Package 'downscaledl'

October 13, 2022

Type Package

Title Downscale of RS Images using Deep Learning

Version 1.0

Date 2019-10-11

Author Lianfa Li

Maintainer Lianfa Li <lspatial@gmail.com>

Description Downscaling the coarse-resolution remote sensing images into fined-resolution images is important for preprocessing. This package provides the functions for this with high accuracy. Currently, the resautonet is used as the deep learning algorithm for downscaling. This work was done based on this paper, Lianfa Li (2019) <doi:10.3390/rs11111378>.

Depends R (>= 3.1)

Imports Rcpp (>= 1.0.0),rstack,dplyr,magrittr,keras,raster,parallel,tensorflow,sp,rgdal

LinkingTo Rcpp, RcppArmadillo

SystemRequirements C++11

License GPL

Encoding UTF-8

LazyData true

NeedsCompilation yes

RoxygenNote 6.1.1

Repository CRAN

Date/Publication 2019-10-24 16:20:10 UTC

R topics documented:

`downscaledl-package` *Downscale of RS Images using Deep Learning*

Description

Downscaling the coarse-resolution remote sensing images into fined-resolution images is important for preprocessing. This package provides the functions for this with high accuracy. Currently, the resautonet is used as the deep learning algorithm for downscaling. This work was done based on this paper, Lianfa Li (2019) <doi:10.3390/rs11111378>.

Details

This package provides the downscaling functions for the coarse-resolution remote sensing images. Currently, we used the deep learning algorithm as the base learner in downscaling.

The major functions defined in this package include:

`AutoEncoderModel`:The residual network function used in downscaling;

`ResautoDownscale`:The major downscaling function that uses `AutoEncoderModel`;

`r2_squ`:The function calculating R2 ;

`rmse`:The function calculate RMSE.

Author(s)

Lianfa Li

Maintainer: Lianfa Li <lspatial@gmail.com>

References

~~ Literature or other references for background information ~~

Examples

```
#Load the high-resolution raster of a covariate,
# elevation to be used for downscaling
eleFile=file.path(system.file(package = "downscaledl"), "extdata", "sample_ele.tif")
ele=raster::raster(eleFile)

#Load the coarse-resolution raster of the target variable to be downscaled
```

```
coarseFile=file.path(system.file(package = "downscaledl"), "extdata", "sample_coarse_res.tif")
 coarseRst=raster::raster(coarseFile)

 #Extract x and y to be used as two predictors in downscaling
 xRast=ele
 yRast=ele
 pos=raster::rasterToPoints(ele,spatial=TRUE)
 cell=raster::cellFromXY(xRast,pos)
 xyc=sp::coordinates(pos)
 xRast[cell]=xyc[,"x"]
 yRast[cell]=xyc[,"y"]

 #Merge the covariates
 covStk=raster::stack(xRast,yRast,ele)
 names(covStk)=c("x","y","ele")

 #Use the fine-resolution covariate (elevation) as the target image
 fineTarget=ele

 #Set the paramneters and start to downscale ...
ares=ResautoDownscale(covStk,fineTarget,coarseRst,ss= 0.2, cores= 5, thresh = 0.01,ntime=3)

 #Show the iteration results
 message(paste(capture.output(ares$diogRMSE), collapse = "\n",sep=""))

 #Show the optimal results in the final predictions
 message(paste("test R2:",round(ares$r2,2),",test RMSE:",round(ares$rmse,4),sep=""))

 #Obtain the downscaled
 downscaled_img=ares$raster

 #Save the current par setting
 curpar = par(no.readonly = TRUE)

 #Set the new par setting
 par(mfrow=c(1,2),mar=c(1,1,1,1))

 #Show the final predictions of fine resolution and
 # original coarse-resolution image for a comparison
 raster::plot(coarseRst)
 raster::plot(downscaled_img)

 #Restore the previous par setting
 par(curpar)
```

AutoEncoderModel *AutoEncoderModel*

Description

This function is to construct a residual autoencoder-based deep network.

Usage

```
AutoEncoderModel(nfea, nout, nodes, acts, mdropout = 0, reg = NULL,
  batchnorm = TRUE, isres = TRUE, outtype = 0, fact = "linear")
```

Arguments

nfea	integer, Number of features
nout	integer, Number of output units
nodes	list of integers, list of the number of nodes for the hidden layers in encoding component
acts	list of strings, list of activation function names
mdropout	double, dropout rate of the coding (middle) layer (default:0)
reg	string, regularization string (default: NULL)
batchnorm	bool, flag to conduct batch normalization (default:TRUE)
isres	bool, flag to conduct the residual connections (default: TRUE)
outtype	integer, output type, 0 indicating nout outputs and 1 nout+nfea outputs (default: 0)
fact	string, activation for output layer (default:"linear")

Value

keras model, model of (residual) autoencoder-based deep network

See Also

hiddenBlock for the internal function used in this function.

Examples

```
# This is an example to simulate a dataset to demonstrate use of autoencoder

#Set the sample size as 1000 for the simulated dataset
n=1000

#Get the simulated data using random functions
dataDf=data.frame(id=c(1:n),x1=runif(n),x2=rnorm(n,100,10),
           x3=runif(n,100,200),x4=rnorm(n,1000,30))

#Set the proportion of the test samples
testProp=0.1
ntest=as.integer(n*testProp)
ntrain=n-ntest

#Obtain the index for the training and testing samples
index_train=sample(c(1:n),ntrain)
index_test=setdiff(c(1:n),index_train)

#Obtain y as analytic solution for x plus random noise
```

```
dataDf$y=sqrt(dataDf$x1)+dataDf$x2^0.3+log(dataDf$x3)+dataDf$x4^2+rnorm(n)

#Scale the dataset
scalev = scale(dataDf[,c(2:6)])
col_means = attr(scalev, "scaled:center")
col_stddevs = attr(scalev, "scaled:scale")


#Set the early stopping and learning rate adjustement functions
early_stopping = keras::callback_early_stopping(monitor ='loss', min_delta=0.000001)
reduce=keras::callback_reduce_lr_on_plateau(patience=20)

#Set the parameters
nfea=4;nout=1;nodes=c(32,16,8,4);mdropout=0.2;isres=TRUE;outtype=0;fact="linear"
acts=rep("relu",length(nodes));fact="linear";reg=NULL;batchnorm=TRUE

#Define the residual autoencoder and show its network structure
autoresmodel=AutoEncoderModel(nfea,nout,nodes,acts,mdropout,reg,batchnorm,isres,outtype,fact=fact)
#summary(autoresmodel) #Optional function to show the model

#Define the loss function and compile the models
metric_r2= keras::custom_metric("rsquared", function(y_true, y_pred) {
  SS_res =keras::k_sum(keras::k_square(y_true-y_pred ))
  SS_tot =keras::k_sum(keras::k_square(( y_true - keras::k_mean(y_true))))
  return ( 1 - SS_res/(SS_tot + keras::k_epsilon()))
})
keras::compile(autoresmodel,
  loss = "mean_squared_error",
  optimizer = keras::optimizer_rmsprop(),
  metrics = c("mean_squared_error",metric_r2)
)

#Set the number of maximum epochs
nepoch=70

# Set the train samples and train the model
x_train=scalev[index_train,c(1:4)]
y_train=scalev[index_train,5]
history = keras::fit(autoresmodel,x_train, y_train,
  epochs = nepoch, batch_size = 20,
  validation_split = 0.2,verbose=1,callbacks=list(early_stopping,reduce)
)

# Show the training curves
trainLoss=data.frame(r2=history$metrics$rsquared)
trainLoss$epoch=c(1:length(history$metrics$rsquared))
trainLoss$val_r2=history$metrics$val_rsquared

#Save the current par setting
curpar = par(no.readonly = TRUE)

#Set the new par setting and make the plots
par(mar=c(4,4,1,1))
```

```
plot(trainLoss$epoch,trainLoss$r2,type="l",
     xlab="Training epoch",ylab=expression("R"^2))
lines(trainLoss$epoch,trainLoss$val_r2,col="red")

#Predict the test dataset
x_test=scalev[index_test,c(1:4)]
y_test=dataDf[index_test,"y"]

y_pred=predict(autoresmodel,x_test)

#Make inverse scaling
y_pred=y_pred*col_stddevs[5]+col_means[5]

#Show the test results
test_r2=rSquared(y_test,y_test-y_pred)
test_rmse=rmse(y_test,y_pred)
message(paste("test r2:",round(test_r2,2),
            "; test RMSE:",round(test_rmse,2),sep=""))

#Restore the previous par setting
par(curpar)
```

---

hiddenBlock *hiddenBlock*

Description

This function is to construct the hidden block .

Usage

```
hiddenBlock(inlayer, nodes, acts, idepth, orginlayer = NULL,
  reg = NULL, dropout = 0, batchnorm = TRUE)
```

Arguments

inlayer	input layer, keras layer
nodes	list of integers, list of the number of nodes for all the hidden layers
acts	list of strings, list of activations for hidden layers
idepth	integer, index of the hidden layer
orginlayer	keras layer, original layer to be added to decoding layer (default: NULL)
reg	string, regularization (default: NULL)
dropout	double, dropout rate for the target hidden layer (default 0)
batchnorm	bool, flag to conduct batch normalization (default: TRUE)

Value

keras layer, block of a hidden layer (with addtion of actionvation or/and batch normalization)

See Also

AutoEncoderModel for major function using this function.

`keras_r2` *keras_r2*

Description

This function is to calculate rsquared value for regression models.

Usage

```
keras_r2(y_true, y_pred)
```

Arguments

`y_true`	tensor of ground truth
`y_pred`	tensor of predicted values

Value

keras tensor, double for rsquared

`r2_squ` *r2_squ*

Description

This function is to calculate rsquared value for regression models.

Usage

```
r2_squ(obs, res)
```

Arguments

`obs`	vector, observed values
`res`	residual (observed values-predicted values)

Value

double for rsquared

Examples

```
n=300
x=runif(n,1,100)
y=3*x^2+sqrt(abs(x))+rnorm(n)
lmodel=lm(y~x)
ypre=predict(lmodel,data=x)
r2=r2_squ(y,y-ypre)
message(paste("r2 is :",r2))
```

RcppArmadillo-Functions

Set operation of functions in RcppArmadillo package

Description

These four functions are created when `RcppArmadillo.package.skeleton()` is invoked to create a skeleton packages.

Usage

```
rcpparmabasic_test()
rcpparmabasic_outerproduct(x)
rcpparmabasic_innerproduct(x)
rcpparmabasic_bothproducts(x)
```

Arguments

`x`	a numeric vector

Details

These are example functions which should be largely self-explanatory.

Value

`rcpparmabasic_test()` does not return a value, but displays a message to the console.

`rcpparmabasic_outerproduct()` returns a numeric matrix computed as the outer (vector) product of `x`.

`rcpparmabasic_innerproduct()` returns a double computer as the inner (vector) product of `x`.

`rcpparmabasic_bothproducts()` returns a list with both the outer and inner products.

Author(s)

Lianfa Li

References

See the documentation for Armadillo, and RcppArmadillo, for more details.

Examples

```
x <- sqrt(1:4)
rcpparmabasic_innerproduct(x)
rcpparmabasic_outerproduct(x)
```

ResautoDownscale *ResautoDownscale*

Description

This function is the iterative implementation of downscaling with autoencoder based residual network.

Usage

```
ResautoDownscale(r2, fpredict0, c.grid, ss = 0.2, nepoch = 30,
  cores = 1, thresh = 0.01, ntime = 5)
```

Arguments

r2	Stack for the covariates for downscaling
fpredict0	Target fine-resolution image to be downscaled, you can just provide a NA image data (this function can fill this NA) or an initial predictions obtained by other methods.
c.grid	Coarsely resolved grid
ss	Sampling propotion for independent test
nepoch	Numder of epoch for residual network training (default: 30)
cores	Number of CPU cores used for computing (default: 1)
thresh	Stopping creation shreshold (default: 0.01)
ntime	Maximum number of iterations (default: 5)

Value

List(performance metrics such as R2, RMSE, and downscaled images)

Author(s)

Lianfa Li <lspatial@gmail.com>

See Also

AutoEncoderModel for the residual network function used in downscaling.

Examples

```
#Load the high-resolution raster of a covariate,
# elevation to be used for downscaling
eleFile=file.path(system.file(package = "downscaledl"), "extdata", "sample_ele.tif")
ele=raster::raster(eleFile)

#Load the coarse-resolution raster of the target variable to be downscaled
coarseFile=file.path(system.file(package = "downscaledl"), "extdata", "sample_coarse_res.tif")
coarseRst=raster::raster(coarseFile)

#Extract x and y to be used as two predictors in downscaling
xRast=ele
yRast=ele
pos=raster::rasterToPoints(ele,spatial=TRUE)
cell=raster::cellFromXY(xRast,pos)
xyc=sp::coordinates(pos)
xRast[cell]=xyc[,"x"]
yRast[cell]=xyc[,"y"]

#Merge the covariates
covStk=raster::stack(xRast,yRast,ele)
names(covStk)=c("x","y","ele")

#Use the fine-resolution covariate (elevation) as the target image
fineTarget=ele

#Set the paramneters and start to downscale ...
ares=ResautoDownscale(covStk,fineTarget,coarseRst,ss= 0.2, cores= 5, thresh = 0.01,ntime=3)

#Show the iteration results
message(paste(capture.output(ares$diogRMSE), collapse = "\n",sep=""))

#Show the optimal results in the final predictions
message(paste("test R2:",round(ares$r2,2),",test RMSE:",round(ares$rmse,4),sep=""))

#Obtain the downscaled
downscaled_img=ares$raster

#Save the current par setting
curpar = par(no.readonly = TRUE)

#Set the new par setting
par(mfrow=c(1,2),mar=c(1,1,1,1))

#Show the final predictions of fine resolution and
# original coarse-resolution image for a comparison
raster::plot(coarseRst)
raster::plot(downscaled_img)

#Restore the previous par setting
par(curpar)
```

rmse *rmse*

Description

This function is to calculate rmse value for regression models.

Usage

```
rmse(obs, pre)
```

Arguments

obs	vector, observed values
pre	vector, predicted values

Value

double for rmse

Examples

```
n=200
x=runif(n,1,10)
y=log(3*x)+x^2+rnorm(n)
lmodel=lm(y~x)
ypre=predict(lmodel,data=x)
rmse=rmse(y,ypre)
message(paste("rmse is :",rmse))
```

rSquared *rSquared*

Description

This function is to calculate rsquared value for regression models.

Usage

```
rSquared(obs, res)
```

Arguments

obs	vector, observed values
res	residual (observed values-predicted values)

Value

double for rSquared

Index