Package 'parallelSVM'

June 26, 2015

Title A Parallel-Voting Version of the Support-Vector-Machine Algorithm

Version 0.1-9

Date 2015-06-26

Author Wannes Rosiers (InfoFarm)

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

Description
By sampling your data, running the Support-Vector-Machine 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 Support-Vector-Machine and possibly even more accurate predictions.

License GPL-2

Depends R (>= 2.14.0), e1071 (>= 1.6-3)

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 13:34:36

R topics documented:

`parallelSVM-package` *Parallel-voting version of Support-Vector-Machine*

Description

By sampling your data, running the Support-Vector-Machine 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 Support-Vector-Machine and possibly even more accurate predictions.

Details

Package:	parallelSVM
Type:	Package
Version:	1.0
Date:	2015-02-09
License:	GPL-2

This package consists of two main functions: `parallelSVM` A function which allows you to create multiple Support-Vector-Machine models: one for each core you provide. It returns a list of Support-Vector-Machine models. predict: An extension of the predict function, which uses the prediction of each Support-Vector-Machine model. When probability is TRUE, it returns the average of all predictions, otherwise it returns the class most models agree upon.

Author(s)

Wannes Rosiers

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

See Also

This package can be regarded as a parallel extension of `svm`

Examples

```
## Not run:
# Create your data
data(iris)
x <- subset(iris, select = -Species)
y <- iris$Species

# Create a model
model <- parallelSVM(x, y)

# Get prediction
predictions <- predict(model, x)
```

```
# Check the quality
table(predictions,y)

## End(Not run)
```

`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
```

parallelSVM *Parallel-voting version of Support-Vector-Machine*

Description

By sampling your data, running the Support-Vector-Machine 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 Support-Vector-Machine and possibly even more accurate predictions.

Usage

```
## S3 method for class 'formula'
## S3 method for class 'formula'
parallelSVM(formula, data= NULL, numberCores = detectCores(),
samplingSize = 0.2, ...,
subset, na.action = na.omit, scale = TRUE)
## Default S3 method
## Default S3 method:
parallelSVM(x, y = NULL, numberCores = detectCores(),
samplingSize = 0.2, scale = TRUE, type = NULL,
kernel = "radial", degree = 3,
gamma = if (is.vector(x)) 1 else 1/ncol(x),
coef0 = 0, cost = 1, nu = 0.5, class.weights = NULL,
cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE,
fitted = TRUE, seed = 1L, ..., subset, na.action = na.omit)
```

Arguments

formula	a symbolic description of the model to be fit
data	An optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.
x	A data matrix, a vactor ar a sparse matrix.
y	A response vector with one label for each row/component of x. Can be either a factor (for calssification tasks) or a numeric vector (for regression).
numberCores	Number of cores of your machine you want to use. Is set equal to the number of samples you take.
samplingSize	Size of your data or of x you will take in each sample.
scale	A logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.
type	Support-Vector-Machine can be used as a classification machine, as a regression machine, or for novelty detection. Depending of whether y is a factor or not, the default setting for type is C-classification or eps-regression, respectively,

	but may be overwritten by setting an explicit value. Valid options are: - C-classification - nu-classification - one-classification (for novelty detection) - eps-regression - nu-regression
kernel	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. - linear - polynomial - radial basis - sigmoid
degree	parameter needed for kernel of type polynomial (default: 3)
gamma	parameter needed for all kernels except linear (default: 1/(data dimension))
coef0	parameter needed for kernels of type polynomial and sigmoid (default: 0)
cost	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
nu	parameter needed for nu-classification, nu-regression, and one-classification
class.weights	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.
cachesize	cache memory in MB (default 40)
tolerance	tolerance of termination criterion (default: 0.001)
epsilon	epsilon in the insensitive-loss function (default: 0.1)
shrinking	option whether to use the shrinking-heuristics (default: TRUE)
cross	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression
probability	logical indicating whether the model should allow for probability predictions.
fitted	logical indicating whether the fitted values should be computed and included in the model or not (default: TRUE)
seed	integer seed for libsvm (used for cross-validation and probability prediction models).
...	additional parameters for the low level fitting function svm.default
subset	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
na.action	A function to specify the action to be taken if NAs are found. The default action is na.omit, which leads to rejection of cases with missing values on any required variable. An alternative is na.fail, which causes an error if NA cases are found. (NOTE: If given, this argument must be named.)

Value

A list containing of numberCores Support Vector Machine models.

Note

Usage is just like svm, the only difference is the numberCores you want to use (equal to the number of models you build), and the sampleSize (the size of the sample you want to use to create each model)

Author(s)

Wannes Rosiers

See Also

This package can be regarded as a parallel extension of svm.

Examples

```
## Not run:
# Load the normal svm function
library(e1071)

# Example with formula
# load trainData and testData
data(magicData)

# Calculate the model
# Here we use it on bigger data
system.time(serialSvm   <- svm(V11 ~ ., trainData[,-1],
probability=TRUE, cost=10, gamma=0.1))
system.time(parallelSvm <- parallelSVM(V11 ~ ., data = trainData[,-1],
numberCores = 8, samplingSize = 0.2,
probability = TRUE, gamma=0.1, cost = 10))

# Calculate predictions
system.time(serialPredictions <- predict(serialSvm, testData))
system.time(parallelPredicitions <- predict(parallelSvm, testData))

# Check for quality
table(serialPredictions,testData[,"V11"])
table(parallelPredicitions,testData[,"V11"])

# Example without formula
# load data
data(iris)
x <- subset(iris, select = -Species)
y <- iris$Species

# estimate model and predict input values
system.time(model       <- parallelSVM(x, y))
system.time(serialmodel <- svm(x, y))

fitted(model)
fitted(serialmodel)

# Calculate predictions
system.time(serialPredictions <- predict(serialmodel, x))
system.time(parallelPredicitions <- predict(model, x))

# Check for quality
table(serialPredictions,y)
```

```
table(parallelPredicitions,y)

## 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(x, y = NULL, numberCores = detectCores(), samplingSize = 0.2)
```

Arguments

`x`	A data frame, or structure convertable to a data frame, which you want to sample upon.
`y`	An vector containing a target variable for predictions later on. This target variable could be contained in x as well, then y is set to NULL.
`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.

Value

If y is null, 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 x. If y is not null, again you get a list of length numberCores. Each core has created one item of your list, namely:

`xSample`	A data frame containing a samplingSize size sample of x.
`ySample`	A vector with the corresponding y values (corresponding indices with x).

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)
y <- 1:10

# Sampling with provided y
trainSample(x,y,numberCores=2,samplingSize = 0.5)
```

```
# Sampling without provided y
trainSample(x,numberCores=2,samplingSize = 0.5)

## End(Not run)
```

Index