Machine Learning con R

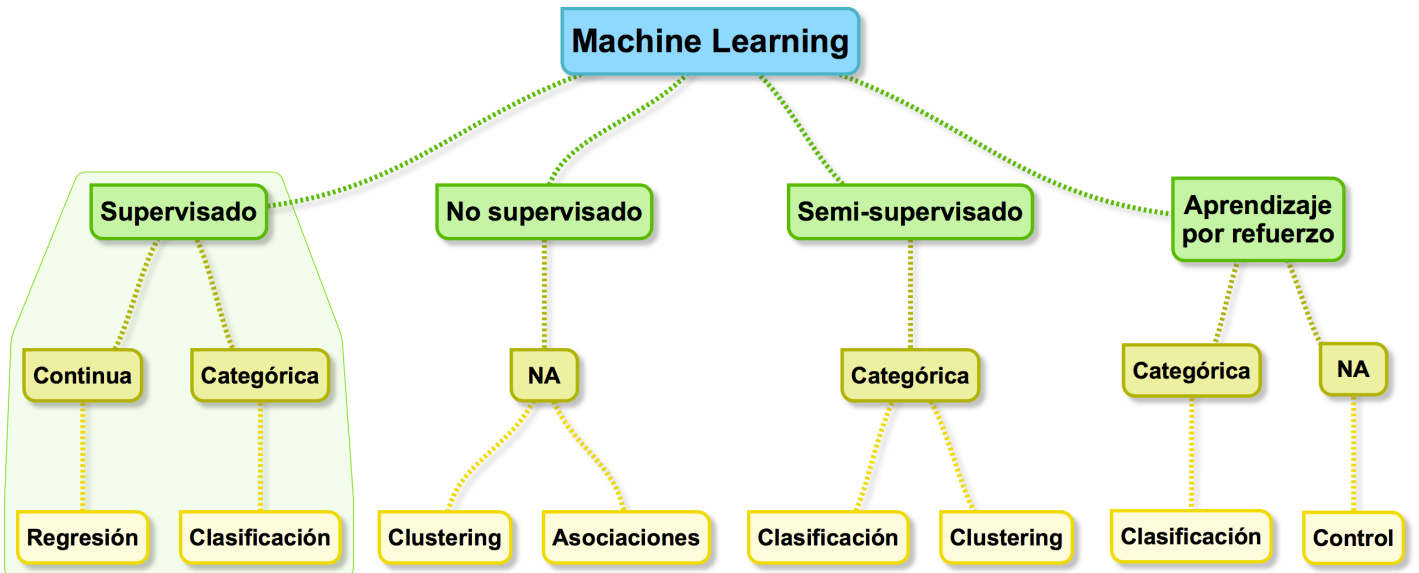
Domingo López Rodríguez

13 de diciembre de 2018

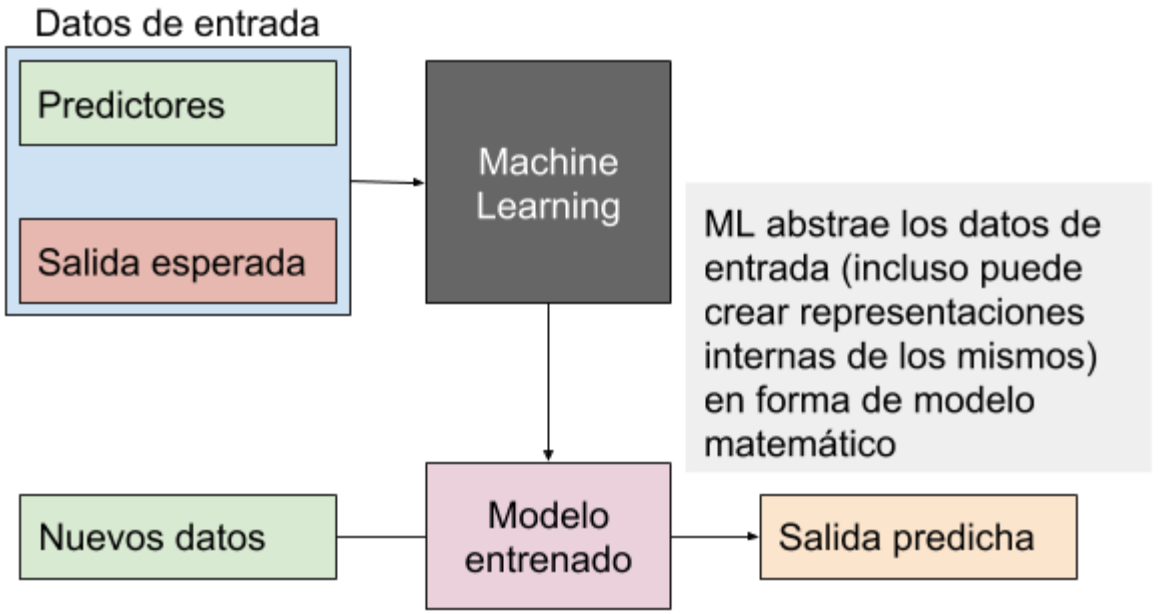
Machine Learning

Una máquina *aprende* si es capaz de utilizar experiencias pasadas para mejorar su desempeño futuro en una tarea

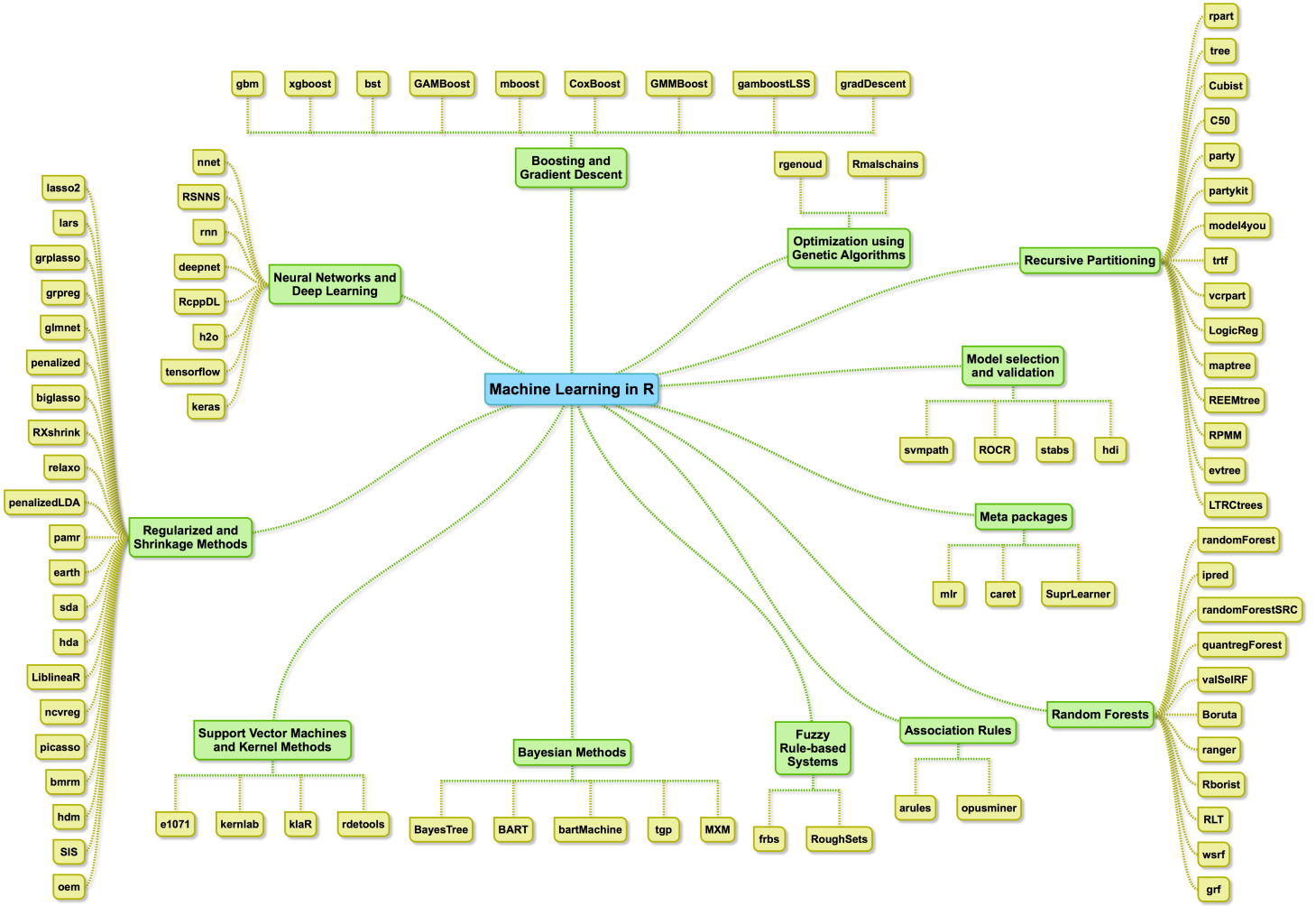
Tipos de machine learning



El proceso


Datos de entrada
Predictores
Salida esperada
Machine Learning
ML abstrae los datos de entrada (incluso puede crear representaciones internas de los mismos) en forma de modelo matemático
Nuevos datos
Modelo entrenado
Salida predicha

Paquetes de R para Machine Learning



Paquetes de R que vamos a usar

- mlbench

Una colección de problemas de prueba, tanto artificiales como reales, que incluyen, por ejemplo, algunos de los datasets del repositorio UCI.

- mlr

Machine Learning en R

- caret

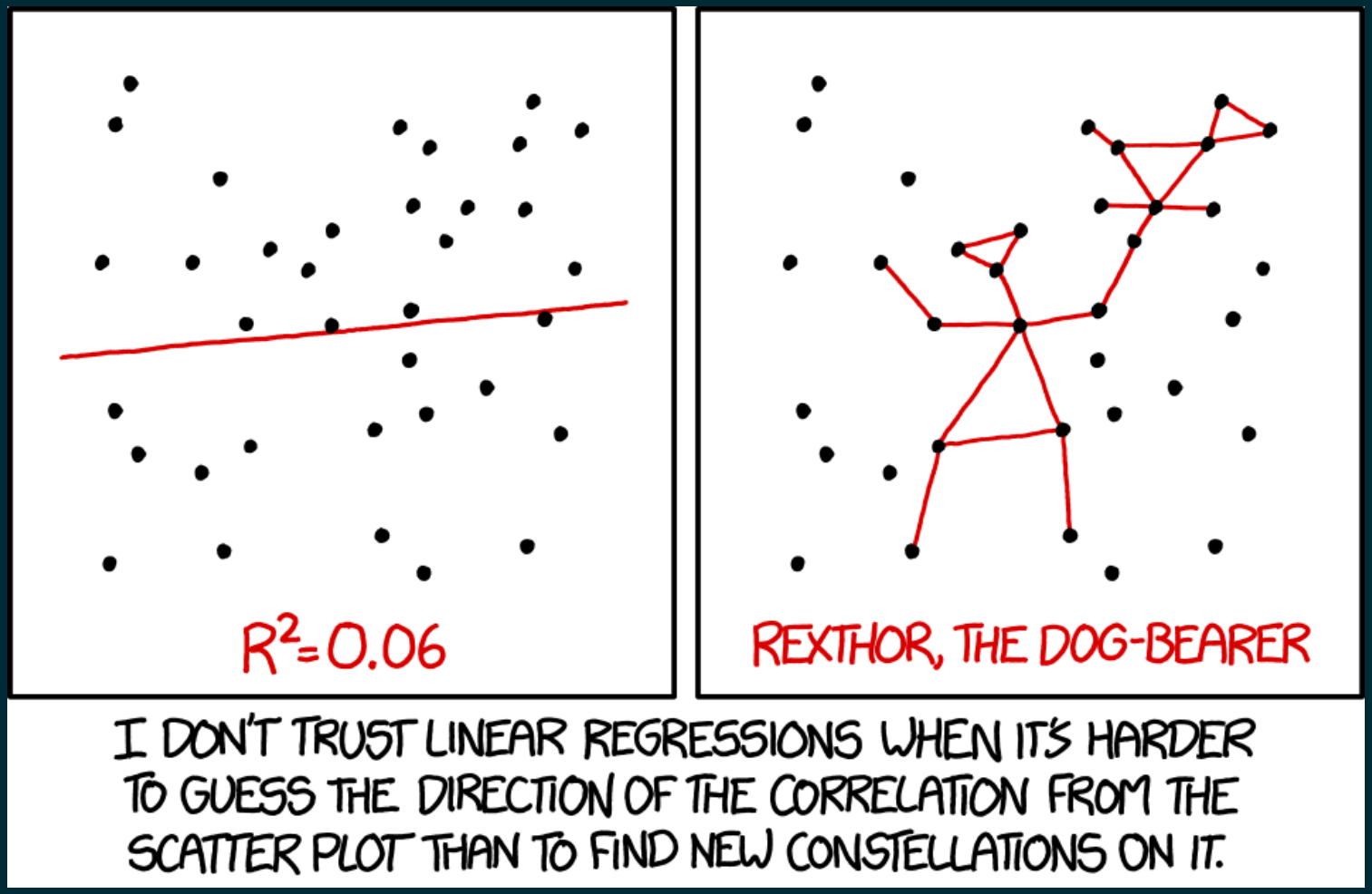
Diversas funciones para entrenar modelos y para representar gráficamente modelos regresión y clasificación.

- Y la distribución *base*!

Los instalamos antes de empezar:

```
install.packages(c("mlbench", "mlr", "caret"))
```

Regresión


$R^2 = 0.06$
REXTHOR, THE DOG-BEARER
I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

Datos

Cargamos los datos:

```
library(mlbench)
data("BostonHousing")
```

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

Datos sobre el precio de la vivienda en los suburbios de Boston, expresados en miles de dólares, con variables numéricas que indican aspectos del vecindario, de la criminalidad, etc.

El objetivo es predecir dicho precio de la vivienda a partir del resto de parámetros proporcionados.

R - Base

Construimos el modelo:

```
base_model ← lm(medv ~ ., data = BostonHousing)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	36.4594884	5.1034588	7.1440742	0.0000000
crim	-0.1080114	0.0328650	-3.2865169	0.0010868
zn	0.0464205	0.0137275	3.3815763	0.0007781
indus	0.0205586	0.0614957	0.3343100	0.7382881
chas1	2.6867338	0.8615798	3.1183809	0.0019250
nox	-17.7666112	3.8197437	-4.6512574	0.0000042
rm	3.8098652	0.4179253	9.1161402	0.0000000
age	0.0006922	0.0132098	0.0524024	0.9582293
dis	-1.4755668	0.1994547	-7.3980036	0.0000000
rad	0.3060495	0.0663464	4.6128998	0.0000051
tax	-0.0123346	0.0037605	-3.2800091	0.0011116
ptratio	-0.9527472	0.1308268	-7.2825106	0.0000000
b	0.0093117	0.0026860	3.4667926	0.0005729
lstat	-0.5247584	0.0507153	-10.3471458	0.0000000

Evaluación del desempeño

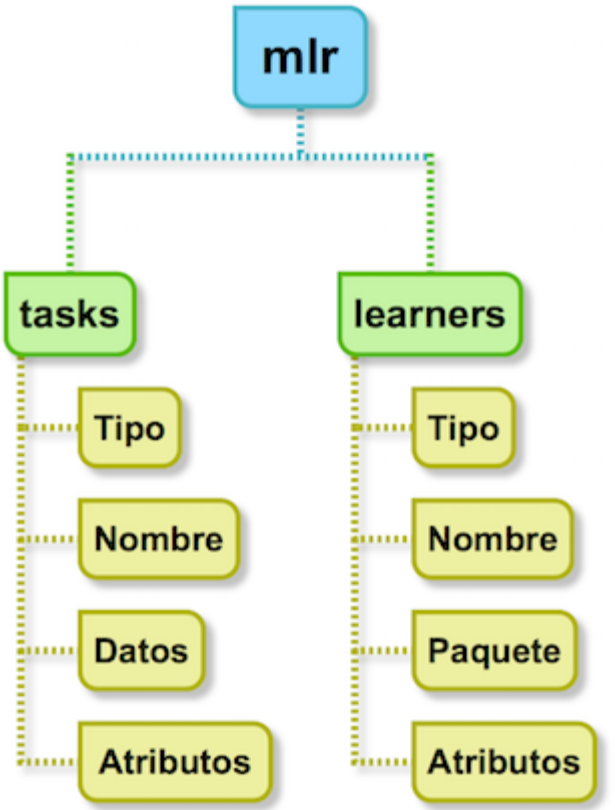
La regresión lineal tiene $R^2 = 0.73$.

Evaluamos su desempeño:

```
predictions ← base_model %>% predict(newdata = BostonHousing)

target_var ← BostonHousing$medv

mse_base ← mean((predictions - target_var) ^ 2)
```

El error cuadrático medio (MSE) obtenido por el modelo lineal es 21.89.

El paquete `mlr`



mlr - Tasks

```
library(mlr)
boston_task ← makeRegrTask(data = BostonHousing,
                           target = "medv")
```

```
Supervised task: BostonHousing
Type: regr
Target: medv
Observations: 506
Features:
   numerics     factors     ordered functionals
         12           1           0           0
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Has coordinates: FALSE
```

mlr - Learners

Lista de métodos (*learners*) que son capaces de hacer regresión:

```
regr_learners ← listLearners() %>% subset(type == "regr")
```

	class	name	package	type	installed	numerics	factors
95	regr.bartMachine	Bayesian Additive Regression Trees	bartMachine	regr	FALSE	TRUE	TRUE
96	regr.bcart	Bayesian CART	tgp	regr	FALSE	TRUE	TRUE
97	regr.bgp	Bayesian Gaussian Process	tgp	regr	FALSE	TRUE	FALSE
98	regr.bgpllm	Bayesian Gaussian Process with jumps to the Limiting Linear Model	tgp	regr	FALSE	TRUE	FALSE
99	regr.blm	Bayesian Linear Model	tgp	regr	FALSE	TRUE	FALSE
100	regr.brnn	Bayesian regularization for feed-forward neural networks	brnn	regr	FALSE	TRUE	TRUE

Sólo aquellos que están instalados:

	class	name	package	type	installed	numerics	factors
108	regr.cubist	Cubist	Cubist	regr	TRUE	TRUE	TRUE
114	regr.featureless	Featureless regression	mlr	regr	TRUE	TRUE	TRUE
115	regr.fnn	Fast k-Nearest Neighbor	FNN	regr	TRUE	TRUE	FALSE
116	regr.frbs	Fuzzy Rule-based Systems	frbs	regr	TRUE	TRUE	FALSE
118	regr.gausspr	Gaussian Processes	kernlab	regr	TRUE	TRUE	TRUE
119	regr.gbm	Gradient Boosting Machine	gbm	regr	TRUE	TRUE	TRUE

mlr - Entrenamiento

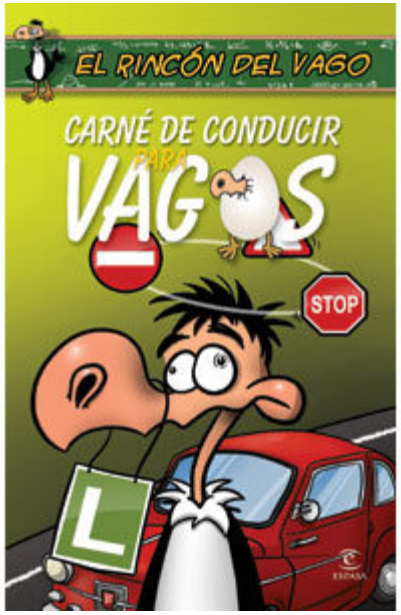
```
models ← benchmark(learners = "regr.lm",
                   tasks = boston_task,
                   show.info = FALSE)
```

iter	mse
1	25.24335
2	18.11463
3	30.87751
4	29.43987
5	21.39912
6	19.66457
7	36.42655
8	18.55697
9	16.17549
10	23.98655

El MSE final es: 23.99. (El encontrado usando `lm` directamente era 21.89).

¿Por qué la diferencia con lo anterior? *Hemos hecho trampas*.

Validación cruzada



Human Learning

Validación

Sobreentrenamiento

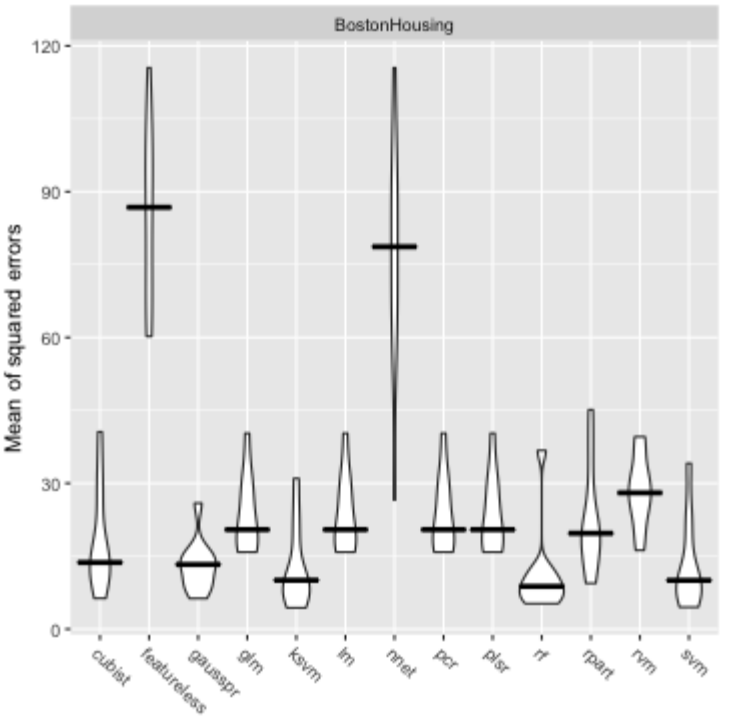
mlr - Comparación entre métodos:

```
my_learners ← regr_learners %>%
  subset(factors == TRUE & installed == TRUE)

many_models ← benchmark(learners = my_learners$class,
                        tasks = boston_task)
```

	MSE
Cubist	18.40429
Featureless regression	84.88674
Gaussian Processes	13.13554
Generalized Linear Regression	23.59927
Support Vector Machines	12.84640
Simple Linear Regression	23.59927
Neural Network	76.65893
Principal Component Regression	23.59927
Partial Least Squares Regression	23.59927
Random Forest	11.27763
Decision Tree	22.59178
Relevance Vector Machine	28.26751
Support Vector Machines (libsvm)	13.38069

mlr - Visualización de resultados

```
plotBMRBoxplots(many_models, style = "violin")
```



El paquete `caret`

```
library(caret)
modelLookup()
```

class	name	package	installed	regression	classification
ada	Boosted Classification Trees	ada,plyr	TRUE	FALSE	TRUE
AdaBag	Bagged AdaBoost	adabag,plyr	TRUE	FALSE	TRUE
AdaBoost.M1	AdaBoost.M1	adabag,plyr	TRUE	FALSE	TRUE
adaboost	AdaBoost Classification Trees	fastAdaboost	TRUE	FALSE	TRUE
amdai	Adaptive Mixture Discriminant Analysis	adaptDA	FALSE	FALSE	TRUE
ANFIS	Adaptive-Network-Based Fuzzy Inference System	frbs	TRUE	TRUE	FALSE
avNNet	Model Averaged Neural Network	nnet	TRUE	TRUE	TRUE
awnb	Naive Bayes Classifier with Attribute Weighting	bnclassify	TRUE	FALSE	TRUE
awtan	Tree Augmented Naive Bayes Classifier with Attribute Weighting	bnclassify	TRUE	FALSE	TRUE
bag	Bagged Model	caret	TRUE	TRUE	TRUE
bagEarth	Bagged MARS	earth	TRUE	TRUE	TRUE
bagEarthGCV	Bagged MARS using gCV Pruning	earth	TRUE	TRUE	TRUE
bagFDA	Bagged Flexible Discriminant Analysis	earth,mda	TRUE	FALSE	TRUE
bagFDAGCV	Bagged FDA using gCV Pruning	earth	TRUE	FALSE	TRUE
bam	Generalized Additive Model using Splines	mgcv	TRUE	TRUE	TRUE

`caret` - Entrenamiento

Debemos especificar el método de validación que queremos usar, y usamos la función `train` para entrenar dicho método.

```
fitControl ← trainControl(method = "repeatedcv",
                          number = 10,
                          repeats = 10)

caret_lm_model ← train(medv ~ .,
                       data = BostonHousing,
                       method = "lm",
                       trControl = fitControl)
```

caret - Resultado

El resultado de entrenar el modelo de regresión lineal es:

```
Linear Regression

506 samples
 13 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 455, 455, 455, 454, 456, 456, ...
Resampling results:

  RMSE      Rsquared   MAE
  4.796073  0.7320042  3.382283

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Podemos observar que se ha conseguido un MSE = 23.

`caret` - Comparación entre métodos

En `caret`, a diferencia de `mlr`, no existe un método para comparar todos los métodos que queremos, así que hay que ejecutar todos los métodos uno a uno:

```
methods ← c("cubist", "gbm", "xgbTree")

methods %>%
  lapply(function(x) {

    train(medv ~ .,
          data = BostonHousing,
          method = x,
          trControl = fitControl,
          verbose = FALSE)

  }) → caret_results
```

Hemos tenido que utilizar un `lapply` para ejecutar todos los métodos que nos interesaban.

caret - Resultados

```
Cubist

506 samples
 13 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 456, 454, 454, 455, 457, 456, ...
Resampling results across tuning parameters:

  committees  neighbors  RMSE      Rsquared   MAE
   1          0          3.939305  0.8135201  2.478549
   1          5          3.600509  0.8416157  2.221254
   1          9          3.639795  0.8385131  2.240542
  10          0          3.320117  0.8629138  2.213080
  10          5          3.009630  0.8851357  1.948938
  10          9          3.043609  0.8824804  1.979845
  20          0          3.257859  0.8682433  2.183036
  20          5          2.955227  0.8893099  1.923072
  20          9          2.986594  0.8868305  1.951879

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were committees = 20 and neighbors = 5.
```

caret-tuneGrid

- Los métodos tienen usualmente un número de *hiperparámetros* que ajustar.
- Por defecto, `caret` se encarga de estudiar una serie de configuraciones para cada método que utiliza en el entrenamiento.
- Si el método que queremos entrenar tiene p posibles hiperparámetros que configurar, `caret` usa de entrada 3^p configuraciones.
- Podemos especificar otras configuraciones de prueba mediante el argumento `tuneGrid` de la función `train`.

```
train(x, y, method = "rf", preProcess = NULL, ...,
weights = NULL, metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
maximize = ifelse(metric %in% c("RMSE", "logLoss", "MAE"), FALSE, TRUE
trControl = trainControl(), tuneGrid = NULL,
tuneLength = ifelse(trControl$method == "none", 1, 3))
```

caret - Hiperparámetros

```
getModelInfo(model = "xgbTree")[[1]]$parameters
```

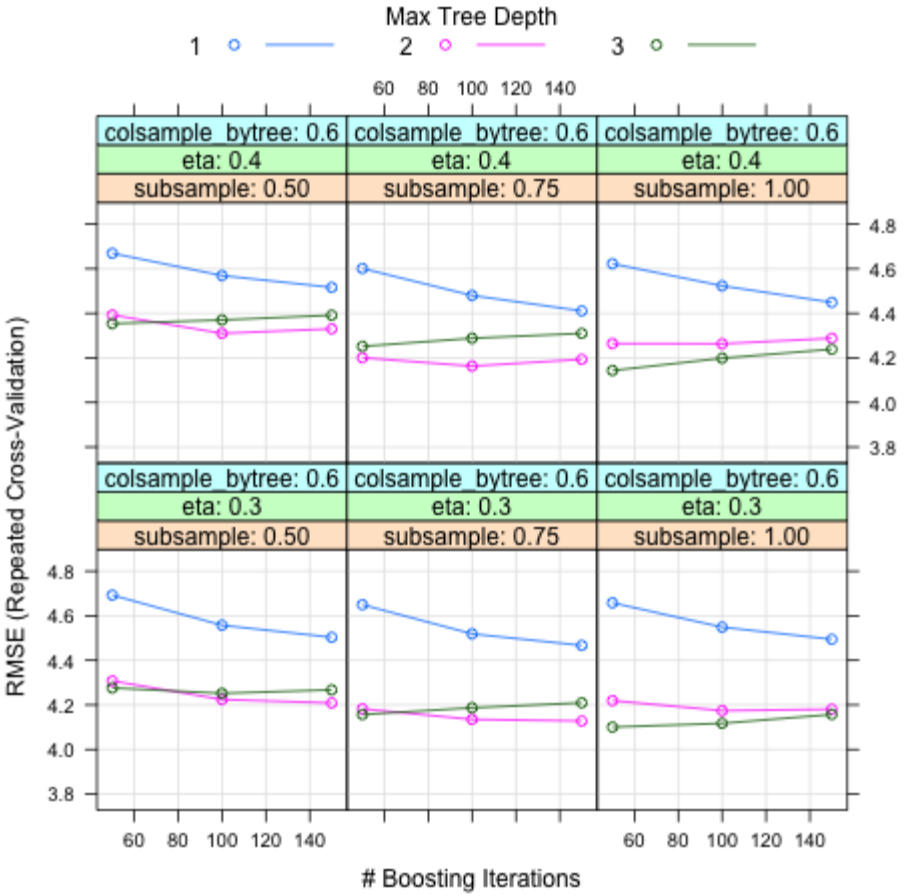
parameter	class	label
nrounds	numeric	# Boosting Iterations
max_depth	numeric	Max Tree Depth
eta	numeric	Shrinkage
gamma	numeric	Minimum Loss Reduction
colsample_bytree	numeric	Subsample Ratio of Columns
min_child_weight	numeric	Minimum Sum of Instance Weight
subsample	numeric	Subsample Percentage

caret - Resultados (II)

```
eXtreme Gradient Boosting

506 samples
 13 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 10 times)
Summary of sample sizes: 456, 455, 455, 455, 455, 456, ...
Resampling results across tuning parameters:

  eta  max_depth  colsample_bytree  subsample  nrounds  RMSE      Rsquared   MAE
  0.3  1          0.6               0.50        50      4.692556  0.7361644  3.148563
  0.3  1          0.6               0.50       100      4.557319  0.7507586  3.061440
  0.3  1          0.6               0.50       150      4.503253  0.7570643  3.031274
  0.3  1          0.6               0.75        50      4.649617  0.7391848  3.105018
  0.3  1          0.6               0.75       100      4.518725  0.7540496  3.023159
  0.3  1          0.6               0.75       150      4.466924  0.7601459  2.987899
   ...

Tuning parameter 'gamma' was held constant at a value of 0
Tuning parameter 'min_child_weight'
 was held constant at a value of 1
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nrounds = 150, max_depth = 3, eta = 0.3, gamma =
 0, colsample_bytree = 0.8, min_child_weight = 1 and subsample = 0.5.
```

caret - Visualización de resultados

```
plot(caret_results$xgbTree)
```



Clasificación

Datos

```
AlzheimerData ← readRDS("./AlzheimerData2.rds")
```

Usaremos datos procedentes de análisis de imagen médica tanto a pacientes de Alzheimer como sujetos sanos.

Tenemos unos 600 sujetos, de los cuales aproximadamente 330 son sanos.

Los datos incluyen valores demográficos y parámetros morfológicos del cerebro (volumen y grosor) de cada individuo, hasta llegar a unas 340 variables por sujeto.

El objetivo del problema es intentar *predecir* el valor de la variable de clase (AD - *Alzheimer* - o HEALTHY - *sano* -) a partir del resto de variables.

	AGE	SEX	BRAIN_VOLUME	GM_VOLUME	WM_VOLUME
152	75.46	MALE	1607.759	701.2512	493.5096
1895	38.00	FEMALE	1502.818	730.0900	518.2537
1731	62.80	MALE	1648.618	750.3945	566.8752
856	75.00	MALE	1787.767	778.1313	543.2225
1702	70.25	FEMALE	1177.902	545.2222	365.4857
67	68.00	MALE	1410.726	598.9312	433.2638

mlr - Definición del problema

Vamos ahora a crear una tarea de clasificación asociada a los datos que hemos importado, y seleccionamos qué métodos queremos/podemos usar:

```
ad_task ← makeClassifTask(id = "Alzheimer Disease",
                          data = AlzheimerData,
                          target = "CLASS")

my_class_learners ← listLearners() %>%
  subset(type == "classif" & installed == TRUE & factors == TRUE)
```

```
Supervised task: Alzheimer Disease
Type: classif
Target: CLASS
Observations: 600
Features:
   numerics     factors     ordered functionals
        226           1           0           0
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Has coordinates: FALSE
Classes: 2
     AD HEALTHY
    268     332
Positive class: AD
```

mlr - Entrenamiento

Vamos a usar los métodos disponibles para clasificar los sujetos:

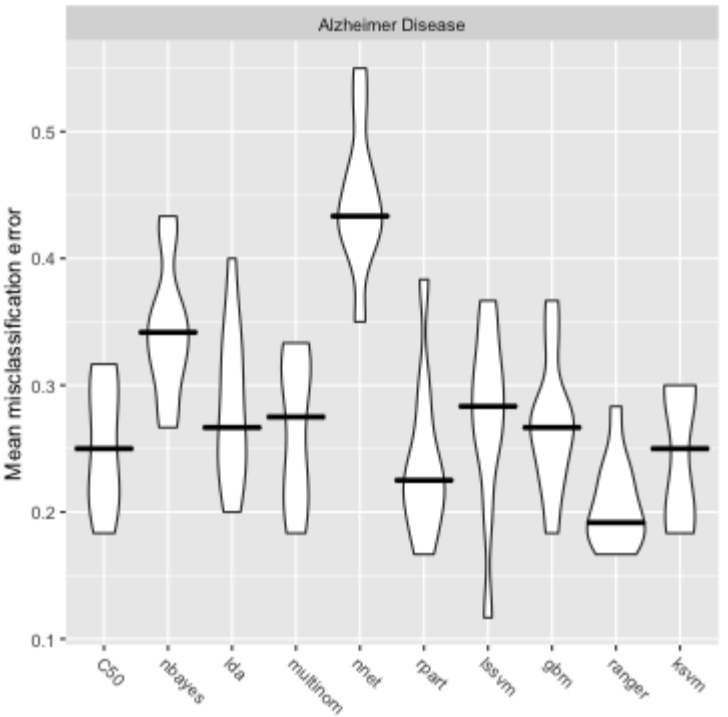
```
results_ad ← benchmark(learners = my_class_learners$class,
                       tasks = ad_task,
                       measures = list(mmce))
```

	MMCE
classif.C50	0.2533333
classif.naiveBayes	0.3433333
classif.lda	0.2833333
classif.multinom	0.2650000
classif.nnet	0.4466667
classif.rpart	0.2400000
classif.lssvm	0.2733333
classif.gbm	0.2683333
classif.ranger	0.2066667
classif.ksvm	0.2466667

El error mínimo obtenido ha sido de 20.67% y lo ha obtenido ranger.

mlr - Visualización de resultados

```
plotBMRBoxplots(results_ad, mmce, style = "violin")
```



mlr - Matriz de confusión

Otra forma de ver cómo se ha comportado el mejor modelo es a través de la matriz de confusión:

	AD	HEALTHY
AD	191	77
HEALTHY	75	257

Este tipo de tablas nos proporciona información acerca de la sensibilidad (tasa de falsos negativos) y la especificidad (tasa de falsos positivos) del método.

mlr - Predicción en nuevos sujetos

Tomamos algunos individuos de forma aleatoria:

```
indices ← sample(nrow(AlzheimerData), size = 15)

test ← AlzheimerData[indices, ]
truth ← AlzheimerData[indices, "CLASS"]
```

Y predecimos la variable de respuesta $CLASS$ para el mejor clasificador, sobre estos sujetos:

```
library(C50)
predicted_y ← predictLearner(
  .learner = results_ad$learners[["classif.C50"]],
  .model = results_ad$results$`Alzheimer Disease`[["classif.C50"]]$models[[1]],
  .newdata = test
)
```

mlr - Resultados sobre los nuevos sujetos

truth	predicted_y
HEALTHY	HEALTHY
HEALTHY	HEALTHY
HEALTHY	HEALTHY
HEALTHY	HEALTHY
AD	AD
HEALTHY	HEALTHY
AD	AD
HEALTHY	HEALTHY
AD	AD
AD	AD
HEALTHY	HEALTHY
AD	AD
AD	AD
HEALTHY	HEALTHY
HEALTHY	HEALTHY

	AD	HEALTHY
AD	6	0
HEALTHY	0	9

caret - Entrenamiento

Igual que en el caso de la regresión, debemos especificar los métodos a entrenar de antemano, y entrenar cada uno por separado.

```
methods ← c("gbm", "nnet", "rpart", "regLogistic", "xgbTree")

methods %>%
  lapply(function(x) {

    print(x)

    caret::train(CLASS ~ .,
                 data = AlzheimerData2,
                 method = x,
                 trControl = fitControl)

  }) → caret_class_results

names(caret_class_results) ← methods
```

caret - Resultados (individual)

```
print(caret_class_results$gbm)
```

```
Stochastic Gradient Boosting

600 samples
227 predictors
  2 classes: 'AD', 'HEALTHY'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 540, 540, 540, 540, 540, 539, ...
Resampling results across tuning parameters:

  interaction.depth  n.trees  Accuracy   Kappa
  1                   50      0.7115398  0.4169400
  1                  100      0.7299032  0.4525592
  1                  150      0.7533204  0.5013655
  2                   50      0.7498750  0.4952676
  2                  100      0.7766273  0.5489929
  2                  150      0.7980143  0.5906477
  3                   50      0.7799898  0.5546460
  3                  100      0.7948777  0.5840294
  3                  150      0.8082129  0.6109048

Tuning parameter 'shrinkage' was held constant at a value of 0.1

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 150,
 interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

caret - Resultados (comparativo)

Si comparamos los resultados obtenidos por todos los clasificadores, los resultados son como siguen:

	MMCE
gbm	0.1917871
nnet	0.2697245
rpart	0.2929777
regLogistic	0.2280995
xgbTree	0.1766352

En este caso, el mejor clasificador ha sido xgbTree con un error (MMCE) de 17.66%.

`caret` - Matriz de confusión

Vemos la matriz de confusión obtenida por el mejor método, usando todo el conjunto de datos:

	AD	HEALTHY
AD	267	1
HEALTHY	4	328

caret - Predicción en nuevos sujetos

```
predicted_y ← caret_class_results[[best_classifier_caret]] %>%
  predict(newdata = test)
```

truth	predicted_y
HEALTHY	HEALTHY
HEALTHY	HEALTHY
HEALTHY	HEALTHY
HEALTHY	HEALTHY
AD	AD
HEALTHY	HEALTHY
AD	AD
HEALTHY	HEALTHY
AD	AD
AD	AD
HEALTHY	HEALTHY
AD	AD
AD	AD
HEALTHY	HEALTHY
HEALTHY	HEALTHY

	AD	HEALTHY
AD	6	0
HEALTHY	0	9

Conclusiones

- Existe multitud de paquetes para aplicar métodos de Machine Learning dentro de R, con muy diversas especializaciones.
- Para alguien no experto, lo mejor es utilizar uno de los paquetes `caret` o `mlr`: Nos proporcionan *interfaces* de alto nivel con los paquetes que implementan los diversos métodos.
- Para quien ya tenga experiencia, se puede empezar por comparar diversos métodos usando uno de los paquetes anteriores y, posteriormente, usar el paquete concreto del método que haya demostrado mejores resultados.
- `caret` facilita hacer búsqueda de la mejor configuración de un método, pero `mlr` es más útil a la hora de comparar varios métodos entre sí.
- Lo mismo se aplica a la visualización de los resultados: en `mlr` se pueden representar varios métodos a la vez y compararlos, mientras que en `caret` no es sencillo esto último, sin embargo se puede usar para comparar las distintas configuraciones testeadas.

¡Gracias!

twitter: `@dominlopez` - `@_RMlg`

github: `neuroimaginador` - `RMalagaGroup`

Slides created via the R package **xaringan**.