Package ‘Buddle’

October 12, 2022

Type Package

Title A Deep Learning for Statistical Classification and Regression Analysis with Random Effects

Version 2.0.1

Date 2020-02-04

Author Jiwoong Kim <jwboys26 at gmail.com>

Maintainer Jiwoong Kim <jwboys26@gmail.com>

Description Statistical classification and regression have been popular among various fields and stayed in the limelight of scientists of those fields. Examples of the fields include clinical trials where the statistical classification of patients is indispensable to predict the clinical courses of diseases. Considering the negative impact of diseases on performing daily tasks, correctly classifying patients based on the clinical information is vital in that we need to identify patients of the high-risk group to develop a severe state and arrange medical treatment for them at an opportune moment. Deep learning - a part of artificial intelligence - has gained much attention, and research on it burgeons during past decades: see, e.g, Kazemi and Mirroshandel (2018) <DOI:10.1016/j.artmed.2017.12.001>. It is a veritable technique which was originally designed for the classification, and hence, the Buddle package can provide sublime solutions to various challenging classification and regression problems encountered in the clinical trials. The Buddle package is based on the back-propagation algorithm - together with various powerful techniques such as batch normalization and dropout - which performs a multi-layer feed-forward neural network: see Krizhevsky et. al (2017) <DOI:10.1145/3065386>, Schmidhuber (2015) <DOI:10.1016/j.neunet.2014.09.003> and LeCun et al. (1998) <DOI:10.1109/5.726791> for more details. This package contains two main functions: TrainBuddle() and FetchBuddle(). TrainBuddle() builds a feed-forward neural network model and trains the model. FetchBuddle() recalls the trained model which is the output of TrainBuddle(), classifies or regresses given data, and make a final prediction for the data.

Depends R (>= 3.5.0)

License GPL-2

LazyData TRUE

Imports Rcpp (>= 0.12.17), plyr, stats, graphics

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.0.2

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-02-13 09:30:10 UTC

R topics documented:

`CheckNonNumeric` *Detecting Non-numeric Values.*

Description

Check whether or not an input matrix includes any non-numeric values (NA, NULL, "", character, etc) before being used for training. If any non-numeric values exist, then TrainBuddle() or FetchBuddle() will return non-numeric results.

Usage

```
CheckNonNumeric(X)
```

Arguments

`X`	an n-by-p matrix.

Value

A list of (n+1) values where n is the number of non-numeric values. The first element of the list is n, and all other elements are entries of X where non-numeric values occur. For example, when the (1,1)th and the (2,3)th entries of a 5-by-5 matrix X are non-numeric, then the list returned by CheckNonNumeric() will contain 2, (1,1), and (2,3).

See Also

GetPrecision(), FetchBuddle(), MakeConfusionMatrix(), OneHot2Label(), Split2TrainTest(), TrainBuddle()

Examples

```
n = 5;
p = 5;
X = matrix(0, n, p)       #### Generate a 5-by-5 matrix which includes two NA's.
X[1,1] = NA
X[2,3] = NA

lst = CheckNonNumeric(X)

lst
```

`FetchBuddle` *Predicting Classification and Regression.*

Description

Yield prediction (softmax value or value) for regression and classification for given data based on the results of training.

Usage

```
FetchBuddle(X, lW, lb, lParam)
```

Arguments

`X`	a matrix of real values which will be used for predicting classification or regression.
`lW`	a list of weight matrices obtained after training.
`lb`	a list of bias vectors obtained after training.
`lParam`	a list of parameters used for training. It includes: label, hiddenlayer, batch, drop, drop.ratio, lr, init.weight, activation, optim, type, rand.eff, distr, and disp.

Value

A list of the following values:

predicted predicted real values (regression) or softmax values (classification).

One.Hot.Encoding one-hot encoding values of the predicted softmax values for classification. For regression, a zero matrix will be returned. To convert the one-hot encoding values to labels, use OneHot2Label().

References

[1] Geron, A. Hand-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol: O'Reilly, 2017. Print.

[2] Han, J., Pei, J, Kamber, M. Data Mining: Concepts and Techniques. New York: Elsevier, 2011. Print.

[3] Weilman, S. Deep Learning from Scratch. O'Reilly Media, 2019. Print.

See Also

CheckNonNumeric(), GetPrecision(), MakeConfusionMatrix(), OneHot2Label(), Split2TrainTest(), TrainBuddle()

Examples

```
### Using mnist data again

data(mnist_data)

X1 = mnist_data$Images       ### X1: 100 x 784 matrix
Y1 = mnist_data$Labels       ### Y1: 100 x 1 vector



############################# Train Buddle

lst = TrainBuddle(Y1, X1, train.ratio=0.6, arrange=TRUE, batch.size=10, total.iter = 100,
                 hiddenlayer=c(20, 10), batch.norm=TRUE, drop=TRUE,
                 drop.ratio=0.1, lr=0.1, init.weight=0.1,
                 activation=c("Relu","SoftPlus"), optim="AdaGrad",
                 type = "Classification", rand.eff=TRUE, distr = "Logistic", disp=TRUE)

lW = lst[[1]]
lb = lst[[2]]
lParam = lst[[3]]


X2 = matrix(rnorm(20*784,0,1), 20,784)  ## Genderate a 20-by-784 matrix

lst = FetchBuddle(X2, lW, lb, lParam)   ## Pass X2 to FetchBuddle for prediction
```

`GetPrecision` *Obtaining Accuracy.*

Description

Compute measures of accuracy such as precision, recall, and F1 from a given confusion matrix.

Usage

```
GetPrecision(confusion.matrix)
```

Arguments

`confusion.matrix`	a confusion matrix.

Value

An (r+1)-by-3 matrix when the input is an r-by-r confusion matrix.

See Also

CheckNonNumeric(), FetchBuddle(), MakeConfusionMatrix(), OneHot2Label(), Split2TrainTest(), TrainBuddle()

Examples

```
data(iris)

Label = c("setosa", "versicolor", "virginica")

predicted.label = c("setosa", "setosa",   "virginica", "setosa", "versicolor", "versicolor")
true.label    = c("setosa", "virginica", "versicolor","setosa", "versicolor", "virginica")

confusion.matrix = MakeConfusionMatrix(predicted.label, true.label, Label)
precision = GetPrecision(confusion.matrix)

confusion.matrix
precision
```

`MakeConfusionMatrix` *Making a Confusion Matrix.*

Description

Create a confusion matrix from two vectors of labels: predicted label obtained from FetchBuddle() as a result of prediction and true label of a test set.

Usage

```
MakeConfusionMatrix(predicted.label, true.label, Label)
```

Arguments

`predicted.label`	a vector of predicted labels.
`true.label`	a vector of true labels.
`Label`	a vector of all possible values or levels which a label can take.

Value

An r-by-r confusion matrix where r is the length of Label.

See Also

CheckNonNumeric(), GetPrecision(), FetchBuddle(), OneHot2Label(), Split2TrainTest(), TrainBuddle()

Examples

```
data(iris)

Label = c("setosa", "versicolor", "virginica")

predicted.label = c("setosa", "setosa",   "virginica", "setosa", "versicolor", "versicolor")
true.label    = c("setosa", "virginica", "versicolor","setosa", "versicolor", "virginica")

confusion.matrix = MakeConfusionMatrix(predicted.label, true.label, Label)
precision = GetPrecision(confusion.matrix)

confusion.matrix
precision
```

`mnist_data` *Image data of handwritten digits.*

Description

A dataset containing 100 images of handwritten digits.

Usage

```
data(mnist_data)
```

Details

#'@format A list containing a matrix of image data and a vector of labels:

Images 100-by-784 matrix of image data of handwritten digits.

Labels 100-by-1 vector of labels of handwritten digits.

Source

http://yann.lecun.com/exdb/mnist/

Examples

```
data(mnist_data)

Img_Mat = mnist_data$Images
Img_Label = mnist_data$Labels

digit_data = Img_Mat[1, ]    ### image data (784-by-1 vector) of the first handwritten digit (=5)
label = Img_Label[1]           ### label of the first handwritten digit (=5)
imgmat = matrix(digit_data, 28, 28)    ### transform the vector of image data to a matrix
```

`OneHot2Label` *Obtaining Labels*

Description

Convert a one-hot encoding matrix to a vector of labels.

Usage

```
OneHot2Label(OHE, Label)
```

Arguments

`OHE`	an r-by-n one-hot encoding matrix.
`Label`	an r-by-1 vector of values or levels which a label can take.

Value

An n-by-1 vector of labels.

See Also

CheckNonNumeric(), GetPrecision(), FetchBuddle(), MakeConfusionMatrix(), Split2TrainTest(), TrainBuddle()

`Split2TrainTest` *Splitting Data into Training and Test Sets.*

Description

Convert data into training and test sets so that the training set contains approximately the specified ratio of all labels.

Usage

```
Split2TrainTest(Y, X, train.ratio)
```

Arguments

`Y`	an n-by-1 vector of responses or labels.
`X`	an n-by-p design matrix of predictors.
`train.ratio`	a ratio of the size of the resulting training set to the size of data.

Value

A list of the following values:

y.train the training set of Y.

y.test the test set of Y.

x.train the training set of X.

x.test the test set of X.

See Also

CheckNonNumeric(), GetPrecision(), FetchBuddle(), MakeConfusionMatrix(), OneHot2Label(), TrainBuddle()

Examples

```
data(iris)

Label = c("setosa", "versicolor", "virginica")


train.ratio=0.8
Y = iris$Species
X = cbind( iris$Sepal.Length, iris$Sepal.Width, iris$Petal.Length, iris$Petal.Width)

lst = Split2TrainTest(Y, X, train.ratio)

Ytrain = lst$y.train
Ytest = lst$y.test

length(Ytrain)
length(Ytest)

length(which(Ytrain==Label[1]))
length(which(Ytrain==Label[2]))
length(which(Ytrain==Label[3]))

length(which(Ytest==Label[1]))
length(which(Ytest==Label[2]))
length(which(Ytest==Label[3]))
```

`TrainBuddle` *Implementing Statistical Classification and Regression.*

Description

Build a multi-layer feed-forward neural network model for statistical classification and regression analysis with random effects.

Usage

```
TrainBuddle(
  formula.string,
  data,
  train.ratio = 0.7,
  arrange = 0,
  batch.size = 10,
  total.iter = 10000,
  hiddenlayer = c(100),
  batch.norm = TRUE,
  drop = TRUE,
  drop.ratio = 0.1,
  lr = 0.1,
  init.weight = 0.1,
  activation = c("Sigmoid"),
  optim = "SGD",
  type = "Classification",
  rand.eff = FALSE,
  distr = "Normal",
  disp = TRUE
)
```

Arguments

`formula.string`	a formula string or a vector of numeric values. When it is a string, it denotes a classification or regression equation, of the form label ~ predictors or response ~ predictors, where predictors are separated by + operator. If it is a numeric vector, it will be a label or a response variable of a classification or regression equation, respectively.
`data`	a data frame or a design matrix. When formula.string is a string, data should be a data frame which includes the label (or the response) and the predictors expressed in the formula string. When formula.string is a vector, i.e. a vector of labels or responses, data should be an nxp numeric matrix whose columns are predictors for further classification or regression.
`train.ratio`	a ratio that is used to split data into training and test sets. When data is an n-by-p matrix, the resulting train data will be a (train.ratio x n)-by-p matrix. The default is 0.7.
`arrange`	a logical value to arrange data for the classification only (automatically set up to FALSE for regression) when splitting data into training and test sets. If it is true, data will be arranged for the resulting training set to contain the specified ratio (train.ratio) of labels of the whole data. See also Split2TrainTest().
`batch.size`	a batch size used for training during iterations.
`total.iter`	a number of iterations used for training.
`hiddenlayer`	a vector of numbers of nodes in hidden layers.
`batch.norm`	a logical value to specify whether or not to use the batch normalization option for training. The default is TRUE.
`drop`	a logical value to specify whether or not to use the dropout option for training. The default is TRUE.
`drop.ratio`	a ratio for the dropout; used only if drop is TRUE. The default is 0.1.
`lr`	a learning rate. The default is 0.1.
`init.weight`	a weight used to initialize the weight matrix of each layer. The default is 0.1.
`activation`	a vector of activation functions used in all hidden layers. For two hidden layers (e.g., hiddenlayer=c(100, 50)), it is a vector of two activation functions, e.g., c("Sigmoid", "SoftPlus"). The list of available activation functions includes

	Sigmoid, Relu, LeakyRelu, TanH, ArcTan, ArcSinH, ElliotSig, SoftPlus, BentIdentity, Sinusoid, Gaussian, Sinc, and Identity. For details of the activation functions, please refer to Wikipedia.
`optim`	an optimization method which is used for training. The following methods are available: "SGD", "Momentum", "AdaGrad", "Adam", "Nesterov", and "RMSprop."
`type`	a statistical model for the analysis: "Classification" or "Regression."
`rand.eff`	a logical value to specify whether or not to add a random effect into classification or regression.
`distr`	a distribution of a random effect; used only if rand.eff is TRUE. The following distributions are available: "Normal", "Exponential", "Logistic", and "Cauchy."
`disp`	a logical value which specifies whether or not to display intermediate training results (loss and accuracy) during the iterations.

Value

A list of the following values:

lW a list of n terms of weight matrices where n is equal to the number of hidden layers plus one.

lb a list of n terms of bias vectors where n is equal to the number of hidden layers plus one.

lParam a list of parameters used for the training process.

train.loss a vector of loss values of the training set obtained during iterations where its length is eqaul to number of epochs.

train.accuracy a vector of accuracy values of the training set obtained during during iterations where its length is eqaul to number of epochs.

test.loss a vector of loss values of the test set obtained during the iterations where its length is eqaul to number of epochs.

test.accuracy a vector of accuracy values of the test set obtained during the iterations where its length is eqaul to number of epochs.

predicted.softmax an r-by-n numeric matrix where r is the number of labels (classification) or 1 (regression), and n is the size of the test set. Its entries are predicted softmax values (classification) or predicted values (regression) of the test sets, obtained by using the weight matrices (lW) and biases (lb).

predicted.encoding an r-by-n numeric matrix which is a result of one-hot encoding of the predicted.softmax; valid for classification only.

confusion.matrix an r-by-r confusion matrix; valid classification only.

precision an (r+1)-by-3 matrix which reports precision, recall, and F1 of each label; valid classification only.

References

[1] Geron, A. Hand-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol: O'Reilly, 2017. Print.

[2] Han, J., Pei, J, Kamber, M. Data Mining: Concepts and Techniques. New York: Elsevier, 2011. Print.

[3] Weilman, S. Deep Learning from Scratch. O'Reilly Media, 2019. Print.

See Also

CheckNonNumeric(), GetPrecision(), FetchBuddle(), MakeConfusionMatrix(), OneHot2Label(), Split2TrainTest()

Examples

```
####################
# train.ratio = 0.6                    ## 60% of data is used for training
# batch.size = 10
# total.iter = 100
# hiddenlayer=c(20,10)                ## Use two hidden layers
# arrange=TRUE                         #### Use "arrange" option
# activations = c("Relu","SoftPlus")   ### Use Relu and SoftPlus
# optim = "Nesterov"                  ### Use the "Nesterov" method for the optimization.
# type = Classification
# rand.eff = TRUE                      #### Add some random effect
# distr="Normal"                      #### The random effect is a normal random variable
# disp = TRUE                         #### Display intemeidate results during iterations.


data(iris)

lst = TrainBuddle("Species~Sepal.Width+Petal.Width", iris, train.ratio=0.6,
         arrange=TRUE, batch.size=10, total.iter = 100, hiddenlayer=c(20, 10),
         batch.norm=TRUE, drop=TRUE, drop.ratio=0.1, lr=0.1, init.weight=0.1,
         activation=c("Relu","SoftPlus"), optim="Nesterov",
         type = "Classification", rand.eff=TRUE, distr = "Normal", disp=TRUE)

lW = lst$lW
lb = lst$lb
lParam = lst$lParam

confusion.matrix = lst$confusion.matrix
precision = lst$precision

confusion.matrix
precision


### Another classification example
### Using mnist data


data(mnist_data)

Img_Mat = mnist_data$Images
Img_Label = mnist_data$Labels

                             ##### Use 100 images

X = Img_Mat                   ### X: 100 x 784 matrix
Y = Img_Label                 ### Y: 100 x 1 vector
```

```
lst = TrainBuddle(Y, X, train.ratio=0.6, arrange=TRUE, batch.size=10, total.iter = 100,
                 hiddenlayer=c(20, 10), batch.norm=TRUE, drop=TRUE,
                 drop.ratio=0.1, lr=0.1, init.weight=0.1,
                 activation=c("Relu","SoftPlus"), optim="AdaGrad",
                 type = "Classification", rand.eff=TRUE, distr = "Logistic", disp=TRUE)


confusion.matrix = lst$confusion.matrix
precision = lst$precision

confusion.matrix
precision







###############   Regression example


n=100
p=10
X = matrix(rnorm(n*p, 1, 1), n, p)  ## X is a 100-by-10 design matrix
b = matrix( rnorm(p, 1, 1), p,1)
e = matrix(rnorm(n, 0, 1), n,1)
Y = X %*% b + e                     ### Y=X b + e
######### train.ratio=0.7
######### batch.size=20
######### arrange=FALSE
######### total.iter = 100
######### hiddenlayer=c(20)
######### activation = c("Identity")
######### "optim" = "Adam"
######### type = "Regression"
######### rand.eff=FALSE

lst = TrainBuddle(Y, X, train.ratio=0.7, arrange=FALSE, batch.size=20, total.iter = 100,
                 hiddenlayer=c(20), batch.norm=TRUE, drop=TRUE, drop.ratio=0.1, lr=0.1,
                 init.weight=0.1, activation=c("Identity"), optim="AdaGrad",
                 type = "Regression", rand.eff=FALSE, disp=TRUE)
```

Index