$mejor_individuo_global
```

```
## [1] "Pclass" "Sex"    "Age"    "SibSp"  "Fare"
```

Evolución del error

```
library(ggplot2)
ggplot(data = seleccion$df_resultados,
       aes(x = generacion, y = fitness)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  labs(x = "generación",
       title = "Evolución del fitness a lo largo de las generaciones") +
  theme_bw()
```

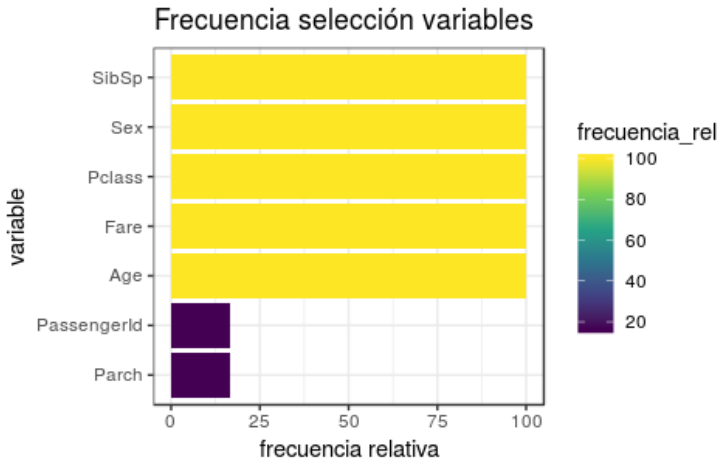


Frecuencia selección variables

```
frecuencia_abs <- table(unlist(seleccion$mejor_individuo))
n_generaciones <- sum(!sapply(seleccion$mejor_individuo, is.null))
frecuencia_rel <- (frecuencia_abs / n_generaciones) * 100

as.data.frame(frecuencia_rel) %>%
  rename(variable = Var1, frecuencia_rel = Freq) %>%
  arrange(desc(frecuencia_rel))
```

```
##      variable frecuencia_rel
## 1         Age      100.00000
## 2        Fare      100.00000
## 3      Pclass      100.00000
## 4         Sex      100.00000
## 5       SibSp      100.00000
## 6       Parch       16.66667
## 7 PassengerId       16.66667
```

```
as.data.frame(frecuencia_rel) %>%
  rename(variable = Var1, frecuencia_rel = Freq) %>%
  arrange(desc(frecuencia_rel)) %>%
  ggplot(aes(x = reorder(variable, frecuencia_rel), y = frecuencia_rel,
         fill = frecuencia_rel)) +
    geom_col() +
    scale_fill_viridis_c() +
    labs(title = "Frecuencia selección variables",
         x = "variable",
         y = "frecuencia relativa") +
    coord_flip() +
    theme_bw()
```



Frecuencia esperada por azar:

```
calcular_frecuencia_esperada(n_variables = ncol(datos)-1,
                             individuos  = seleccion$mejor_individuo)
```

```
## [1] 0.6666667
```

Paralelización

Uno de los inconvenientes de los algoritmos genéticos es su alto requerimiento computacional. Por ejemplo, si se establecen 20 generaciones con 50 individuos por generación y una estimación del *fitness* mediante validación cruzada de 5 particiones, en total, se tienen que ajustar $50x20x5 = 5000$ modelos.

Dos de las estrategias que se pueden emplear para agilizar el proceso son:

- Parada temprana: detener el algoritmo si tras n generaciones consecutivas no se ha conseguido mejora. Esta estrategia está implementada en los ejemplos anteriores.
- Paralelización: ajustar varios modelos de forma simultánea empleando múltiples cores del ordenador.

El algoritmo genético propuesto en este documento tiene dos etapas que pueden ser paralelizadas: la evaluación de los individuos de una población (`calcular_fitness_poblacion()`) y la validación cruzada de cada individuo (`calcular_fitness_individuo()`). Se procede a paralelizar la primera y comparar el impacto en el tiempo de ejecución.

Versión paralelizada

ESTA IMPLEMENTACIÓN NO FUNCIONA EN WINDOWS

```
# Paquetes necesarios para paralelizar.
library(foreach)
library(doParallel)

calcular_fitness_poblacion_paral <- function(poblacion, x, y, cv, seed = 123,
                                             modelo = "rf", metrica = NULL,
                                             nivel_referencia = NULL,
                                             verbose = TRUE) {
  # ARGUMENTOS
  # ============================================================================
  # poblacion: matriz que representa la población de individuos.
  # x:         matriz de predictores.
  # y:         variable respuesta.
  # cv:        número de particiones de validación cruzada.
  # seed:      semilla para garantizar reproducibilidad en el proceso de CV.
  # verbose:   mostrar información del proceso por pantalla.
  # modelo:    tipo de modelo empleado para calcular el fitness. Puede ser
```

```
#            lm, glm o rf.
# metrica:   métrica utilizad apara calcular el fitness. Por defecto es el
#            -MSE para regresión y Accuracy para clasificación. En el caso de
#            clasificación binaria, se acepta también f1.
# nivel_referencia: valor de la variable respuesta considerado como referencia.
#                   Necesario cuando la métrica es f1 o kappa.

# RETORNO
# ============================================================================:
#  Vector con el fitness de todos los individuos de la población, obtenido por
#  validación cruzada. El orden de los valores se corresponde con el orden de
#  las filas de la matriz población.

# Tipo de modelo utilizado para calcular el fitness.
if (modelo == "lm") {
  calcular_fitness_individuo <- calcular_fitness_individuo_lm
} else if (modelo == "rf") {
  calcular_fitness_individuo <- calcular_fitness_individuo_rf
} else if (modelo == "glm") {
  calcular_fitness_individuo <- calcular_fitness_individuo_glm
} else {
  stop(paste(
    "El modelo empleado para calcular el fitness debe ser",
    "lm (linear model)",
    "glm (logistic regresion)",
    "o rf (randomforest)."
  ))
}

# Se emplean todos los cores del ordenador menos 1.
registerDoParallel(parallel::detectCores() - 1)
# Se emplea la función foreach para paralelizar.
fitness_poblacion <- foreach::foreach(i = 1:nrow(poblacion)) %dopar% {
  individuo <- poblacion[i, ]
  if (verbose) {
    print(paste("Individuo", i, ":", paste(individuo, collapse = " ")))
  }

  fitness_individuo <- calcular_fitness_individuo(
    x = x[, individuo, drop = FALSE],
    y = y,
    cv = cv,
    seed = seed,
    metrica = metrica,
    nivel_referencia = nivel_referencia,
    verbose = verbose
  )
  fitness_individuo
}
```

```
  if (verbose) {
    print(paste(
      "Fitness calculado para los",
      nrow(poblacion),
      "individuos de la población."
    ))
    print(paste("Modelo empleado para el cálculo del fitness:", modelo))
    cat("\n")
  }

  # Se vuelve a un único core.
  options(cores = 1)

  return(unlist(fitness_poblacion))
}
```

Para incluir la opción de paralelizado, se repite la función del algoritmo completo, esta vez, incluyendo el argumento paralelizado con el que el usuario pueda especificar que se emplee la función `calcular_fitness_poblacion` o `calcular_fitness_poblacion_paral`.

```
selecionar_predictores <- function(x,
                                   y,
                                   n_poblacion = 20,
                                   n_generaciones = 10,
                                   n_max_predictores = NULL,
                                   n_min_predictores = NULL,
                                   modelo = "lm",
                                   cv = 5,
                                   metrica = NULL,
                                   nivel_referencia = NULL,
                                   elitismo = 0.1,
                                   prob_mut = 0.01,
                                   metodo_seleccion = "tournament",
                                   metodo_cruce = "uniforme",
                                   parada_temprana = FALSE,
                                   rondas_parada = NULL,
                                   tolerancia_parada = NULL,
                                   paralelizado = FALSE,
                                   verbose = TRUE,
                                   ...) {


  library(foreach)
  library(doParallel)
  library(randomForest)
  library(caret)
  library(MLmetrics)
```

```
  # COMPROBACIONES INICIALES
  # ============================================================================
  # Comprobación de que la variable respuesta es numérica si el modelo es lm.
  if (!is.numeric(y) & modelo == "lm") {
    stop(paste(
      "El modelo lm solo puede aplicarse a problemas de regresión,",
      "(variable respuesta numérica)."
    ))
  }

  if (is.numeric(y) & modelo == "glm") {
    stop("El modelo glm solo puede emplearse para clasificación binaria.")
  }

  if ((length(levels(y)) != 2) & modelo == "glm") {
    stop("El modelo glm solo puede emplearse para clasificación binaria.")
  }

  # El número máximo de predictores no puede superar el número de columnas de x.
  if (n_max_predictores > ncol(x)) {
    stop(paste(
      "El número máximo de predictores no puede superar al número de",
      "variables disponibles en x."
    ))
  }

  # Si se activa la parada temprana, hay que especificar los argumentos
  # rondas_parada y tolerancia_parada.
  if (isTRUE(parada_temprana) & (is.null(rondas_parada) |
is.null(tolerancia_parada))) {
    stop(paste(
      "Para activar la parada temprana es necesario indicar un valor",
      "de rondas_parada y de tolerancia_parada."
    ))
  }

  # ALMACENAMIENTO DE RESULTADOS
  # ============================================================================
  # Por cada generación se almacena el mejor individuo, su fitness, y el porcentaje
  # de mejora respecto a la última generación.
  resultados_fitness   <- vector(mode = "list", length = n_generaciones)
  resultados_individuo <- vector(mode = "list", length = n_generaciones)
  porcentaje_mejora    <- vector(mode = "list", length = n_generaciones)

  # CREACIÓN DE LA POBLACIÓN INICIAL
  # ============================================================================
  poblacion <- crear_poblacion(
    n_poblacion = n_poblacion,
    n_variables = ncol(x),
```

```
  n_max = n_max_predictores,
  n_min = n_min_predictores,
  verbose = verbose
)
# ITERACIÓN DE POBLACIONES
# ==============================================================================
for (i in 1:n_generaciones) {
  if (verbose) {
    print("----------------------")
    print(paste("Generación:", i))
    print("----------------------")
  }


  # CALCULAR FITNESS DE LOS INDIVIDUOS DE LA POBLACIÓN
  # ============================================================================
  if (!paralelizado) {
    fitness_ind_poblacion <- calcular_fitness_poblacion(
      poblacion = poblacion,
      x = x,
      y = y,
      modelo = modelo,
      cv = cv,
      metrica = metrica,
      nivel_referencia = nivel_referencia,
      verbose = verbose
    )
  }

  if (paralelizado) {
    fitness_ind_poblacion <- calcular_fitness_poblacion_paral(
      poblacion = poblacion,
      x = x,
      y = y,
      modelo = modelo,
      cv = cv,
      metrica = metrica,
      nivel_referencia = nivel_referencia,
      verbose = verbose
    )
  }

  # SE ALMACENA EL MEJOR INDIVIDUO DE LA POBLACIÓN ACTUAL
  # ============================================================================
  fitness_mejor_individuo   <- max(fitness_ind_poblacion)
  mejor_individuo           <- poblacion[which.max(fitness_ind_poblacion), ]
  resultados_fitness[[i]]   <- fitness_mejor_individuo
  resultados_individuo[[i]] <- colnames(x)[mejor_individuo]
```

```r
  # SE CALCULA LA MEJORA RESPECTO A LA GENERACIÓN ANTERIOR
  # ============================================================================
  # El porcentaje de mejora solo puede calcularse a partir de la segunda
  # generación.
  if (i > 1) {
    porcentaje_mejora[[i]] <- 1 - (resultados_fitness[[i]] /
                                     resultados_fitness[[i - 1]])
  }

  # NUEVA POBLACIÓN
  # ============================================================================
  nueva_poblacion <- matrix(
    data = NA,
    nrow = nrow(poblacion),
    ncol = ncol(poblacion)
  )

  # ELITISMO
  # ============================================================================
  # El elitismo indica el porcentaje de mejores individuos de la población
  # actual que pasan directamente a la siguiente población. De esta forma, se
  # asegura que, la siguiente generación, no sea nunca inferior.

  if (elitismo > 0) {
    n_elitismo <- ceiling(nrow(poblacion) * elitismo)
    posicion_n_mejores <- order(fitness_ind_poblacion, decreasing = TRUE)
    posicion_n_mejores <- posicion_n_mejores[1:n_elitismo]
    nueva_poblacion[1:n_elitismo, ] <- poblacion[posicion_n_mejores, ]
  } else {
    n_elitismo <- 0
  }

  # CREACIÓN DE NUEVOS INDIVIDUOS POR CRUCES
  # ============================================================================
  for (j in (n_elitismo + 1):nrow(nueva_poblacion)) {
    # Seleccionar parentales
    metrica <- ifelse(test = is.numeric(y), "mse", "accuracy")
    indice_parental_1 <- seleccionar_individuo(
      vector_fitness = fitness_ind_poblacion,
      metrica = metrica,
      metodo_seleccion = metodo_seleccion,
      verbose = verbose
    )
    indice_parental_2 <- seleccionar_individuo(
      vector_fitness = fitness_ind_poblacion,
      metrica = metrica,
      metodo_seleccion = metodo_seleccion,
      verbose = verbose
    )
```

```
      parental_1 <- poblacion[indice_parental_1, ]
      parental_2 <- poblacion[indice_parental_2, ]

      # Cruzar parentales para obtener la descendencia
      descendencia <- cruzar_individuos(
                        parental_1 = parental_1,
                        parental_2 = parental_2,
                        metodo_cruce = metodo_cruce,
                        verbose = verbose
                      )
      # Mutar la descendencia
      descendencia <- mutar_individuo(
                        individuo = descendencia,
                        prob_mut = prob_mut
                      )

      # Almacenar el nuevo descendiente en la nueva población: puede ocurrir que
      # el individuo resultante del cruce y posterior mutación no contenga ningún
      # predictor (todas sus posiciones son FALSE). Si esto ocurre, y para evitar
      # que se produzca un error al ajustar el modelo, se introduce aleatoriamente
      # un valor TRUE.
      if (all(descendencia == FALSE)) {
        descendencia[sample(x = 1:length(descendencia), size = 1)] <- TRUE
      }
      nueva_poblacion[j, ] <- descendencia
    }
    poblacion <- nueva_poblacion

    # CRITERIO DE PARADA
    # ============================================================================
    # Si durante las últimas n generaciones el mejor individuo no ha mejorado por
    # una cantidad superior a tolerancia_parada, se detiene el algoritmo y no se
    # crean nuevas generaciones.

    if (parada_temprana && (i > rondas_parada)) {
      ultimos_n <- tail(unlist(porcentaje_mejora), n = rondas_parada)
      if (all(ultimos_n < tolerancia_parada)) {
        print(paste(
          "Algoritmo detenido en la generacion", i,
          "por falta de mejora mínima de", tolerancia_parada,
          "durante", rondas_parada,
          "generaciones consecutivas."
        ))
        break()
      }
    }
  }

  # IDENTIFICACIÓN DEL MEJOR INDIVIDUO DE TODO EL PROCESO
```

```
  # ============================================================================
  mejor_individuo_global <-
resultados_individuo[[which.max(unlist(resultados_fitness))]]

  # RESULTADOS
  # ============================================================================
  # La función devuelve una lista con 4 elementos:
  #   fitness:           una lista con el fitness del mejor individuo de cada
  #                      generación.
  #   mejor_individuo:   una lista con la combinación de predictores  del mejor
  #                      individuo de cada generación.
  #   porcentaje_mejora: una lista con el porcentaje de mejora entre el mejor
  #                      individuo de cada generación.
  #   df_resultados:     un dataframe con todos los resultados anteriores.
  #   mejor_individuo_global: la combinación de predictores  del mejor individuo
  #                           encontrado en todo el proceso.

  # Para crear el dataframe se convierten las listas a vectores del mismo tamaño.
  fitness <- unlist(resultados_fitness)
  predictores <- resultados_individuo[!sapply(resultados_individuo, is.null)]
  predictores <- sapply(predictores, function(x) {
                                       paste(x, collapse = ", ")
                                     })
  mejora <- c(NA, unlist(porcentaje_mejora))

  df_resultados <- data.frame(
    generacion = seq_along(fitness),
    fitness = fitness,
    predictores = predictores,
    mejora = mejora
  )

  return(list(
    fitness = resultados_fitness,
    mejor_individuo = resultados_individuo,
    porcentaje_mejora = porcentaje_mejora,
    df_resultados = df_resultados,
    mejor_individuo_global = mejor_individuo_global
  ))
}
```

Comparación

Se compara el tiempo necesario para ejecutar el ejemplo del Titanic empleando paralelización y sin ella.

```
library(titanic)
library(dplyr)

# Se eliminan para este ejemplo 3 variables.
datos <- titanic_train %>% select(-Name, -Ticket, -Cabin)
# Se eliminan observaciones incompletas.
datos <- na.omit(datos)
# Se convierten a factor las variables de tipo character.
datos <- datos %>% mutate_if(is.character, as.factor)
datos <- datos %>% mutate(Survived = as.factor(Survived))
```

Sin paralelización

```
library(tictoc)
tic()

seleccion <- selecionar_predictores(x = datos[, -2],
                                    y = datos[, 2],
                                    n_poblacion = 50,
                                    n_generaciones = 10,
                                    n_max_predictores = 8,
                                    n_min_predictores = 1,
                                    elitismo = 0.01,
                                    prob_mut = 0.01,
                                    metodo_seleccion = "tournament",
                                    metodo_cruce = "uniforme",
                                    modelo = "rf",
                                    metrica = "accuracy",
                                    nivel_referencia = "1",
                                    cv = 5,
                                    parada_temprana = FALSE,
                                    rondas_parada = NULL,
                                    tolerancia_parada = NULL,
                                    paralelizado = FALSE,
                                    verbose = FALSE
                                    )
toc()
```

```
## 125.35 sec elapsed
```

Con paralelización

```
tic()
seleccion <- selecionar_predictores(x = datos[, -2],
                                    y = datos[, 2],
                                    n_poblacion = 50,
                                    n_generaciones = 10,
                                    n_max_predictores = 8,
                                    n_min_predictores = 1,
                                    elitismo = 0.01,
                                    prob_mut = 0.01,
                                    metodo_seleccion = "tournament",
                                    metodo_cruce = "uniforme",
                                    modelo = "rf",
                                    metrica = "accuracy",
                                    nivel_referencia = "1",
                                    cv = 5,
                                    parada_temprana = FALSE,
                                    rondas_parada = NULL,
                                    tolerancia_parada = NULL,
                                    paralelizado = TRUE,
                                    verbose = FALSE
                                    )
toc()
```

```
## 64.032 sec elapsed
```

Puede observarse que, para las mismas generaciones, la versión paralelizada tarda la mitad de tiempo.

Bibliografía

John McCall, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics, Volume 184, Issue 1, 2005

https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)