uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
APRENDIZAJE AUTOMÁTICO PARA EL ANÁLISIS DE DATOS
GRADO EN ESTADÍSTICA Y EMPRESA
Ricardo Aler

PRESENTACIÓN

APRENDIZAJE AUTOMÁTICO
GRADO EN ESTADÍSTICA Y EMPRESA

RICARDO ALER MUR

APRENDIZAJE AUTOMÁTICO

- En general, es un subcampo de la Inteligencia Artificial, cuyo objetivo es hacer que las máquinas aprendan.

- En la práctica, consiste en crear modelos a partir de datos, y por tanto es cercana a algunas partes de la Estadística. Este es el punto de vista de la asignatura.

SISTEMAS DE RECOMENDACIÓN

- Ejemplo: **Santander Product Recommendation (Sistemas recomendadores)**
  - https://www.**kaggle**.com/c/santander-product-recommendation/data
  - Premio de $60000. 1787 equipos. Hace 2 años.
- Proporcionaban 1.5 años del comportamiento de clientes del banco: qué productos habían comprado: cuenta de ahorro, tarjeta de crédito, fondo de pensión, … y datos demográficos (renta, edad, sexo, localización, …)
- El objetivo era predecir que productos adicionales compraría el cliente el último mes (Junio de 2016)

Llega nuevo cliente:

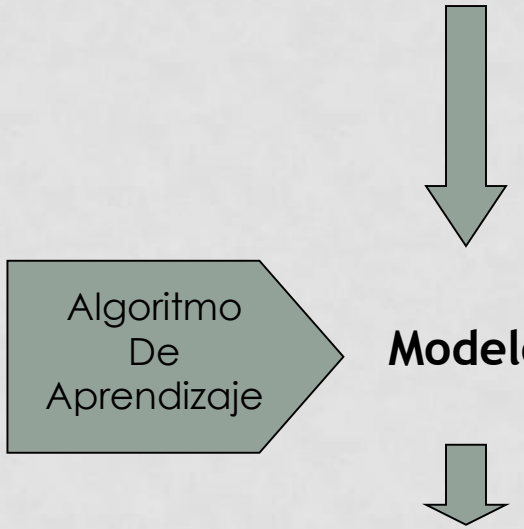
- Datos específicos: edad=50 años, sexo=mujer, localidad=22500, …
- Productos que compró / usó: tarjeta crédito y fondo de pensión

?

Datos Entrenamiento

Base de datos bancaria: para cada cliente:

- Datos específicos: edad, sexo, localidad, …
- Productos que compró / usó: tarjeta crédito, fondo de pensión, acciones, cuenta de ahorro, …

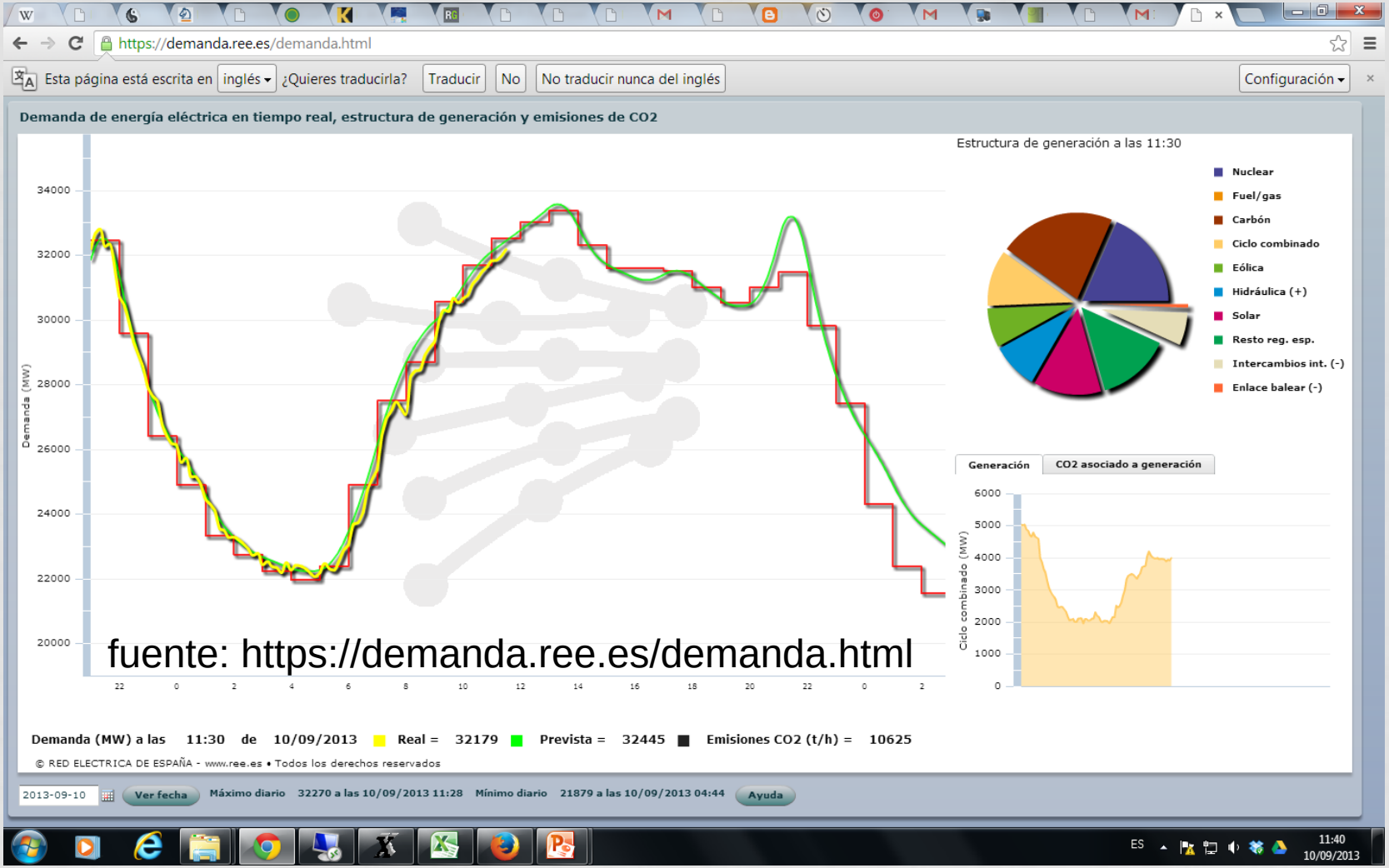


Comprará acciones y nuevo fondo de pensión

APLICACIONES DE MINERÍA DE DATOS (TÉCNICA DE CARÁCTER HORIZONTAL)

- Financieras y banca
  - Obtención de patrones de uso fraudulento de tarjetas de crédito
  - Predicción de devolución de créditos
- Análisis de mercado:
  - Sistemas recomendadores (análisis de cesta de la compra)
  - Segmentación de mercado
- Seguros y salud privada: determinación de clientes potencialmente caros
- Educación: detección de abandonos
- Industria: Predicción de la demanda eléctrica, de gas, etc.

PREDICCIÓN DEMANDA ELÉCTRICA



APLICACIONES II

- Medicina: diagnóstico de enfermedades (ej: diagnóstico de dolor abdominal)
- Ciencia:
  - Análisis de secuencias de proteínas
  - Predecir si un compuesto químico causa cáncer
  - Predecir si una persona puede tener potencialmente una enfermedad a partir de su DNA
  - Clasificación de cuerpos celestes (SKYCAT)
- Internet:
  - Detección de spam (SpamAssassin, bayesiano)
  - Web: asociar libros que compran usuarios en e-tiendas (amazon.com)

TEMARIO

1. **Introducción al aprendizaje automático**
   a) **Tareas, modelos y algoritmos**
   b) **Conceptos básicos en clasificación.**

2. Métodos básicos para clasificación y regresión:
   1. Modelos basados en similaridad: vecino más cercano (K-Nearest Neighbor KNN)
   2. Modelos basados en árboles y reglas

3. Metodología en aprendizaje automático (machine learning pipeline)
   1. Entrenamiento
   2. Ajuste de hiper-parámetros
   3. Evaluación (validación cruzada)

4. Selección y transformación de atributos

TEMARIO

1. Introducción al aprendizaje automático
   a) Tareas, modelos y algoritmos
   b) Conceptos básicos en clasificación.

2. **Métodos básicos para clasificación y regresión:**
   1. **Modelos basados en similaridad: vecino más cercano (K-Nearest Neighbor KNN)**
   2. **Modelos basados en árboles y reglas**

3. Metodología en aprendizaje automático (machine learning pipeline)
   1. Entrenamiento
   2. Ajuste de hiper-parámetros
   3. Evaluación (validación cruzada)

4. Selección y transformación de atributos

TEMARIO

1. Introducción al aprendizaje automático
   a) Tareas, modelos y algoritmos
   b) Conceptos básicos en clasificación.

2. Métodos básicos para clasificación y regresión:
   1. Modelos basados en similaridad: vecino más cercano (K-Nearest Neighbor KNN)
   2. Modelos basados en árboles y reglas

3. **Metodología en aprendizaje automático (machine learning pipeline)**
   1. **Entrenamiento**
   2. **Ajuste de hiper-parámetros**
   3. **Evaluación (validación cruzada)**

4. Pre-proceso de datos
   - Imputación, normalización, …
   - Selección y transformación de atributos (o predictores, o características)

TEMARIO

1. Introducción al aprendizaje automático
   a) Tareas, modelos y algoritmos
   b) Conceptos básicos en clasificación.

2. Métodos básicos para clasificación y regresión:
   1. Modelos basados en similaridad: vecino más cercano (K-Nearest Neighbor KNN)
   2. Modelos basados en árboles y reglas

3. Metodología en aprendizaje automático (machine learning pipeline)
   1. Entrenamiento
   2. Ajuste de hiper-parámetros
   3. Evaluación (validación cruzada)

4. **Pre-proceso de datos**
   - **Imputación, normalización, …**
   - **Selección y transformación de atributos (o predictores, o características)**

TEMARIO

5. **Métodos avanzados para clasificación y regresión.**
   a) **Conjuntos de modelos (ensembles):**
      a) **Bagging, Random Forests**
      b) **Boosting, Gradient Boosting**
      c) **Stacking**
   b) **Otros métodos avanzados:**
      a) **Redes de neuronas**
      b) **Máquinas de vectores de soporte**

6. Clasificación y evaluación con coste y muestras desbalanceadas. Curvas ROC

7. Introducción a técnicas de Big Data:
   a) MapReduce
   b) Spark

TEMARIO

5. Métodos avanzados para clasificación y regresión. Conjuntos de modelos (ensembles):
   a) Bagging, Random Forests
   b) Boosting, Gradient Boosting
   c) Stacking
   d) (Otros métodos avanzados: redes de neuronas, máquinas de vectores de soporte)

6. **Clasificación y evaluación con coste y muestras desbalanceadas. Curvas ROC**

7. Introducción a técnicas de Big Data:
   a) MapReduce
   b) Spark

TEMARIO

5. Métodos avanzados para clasificación y regresión. Conjuntos de modelos (ensembles):
   a) Bagging, Random Forests
   b) Boosting, Gradient Boosting
   c) Stacking
   d) (Otros métodos avanzados: redes de neuronas, máquinas de vectores de soporte)

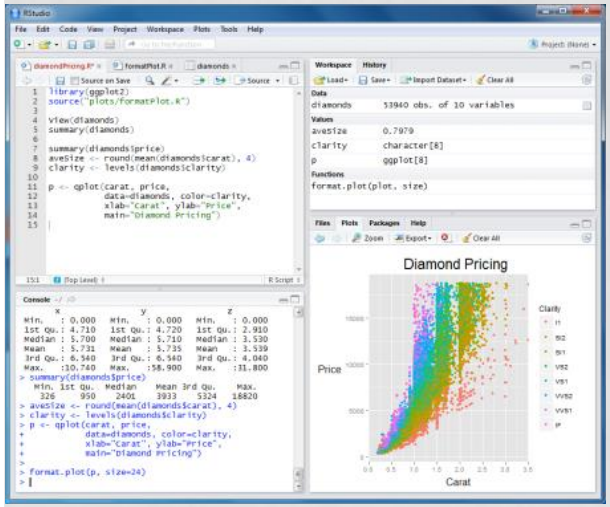
6. Clasificación y evaluación con coste y muestras desbalanceadas. Curvas ROC

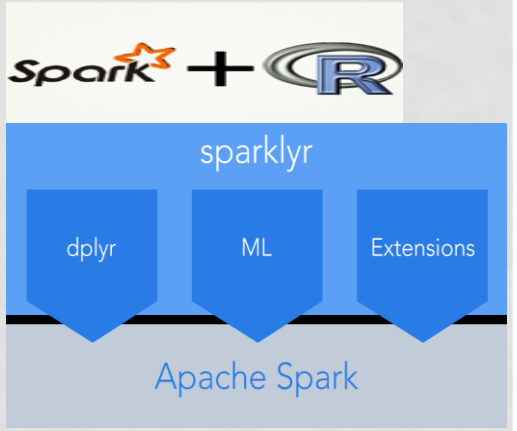
7. **Introducción a técnicas de Big Data:**
   a) **MapReduce**
   b) **Spark**

EVALUACIÓN

- **EXAMEN.**
- **PRÁCTICAS:**
  - **Práctica 1:** aprendizaje automático
  - **Práctica 2:** clasificación con muestras desbalanceadas
  - **Práctica 3:** Big Data en R (Sparklyr)Rstudio / SparkR

RSTUDIO



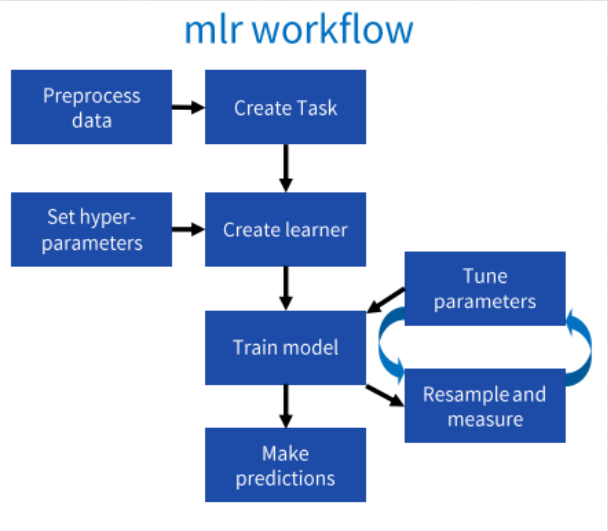


¿POR QUÉ MLR?

- Hay muchas librerías de aprendizaje automático para R, pero su uso no es siempre uniforme:
  - modelo = metodo(output ~ . , datos)
  - modelo = metodo(output ~ input1+input2+input3+input4 , datos)
  - modelo = metodo(datos[,inputs], datos$output)
- Algunos necesitan dataframes, otros matrices
- Con MLR, se hace igual con todos los métodos:
  - tarea = makeClassifTask(data = datos, target = "output")
  - metodo = makeLearner("metodo")
  - modelo = train(metodo, tarea)

¿POR QUÉ MLR?

- Hay muchas operaciones que hay que hacer típicamente en aprendizaje automático:
  - Ajustar hiper-parámetros
  - Obtener predicciones de los modelos
  - Evaluar a los modelos. Dividir en entrenamiento y test, …
  - Tratar problemas con muestras desbalanceadas
  - …



Source: https://github.com/mlr-org/mlr/blob/master/addon/cheatsheet/MlrCheatsheet.pdf

BIBLIOGRAFÍA PRINCIPAL

- **Applied predictive modeling. Max Kuhn. 2013**
- **Machine Learning with R Cookbook. Chiu Yu-Wei. 2015**
  - **http://proquest.safaribooksonline.com/9781783982042**
- **MLR: Machine Learning in R**
  - **Tutorial: https://mlr-org.github.io/mlr/**
  - **Cheatsheet:**
    - **https://github.com/mlr-org/mlr/blob/master/inst/cheatsheet/MlrCheatsheet.pdf**
    - **Artículo:** MLR: Machine Learning in R. Journal of Machine Learning Research 17 (2016) 1-5

Source: https://github.com/mlr-org/mlr/blob/master/addon/cheatsheet/MlrCheatsheet.pdf

Machine Learning with R

mlr

Introduction

***mlr* offers a unified interface for the basic building blocks of machine learning: tasks, learners, hyperparameters, etc.**

Tasks contain a description of a task (classification, regression, clustering, etc.) and a data set.

Learners specify a machine learning algorithm (GLM, SVM, xgboost, etc.) and its parameters.

Hyperparameters are learner settings that can be specified directly or tuned. A **parameter set** lists the possible hyperparameters for a given learner.

Wrapped Models are learners that have been trained on a task and can be used to make predictions.

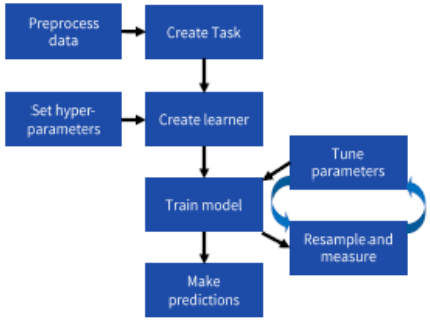
Predictions are the results of applying a model to either new data or the original training data.

Measures control how learner performance is evaluated, e.g. RMSE, LogLoss, AUC, etc.

Resampling estimates generalization performance by separating training data from test data. Common strategies include holdout and cross-validation.

Links: Tutorial | CRAN | Github

mlr workflow



Setup

Preprocessing data

`createDummyFeatures(obj=,target=,method=,cols=)`
Creates (0,1) flags for each non-numeric variable excluding `target`. Can be applied to entire dataset or only specific `cols`

`normalizeFeatures(obj=,target=,method=,cols=, range=,on.constant=)`
Normalizes numerical features according to specified `method`:
- `"center"` (subtract mean)
- `"scale"` (divide by std. deviation)
- `"standardize"` (center and scale)
- `"range"` (linear scale to given range, default `range=c(0,1)`)

`mergeSmallFactorLevels(task=,cols=,min.perc=)`
Combine infrequent factor levels into a single merged level

`summarizeColumns(obj=)` where `obj` is a data.frame or task. Provides type, NA, and distributional data about each column

See also `capLargeValues` `dropFeatures` `removeConstantFeatures` `summarizeLevels`

Creating a task

`makeClassifTask(data=,target=)`
Classification of a target variable, with optional positive class `positive`

`makeRegrTask(data=,target=)`
Regression on a target variable

`makeMultilabelTask(data=,target=)`
Classification where the target can belong to more than one class per observation

`makeClusterTask(data=)`
Unsupervised clustering on a data set

`makeSurvTask(data=,target= c("time","event"))`
Survival analysis with a survival time column and an event column

`makeCostSensTask(data=,costs=)`
Cost-sensitive classification where each observation-cost pair has a specified cost

Other arguments that can be passed to a `task`:
- `weights=` Weighting vector to apply to observations
- `blocking=` Factor vector where each level indicates a block of observations that will not be split up in resampling

Making a learner

`makeLearner(cl=,predict.type=,...,par.vals=)`
Choose an algorithm class to perform the task and determine what that algorithm will predict
- `cl=` name of algorithm, e.g. `"classif.xgboost"` `"regr.randomForest"` `"cluster.kmeans"`
- `predict.type="response"` returns a prediction type that matches the source data; `"prob"` returns a predicted probability for classification problems only; `"se"` returns the a standard error of the prediction for regression problems only. Only certain learners can return `"prob"` and `"se"`
- `par.vals=` takes a list of hyperparameters and passes them to the learner; parameters can also be passed directly (`...`)

You can make multiple learners at once with `makeLearners()`

mlr has integrated over 170 different learning algorithms
- Full list: `View(listLearners())` shows all learners
- Available learners for a task: `View(listLearners(task))`
- Filtered list: `View(listLearners("classif", properties=c("prob", "factors")))` shows all classification learners `"classif"` which can predict probabilities `"prob"` and handle factor inputs `"factors"`
- See also `getLearnerProperties()`

Training & Testing

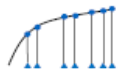
Setting hyperparameters

`setHyperPars(learner=, ...)`
Set the hyperparameters (settings) for each learner, if you don't want to use the defaults. You can also specify hyperparameters in the `makeLearner()` call

`getParamSet(learner=)`
Show the possible universe of parameters for your learner; can take a learner directly, or a text string such as `"classif.qda"`

Train a model and predict

`train(learner=,task=)`
Train a model (`WrappedModel`) by applying a learner to a task. By default, the model will train on all observations. The underlying model can be extracted with `getLearnerModel()`



`predict(object=,task=,newdata=)`
Use a trained model to make predictions on a task or dataset. The resulting `pred` object can be viewed with `View(pred)` or accessed by `as.data.frame(pred)`

Measuring performance

`performance(pred=,measures=)`
Calculate performance of predictions according to one or more of several measures (use `listMeasures()` for full list):
- `classif` `acc` `auc` `bac` `ber` `brier[.scaled]` `f1` `fdr` `fn` `fnr` `fp` `fpr` `gmean` `multiclass[.au1u .au1p .aunu .aunp .brier]` `npv` `ppv` `qsr` `ssr` `tn` `tnr` `tp` `tpr` `wkappa`
- `regr` `arsq` `expvar` `kendalltau` `mae` `mape` `medae` `medse` `mse` `msle` `rae` `rmse` `rmsle` `rrse` `rsq` `sae` `spearmanrho` `sse`
- `cluster` `db` `dunn` `G1` `G2` `silhouette`
- `multilabel` `multilabel[.f1 .subset01 .tpr .ppv .acc .hamloss]`
- `costsens` `mcp` `meancosts`
- `surv` `cindex`
- `other` `featperc` `timeboth` `timepredict` `timetrain`

For detailed performance data on classification tasks, use:
- `calculateConfusionMatrix(pred=)`
- `calculateROCMeasures(pred=)`

Resampling a learner

`makeResampleDesc(method=, ..., stratify=)`
`method` must be one of the following:
- `"CV"` (cross-validation, for number of folds use `iters=`)
- `"LOO"` (leave-one-out cross-validation, for folds use `iters=`)
- `"RepCV"` (repeated cross-validation, for number of repetitions use `reps=`, for folds use `folds=`)
- `"Subsample"` (aka Monte-Carlo cross-validation, for iterations use `iters=`, for train % use `split=`)
- `"Bootstrap"` (out-of-bag bootstrap, uses `iters=`)
- `"Holdout"` (for train % use `split=`)

`stratify` keeps target proportions consistent across samples.

`makeResampleInstance(desc=,task=)` can reduce noise by ensuring the resampling is done identically every time.

`resample(learner=,task=,resampling=,measures=)`
Train and test model according to specified resampling strategy.

mlr includes several pre-specified resample descriptions: `cv2` (2-fold cross-validation), `cv3`, `cv5`, `cv10`, `hout` (holdout with split 2/3 for training, 1/3 for testing).
Convenience functions also exist to `resample()` with a specific strategy: `crossval()`, `repcv()`, `holdout()`, `subsample()`, `bootstrapOOB()`, `bootstrapB632()`, `bootstrapB632plus()`

Refining Performance

Tuning hyperparameters

Set search space using `makeParamSet(make<type>Param())`
- `makeNumericParam(id=,lower=,upper=,trafo=)`
- `makeIntegerParam(id=,lower=,upper=,trafo=)`
- `makeIntegerVectorParam(id=,len=,lower=,upper=, trafo=)`
- `makeDiscreteParam(id=,values=c(...))` (can also be used to test discrete values of numeric or integer parameters)

`trafo` transforms the parameter output using a specified function, e.g. `lower=-2,upper=2,trafo=function(x) 10^x` would test values between 0.01 and 100, scaled exponentially
Other acceptable parameter types include `Logical` `LogicalVector` `CharacterVector` `DiscreteVector`

Set a search algorithm with `makeTuneControl<type>()`
- `Grid(resolution=10L)` Grid of all possible points
- `Random(maxit=100)` Randomly sample search space
- `MBO(budget=)` Use Bayesian model-based optimization
- `Irace(n.instances=)` Iterated racing process
- Other types: `CMAES`, `Design`, `GenSA`

Tune using `tuneParams(learner=,task=,resampling=, measures=,par.set=,control=)`

Quickstart

Prepare data for training and testing

```
library(mlbench)
data(Soybean)
soy = createDummyFeatures(Soybean,target="Class")
tsk = makeClassifTask(data=soy,target="Class")
ho = makeResampleInstance("Holdout",tsk)
tsk.train = subsetTask(tsk,ho$train.inds[[1]])
tsk.test = subsetTask(tsk,ho$test.inds[[1]])
```

Convert the factor inputs in the Soybean dataset into (0,1) dummy features which can be used by the XGboost algorithm. Create a task to predict the "Class" column. Create a train set with 2/3 of data and a test set with the remaining 1/3 (default).

Create learner and evaluate performance

```
lrn = makeLearner("classif.xgboost",nrounds=10)
cv = makeResampleDesc("CV",iters=5)
res = resample(lrn,tsk.train,cv,acc)
```

Create an XGboost learner which will build 10 trees. Then test performance using 5-fold cross-validation. Accuracy should be between 0.904-0.92.

Tune hyperparameters and retrain model

```
ps = makeParamSet(makeNumericParam("eta",0,1),
  makeNumericParam("lambda",0,200),
  makeIntegerParam("max_depth",1,20))
tc = makeTuneControlMBO(budget=100)
tr = tuneParams(lrn,tsk.train,cv5,acc,ps,tc)
lrn = setHyperPars(lrn,par.vals=tr$x)
```

Tune hyperparameters `eta`, `lambda`, and `max_depth` by defining a search space and using Model Based Optimization (MBO) to control the search. Then perform 100 rounds of 5-fold cross-validation, improving accuracy to ~0.93. Update the XGboost learner with the tuned hyperparameters.

```
mdl = train(lrn,tsk.train)
prd = predict(mdl,tsk.test)
calculateConfusionMatrix(prd)
mdl = train(lrn,tsk)
```

Train the model on the train set and make predictions on the test set. Show performance as a confusion matrix. Finally, re-train model on the full set to use on new data. You are now ready to go out into the real world and make 93% accurate predictions!

Legend for functions (not all parameters shown):
`function(required_parameters=,optional_parameters=)`

BIBLIOGRAFÍA SECUNDARIA

- **Mastering Machine Learning with R. Cory Lesmeister. 2015**
  - **http://proquest.safaribooksonline.com/9781783984527**
- **R for Data Science. 2017**
  - **http://r4ds.had.co.nz/**
- **Data Mining: Practical Machine Learning Tools and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems)**
  - **En esta referencia se usa WEKA (en lugar de R), pero los conceptos principales de Machine Learning están bien explicados.**