Package 'parallelML'

October 14, 2022

Type Package

Title A Parallel-Voting Algorithm for many Classifiers

Version 1.2

Date 2015-06-26

Author Wannes Rosiers (InfoFarm)

Maintainer Wannes Rosiers <wannes.rosiers@infofarm.be>

Description By sampling your data, running the provided classifier on these samples in parallel on your own machine and letting your models vote on a prediction, we return much faster predictions than the regular machine learning algorithm and possibly even more accurate predictions.

License GPL-2

Depends R (>= 2.14.0)

Imports parallel (>= 3.1.1), foreach (>= 1.2.0), doParallel (>= 1.0.8)

URL www.infofarm.be

NeedsCompilation no

Repository CRAN

Date/Publication 2015-06-26 14:02:47

R topics documented:

`parallelML-package` *Parallel-Voting version of machine learning algorithms*

Description

By sampling your data, running the machine learning algorithm on these samples in parallel on your own machine and letting your models vote on a prediction, we return much faster predictions than the regular machine learning algorithm and possibly even more accurate predictions.

Details

Package:	parallelML
Type:	Package
Version:	1.0
Date:	2015-09-06
License:	GPL-2

This package consists of two main functions: `parallelML` A function which allows you to create multiple machine learning models: one for each core you provide. It returns a list of machine learning models. `predictML`: A function which creates predictions for all models you created. When combine is "vote", it returns the class most models agree upon, otherwise it gives a list of predictions of all models

Author(s)

Wannes Rosiers

Maintainer: Wannes Rosiers <wannes.rosiers@infofarm.be>

See Also

This package can be regarded as a parallel extension of machine learning algorithms, therefor check the package of the machine learning algorithm you want to use.

Examples

```
## Not run:
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)

# Create a model
parSvmModel <- parallelML("svm(formula = Species ~ ., data = iris)",
                    "e1071",samplingSize = 0.8)
```

```
# Get prediction
parSvmPred   <- predictML("predict(parSvmModel,newdata=iris)",
                          "e1071","vote")

# Check the quality
table(parSvmPred,iris$Species)

## End(Not run)
```

---

caller *Convert arguments of a function to a list*

Description

It strips the function call of a function and returns a named list of only the used arguments

Usage

```
caller(...)
```

Arguments

...	All arguments you would pass to another function.

Value

A named list of all arguments

Author(s)

Wannes Rosiers

Examples

```
caller(formula = Species ~ ., data = iris)
```

convertAverage *Take averages over a list of data.frames*

Description

Given a list of similar data.frames (same schema and dimension) or vectors (same length), create a single data.frame or vector with same dimension or length consisting of averages over the list.

Usage

```
convertAverage(data)
```

Arguments

data	A list of equally-sized data.frames or vectors

Value

A single data.frame or vector containing averages over the list.

Author(s)

Wannes Rosiers

Examples

```
# Create a list of data.frames
frame1 <- data.frame(a=1:10,b=2:11)
frame2 <- data.frame(a=11:20,b=3:12)
frameList <- list(frame1,frame2)

# Take the respective averages
convertAverage(frameList)

# Create a list of vectors
vector1 <- 1:10
vector2 <- 11:20
vectorList <- list(vector1,vector2)

# Take the respective averages
convertAverage(vectorList)
```

getArgs *Get all arguments of a machine learning call*

Description

Converts a machine learning call to a named list of all its arguments.

Usage

```
getArgs(MLCall)
```

Arguments

MLCall	Your call to a machine learning algorithm. All arguments in this call should be named.

Value

A named list of all arguments present in your machine learning call

Author(s)

Wannes Rosiers

Examples

```
## Not run:
library(e1071)
MLCall <- "svm(formula = Species ~ ., data = iris)"
getArgs(MLCall)

## End(Not run)
```

getCall *Convert string to function call*

Description

Convert string to function call, where you can acces easily all entries

Usage

```
getCall(MLCall)
```

Arguments

MLCall	Your call to a machine learning algorithm. All arguments in this call should be named.

Value

A function Call of class "call"

Author(s)

Wannes Rosiers

Examples

```
## Not run:
library(e1071)
MLCall <- "svm(formula = Species ~ ., data = iris)"
functionCall <- getCall(MLCall)
functionCall$formula

## End(Not run)
```

---

get_args *Get all arguments of a call*

Description

This function can only be called within another function, it then returns a named list of all arguments in the super-function.

Usage

```
get_args()
```

Value

A named list of all arguments in the super-function.

Author(s)

Wannes Rosiers

`iris` *Edgar Anderson's Iris Data*

Description

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Usage

```
iris
```

Format

iris is a data frame with 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species. iris3 gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names Sepal L., Sepal W., Petal L., and Petal W., and the third the species.

Source

Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7, Part II, 179–188. The data were collected by Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59, 2–5.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole. (has iris3 as iris.)

Examples

```
dni3 <- dimnames(iris3)
ii <- data.frame(matrix(aperm(iris3, c(1,3,2)), ncol = 4,
                        dimnames = list(NULL, sub(" L.",".Length",
                                                  sub(" W.",".Width", dni3[[2]])))),
    Species = gl(3, 50, labels = sub("S", "s", sub("V", "v", dni3[[3]]))))
all.equal(ii, iris) # TRUE
```

parallelML *Parallel-Voting version of machine learning algorithms*

Description

By sampling your data, running the machine learning algorithm on these samples in parallel on your own machine and letting your models vote on a prediction, we return much faster predictions than the regular machine learning algorithm and possibly even more accurate predictions.

Usage

```
## Default S3 method:
parallelML(MLCall, MLPackage,
samplingSize = 0.2,numberCores = detectCores(),
underSample = FALSE, underSampleTarget = NULL,
sampleMethod = "bagging")
```

Arguments

MLCall	Your call to a machine learning algorithm. All arguments in this call should be named and the package only allows formula calls. Hence the call should look like "machineLearningAlgorithm(formula = ..., data = ..., ...)".
MLPackage	A character string of the package which provides your machine learning algorithm. This is needed since all cores should load the package.
samplingSize	Size of your data you will take in each sample.
numberCores	Number of cores of your machine you want to use. Is set equal to the number of samples you take.
underSample	Logical wether you want to take an undersample on your desired target.
underSampleTarget	When you set underSample to TRUE, underSampleTarget takes your target you want to keep in every sample. e.g. If you have 5 elements of category1 and 100 elements of category2 and your sampleSize is 0.2, then every sample will contain 25 elements, namely the 5 of category1 and 20 of category2.
sampleMethod	String which decides wether you sample on your observations (bagging) or on your variables (random).

Value

A list containing of numberCores machine learning models.

Note

Although it can cope with numeric probability predictions, this package is designed for classification labeling.

Author(s)

Wannes Rosiers

See Also

This package can be regarded as a parallel extension of machine learning algorithms, therefor check the package of the machine learning algorithm you want to use.

Examples

```
## Not run:
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)

# Create a model
parSvmModel <- parallelML("svm(formula = Species ~ ., data = iris)",
                     "e1071",samplingSize = 0.8)

# Get prediction
parSvmPred   <- predictML("predict(parSvmModel,newdata=iris)",
                          "e1071","vote")

# Check the quality
table(parSvmPred,iris$Species)

## End(Not run)
## Not run:
# Load the library which provides rpart
library(rpart)

# Create your data
data("magicData")

# Create a model
parTreeModel  <- parallelML("rpart(formula = V11 ~ ., data = trainData[,-1])",
                            "rpart",samplingSize = 0.8)

# Get prediction
parTrainTreePred <- predictML("predict(parTreeModel,newdata=trainData[,-1],type='class')",
                              "rpart","vote")
parTestTreePred <- predictML("predict(parTreeModel,newdata=testData[,-1],type='class')",
                             "rpart","vote")

# Check the quality
table(parTrainTreePred,trainData$V11)
table(parTestTreePred,testData$V11)

## End(Not run)
## Not run:
```

```
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)
subdata <- iris[1:60,]

# Create a model
parsvmmodel   <- parallelML("svm(formula = Species ~ ., data = subdata)",
                            "e1071",samplingSize = 0.8,
                            underSample = TRUE, underSampleTarget = "versicolor")

# Get prediction
parsvmpred    <- predictML("predict(parsvmmodel,newdata=subdata)",
                           "e1071","vote")

# Check the quality
table(parsvmpred,subdata$Species)

## End(Not run)
## Not run:
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)

# Create a model
parsvmmodel   <- parallelML("svm(formula = Species ~ ., data = iris)",
                            "e1071",samplingSize = 0.6,
                            sampleMethod = "random")

# Get prediction
parsvmpred    <- predictML("predict(parsvmmodel,newdata=iris)",
                           "e1071","vote")

# Check the quality
table(parsvmpred,iris$Species)

## End(Not run)
```

---

predictML *Parallel-Voting prediction of multiple machine learning models*

Description

By sampling your data, running the machine learning algorithm on these samples in parallel on your own machine and letting your models vote on a prediction, we return much faster predictions than the regular machine learning algorithm and possibly even more accurate predictions.

Usage

```
predictML(predictCall, MLPackage, combine = "raw")
```

Arguments

predictCall	Your call to a predict function. The object however should be an element of class parallelML.
MLPackage	A character string of the package which provides your machine learning algorithm. This is needed since all cores should load the package.
combine	A character string: "raw" when you want to return a list of predictions, "vote" when you want to return the class upon which most models agree and "avg" when you want to return the average of numeric predictions (probabilities).

Value

Either a list of vectors of predictions for each model or a vector of class upon which most models agree.

Note

Although it can cope with numeric probability predictions, this package is designed for classification labeling.

Author(s)

Wannes Rosiers

See Also

This package can be regarded as a parallel extension of machine learning algorithms, therefor check the package of the machine learning algorithm you want to use.

Examples

```
## Not run:
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)

# Create a model
parSvmModel <- parallelML("svm(formula = Species ~ ., data = iris)",
                    "e1071",samplingSize = 0.8)

# Get prediction
parSvmPred   <- predictML("predict(parSvmModel,newdata=iris)",
                          "e1071","vote")

# Check the quality
```

```
table(parSvmPred,iris$Species)

## End(Not run)
## Not run:
# Load the library which provides rpart
library(rpart)

# Create your data
data("magicData")

# Create a model
parTreeModel  <- parallelML("rpart(formula = V11 ~ ., data = trainData[,-1])",
                            "rpart",samplingSize = 0.8)

# Get prediction
parTrainTreePred <- predictML("predict(parTreeModel,newdata=trainData[,-1],type='class')",
                              "rpart","vote")
parTestTreePred <- predictML("predict(parTreeModel,newdata=testData[,-1],type='class')",
                             "rpart","vote")

# Check the quality
table(parTrainTreePred,trainData$V11)
table(parTestTreePred,testData$V11)

## End(Not run)
## Not run:
# Load the library which provides svm
library(e1071)

# Create your data
data(iris)
data("magicData")

# Get nummeric predicitions of Support Vector Machine
parsvmmodel   <- parallelML("svm(formula = Species ~ ., data = iris,probability=TRUE)",
                            "e1071",samplingSize = 0.8,
                            underSample = TRUE, underSampleTarget = "versicolor")
parsvmpred    <- predictML("predict(parsvmmodel,newdata=iris,probability=TRUE)",
                           "e1071","avg")

# Get numeric predictions of a generalized linear model
parglmmodel   <- parallelML("glm(formula = V11 ~ ., data = trainData[,-1],
                            family = binomial(link='logit'))","stats",samplingSize = 0.8)

parglmpred   <- predictML("predict(parglmmodel,newdata=trainData[,-1],type='response')",
                          "stats","avg")

## End(Not run)
```

---

registerCores *registerCores*

Description

The registerCores function is used to register the multicore parallel backend via either the doMC or doSNOW package.

Usage

```
registerCores(numberCores)
```

Arguments

numberCores	The number of cores to use for parallel execution. If not specified, the number of cores is set to the number of cores of your machine.

Details

The multicore functionality, originally written by Simon Urbanek and subsumed in the parallel package in R 2.14.0, provides functions for parallel execution of R code on machines with multiple cores or processors, using the system fork call to spawn copies of the current process. The multicore functionality, and therefore registerCores, should not be used in a GUI environment, because multiple processes then share the same GUI.

Author(s)

Wannes Rosiers

---

testData *MagicGamma test data*

Description

The data are MC generated (see below) to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. Cherenkov gamma telescope observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electromagnetic showers initiated by the gammas, and developing in the atmosphere. This Cherenkov radiation (of visible to UV wavelengths) leaks through the atmosphere and gets recorded in the detector, allowing reconstruction of the shower parameters. The available information consists of pulses left by the incoming Cherenkov photons on the photomultiplier tubes, arranged in a plane, the camera. Depending on the energy of the primary gamma, a total of few hundreds to some 10000 Cherenkov photons get collected, in patterns (called the shower image), allowing to discriminate statistically those caused by primary gammas (signal) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (background).

Typically, the image of a shower after some pre-processing is an elongated cluster. Its long axis is oriented towards the camera center if the shower axis is parallel to the telescope's optical axis, i.e. if the telescope axis is directed towards a point source. A principal component analysis is performed in the camera plane, which results in a correlation axis and defines an ellipse. If the depositions were distributed as a bivariate Gaussian, this would be an equidensity ellipse. The characteristic parameters of this ellipse (often called Hillas parameters) are among the image parameters that can

be used for discrimination. The energy depositions are typically asymmetric along the major axis, and this asymmetry can also be used in discrimination. There are, in addition, further discriminating characteristics, like the extent of the cluster in the image plane, or the total sum of depositions.

The data set was generated by a Monte Carlo program, Corsika, described in: D. Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998). [Web Link]

The program was run with parameters allowing to observe events with energies down to below 50 GeV.

Usage

```
testData
```

Format

1. fLength: continuous # major axis of ellipse [mm] 2. fWidth: continuous # minor axis of ellipse [mm] 3. fSize: continuous # 10-log of sum of content of all pixels [in #phot] 4. fConc: continuous # ratio of sum of two highest pixels over fSize [ratio] 5. fConc1: continuous # ratio of highest pixel over fSize [ratio] 6. fAsym: continuous # distance from highest pixel to center, projected onto major axis [mm] 7. fM3Long: continuous # 3rd root of third moment along major axis [mm] 8. fM3Trans: continuous # 3rd root of third moment along minor axis [mm] 9. fAlpha: continuous # angle of major axis with vector to origin [deg] 10. fDist: continuous # distance from origin to center of ellipse [mm] 11. class: g,h # gamma (signal), hadron (background)

g = gamma (signal): 12332 h = hadron (background): 6688

For technical reasons, the number of h events is underestimated. In the real data, the h class represents the majority of the events.

The simple classification accuracy is not meaningful for this data, since classifying a background event as signal is worse than classifying a signal event as background. For comparison of different classifiers an ROC curve has to be used. The relevant points on this curve are those, where the probability of accepting a background event as signal is below one of the following thresholds: 0.01, 0.02, 0.05, 0.1, 0.2 depending on the required quality of the sample of the accepted events for different experiments.

Source

Bock, R.K., Chilingarian, A., Gaug, M., Hakl, F., Hengstebeck, T., Jirina, M., Klaschka, J., Kotrc, E., Savicky, P., Towers, S., Vaicilius, A., Wittek W. (2004). Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. Nucl.Instr.Meth. A, 516, pp. 511-528.

P. Savicky, E. Kotrc. Experimental Study of Leaf Confidences for Random Forest. Proceedings of COMPSTAT 2004, In: Computational Statistics. (Ed.: Antoch J.) - Heidelberg, Physica Verlag 2004, pp. 1767-1774.

J. Dvorak, P. Savicky. Softening Splits in Decision Trees Using Simulated Annealing. Proceedings of ICANNGA 2007, Warsaw, (Ed.: Beliczynski et. al), Part I, LNCS 4431, pp. 721-729.

References

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

`trainData` *MagicGamma training data*

Description

The data are MC generated (see below) to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. Cherenkov gamma telescope observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electromagnetic showers initiated by the gammas, and developing in the atmosphere. This Cherenkov radiation (of visible to UV wavelengths) leaks through the atmosphere and gets recorded in the detector, allowing reconstruction of the shower parameters. The available information consists of pulses left by the incoming Cherenkov photons on the photomultiplier tubes, arranged in a plane, the camera. Depending on the energy of the primary gamma, a total of few hundreds to some 10000 Cherenkov photons get collected, in patterns (called the shower image), allowing to discriminate statistically those caused by primary gammas (signal) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (background).

Typically, the image of a shower after some pre-processing is an elongated cluster. Its long axis is oriented towards the camera center if the shower axis is parallel to the telescope's optical axis, i.e. if the telescope axis is directed towards a point source. A principal component analysis is performed in the camera plane, which results in a correlation axis and defines an ellipse. If the depositions were distributed as a bivariate Gaussian, this would be an equidensity ellipse. The characteristic parameters of this ellipse (often called Hillas parameters) are among the image parameters that can be used for discrimination. The energy depositions are typically asymmetric along the major axis, and this asymmetry can also be used in discrimination. There are, in addition, further discriminating characteristics, like the extent of the cluster in the image plane, or the total sum of depositions.

The data set was generated by a Monte Carlo program, Corsika, described in: D. Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998). [Web Link]

The program was run with parameters allowing to observe events with energies down to below 50 GeV.

Usage

```
trainData
```

Format

1. fLength: continuous # major axis of ellipse [mm] 2. fWidth: continuous # minor axis of ellipse [mm] 3. fSize: continuous # 10-log of sum of content of all pixels [in #phot] 4. fConc: continuous # ratio of sum of two highest pixels over fSize [ratio] 5. fConc1: continuous # ratio of highest pixel over fSize [ratio] 6. fAsym: continuous # distance from highest pixel to center, projected onto major axis [mm] 7. fM3Long: continuous # 3rd root of third moment along major axis [mm] 8.

fM3Trans: continuous # 3rd root of third moment along minor axis [mm] 9. fAlpha: continuous # angle of major axis with vector to origin [deg] 10. fDist: continuous # distance from origin to center of ellipse [mm] 11. class: g,h # gamma (signal), hadron (background)

g = gamma (signal): 12332 h = hadron (background): 6688

For technical reasons, the number of h events is underestimated. In the real data, the h class represents the majority of the events.

The simple classification accuracy is not meaningful for this data, since classifying a background event as signal is worse than classifying a signal event as background. For comparison of different classifiers an ROC curve has to be used. The relevant points on this curve are those, where the probability of accepting a background event as signal is below one of the following thresholds: 0.01, 0.02, 0.05, 0.1, 0.2 depending on the required quality of the sample of the accepted events for different experiments.

Source

Bock, R.K., Chilingarian, A., Gaug, M., Hakl, F., Hengstebeck, T., Jirina, M., Klaschka, J., Kotrc, E., Savicky, P., Towers, S., Vaicilius, A., Wittek W. (2004). Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope. Nucl.Instr.Meth. A, 516, pp. 511-528.

P. Savicky, E. Kotrc. Experimental Study of Leaf Confidences for Random Forest. Proceedings of COMPSTAT 2004, In: Computational Statistics. (Ed.: Antoch J.) - Heidelberg, Physica Verlag 2004, pp. 1767-1774.

J. Dvorak, P. Savicky. Softening Splits in Decision Trees Using Simulated Annealing. Proceedings of ICANNGA 2007, Warsaw, (Ed.: Beliczynski et. al), Part I, LNCS 4431, pp. 721-729.

References

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

`trainSample` *Sample data in parallel*

Description

Sample data or data and output in parallel: each core provides one sample of your desired size.

Usage

```
trainSample(data, numberCores = detectCores(), samplingSize = 0.2,
underSample = FALSE, toPredict = NULL, underSampleTarget = NULL,
sampleMethod = "bagging")
```

Arguments

data	A data frame, or structure convertable to a data frame, which you want to sample upon.
numberCores	In this setting equal to number of different training samples you are creating: one for each core you are using.
samplingSize	Size of your training sample in percentage.
underSample	Logical wether you want to take an undersample on your desired target.
toPredict	The column of your dataset you want to predict
underSampleTarget	When you set underSample to TRUE, underSampleTarget takes your target you want to keep in every sample. e.g. If you have 5 elements of category1 and 100 elements of category2 and your sampleSize is 0.2, then every sample will contain 25 elements, namely the 5 of category1 and 20 of category2.
sampleMethod	String which decides wether you sample on your observations (bagging) or on your variables (random).

Value

You get a list of length numberCores. Each core has created one item of your list, namely a data frame containing a a samplingSize size sample of data.

Author(s)

Wannes Rosiers

See Also

Under the hood this function uses foreach, and sample

Examples

```
## Not run:
# Create your data
x <- data.frame(1:10,10:1)

# Sampling on observations
trainSample(x,numberCores=2,samplingSize = 0.5)

#Create your data
data(iris)

# Sampling on variables
trainSample(iris,numberCores=2,samplingSize = 0.6,
            toPredict = "Species", sampleMethod = "random")

# Create your data
data(iris)
data <- iris[1:110,]
```

```
# Sampling
trainSamples <- trainSample(data,2,0.2,TRUE,"Species","virginica")

## End(Not run)
```

Index