Package 'DeepGS'

June 8, 2018

Title Predicting phenotypes from genotypes using Deep Learning

Version 1.2

Author Chuang Ma, Zhixu Qiu, Qian Cheng and Wenlong Ma

Maintainer Zhixu Qiu <zhixuqiu2015@gmail.com>, Chuang Ma

<chuangma2006@gmail.com>

Description The R package 'DeepGS' can be used to perform genomic selection (GS), which is a promising breeding strategy in plants and animals. DeepGS predicts phenotypes using genome-wide genotypic markers with an advanced machine learning technique (deep learning). The effectiveness of DeepGS has been demonstrated in predicting eight phenotypic traits on a population of 2,000 Iranian bread wheat (Triticum aestivum) lines from the wheat gene bank of the International Maize and Wheat Improvement Center (CIMMYT). Further, ensemble learning based on particle swarm optimization (ELBPSO) can be used linearly combining the predictions of different GS models.

Depends R (>= 3.0.0)

Suggests mxnet

License GPL-2|GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

R topics documented:

cvSampleIndex *Generate Sample Indices for Training Sets and Testing Sets*

Description

This function generates indices for samples in training and testing sets for performing the N-fold cross validation experiment.

Usage

```
cvSampleIndex(sampleNum, cross = 5, seed = 1, randomSeed = FALSE)
```

Arguments

sampleNum	The number of samples needed to be partitioned into training and testing sets.
cross	The fold of cross validation.
seed	An integer used as the seed for data partition. The default value is 1.
randomSeed	Logical variable. The default value is FALSE.

Value

A list and each element including $trainIdx, $testIdx and $cvIdx.

$trainIdx The index of training samples.

$testIdx The index of testing samples.

$cvIdx The index of cross validation.

Author(s)

Chuang Ma, Zhixu Qiu, Qian Cheng and Wenlong Ma

Examples

```
#' ## Load example data ##
data(wheat_example)
## 5-fold cross validation
b <- cvSampleIndex(sampleNum = 2000, cross = 5, seed = 1)
```

ELBPSO *Ensemble Learning Based on Particle Swarm Optimization (ELBPSO)*

Description

The algorithm combines the predicted phenotypic values from different GS models and return the best weights between these models.

Usage

```
ELBPSO(rep_times = 100, interation_times, weight_dimension, weight_min,
  weight_max, rate_min, rate_max, paticle_number, pred_matrix, IW = 1,
  AF1 = 2, AF2 = 2)
```

Arguments

rep_times	ELBPSO repeat times default is 100.
interation_times	Each ELBPSO optimized iteration times.
weight_dimension	Weight dimension corresponding to the number of GS models.
weight_min	Minimum weight.
weight_max	Maximum weight.
rate_min	Minimum update rate.
rate_max	Maximum update rate.
paticle_number	Patricle number.
pred_matrix	The first column represents the real values the other columns represent the predictd values of other GS models.
IW	Inertia weight default is 1.
AF1	Accelerated factor 1 default is 2.
AF2	Accelerated factor 2 default is 2.

Examples

```
# Not run
# library(DeepGS)
# library(rrBLUP)
# data("wheat_example")
# Markers <- wheat_example$Markers
# y <- wheat_example$y
# cvSampleList <- cvSampleIndex(length(y),10,1)
# # select one fold
# cvIdx <- 1
# trainIdx <- cvSampleList[[cvIdx]]$trainIdx
# testIdx <- cvSampleList[[cvIdx]]$testIdx
```

```
# trainMat = Markers[trainIdx,]
# trainPheno = y[trainIdx]
# validIdx <- sample(1:length(trainIdx),floor(length(trainIdx)*0.1))
# validMat <- trainMat[validIdx,]
# validPheno <- trainPheno[validIdx]
# testMat = Markers[testIdx,]
# testPheno = y[testIdx]
# # design DeepGS architecture
# conv_kernel <- c("1*18") # convolution kernels (fileter shape)
# conv_stride <- c("1*1")
# conv_num_filter <- c(8)  # number of filters
# pool_act_type <- c("relu") # active function for next pool
# pool_type <- c("max") # max pooling shape
# pool_kernel <- c("1*4") # pooling shape
# pool_stride <- c("1*4") # number of pool kernerls
# fullayer_num_hidden <- c(32,1)
# fullayer_act_type <- c("sigmoid")
# drop_float <- c(0.2,0.1,0.05)
# cnnFrame <- list(conv_kernel =conv_kernel,conv_num_filter = conv_num_filter,
#                  conv_stride = conv_stride,pool_act_type = pool_act_type,
#                  pool_type = pool_type,pool_kernel =pool_kernel,
#                  pool_stride = pool_stride,fullayer_num_hidden= fullayer_num_hidden,
#                  fullayer_act_type = fullayer_act_type,drop_float = drop_float)
#
# markerImage = paste0("1*",ncol(trainMat))
# # train DeepGS model
# DeepGS_obj <- train_deepGSModel(trainMat = trainMat,trainPheno = trainPheno,
#                           validMat = validMat,validPheno = validPheno, markerImage = markerImage,
#                           cnnFrame = cnnFrame,device_type = "cpu",gpuNum = 1, eval_metric = "mae",
#                             num_round = 6000,array_batch_size= 30,learning_rate = 0.01,
#                           momentum = 0.5,wd = 0.00001, randomseeds = 0,initializer_idx = 0.01,
#                                 verbose =TRUE)
# # make predictions based on the trained model
# DeepGS_pred <- predict_GSModel(GSModel = DeepGS_obj,testMat = Markers[testIdx,],
#                                markerImage = markerImage )
# # train RR-BLUP model#'
# rrBLUP_obj <-mixed.solve(trainPheno, Z=trainMat, K=NULL, SE = FALSE, return.Hinv=FALSE)
# # make predictions based on the trained model
# rrBLUP_pred <-  testMat %*% rrBLUP_obj$u + as.numeric(rrBLUP_obj$beta )
# # prepare the prediction matrix
# test_predMat <- cbind(t(DeepGS_pred), rrBLUP_pred)
# train_predMat <- cbind(testPheno, t(DeepGS_pred), rrBLUP_pred)
# colnames(train_predMat) <- c("real", "DeepGS", "RR-BLUP")
## End not run
# calculating the weight of different training model by using their predict socres

test_datapath <- system.file("exdata", "test_ELBPSO.RData",
                             package = "DeepGS")
load(test_datapath)

weight <- ELBPSO(rep_times = 100,interation_times = 25,weight_dimension = 2,
                 weight_min = 0,weight_max=1,rate_min = -0.01,rate_max = 0.01,
                 paticle_number = 10, pred_matrix = train_predMat,IW = 1,
```

```
                AF1 = 2, AF2 = 2)

ensemble_pred <- (test_predMat %*% weight)/sum(weight)
predMat <- cbind(testPheno, t(DeepGS_pred), rrBLUP_pred, ensemble_pred)
colnames(predMat) <- c("real", "DeepGS", "RR-BLUP", "ensemble")
cor(predMat)
```

---

meanNDCG *Calculate mean Normalied Doscounted Cumulative Gain*

Description

This function calculates the mean normalied doscounted cumulative gain(meanNDCG) value for evaluting the prediction performance of a genomic selection prediction model in selecting top k individuals with high breeding value.

Usage

```
meanNDCG(realScores, predScores, topAlpha = c(10))
```

Arguments

realScores	A numeric vector is the real breeding values of the validation individual for a trait.
predScores	A numeric vector or matrix is the prediction breeding value predicted by genomic selection model of the individuals.
topAlpha	A numeric vector (one or mutliple arguments) is the percentage of excellent individuals, default 10.

Author(s)

Chuang Ma, Zhixu Qiu, Qian Cheng and Wenlong Ma

Examples

```
## Not run
refer_value <- runif(100)
pred_value <- sin(refer_value) + cos(refer_value)
meanNDCG(realScores = refer_value,predScores = pred_value, topAlpha = c(10,20,30))
## End not run
```

predict_GSModel *Predict GSModel*

Description

Predict trait values using trained deep learning genomic selection prediction model.

Usage

```
predict_GSModel(GSModel, testMat, markerImage)
```

Arguments

GSModel	Trained prediction model obtained from the DeepGSModel function.
testMat	A genotype matrix (T * M; T individuals, M markers)
markerImage	(String) This gives a "i * j" image format that the (M x1) markers informations of each individual will be encoded.

Author(s)

Chuang Ma, Qian Cheng, Zhixu Qiu and Wenlong Ma

train_deepGSModel *Build a genomic selection prediction model using the deep learning technique*

Description

The function applies the deep convolutional neural network to build a prediction model for genomic selection.

Usage

```
train_deepGSModel(trainMat, trainPheno, validMat, validPheno, markerImage,
  cnnFrame, device_type = "cpu", gpuNum = "max", eval_metric = "mae",
  num_round = 6000, array_batch_size = 30, learning_rate = 0.01,
  momentum = 0.5, wd = 1e-05, randomseeds = NULL,
  initializer_idx = 0.01, verbose = TRUE...)
```

Arguments

trainMat	A genotype matrix (N x M; N individuals, M markers) for training model.
trainPheno	Vector (N * 1) of phenotype for training model.
validMat	A genotype matrix for valiting trained model.
validPheno	Vector (N * 1) of phenotype for valiting trained model.
markerImage	(String) This gives a "i * j" image format that the (M x1) markers informations of each individual will be encoded. if the image size exceeds the original snp number, 0 will be polished the lack part, if the image size is less than the original snp number, the last snp(s) will be descaled.
cnnFrame	A list containing the following element for convolutional neural network (CNN) framework: • conv_kernel: A vector (K * 1) gives convolutional kernel sizes (width x height) to filter image matrix for K convolutional layers, respectively. • conv_num_filter: A vector (K * 1) gives number of convolutional kernels for K convolutional layers, respectively. • pool_act_type: A vector (K * 1) gives types of active function will define outputs of K convolutional layers which will be an input of corresponding pool layer, respectively. It include "relu", "sigmoid", "softrelu" and "tanh". • conv_stride: A character (K * 1) strides for K convolutional kernel. • pool_type: A character (K * 1) types of K pooling layers select from "avg", "max", "sum", respectively. • pool_kernel: A character (K * 1) K pooling kernel sizes (width * height) for K pooling layers. • pool_stride: A Character (K * 1) strides for K pooling kernels. • fullayer_num_hidden: A numeric (H * 1) number of hidden neurons for H full connected layers, respectively. The last full connected layer's number of hidden nerurons must is one. • fullayer_act_type: A numeric ((H-1) * 1) selecting types of active function from "relu", "sigmoid", "softrelu" and "tanh" for full connected layers. • drop_float: Numeric.
device_type	Selecting "cpu" or "gpu" device to construct predict model.
gpuNum	(Integer) Number of GPU devices, if using multiple GPU (gpuNum > 1), the parameter momentum must greater than 0.
eval_metric	(String) A approach for evaluating the performance of training process, it include "mae", "rmse" and "accuracy", default "mae".
num_round	(Integer) The number of iterations over training data to train the model, default = 10.
array_batch_size	(Integer) It defines number of samples that going to be propagated through the network for each update weight, default 128.
learning_rate	The learn rate for training process.
momentum	(Float, 0~1) Momentum for moving average, default 0.9.
wd	(Float, 0~1) Weight decay, default 0.

randomseeds	Set the seed used by mxnet device-specific random number.
initializer_idx	The initialization scheme for parameters.
verbose	logical (default=TRUE) Specifies whether to print information on the iterations during training.
...	Parameters for construncting neural networks used in package "mxnet" (http://mxnet.io/).

Author(s)

Chuang Ma , Zhixu Qiu, Qian Cheng and Wenlong Ma

Examples

```
data(wheat_example)
Markers <- wheat_example$Markers
y <- wheat_example$y
cvSampleList <- cvSampleIndex(length(y),10,1)
# cross validation set
cvIdx <- 1
trainIdx <- cvSampleList[[cvIdx]]$trainIdx
testIdx <- cvSampleList[[cvIdx]]$testIdx
trainMat <- Markers[trainIdx,]
trainPheno <- y[trainIdx]
validIdx <- sample(1:length(trainIdx),floor(length(trainIdx)*0.1))
validMat <- trainMat[validIdx,]
validPheno <- trainPheno[validIdx]
trainMat <- trainMat[-validIdx,]
trainPheno <- trainPheno[-validIdx]
conv_kernel <- c("1*18") ## convolution kernels (fileter shape)
conv_stride <- c("1*1")
conv_num_filter <- c(8)  ## number of filters
pool_act_type <- c("relu") ## active function for next pool
pool_type <- c("max") ## max pooling shape
pool_kernel <- c("1*4") ## pooling shape
pool_stride <- c("1*4") ## number of pool kernerls
fullayer_num_hidden <- c(32,1)
fullayer_act_type <- c("sigmoid")
drop_float <- c(0.2,0.1,0.05)
cnnFrame <- list(conv_kernel =conv_kernel,conv_num_filter = conv_num_filter,
                 conv_stride = conv_stride,pool_act_type = pool_act_type,
                 pool_type = pool_type,pool_kernel =pool_kernel,
                 pool_stride = pool_stride,fullayer_num_hidden= fullayer_num_hidden,
                 fullayer_act_type = fullayer_act_type,drop_float = drop_float)

markerImage = paste0("1*",ncol(trainMat))

trainGSmodel <- train_deepGSModel(trainMat = trainMat,trainPheno = trainPheno,
                validMat = validMat,validPheno = validPheno, markerImage = markerImage,
                cnnFrame = cnnFrame,device_type = "cpu",gpuNum = 1, eval_metric = "mae",
                num_round = 6000,array_batch_size= 30,learning_rate = 0.01,
```

```
                momentum = 0.5,wd = 0.00001, randomseeds = 0,initializer_idx = 0.01,
                verbose =TRUE)
predscores <- predict_GSModel(GSModel = trainGSmodel,testMat = Markers[testIdx,],
              markerImage = markerImage )
```

---

wheat_example *Run a examples for an in-development function.*

Description

A list including:

- Marker: A matrix (599 * 1225), each row represent 1,225 markers information for one individuals.
- y: The real phenotype value for each individual.

Usage

```
data(wheat_example)
```

Index