uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE

APRENDIZAJE AUTOMÁTICO PARA EL ANÁLISIS DE DATOS

GRADO EN ESTADÍSTICA Y EMPRESA

Ricardo Aler

AjusteHiperGBM

Ricardo Aler

1/8/2019

Ajuste hiper-parámetros GBMs

```
library(mlr)
```

```
## Warning: package 'mlr' was built under R version 3.5.3
```

```
## Loading required package: ParamHelpers
```

```
## Warning: package 'ParamHelpers' was built under R version 3.5.3
```

```
library(mlbench)
library(mlrHyperopt) # Esta librería ayuda en la obtención de hiper-parámetros
library(ggplot2)     # Librería para hacer plots
library(mlrMBO)      # Para hacer ajuste de hiper-parámetros con model-based optimization
```

```
## Loading required package: smoof
```

```
## Loading required package: BBmisc
```

```
##
## Attaching package: 'BBmisc'
```

```
## The following object is masked from 'package:base':
##
##     isFALSE
```

```
## Loading required package: checkmate
```

```
# Esto es para que no haya demasiados mensajes de información en el documento
configureMlr(show.info = FALSE)

data(BostonHousing)
# Primero, definimos una tarea de regresión, cuyos datos están contenidos en el data.frame BostonHousin
task_bh <- makeRegrTask(data= BostonHousing, target="medv")

# Segundo, definimos el nombre delgoritmo de aprendizaje ("learner").
learner_name <- "regr.gbm"

# Tercero, vemos que hiper-parámetros ajustables tiene
filterParams(getParamSet(learner_name), tunable = TRUE)
```

```
##                       Type len      Def                         Constr Req
## distribution      discrete   - gaussian gaussian,laplace,poisson,tdist   -
## n.trees            integer   -      100                      1 to Inf   -
## cv.folds           integer   -        0                    -Inf to Inf   -
## interaction.depth  integer   -        1                      1 to Inf   -
## n.minobsinnode     integer   -       10                      1 to Inf   -
## shrinkage          numeric   -    0.001                      0 to Inf   -
## bag.fraction       numeric   -      0.5                        0 to 1   -
## train.fraction     numeric   -        1                        0 to 1   -
##                   Tunable Trafo
```

```
## distribution         TRUE     -
## n.trees              TRUE     -
## cv.folds             TRUE     -
## interaction.depth    TRUE     -
## n.minobsinnode       TRUE     -
## shrinkage            TRUE     -
## bag.fraction         TRUE     -
## train.fraction       TRUE     -
```

```
# Nuestro learner inicial va a ser una GBM. Lo definimos así:

learner_gbm <- makeLearner(learner_name)
```

Vemos que tiene muchos hiper-parámetros, pero sabemos que los más importantes son el número de árboles **n.trees** y el **shrinkage**. Más adelante tomaremos una decisión acerca de cuales ajustar.

Es interesante saber que podemos conseguir ayuda sobre el método y de sus parámetros así:

```
helpLearner(learner_name)
helpLearnerParam(learner_name, "ntree")
```

```
# Ahora habría que definir un espacio de búsqueda para los hiper-parámetros
# Una opción es ir a la página web # http://mlrhyperopt.jakob-r.de/parconfigs

# Una segunda opción es descargar directamente el espacio de búsqueda desde la web así:

pc = downloadParConfigs(learner.name=learner_name) # http://mlrhyperopt.jakob-r.de/parconfigs
print(pc)
if(length(pc)>0){
  ps = getParConfigParSet(pc[[1]], task=task_bh)
  print(ps)
}
# Desgraciadamente, en este caso no hay ninguno exactamente para ese learner
# Tendríamos que ir a la página web http://mlrhyperopt.jakob-r.de/parconfigs
# y buscarlo a mano, para gbm.
```

En este caso, usaremos una tercera opción, más sencilla: la función **generateParConfig** nos da un espacio de búsqueda básico para un learner concreto.

```
pc_gbm <- generateParConfig(learner=learner_name, task=task_bh)
ps_gbm <- getParConfigParSet(pc_gbm, task=task_bh)
print(ps_gbm)
```

```
##                      Type len   Def       Constr Req Tunable Trafo
## n.trees           numeric   -  5.64    0 to 6.64   -    TRUE     Y
## interaction.depth integer   -     1      1 to 10   -    TRUE     -
## shrinkage         numeric   - 0.001 0.001 to 0.6   -    TRUE     -
## n.minobsinnode    integer   -    10      5 to 25   -    TRUE     -
```

Vemos que aparecen 4 hiper-parámetros, pero sabemos que los más importantes son el número de árboles **n.trees** y el **shrinkage**. También el **n.minobsinnode** (cuántas instancias hacen falta en un nodo para seguir subdividiendo) e **interaction.depth** (controla la profundidad). Ambos controlan la profundidad de dos maneras distintas. El valor por omisión de **interaction depth** es 1, lo cual implica construir árboles de profundidad 1. Dado que Boosting utiliza modelos débiles (weak learners) como modelos base, podemos dejar que se sigan construyendo árboles de dicha profundidad (o sea, no ajustar este hiper-parámetro). Así que, en principio, ajustaremos sólo **n.trees** y **shrinkage**. Lo haremos así:

```
ps_gbm$pars <- ps_gbm$pars[c("n.trees", "shrinkage")]
print(ps_gbm)
```

```
##              Type len   Def       Constr Req Tunable Trafo
## n.trees   numeric   -  5.64    0 to 6.64   -    TRUE     Y
## shrinkage numeric   - 0.001 0.001 to 0.6   -    TRUE     -
```

Vemos que para **n.trees** se usa Trafo (transformación), veamos cual es:

```
print(ps_gbm$pars$n.trees$trafo)
```

```
## function (x)
## round(2^x * 10)
## <environment: 0x000000001ff74aa0>
```

También puede ser interesante saber cómo tendríamos que definir este espacio de búsqueda si hubiera que hacerlo a mano, o quisieramos cambiarlo. Sería así:

```
ps_amano <- makeParamSet(
  makeNumericParam("n.trees", lower=0, upper=6.64, trafo = function(x) round(2^x * 10)),
  makeNumericParam("shrinkage", lower=0.001, upper=0.6)
)

print(ps_amano)
```

```
##              Type len Def       Constr Req Tunable Trafo
## n.trees   numeric   -   -    0 to 6.64   -    TRUE     Y
## shrinkage numeric   -   - 0.001 to 0.6   -    TRUE     -
```

Ahora definimos cómo se búsca en el espacio de hiper-parámetros (con **randomsearch** en este caso y un **budget** de 50 evaluaciones), y cómo se evalúan los distintos hiper-parámetros (validación cruzada de tres folds).

```
# Ojo, he puesto 1000 evaluaciones para hacer una buena visualización después, pero puede tardar mucho.
# Posiblemente con budget = 50 sea suficiente

control_grid <- makeTuneControlRandom(budget=1000)
inner_desc <- makeResampleDesc("CV", iter=3)

# Aquí construimos una secuencia de ajuste seguida de construcción de modelo
learner_tune_gbm <- makeTuneWrapper(learner_gbm, resampling = inner_desc, par.set = ps_gbm, control = co

# También tenemos que definir cómo se va a evaluar el modelo final. En este caso con train/test (holdou
outer_desc <-  makeResampleDesc("Holdout")
set.seed(0)
outer_inst <- makeResampleInstance(outer_desc, task_bh)

# Por último, usamos resample para entrenar y evaluar learner_tune_gbm
set.seed(0)
error_tune_gbm <- resample(learner_tune_gbm, task_bh, outer_inst, measures = list(rmse), extract = getTu
```

Vemos que los hiper-parámetros que se eligieron en la partición de entrenamiento son los siguientes (también se muestra el error que producen dichos hiper-parámetros en la validación cruzada de 3 folds que se usó para seleccionarlos):

```
error_tune_gbm$extract
```

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

```
## Tune result:
## Op. pars: n.trees=191; shrinkage=0.209
## rmse.test.rmse=3.9481568
```

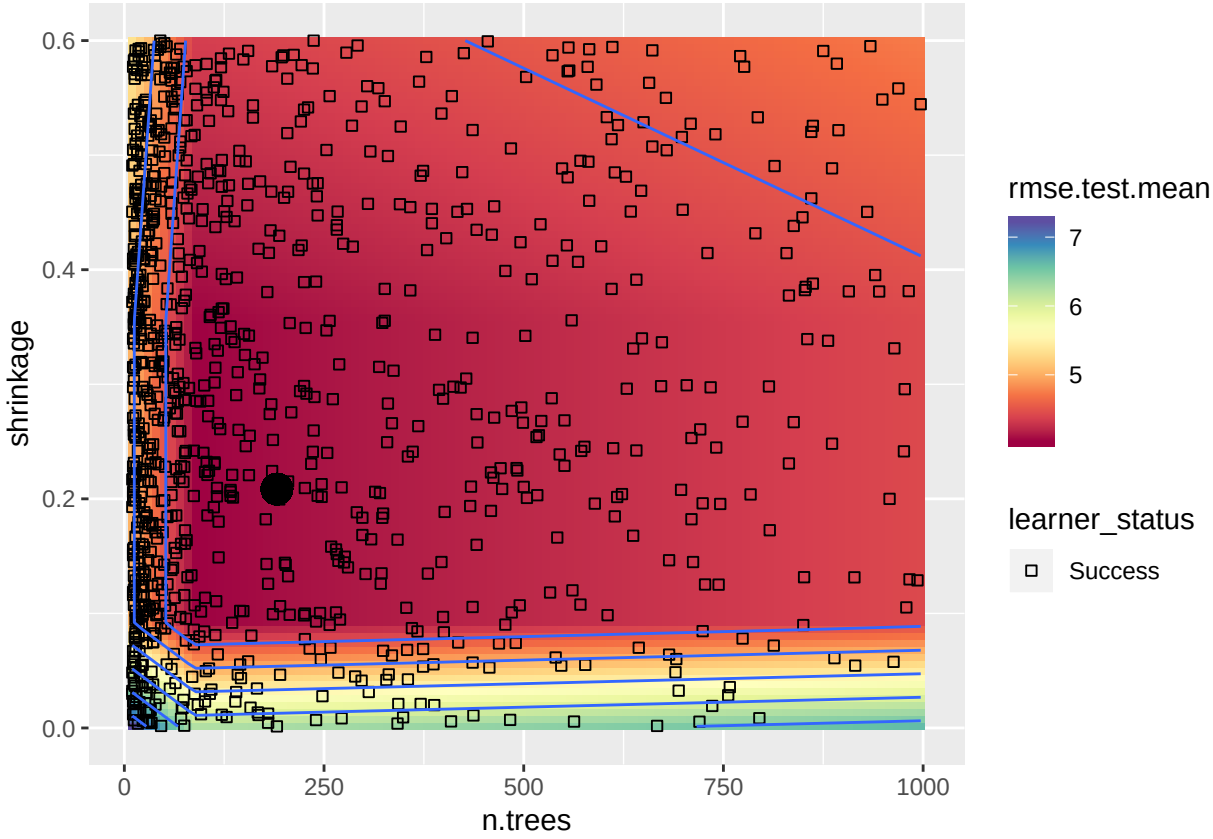
y el error del modelo final es:

```
error_tune_gbm$aggr
```

```
## rmse.test.rmse
##       3.237762
```

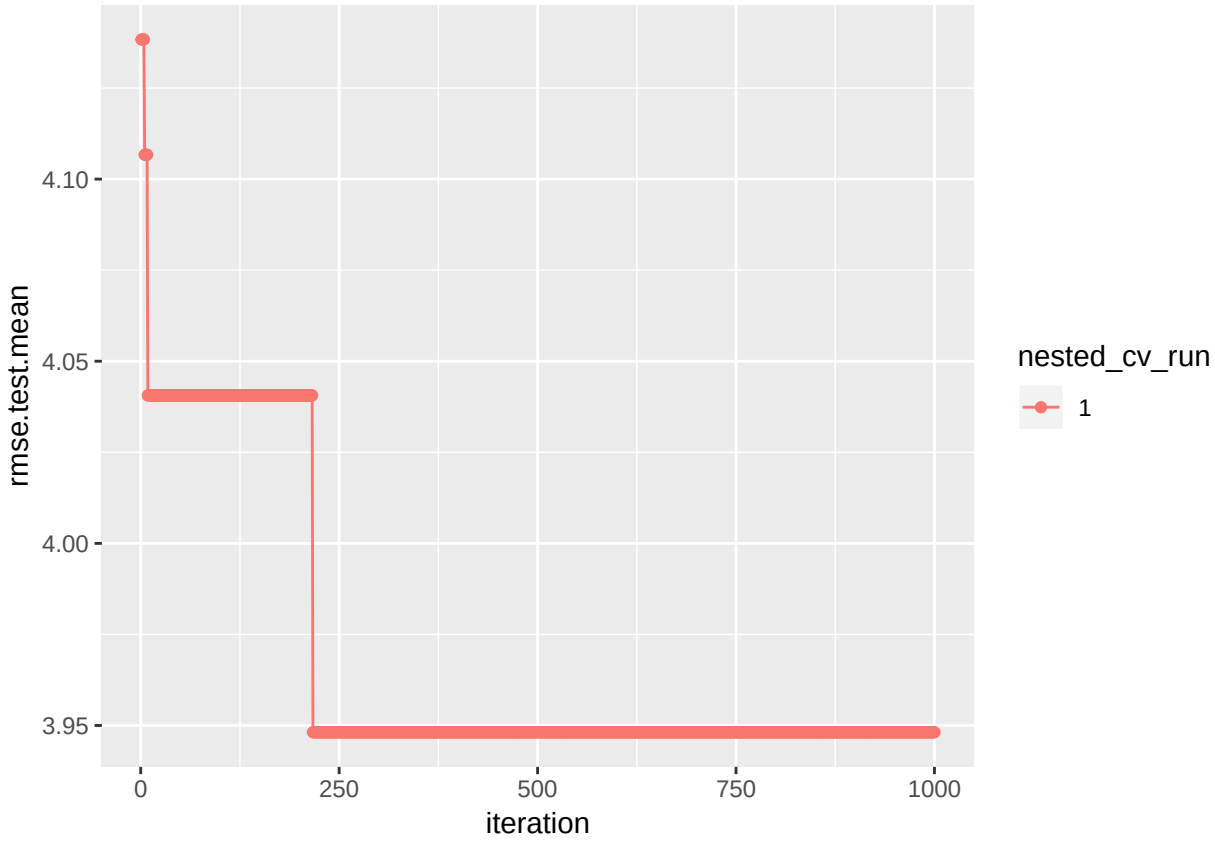
Por último, podemos ver un plot que nos muestra cómo cambia el error con los distintos valores de los hiper-parámetros. Podemos ver que hace falta un número razonable de árboles (al menos $2^3 * 10 = 80$, más o menos) y un **shrinkage** de al menos 0.1. El óptimo son 207 árboles y shrinkage 0.182.

```
data <- generateHyperParsEffectData(error_tune_gbm, include.diagnostics = FALSE, trafo=TRUE)

# Las siguientes dos líneas son para solventar un bug de MLR
names(data$data)[names(data$data)=="rmse.test.rmse"] <- "rmse.test.mean"
data$measures[data$measures=="rmse.test.rmse"] <- "rmse.test.mean"


plt = plotHyperParsEffect(data, x = "n.trees", y = "shrinkage", z = "rmse.test.mean",
  plot.type = "contour", interpolate = "regr.earth", show.experiments = TRUE)


plot(plt + geom_point(x=error_tune_gbm$extract[[1]]$x$n.trees, y=error_tune_gbm$extract[[1]]$x$shrinkag
```



A continuación podemos ver como se ha ido reduciendo el error con las iteraciones.

```
plt = plotHyperParsEffect(data, x = "iteration", y = "rmse.test.mean",
  plot.type = "line")
plt
```



Vamos a hacer el ajuste de hiper-parámetros con Model Based Optimization, del páquete mbo

```
# Dedica 80 iteraciones al ajuste
control = makeMBOControl()
control = setMBOControlTermination(control, iters = 80)
control = setMBOControlInfill(control, crit = makeMBOInfillCritEI())

control_grid <-makeTuneControlMBO(mbo.control = control)

inner_desc <- makeResampleDesc("CV", iter=3)

# Aquí construimos una secuencia de ajuste seguida de construcción de modelo
learner_tune_gbm <- makeTuneWrapper(learner_gbm, resampling = inner_desc, par.set = ps_gbm, control = c

# También tenemos que definir cómo se va a evaluar el modelo final. En este caso con train/test (holdou
outer_desc <-  makeResampleDesc("Holdout")
set.seed(0)
outer_inst <- makeResampleInstance(outer_desc, task_bh)
```

```r
# Por último, usamos resample para entrenar y evaluar learner_tune_gbm
set.seed(0)
error_tune_gbm <- resample(learner_tune_gbm, task_bh, outer_inst, measures = list(rmse), extract = getTu
```

Vemos que los hiper-parámetros que se eligieron en la partición de entrenamiento son los siguientes (también se muestra el error que producen dichos hiper-parámetros en la validación cruzada de 3 folds que se usó para seleccionarlos). No son muy distintos a los que obtuvimos con Random Search.

```r
error_tune_gbm$extract
```

```
## [[1]]
## Tune result:
## Op. pars: n.trees=104; shrinkage=0.25
## rmse.test.rmse=4.0192579
```

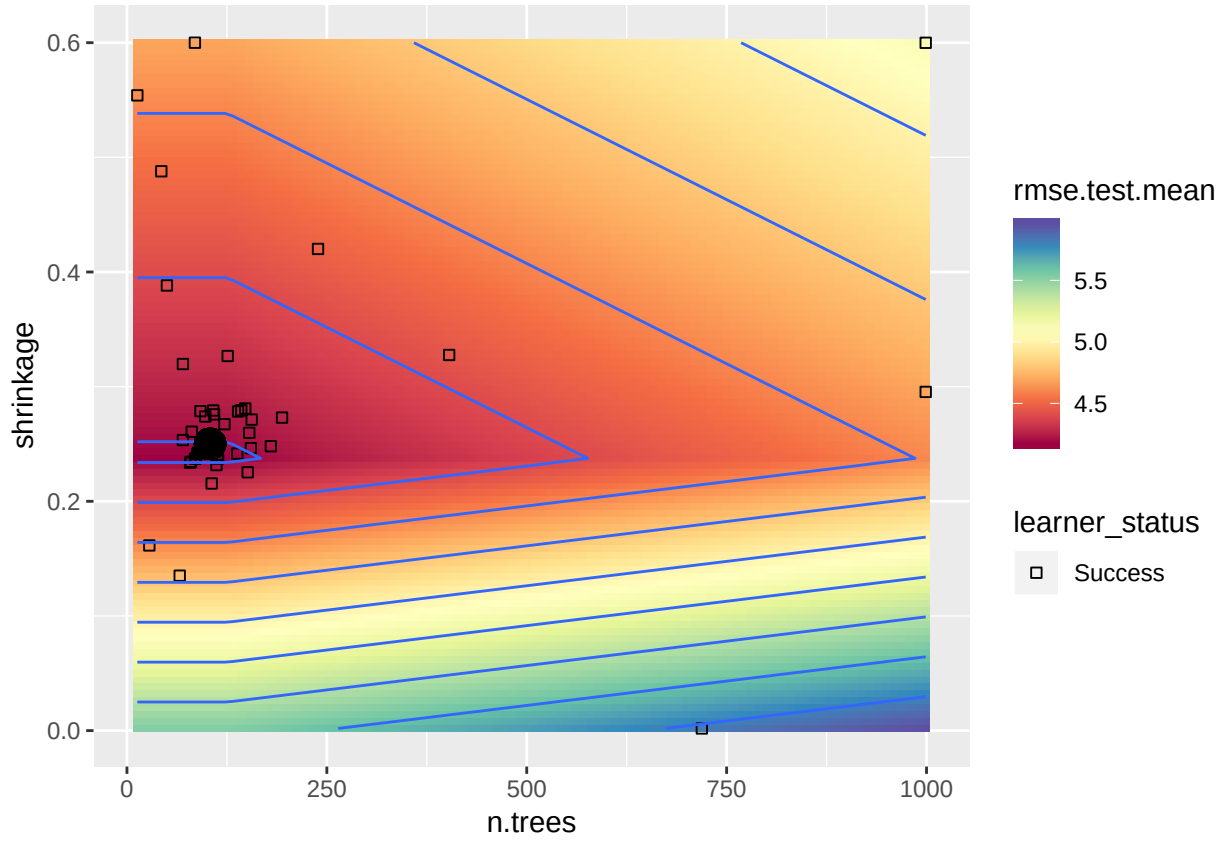
y el error del modelo final es muy parecido también al que obtuvimos con Random Search:

```r
error_tune_gbm$aggr
```

```
## rmse.test.rmse
##       3.312296
```

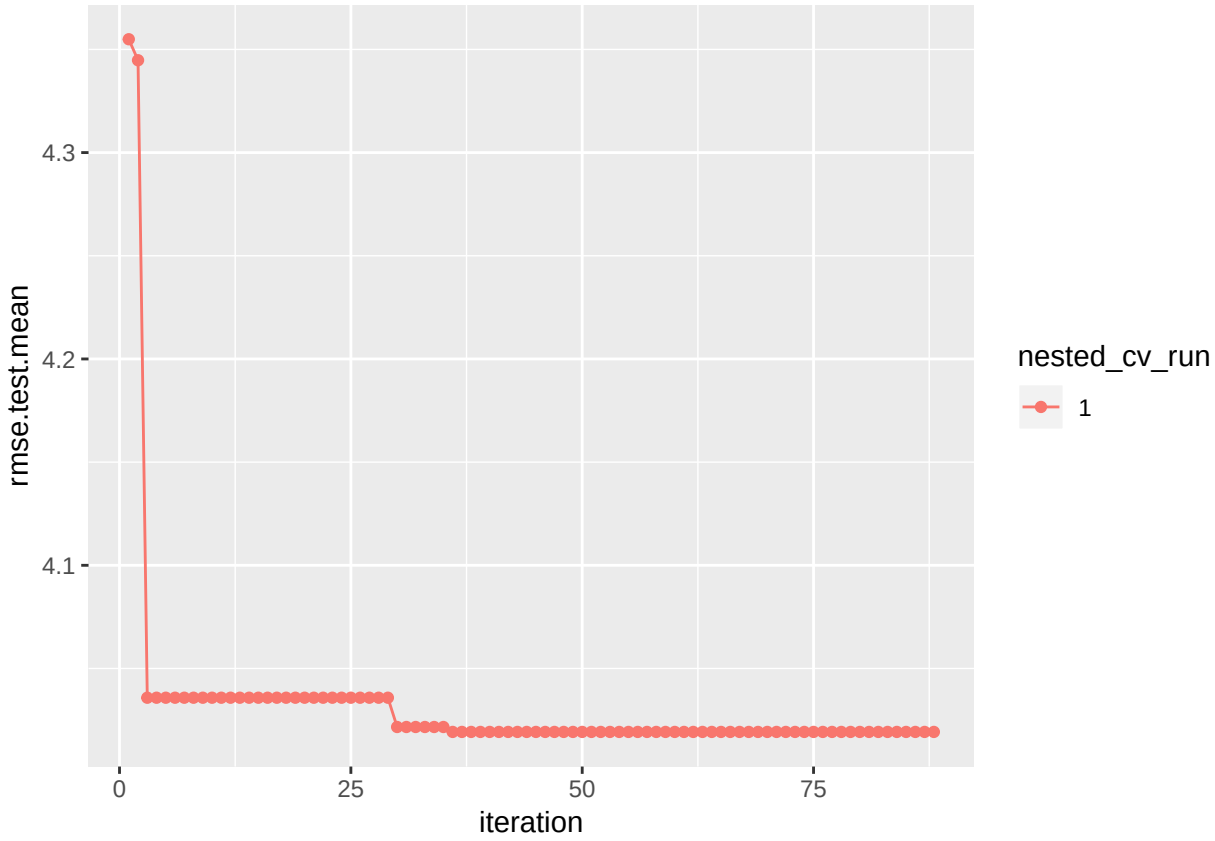
Lo que si parece claro que este tipo de búsqueda se centra mucho mejor en los valores adecuados de n.trees y Por último, podemos ver un plot que nos muestra cómo cambia el error con los distintos valores de los hiper-parámetros. Podemos ver que hace falta un número razonable de árboles (al menos $2^3 * 10 = 80$, más o menos) y un **shrinkage** de al menos 0.1. El óptimo son 207 árboles y shrinkage 0.182.

```r
data <- generateHyperParsEffectData(error_tune_gbm, include.diagnostics = FALSE, trafo=TRUE)

# Las siguientes dos líneas son para solventar un bug de MLR
names(data$data)[names(data$data)=="rmse.test.rmse"] <- "rmse.test.mean"
data$measures[data$measures=="rmse.test.rmse"] <- "rmse.test.mean"


plt = plotHyperParsEffect(data, x = "n.trees", y = "shrinkage", z = "rmse.test.mean",
  plot.type = "contour", interpolate = "regr.earth", show.experiments = TRUE)

plot(plt + geom_point(x=error_tune_gbm$extract[[1]]$x$n.trees, y=error_tune_gbm$extract[[1]]$x$shrinkag
```



A continuación podemos ver como se ha ido reduciendo el error con las iteraciones y parece que con bastantes menos que con Random Search se alcanzan mejores mínimos.

```
plt = plotHyperParsEffect(data, x = "iteration", y = "rmse.test.mean",
  plot.type = "line")
plt
```


4.3
4.2
4.1
rmse.test.mean
0
25
50
75
iteration
nested_cv_run
1