Package ‘FuncNN’

October 12, 2022

Title Functional Neural Networks
Version 1.0
Description A collection of functions which fit functional neural network models. In other words, this package will allow users to build deep learning models that have either functional or scalar responses paired with functional and scalar covariates. We implement the theoretical discussion found in Thind, Multani and Cao (2020) <arXiv:2006.09590> through the help of a main fitting and prediction function as well as a number of helper functions to assist with cross-validation, tuning, and the display of estimated functional weights.
Imports keras, tensorflow, fda.usc, fda, ggplot2, ggpubr, caret, pbapply, reshape2, flux, doParallel, foreach, Matrix
License GPL-3
Encoding UTF-8
LazyData true
RoxygenNote 7.1.0
Suggests knitr, rmarkdown
NeedsCompilation no
Author Richard Groenewald [ctb], Barinder Thind [aut, cre, cph], Jiguo Cao [aut], Sidi Wu [ctb]
Maintainer Barinder Thind <barinder.thi@gmail.com>
Repository CRAN
Date/Publication 2020-09-15 09:40:15 UTC

R topics documented:

daily ............................................................... 2
fnn.cv ............................................................. 2
fnn.fit ............................................................. 5
daily  

*Classic Canadian weather data set.*

**Description**

Classic Canadian weather data set.

**Usage**

data(daily)

**Format**

An object containing temperature and precipitation data for 35 Canadian cities.

**References**


---

**fnn.cv**  

*Functional Neural Networks with Cross-validation*

**Description**

This is a convenience function for the user. The inputs are largely the same as the `fnn.fit()` function with the additional parameter of fold choice. This function only works for scalar responses.

**Usage**

```r
fnn.cv(
    nfolds,  
    resp,  
    func_cov,  
    scalar_cov = NULL,  
    basis_choice = c("fourier"),  
    num_basis = c(7),  
    hidden_layers = 2,  
    neurons_per_layer = c(64, 64),  
)```
activations_in_layers = c("sigmoid", "linear"),
domain_range = list(c(0, 1)),
epochs = 100,
loss_choice = "mse",
metric_choice = list("mean_squared_error"),
val_split = 0.2,
learn_rate = 0.001,
patience_param = 15,
early_stopping = TRUE,
print_info = TRUE,
batch_size = 32,
decay_rate = 0,
func_resp_method = 1,
covariate_scaling = TRUE,
raw_data = FALSE
)

Arguments

nfolds The number of folds to be used in the cross-validation process.

resp For scalar responses, this is a vector of the observed dependent variable. For functional responses, this is a matrix where each row contains the basis coefficients defining the functional response (for each observation).

func_cov The form of this depends on whether the raw_data argument is true or not. If true, then this is a list of k matrices. The dimensionality of the matrices should be the same (n x p) where n is the number of observations and p is the number of longitudinal observations. If raw_data is false, then the input should be a tensor with dimensionality b x n x k where b is the number of basis functions used to define the functional covariates, n is the number of observations, and k is the number of functional covariates.

scalar_cov A matrix contained the multivariate information associated with the data set. This is all of your non-longitudinal data.

basis_choice A vector of size k (the number of functional covariates) with either "fourier" or "bspline" as the inputs. This is the choice for the basis functions used for the functional weight expansion. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates.

num_basis A vector of size k defining the number of basis functions to be used in the basis expansion. Must be odd for fourier basis choices. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates.

hidden_layers The number of hidden layers to be used in the neural network.

neurons_per_layer Vector of size = hidden_layers. The u-th element of the vector corresponds to the number of neurons in the u-th hidden layer.

activations_in_layers Vector of size = hidden_layers. The u-th element of the vector corresponds to the activation choice in the u-th hidden layer.
domain_range  List of size k. Each element of the list is a 2-dimensional vector containing the upper and lower bounds of the k-th functional weight.

epochs  The number of training iterations.

loss_choice  This parameter defines the loss function used in the learning process.

metric_choice  This parameter defines the printed out error metric.

val_split  A parameter that decides the percentage split of the inputted data set.

learn_rate  Hyperparameter that defines how quickly you move in the direction of the gradient.

patience_param  A keras parameter that decides how many additional epochs are eclipsed with minimal change in error before the learning process is stopped. This is only active if early_stopping = TRUE

early_stopping  If TRUE, then learning process will be halted early if error improvement isn’t seen.

print_info  If TRUE, function will output information about the model as it is trained.

batch_size  Size of the batch for stochastic gradient descent.

decay_rate  A modification to the learning rate that decreases the learning rate as more and more learning iterations are completed.

func_resp_method  Set to 1 by default. In the future, this will be set to 2 for an alternative functional response approach.

covariate_scaling  If TRUE, then data will be internally scaled before model development.

raw_data  If TRUE, then user does not need to create functional observations beforehand. The function will internally take care of that pre-processing.

Details

No additional details for now.

Value

The following are returned.

predicted_folds  – The predicted scalar values in each fold.

true_folds  – The true values of the response in each fold.

MSPE  – A list object containing the MSPE in each fold and the overall cross-validated MSPE.

fold_indices  – The generated indices for each fold; for replication purposes.

Examples

# Libraries
library(fda)

# Loading data
**Description**

This is the main function in the FuncNN package. This function fits models of the form: \( f(z, b(x)) \)
where \( z \) are the scalar covariates and \( b(x) \) are the functional covariates. The form of \( f() \) is that of a neural network with a generalized input space.

**Usage**

```r
fnn.fit(
  resp,
  func_cov,
  scalar_cov = NULL,
```

```r
```
```r
cbasis_choice = c(“fourier”),
num_basis = c(7),
hidden_layers = 2,
networks_per_layer = c(64, 64),
activations_in_layers = c(“sigmoid”, “linear”),
domain_range = list(c(0, 1)),
epochs = 100,
loss_choice = “mse”,
metric_choice = list(“mean_squared_error”),
val_split = 0.2,
learn_rate = 0.001,
patience_param = 15,
ear_early_stopping = TRUE,
print_info = TRUE,
batch_size = 32,
decay_rate = 0,
func_res_method = 1,
covariate_scaling = TRUE,
raw_data = FALSE,
dropout = FALSE
```

### Arguments

- **resp**
  For scalar responses, this is a vector of the observed dependent variable. For functional responses, this is a matrix where each row contains the basis coefficients defining the functional response (for each observation).

- **func_cov**
  The form of this depends on whether the `raw_data` argument is true or not. If true, then this is a list of k matrices. The dimensionality of the matrices should be the same (n x p) where n is the number of observations and p is the number of longitudinal observations. If `raw_data` is false, then the input should be a tensor with dimensionality b x n x k where b is the number of basis functions used to define the functional covariates, n is the number of observations, and k is the number of functional covariates.

- **scalar_cov**
  A matrix contained the multivariate information associated with the data set. This is all of your non-longitudinal data.

- **basis_choice**
  A vector of size k (the number of functional covariates) with either "fourier" or "bspline" as the inputs. This is the choice for the basis functions used for the functional weight expansion. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates.

- **num_basis**
  A vector of size k defining the number of basis functions to be used in the basis expansion. Must be odd for `fourier` basis choices. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates.

- **hidden_layers**
  The number of hidden layers to be used in the neural network.

- **neurons_per_layer**
  Vector of size = hidden_layers. The u-th element of the vector corresponds to the number of neurons in the u-th hidden layer.
**activations_in_layers**
Vector of size = hidden_layers. The u-th element of the vector corresponds to the activation choice in the u-th hidden layer.

**domain_range**
List of size k. Each element of the list is a 2-dimensional vector containing the upper and lower bounds of the k-th functional weight.

**epochs**
The number of training iterations.

**loss_choice**
This parameter defines the loss function used in the learning process.

**metric_choice**
This parameter defines the printed out error metric.

**val_split**
A parameter that decides the percentage split of the inputted data set.

**learn_rate**
Hyperparameter that defines how quickly you move in the direction of the gradient.

**patience_param**
A keras parameter that decides how many additional epochs are eclipsed with minimal change in error before the learning process is stopped. This is only active if early_stopping = TRUE

**early_stopping**
If TRUE, then learning process will be halted early if error improvement isn’t seen.

**print_info**
If TRUE, function will output information about the model as it is trained.

**batch_size**
Size of the batch for stochastic gradient descent.

**decay_rate**
A modification to the learning rate that decreases the learning rate as more and more learning iterations are completed.

**func_resp_method**
Set to 1 by default. In the future, this will be set to 2 for an alternative functional response approach.

**covariate_scaling**
If TRUE, then data will be internally scaled before model development.

**raw_data**
If TRUE, then user does not need to create functional observations beforehand. The function will internally take care of that pre-processing.

**dropout**
Keras parameter that randomly drops some percentage of the neurons in a given layer. If TRUE, then 0.1*layer_number will be dropped; instead, you can specify a vector equal to the number of layers specifying what percentage to drop in each layer.

**Details**
Updates coming soon.

**Value**
The following are returned:

**model** – Full keras model that can be used with any functions that act on keras models.

**data** – Adjust data set after scaling and appending of scalar covariates.

**fnc_basis_num** – A return of the original input; describes the number of functions used in each of the k basis expansions.
fnc_type – A return of the original input; describes the basis expansion used to make the functional weights.

parameter_info – Information associated with hyperparameter choices in the model.

per_iter_info – Change in error over training iterations

func_obs – In the case when raw_data is TRUE, the user may want to see the internally developed functional observations. This returns those functions.

Examples

# First, an easy example with raw_data = TRUE

# Loading in data
data("daily")

# Functional covariates (subsetting for time sake)
precip = t(daily$precav)
longtidunal_dat = list(precip)

# Scalar Response
total_prec = apply(daily$precav, 2, mean)

# Running model
fit1 = fnn.fit(resp = total_prec,
func_cov = longtidunal_dat,
scalar_cov = NULL,
learn_rate = 0.0001,
epochs = 10,
raw_data = TRUE)

# Classification Example with raw_data = TRUE

# Loading data
tecator = FuncNN::tecator

# Making classification bins
tecator_resp = as.factor(ifelse(tecator$y$Fat > 25, 1, 0))

# Non functional covariate
tecator_scalar = data.frame(water = tecator$y$Water)

# Splitting data
ind = sample(1:length(tecator_resp), round(0.75*length(tecator_resp)))
train_y = tecator_resp[ind]
test_y = tecator_resp[-ind]
train_x = tecator$absorp.fdata$data[ind,]
test_x = tecator$absorp.fdata$data[-ind,]
scalar_train = data.frame(tecator_scalar[ind,1])
scalar_test = data.frame(tecator_scalar[-ind,1])
# Making list element to pass in
func_covers_train = list(train_x)
func_covers_test = list(test_x)

# Now running model
fit_class = fnn.fit(resp = train_y,
                  func_cov = func_covers_train,
                  scalar_cov = scalar_train,
                  hidden_layers = 6,
                  neurons_per_layer = c(24, 24, 24, 24, 24, 58),
                  activations_in_layers = c("relu", "relu", "relu", "relu", "relu", "linear"),
                  domain_range = list(c(850, 1050)),
                  learn_rate = 0.001,
                  epochs = 100,
                  raw_data = TRUE,
                  early_stopping = TRUE)

# Running prediction, gets probabilities
predict_class = fnn.predict(fit_class,
                         func_cov = func_covers_test,
                         scalar_cov = scalar_test,
                         domain_range = list(c(850, 1050)),
                         raw_data = TRUE)

# Example with Pre-Processing (raw_data = FALSE)

# loading data
tecator = FuncNN::tecator

# libraries
library(fda)

# define the time points on which the functional predictor is observed.
timepts = tecator$absorp.fdata$argvals

# define the fourier basis
nbasis = 29
spline_basis = create.fourier.basis(tecator$absorp.fdata$rangeval, nbasis)

# convert the functional predictor into a fda object and getting deriv
tecator_fd = Data2fd(timepts, t(tecator$absorp.fdata$data), spline_basis)
tecator_deriv = deriv.fd(tecator_fd)
tecator_deriv2 = deriv.fd(tecator_deriv)

# Non functional covariate
tecator_scalar = data.frame(water = tecator$y$Water)

# Response
tecator_resp = tecator$y$Fat

# Getting data into right format
tecator_data = array(dim = c(nbasis, length(tecator_resp), 3))
tecator_data[,,1] = tecator_fd$coefs
fnn.fit

tecator_data[,2] = tecator_deriv$coefs
tecator_data[,3] = tecator_deriv2$coefs

# Splitting into test and train for third FNN
ind = 1:165
tec_data_train <- array(dim = c(nbasis, length(ind), 3))
tec_data_test <- array(dim = c(nbasis, nrow(tecator$absorp.fdata$data) - length(ind), 3))
tec_data_train = tecator_data[, ind, ]
tec_data_test = tecator_data[, -ind, ]
tecResp_train = tecator_resp[ind]
tecResp_test = tecator_resp[-ind]
scalar_train = data.frame(tecator_scalar[ind,1])
scalar_test = data.frame(tecator_scalar[-ind,1])

# Setting up network
tecator_fnn = fnn.fit(resp = tecResp_train,
func_cov = tec_data_train,
scalar_cov = scalar_train,
basis_choice = c("fourier", "fourier", "fourier"),
num_basis = c(5, 5, 7),
hidden_layers = 4,
neurons_per_layer = c(64, 64, 64, 64),
activations_in_layers = c("relu", "relu", "relu", "linear"),
domain_range = list(c(850, 1050), c(850, 1050), c(850, 1050)),
epochs = 300,
learn_rate = 0.002)

# Prediction example can be seen with ?fnn.fit()

# Functional Response Example:

# libraries
library(fda)

# Loading data
data("daily")

# Creating functional data
temp_data = array(dim = c(65, 35, 1))
tempbasis65 = create.fourier.basis(c(0,365), 65)
tempbasis7 = create.bspline.basis(c(0,365), 7, norder = 4)
timepts = seq(1, 365, 1)
temp_fd = Data2fd(timepts, daily$tempav, tempbasis65)
prec_fd = Data2fd(timepts, daily$precav, tempbasis7)
prec_fd$coefs = scale(prec_fd$coefs)

# Data set up
temp_data[,1] = temp_fd$coefs
resp_mat = prec_fd$coefs

# Non functional covariate
weather_scalar = data.frame(total_prec = apply(daily$precav, 2, sum))
# Getting data into proper format
ind = 1:30
nbasis = 65
weather_data_train <- array(dim = c(nbasis, ncol(temp_data), 1))
weather_data_train[,1] = temp_data
scalar_train = data.frame(weather_scalar[,1])
resp_train = t(resp_mat)

# Running model
weather_func_fnn <- fnn.fit(resp = resp_train,
                        func_cov = weather_data_train,
                        scalar_cov = scalar_train,
                        basis_choice = c("bspline"),
                        num_basis = c(7),
                        hidden_layers = 2,
                        neurons_per_layer = c(1024, 1024),
                        activations_in_layers = c("sigmoid", "linear"),
                        domain_range = list(c(1, 365)),
                        epochs = 300,
                        learn_rate = 0.01,
                        func_resp_method = 1)

---

**fnn.fnc**

**Output of Estimated Functional Weights**

**Description**

This function outputs plots and `ggplot()` objects of the functional weights found by the `fnn.fit()` model.

**Usage**

`fnn.fnc(model, domain_range, covariate_scaling = FALSE)`

**Arguments**

- **model**
  A keras model as outputted by `fnn.fit()`.

- **domain_range**
  List of size k. Each element of the list is a 2-dimensional vector containing the upper and lower bounds of the k-th functional weight. Must be the same covariates as input into `fnn.fit()`.

- **covariate_scaling**
  If TRUE, then data will be internally scaled before model development.

**Details**

No additional details for now.
Value

The following are returned:

FNC_Coefficients – The estimated coefficients defining the basis expansion for each of the k functional weights.

saved_plot – A list of size k of ggplot() objects.

Examples

```r
# libraries
library(fda)

# loading data
tecator = FuncNN::tecator

# define the time points on which the functional predictor is observed.
timepts = tecator$absorp.fdata$argvals

# define the fourier basis
nbasis = 29
spline_basis = create.fourier.basis(tecator$absorp.fdata$rangeval, nbasis)

# convert the functional predictor into a fda object and getting deriv
tecator_fd = Data2fd(timepts, t(tecator$absorp.fdata$data), spline_basis)
tecator_deriv = deriv.fd(tecator_fd)
tecator_deriv2 = deriv.fd(tecator_deriv)

# Non functional covariate
tecator_scalar = data.frame(water = tecator$y$Water)

# Response
tecator_resp = tecator$y$Fat

# Getting data into right format
tecator_data = array(dim = c(nbasis, length(tecator_resp), 3))
tecator_data[, , 1] = tecator_fd$coefs
tecator_data[, , 2] = tecator_deriv$coefs
tecator_data[, , 3] = tecator_deriv2$coefs

# Getting data ready to pass into function
ind = 1:165
tec_data_train <- array(dim = c(nbasis, length(ind), 3))
tec_data_train = tecator_data[, ind, ]
tecResp_train = tecator_resp[ind]
scalar_train = data.frame(tecator_scalar[ind, 1])

# Setting up network
tecator_fnn = fnn.fit(resp = tecResp_train,
                     func_cov = tec_data_train,
                     scalar_cov = scalar_train,
                     basis_choice = c("fourier", "fourier", "fourier"),
                     ...)
```
num_basis = c(5, 5, 7),
hidden_layers = 4,
neurons_per_layer = c(64, 64, 64, 64),
activations_in_layers = c("relu", "relu", "relu", "linear"),
domain_range = list(c(850, 1050), c(850, 1050), c(850, 1050)),
epochs = 300,
learn_rate = 0.002)

# Functional weights for this model
est_func_weights = fnn.fnc(tecator_fnn, domain_range = list(c(850, 1050),
c(850, 1050),
c(850, 1050)))

---

**fnn.plot**  
*Plotting Functional Response Predictions*

**Description**
This function is to be used for functional responses. It outputs a ggplot() object of the predicted functional responses.

**Usage**

```r
fnn.plot(
  FNN_Predict_Object,  
  Basis_Type = "fourier",  
  domain_range = c(0, 1),  
  step_size = 0.01
)
```

**Arguments**

- **FNN_Predict_Object**
  An object output by the fnn.predict() function. Must be for when the problem is that of a functional response.

- **Basis_Type**
  The type of basis to use to create the functional response.

- **domain_range**
  The continuum range of the functional responses.

- **step_size**
  The size of the movement from the lower bound of the domain_range to the upper bound.

**Details**
No additional details for now.
Value

The following are returned:

plot – A ggplot() object of the predicted functional responses.
evaluations – The discrete evaluations across the domain of the functional response.

Examples

# libraries
library(fda)

# Loading data
data("daily")

# Creating functional data
temp_data = array(dim = c(65, 35, 1))
tempbasis65 = create.fourier.basis(c(0,365), 65)
tempbasis7 = create.bspline.basis(c(0,365), 7, norder = 4)
timepts = seq(1, 365, 1)
temp_fd = Data2fd(timepts, daily$tempav, tempbasis65)
prec_fd = Data2fd(timepts, daily$precav, tempbasis7)
prec_fd$coefs = scale(prec_fd$coefs)

# Data set up
temp_data[,]1] = temp_fd$coefs
resp_mat = prec_fd$coefs

# Non functional covariate
weather_scalar = data.frame(total_prec = apply(daily$precav, 2, sum))

# Splitting into test and train
ind = 1:30
nbasis = 65
weather_data_train <- array(dim = c(nbasis, length(ind), 1))
weather_data_test <- array(dim = c(nbasis, ncol(daily$tempav) - length(ind), 1))
weather_data_train[,]1] = temp_data[, ind, ]
weather_data_test[,]1] = temp_data[, -ind, ]
scalar_train = data.frame(weather_scalar[ind,1])
scalar_test = data.frame(weather_scalar[-ind,1])
resp_train = t(resp_mat[,ind])
resp_test = t(resp_mat[,,-ind])

# Running model
weather_func_fnn <- fnn.fit(resp = resp_train,
func_cov = weather_data_train,
scalar_cov = scalar_train,
basis_choice = c("bspline"),
num_basis = c(7),
hidden_layers = 2,
neurons_per_layer = c(1024, 1024),
activations_in_layers = c("sigmoid", "linear"),
domain_range = list(c(1, 365)),
epochs = 300,
learn_rate = 0.01,
func_resp_method = 1)

# Getting predictions
predictions = fnn.predict(weather_func_fnn,
                           weather_data_test,
                           scalar_cov = scalar_test,
                           basis_choice = c("bspline"),
                           num_basis = c(7),
                           domain_range = list(c(1, 365)))

# Looking at plot
fnn.plot(predictions, domain_range = c(1, 365), step_size = 1, Basis_Type = "bspline")

---

**Description**

The prediction function associated with the fnn model allowing for users to quickly get scalar or functional outputs.

**Usage**

```r
defn.predict(
    model,
    func_cov,
    scalar_cov = NULL,
    basis_choice = c("fourier"),
    num_basis = c(7),
    domain_range = list(c(0, 1)),
    covariate_scaling = TRUE,
    raw_data = FALSE
)
```

**Arguments**

- `model`: A keras model as outputted by `fnn.fit()`.
- `func_cov`: The form of this depends on whether the `raw_data` argument is true or not. If true, then this is a list of `k` matrices. The dimensionality of the matrices should be the same (n x p) where n is the number of observations and p is the number of longitudinal observations. If `raw_data` is false, then the input should be a tensor with dimensionality `b x n x k` where `b` is the number of basis functions used to define the functional covariates, `n` is the number of observations, and `k`
is the number of functional covariates. Must be the same covariates as input into `fnn.fit()` although here, they will likely be the 'test' observations.

**scalar_cov**
A matrix contained the multivariate information associated with the data set. This is all of your non-longitudinal data. Must be the same covariates as input into `fnn.fit()` although here, they will likely be the 'test' observations.

**basis_choice**
A vector of size k (the number of functional covariates) with either "fourier" or "bspline" as the inputs. This is the choice for the basis functions used for the functional weight expansion. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates. Should be the same choices as input into `fnn.fit()`.

**num_basis**
A vector of size k defining the number of basis functions to be used in the basis expansion. Must be odd for `fourier` basis choices. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates. Should be the same values as input into `fnn.fit()`.

**domain_range**
List of size k. Each element of the list is a 2-dimensional vector containing the upper and lower bounds of the k-th functional weight. Must be the same covariates as input into `fnn.fit()`.

**covariate_scaling**
If TRUE, then data will be internally scaled before model development.

**raw_data**
If TRUE, then user does not need to create functional observations beforehand. The function will internally take care of that pre-processing.

**Details**
No additional details for now.

**Value**
The following is returned:

- **Predictions** – A vector of scalar predictions or a matrix of basis coefficients for functional responses.

**Examples**

```r
# First, we do an example with a scalar response:

# loading data
tecator = FuncNN::tecator

# libraries
library(fda)

# define the time points on which the functional predictor is observed.
timepts = tecator$absorp.fdata$argvals

# define the fourier basis
nbasis = 29
```
spline_basis = create.fourier.basis(tecator$absorp.fdata$rangeval, nbasis)

# convert the functional predictor into a fda object and getting deriv
tecator_fd = Data2fd(timepts, t(tecator$absorp.fdata$data), spline_basis)
tecator_deriv = deriv.fd(tecator_fd)
tecator_deriv2 = deriv.fd(tecator_deriv)

# Non functional covariate
tecator_scalar = data.frame(water = tecator$y$Water)

# Response
tecator_resp = tecator$y$Fat

# Getting data into right format
tecator_data = array(dim = c(nbasis, length(tecator_resp), 3))
tecator_data[,1] = tecator_fd$coefs
tecator_data[,2] = tecator_deriv$coefs
tecator_data[,3] = tecator_deriv2$coefs

# Splitting into test and train for third FNN
ind = 1:165
tec_data_train <- array(dim = c(nbasis, length(ind), 3))
techn_data_test <- array(dim = c(nbasis, nrow(tecator$absorp.fdata$data) - length(ind), 3))
techn_data_train = tecator_data[, ind, ]
techn_data_test = tecator_data[, -ind, ]
techn Resp_train = tecator Resp[ind]
techn Resp_test = tecator Resp[-ind]
scalar_train = data.frame(tecator scalar[ind,1])
scalar_test = data.frame(tecator scalar[-ind,1])

# Setting up network
tecator_fnn = fnn.fit(resp = tecResp_train,
func_cov = tec_data_train,
scalar_cov = scalar_train,
basis_choice = c("fourier", "fourier", "fourier"),
um basis = c(5, 5, 7),
hidden_layers = 4,
neurons_per layer = c(64, 64, 64, 64),
activations_in_layers = c("relu", "relu", "relu", "linear"),
domain_range = list(c(850, 1050), c(850, 1050), c(850, 1050)),
epochs = 300,
learn_rate = 0.002)

# Predicting
pred_tec = fnn.predict(tecator_fnn,
techn_data_test,
scalar_cov = scalar_test,
basis_choice = c("fourier", "fourier", "fourier"),
um basis = c(5, 5, 7),
domain_range = list(c(850, 1050), c(850, 1050), c(850, 1050)))

# Now an example with functional responses
# libraries
library(fda)

# Loading data
data("daily")

# Creating functional data
temp_data = array(dim = c(65, 35, 1))
tempbasis65 = create.fourier.basis(c(0,365), 65)
tempbasis7 = create.bspline.basis(c(0,365), 7, norder = 4)
timepts = seq(1, 365, 1)
temp_fd = Data2fd(timepts, daily$tempav, tempbasis65)
prec_fd = Data2fd(timepts, daily$precav, tempbasis7)
prec_fd$coefs = scale(prec_fd$coefs)

# Data set up
temp_data[,,1] = temp_fd$coefs
resp_mat = prec_fd$coefs

# Non functional covariate
weather_scalar = data.frame(total_prec = apply(daily$precav, 2, sum))

# Splitting into test and train
ind = 1:30
nbasis = 65
weather_data_train <- array(dim = c(nbasis, length(ind), 1))
weather_data_test <- array(dim = c(nbasis, ncol(daily$tempav) - length(ind), 1))
weather_data_train[,,1] = temp_data[, ind, ]
weather_data_test[,,1] = temp_data[, -ind, ]
scalar_train = data.frame(weather_scalar[ind,1])
scalar_test = data.frame(weather_scalar[-ind,1])
resp_train = t(resp_mat[,ind])
resp_test = t(resp_mat[-ind])

# Running model
weather_func_fnn <- fnn.fit(resp = resp_train,
 func_cov = weather_data_train,
 scalar_cov = scalar_train,
 basis_choice = c("bspline"),
 num_basis = c(7),
 hidden_layers = 2,
 neurons_per_layer = c(1024, 1024),
 activations_in_layers = c("sigmoid", "linear"),
 domain_range = list(c(1, 365)),
 epochs = 300,
 learn_rate = 0.01,
 func_resp_method = 1)

# Getting Predictions
predictions = fnn.predict(weather_func_fnn,
 weather_data_test,
 scalar_cov = scalar_test,
 basis_choice = c("bspline"),
# Classification Prediction

# Loading data
tecator = FuncNN::tecator

# Making classification bins
tecator_resp = as.factor(ifelse(tecator$y$Fat > 25, 1, 0))

# Non functional covariate
tecator_scalar = data.frame(water = tecator$y$Water)

# Splitting data
ind = sample(1:length(tecator_resp), round(0.75*length(tecator_resp)))
train_y = tecator_resp[ind]
test_y = tecator_resp[-ind]
train_x = tecator$absorp.fdata$data[ind,]
test_x = tecator$absorp.fdata$data[-ind,]
scalar_train = data.frame(tecator_scalar[ind,1])
scalar_test = data.frame(tecator_scalar[-ind,1])

# Making list element to pass in
func_covs_train = list(train_x)
func_covs_test = list(test_x)

# Now running model
fit_class = fnn.fit(resp = train_y,
                   func_cov = func_covs_train,
                   scalar_cov = scalar_train,
                   hidden_layers = 6,
                   neurons_per_layer = c(24, 24, 24, 24, 24, 58),
                   activations_in_layers = c("relu", "relu", "relu", "relu", "relu", "linear"),
                   domain_range = list(c(850, 1050)),
                   learn_rate = 0.001,
                   epochs = 100,
                   raw_data = TRUE,
                   early_stopping = TRUE)

# Running prediction
predict_class = fnn.predict(fit_class,
                             func_cov = func_covs_test,
                             scalar_cov = scalar_test,
                             domain_range = list(c(850, 1050)),
                             raw_data = TRUE)

# Rounding predictions (they are probabilities)
rounded_preds = ifelse(round(predict_class[,2] == 1, 1, 0)
fnn.tune

Tuning Functional Neural Networks

Description
A convenience function for the user that implements a simple grid search for the purpose of tuning. For each combination in the grid, a cross-validated error is calculated. The best combination is returned along with additional information. This function only works for scalar responses.

Usage
fnn.tune(
  tune_list,
  resp,
  func_cov,
  scalar_cov = NULL,
  basis_choice,
  domain_range,
  batch_size = 32,
  decay_rate = 0,
  nfolds = 5,
  cores = 4,
  raw_data = FALSE
)

Arguments
  
  tune_list This is a list object containing the values from which to develop the grid. For each of the hyperparameters that can be tuned (num_hidden_layers, neurons, epochs, val_split, patience, learn_rate, num_basis, activation_choice), the user inputs a set of values to try. Note that the combinations are found based on the number of hidden layers. For example, if num_hidden_layers = 3 and neurons = c(8, 16), then the combinations will begin as c(8, 8, 8), c(8, 8, 16), ..., c(16, 16, 16). Example provided below.

  resp For scalar responses, this is a vector of the observed dependent variable. For functional responses, this is a matrix where each row contains the basis coefficients defining the functional response (for each observation).

  func_cov The form of this depends on whether the raw_data argument is true or not. If true, then this is a list of k matrices. The dimensionality of the matrices should be the same (n x p) where n is the number of observations and p is the number
of longitudinal observations. If raw_data is false, then the input should be a tensor with dimensionality b x n x k where b is the number of basis functions used to define the functional covariates, n is the number of observations, and k is the number of functional covariates.

scalar_cov: A matrix contained the multivariate information associated with the data set. This is all of your non-longitudinal data.

basis_choice: A vector of size k (the number of functional covariates) with either "fourier" or "bspline" as the inputs. This is the choice for the basis functions used for the functional weight expansion. If you only specify one, with k > 1, then the argument will repeat that choice for all k functional covariates.

domain_range: List of size k. Each element of the list is a 2-dimensional vector containing the upper and lower bounds of the k-th functional weight.

batch_size: Size of the batch for stochastic gradient descent.

decay_rate: A modification to the learning rate that decreases the learning rate as more and more learning iterations are completed.

nfolds: The number of folds to be used in the cross-validation process.

cores: For the purpose of parallelization.

raw_data: If TRUE, then user does not need to create functional observations beforehand. The function will internally take care of that pre-processing.

Details

No additional details for now.

Value

The following are returned:
Parameters – The final list of hyperparameter chosen by the tuning process.
All_Information – A list object containing the errors for every combination in the grid. Each element of the list corresponds to a different choice of number of hidden layers.
Best_Per_Layer – An object that returns the best parameter combination for each choice of hidden layers.
Grid_List – An object containing information about all combinations tried by the tuning process.

Examples

```r
# libraries
library(fda)

# Loading data
data("daily")

data("daily")

# Obtaining response
total_prec = apply(daily$precav, 2, mean)
```
# Creating functional data
temp_data = array(dim = c(65, 35, 1))
tempbasis65 = create.fourier.basis(c(0,365), 65)
timepts = seq(1, 365, 1)
temp_fd = Data2fd(timepts, daily$tempav, tempbasis65)

# Data set up
temp_data[,1] = temp_fd$coefs

# Creating grid
tune_list_weather = list(num_hidden_layers = c(2),
neurons = c(8, 16),
epochs = c(250),
val_split = c(0.2),
patience = c(15),
learn_rate = c(0.01, 0.1),
um_basis = c(7),
activation_choice = c("relu", "sigmoid")

# Running Tuning
weather_tuned = fnn.tune(tune_list_weather,
total_prec,
temp_data,
basis_choice = c("fourier"),
domain_range = list(c(1, 24)),
nfolds = 2)

# Looking at results
weather_tuned

---

tecator  

Class Classic Tecator data set.

Description
Classic Tecator data set.

Usage
data(tecator)

Format
An object containing the response and absorbance curve values.

References
Index

* datasets
  
  daily, 2
  tecator, 22

daily, 2

fnn.cv, 2
fnn.fit, 5
fnn.fit(), 2
fnn.fnc, 11
fnn.plot, 13
fnn.predict, 15
fnn.tune, 20

tecator, 22