Package ‘deepregression’

October 4, 2021

Title Fitting Deep Distributional Regression

Version 0.1

Description Allows for the specification of semi-structured deep distributional regression models which are fitted in a neural network as proposed by Ruegamer et al. (2021) <arXiv:2104.02705>. Predictors can be modeled using structured (penalized) linear effects, structured non-linear effects or using an unstructured deep network model.

Config/reticulate list(packages = list(package = "six", pip = TRUE), list(package = "tensorflow", version = "2.5.0rc0", pip = TRUE), list(package = "tensorflow_probability", version = "0.12", pip = TRUE), list(package = "keras", version = "2.5.0rc0", pip = TRUE))

Depends R (>= 4.0.0)

Suggests testthat, knitr

Imports tensorflow (>= 2.2.0), tfprobability, keras, mgcv, dplyr, purrr, R6, reticulate (>= 1.14), Matrix, magrittr, Metrics, tfruns, methods, utils

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author David Ruegamer [aut, cre], Florian Pfisterer [ctb], Philipp Baumann [ctb], Chris Kolb [ctb]

Maintainer David Ruegamer <david.ruegamer@gmail.com>

Repository CRAN

Date/Publication 2021-10-04 19:50:04 UTC
R topics documented:

check_and_install .................................................. 3
create_family .......................................................... 3
cv ........................................................................ 4
deepregression ......................................................... 4
distfun_to_dist .......................................................... 6
extractval ............................................................... 7
family_to_tfd ............................................................ 8
family_to_trafo .......................................................... 8
fit ......................................................................... 9
from_dist_to_loss ....................................................... 9
from_preds_to_dist ..................................................... 10
get_distribution .......................................................... 11
get_partial_effect ....................................................... 11
get_type_pfc ............................................................. 12
get_weight_by_name ..................................................... 12
handle_gam_term ......................................................... 13
keras_dr ................................................................. 13
layer_add_identity ....................................................... 14
log_score ............................................................... 15
loop_through_pfc_and_call_trafo .................................... 15
makeInputs ............................................................... 16
make_folds ............................................................... 16
make_generator ........................................................... 17
make_generator_from_matrix .......................................... 17
make_tfd_dist ............................................................ 18
names_families .......................................................... 20
orthog_control ........................................................... 20
penalty_control .......................................................... 21
plot.deepregression ...................................................... 22
plot_cv ................................................................. 25
prepare_data ............................................................ 26
prepare_newdata ........................................................ 26
processor ............................................................... 27
quant ................................................................. 28
separate_define_relation .............................................. 28
stddev ................................................................. 29
stop_iter_cv_result ..................................................... 29
subnetwork_init ........................................................ 30
tfd_zinb ............................................................... 30
tfd_zip ................................................................. 31
tf_stride_cols ........................................................... 31

Index 32
check_and_install

Function to check python environment and install necessary packages

Description

Note: The package currently relies on tensorflow version 2.0.0 which is not available for the latest python versions 3.9 and later. If you encounter problems with installing the required python modules please make sure, that a correct python version is configured using 'py_discover_config' and change the python version if required. Internally uses keras::install_keras.

Usage

check_and_install(force = FALSE)

Arguments

force if TRUE, forces the installations

Value

Function that checks if a Python environment is available and contains TensorFlow. If not the recommended version is installed.

create_family

Function to create (custom) family

Description

Function to create (custom) family

Usage

create_family(tfd_dist, trafo_list, output_dim = 1L)

Arguments

tfd_dist a tensorflow probability distribution
trafo_list list of transformations h for each parameter (e.g, exp for a variance parameter)
output_dim integer defining the size of the response

Value

a function that can be used by tfp$layers$DistributionLambda to create a new distribuional layer
cv

Generic cv function

Description

Generic cv function

Usage

cv(x, ...)

Arguments

x model to do cv on
...

... further arguments passed to the class-specific function

deepregression

Fitting Semi-Structured Deep Distributional Regression

Description

Fitting Semi-Structured Deep Distributional Regression

Usage

deepregression(
    y,
    list_of_formulas,
    list_of_deep_models = NULL,
    family = "normal",
    data,
    tf_seed = as.integer(1991 - 5 - 4),
    return_prepoc = FALSE,
    subnetwork_builder = subnetwork_init,
    model_builder = keras_dr,
    fitting_function = utils::getFromNamespace("fit.keras.engine.training.Model",
                                               "keras"),
    additional_processors = list(),
    penalty_options = penalty_control(),
    orthog_options = orthog_control(),
    verbose = FALSE,
    ...
)
)
Arguments

- \( y \): response variable
- `list_of_formulas`: a named list of right hand side formulas, one for each parameter of the distribution specified in `family`; set to \(~ 1\) if the parameter should be treated as constant. Use the `s()`-notation from `mgcv` for specification of non-linear structured effects and `d(\ldots)` for deep learning predictors (predictors in brackets are separated by commas), where `d` can be replaced by an name name of the names in `list_of_deep_models`, e.g., \(~ 1 + s(x) + \text{my\_deep\_mod}(a,b,c)\), where `my\_deep\_mod` is the name of the neural net specified in `list_of_deep_models` and \(a\), \(b\), \(c\) are features modeled via this network.

- `list_of_deep_models`: a named list of functions specifying a keras model. See the examples for more details.
- `family`: a character specifying the distribution. For information on possible distribution and parameters, see `make_tfd_dist`. Can also be a custom distribution.
- `data`: data.frame or named list with input features
- `tf_seed`: a seed for TensorFlow (only works with R version \(\geq 2.2.0\))
- `return_prepoc`: logical; if TRUE only the pre-processed data and layers are returned (default FALSE).
- `subnetwork_builder`: function to build each subnetwork (network for each distribution parameter; per default `subnetwork_init`). Can also be a list of the same size as `list_of_formulas`.
- `model_builder`: function to build the model based on additive predictors (per default `keras_dr`). In order to work with the methods defined for the class `deepregression`, the model should behave like a keras model.
- `fitting_function`: function to fit the instantiated model when calling `fit`. Per default the keras `fit` function.
- `additional_processors`: a named list with additional processors to convert the formula(s). Can have an attribute "controls" to pass additional controls
- `penalty_options`: options for smoothing and penalty terms defined by `penalty_control`
- `orthog_options`: options for the orthogonalization defined by `orthog_control`
- `verbose`: logical; whether to print progress of model initialization to console
- `...`: further arguments passed to the `model_builder` function

References

Examples

```r
library(deepregression)

n <- 1000
data <- data.frame(matrix(rnorm(4*n), c(n,4)))
colnames(data) <- c("x1","x2","x3","xa")
formula <- ~ 1 + deep_model(x1,x2,x3) + s(xa) + x1

deep_model <- function(x) x %>%
  layer_dense(units = 32, activation = "relu", use_bias = FALSE) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 8, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")
y <- rnorm(n) + data$xa^2 + data$x1

mod <- deepregression(
  list_of_formulas = list(loc = formula, scale = ~ 1),
  data = data, y = y,
  list_of_deep_models = list(deep_model = deep_model)
)

if(!is.null(mod)){
  # train for more than 10 epochs to get a better model
  mod %>% fit(epochs = 10, early_stopping = TRUE)
  mod %>% fitted() %>% head()
  cvres <- mod %>% cv()
  mod %>% get_partial_effect(name = "s(xa)"
  mod %>% coef()
  mod %>% plot()
}

mod <- deepregression(
  list_of_formulas = list(loc = ~ 1 + s(xa) + x1, scale = ~ 1,
    dummy = ~ -1 + deep_model(x1,x2,x3) %OZ% 1),
  data = data, y = y,
  list_of_deep_models = list(deep_model = deep_model),
  mapping = list(1,2,1:2)
)
```

---

**distfun_to_dist**  
*Function to define output distribution based on dist.fun*

**Description**  
Function to define output distribution based on dist.fun
extractval

Usage

distfun_to_dist(dist_fun, preds)

Arguments

dist_fun a distribution function as defined by make_tfd_dist
preds tensors with predictions

Value

a symbolic tfp distribution

extractval Extract value in term name

Description

Extract value in term name

Usage

extractval(term, name, null_for_missing = FALSE)

Arguments

term character representing a formula term
name character; the value to extract
null_for_missing logical; if TRUE, returns NULL if argument is missing

Value

the value used for name

Examples

extractval("s(a, la = 2)", "la")
family_to_tfd  

**Character-tfd mapping function**

**Description**

Character-tfd mapping function

**Usage**

family_to_tfd(family)

**Arguments**

- **family**: character defining the distribution

**Value**

a tfp distribution

---

family_to_trafo  

**Character-to-transformation mapping function**

**Description**

Character-to-transformation mapping function

**Usage**

family_to_trafo(family, add_const = 1e-08)

**Arguments**

- **family**: character defining the distribution
- **add_const**: see `make_tfd_dist`

**Value**

a list of transformation for each distribution parameter
**fit**

*Generic train function*

**Description**

Generic train function

**Usage**

```r
fit(object, ...)```

**Arguments**

- `object` : object to apply fit on
- `...` : further arguments passed to the class-specific function

---

**from_dist_to_loss**

*Function to transform a distribution layer output into a loss function*

**Description**

Function to transform a distribution layer output into a loss function

**Usage**

```r
from_dist_to_loss(
  family,
  ind_fun = function(x) tfd_independent(x),
  weights = NULL
)
```

**Arguments**

- `family` : see ?deepregression
- `ind_fun` : function applied to the model output before calculating the log-likelihood. Per default independence is assumed by applying `tfd_independent`.
- `weights` : sample weights

**Value**

loss function
Define Predictor of a Deep Distributional Regression Model

Description

Define Predictor of a Deep Distributional Regression Model

Usage

```r
from_preds_to_dist(
  list_pred_param,
  family = NULL,
  output_dim = 1L,
  mapping = NULL,
  from_distfun_to_dist = distfun_to_dist,
  add_layer_shared_pred = function(x, units) layer_dense(x, units = units, use_bias = FALSE)
)
```

Arguments

- `list_pred_param` list of input-output(-lists) generated from `subnetwork_init`
- `family` see `?deepregression`; if `NULL`, concatenated `list_pred_param` entries are returned (after applying `mapping` if provided)
- `output_dim` dimension of the output
- `mapping` a list of integers. The i-th list item defines which elements of `list_pred_param` are used for the i-th parameter. For example, `map = list(1,2,1:2)` means that `list_pred_param[[1]]` is used for the first distribution parameter, `list_pred_param[[2]]` for the second distribution parameter and `list_pred_param[[3]]` for both distribution parameters (and then added once to `list_pred_param[[1]]` and once to `list_pred_param[[2]]`)
- `from_distfun_to_dist` function creating a `tfp` distribution based on the prediction tensors and `dist_fun`. See `?distfun_to_dist`
- `add_layer_shared_pred` layer to extend shared layers defined in `mapping`

Value

a list with input tensors and output tensors that can be passed to, e.g., `keras_model`
get_distribution  
*Function to return the fitted distribution*

**Description**

Function to return the fitted distribution

**Usage**

```r
get_distribution(x, data = NULL, force_float = FALSE)
```

**Arguments**

- `x`  
  the fitted deepregression object
- `data`  
  an optional data set
- `force_float`  
  forces conversion into float tensors

---

get_partial_effect  
*Return partial effect of one smooth term*

**Description**

Return partial effect of one smooth term

**Usage**

```r
get_partial_effect(
  object, 
  name, 
  return_matrix = FALSE,
  which_param = 1,
  newdata = NULL
)
```

**Arguments**

- `object`  
  deepregression object
- `name`  
  string; for partial match with smooth term
- `return_matrix`  
  logical; whether to return the design matrix or
- `which_param`  
  integer; which distribution parameter the partial effect (FALSE, default)
- `newdata`  
  data.frame; new data (optional)
get_type_pfc  Function to subset parsed formulas

Description
Function to subset parsed formulas

Usage
get_type_pfc(pfc, type = NULL)

Arguments
  pfc  list of parsed formulas
  type either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of

get_weight_by_name  Function to retrieve the weights of a structured layer

Description
Function to retrieve the weights of a structured layer

Usage
get_weight_by_name(mod, name, param_nr = 1)

Arguments
  mod fitted deepregression object
  name name of partial effect
  param_nr distribution parameter number

Value
  weight matrix
handle_gam_term

Function to define smoothness and call mgcv’s smooth constructor

Description

Function to define smoothness and call mgcv’s smooth constructor

Usage

handle_gam_term(object, data, controls)

Arguments

- object: character defining the model term
- data: data.frame or list
- controls: controls for penalization

Value

constructed smooth term

keras_dr

Compile a Deep Distributional Regression Model

Description

Compile a Deep Distributional Regression Model

Usage

keras_dr(
    list_pred_param,
    weights = NULL,
    optimizer = tf$keras$optimizers$Adam(),
    model_fun = keras_model,
    monitor_metrics = list(),
    from_preds_to_output = from_preds_to_dist,
    loss = from_dist_to_loss(family = list(...)$family, weights = weights),
    additional_penalty = NULL,
    ...
)
**layer_add_identity**

**Arguments**

- list_pred_param: list of input-output(-lists) generated from subnetwork_init
- weights: vector of positive values; optional (default = 1 for all observations)
- optimizer: optimizer used. Per default Adam
- model_fun: which function to use for model building (default keras_model)
- monitor_metrics: Further metrics to monitor
- from_preds_to_output: function taking the list_pred_param outputs and transforms it into a single network output
- loss: the model's loss function; per default evaluated based on the arguments family and weights using from_dist_to_loss
- additional_penalty: a penalty that is added to the negative log-likelihood; must be a function of model$trainable_weights with suitable subsetting
- ...: arguments passed to from_preds_to_output

**Value**

a list with input tensors and output tensors that can be passed to, e.g., keras_model

---

**layer_add_identity**  
*Convenience layer function*

**Description**

Convenience layer function

**Usage**

layer_add_identity(inputs)

layer_concatenate_identity(inputs)

**Arguments**

- inputs: list of tensors

**Details**

convenience layers to work with list of inputs where inputs can also have length one

**Value**

tensor
log_score

Function to return the log_score

Description

Function to return the log_score

Usage

log_score(
  x,
  data = NULL,
  this_y = NULL,
  ind_fun = function(x) tfd_independent(x, 1),
  convert_fun = as.matrix,
  summary_fun = function(x) x
)

Arguments

x the fitted deepregression object
data an optional data set
this_y new y for optional data
ind_fun function indicating the dependency; per default (iid assumption) tfd_independent is used.
convert_fun function that converts Tensor; per default as.matrix
summary_fun function summarizing the output; per default the identity

loop_through_pfc_and_call_trafo

Function to loop through parsed formulas and apply data trafo

Description

Function to loop through parsed formulas and apply data trafo

Usage

loop_through_pfc_and_call_trafo(pfc, newdata = NULL)

Arguments

pfc list of processor transformed formulas
newdata list in the same format as the original data
make_folds

Generate folds for CV out of one hot encoded matrix

Usage

make_folds(mat, val_train = 0, val_test = 1)

Arguments

mat  matrix with columns corresponding to folds and entries corresponding to a one
     hot encoding
val_train the value corresponding to train, per default 0
val_test the value corresponding to test, per default 1

Details

val_train and val_test can both be a set of value

makeInputs

Convenience layer function

Description

Convenience layer function

Usage

makeInputs(pp, param_nr)

Arguments

pp  processed predictors
param_nr  integer for the parameter

Value

input tensors with appropriate names
make_generator

creates a generator for training

**Description**

creates a generator for training

**Usage**

```r
make_generator(
  input_x, 
  input_y = NULL, 
  batch_size, 
  sizes, 
  shuffle = TRUE, 
  seed = 42L 
)
```

**Arguments**

- `input_x`: list of matrices
- `input_y`: list of matrix
- `batch_size`: integer
- `sizes`: sizes of the image including colour channel
- `shuffle`: logical for shuffling data
- `seed`: seed for shuffling in generators

**Value**

generator for all x and y

---

**make_generator_from_matrix**

*Make a DataGenerator from a data.frame or matrix*

**Description**

Creates a Python Class that internally iterates over the data.
Usage

```r
make_generator_from_matrix(
    x,
    y = NULL,
    generator = image_data_generator(),
    batch_size = 32L,
    shuffle = TRUE,
    seed = 1L
)
```

Arguments

- **x**: matrix;
- **y**: vector;
- **generator**: generator as e.g. obtained from `keras::image_data_generator`. Used for consistent train-test splits.
- **batch_size**: integer
- **shuffle**: logical; Should data be shuffled?
- **seed**: integer; seed for shuffling data.

Description

Families for deepregression

Usage

```r
make_tfd_dist(family, add_const = 1e-08, output_dim = 1L, trafo_list = NULL)
```

Arguments

- **family**: character vector
- **add_const**: small positive constant to stabilize calculations
- **output_dim**: number of output dimensions of the response (larger 1 for multivariate case)
- **trafo_list**: list of transformations for each distribution parameter. Per default the transformation listed in details is applied.
To specify a custom distribution, define the function as follows:

```r
function(x) do.call(your_tfd_dist,lapply(1:ncol(x),function(i) your_trafo_list_on_inputs[[i]](x[,i,drop=FALSE))))
```

and pass it to `deepregression` via the `dist_fun` argument. Currently the following distributions are supported with parameters (and corresponding inverse link function in brackets):

- "normal": normal distribution with location (identity), scale (exp)
- "bernoulli": bernoulli distribution with logits (identity)
- "bernoulli_prob": bernoulli distribution with probabilities (sigmoid)
- "beta": beta with concentration 1 = alpha (exp) and concentration 0 = beta (exp)
- "betar": beta with mean (sigmoid) and scale (sigmoid)
- "cauchy": location (identity), scale (exp)
- "chi2": cauchy with df (exp)
- "chi": cauchy with df (exp)
- "exponential": exponential with lambda (exp)
- "gamma": gamma with concentration (exp) and rate (exp)
- "gammar": gamma with location (exp) and scale (exp)
- "gumbel": gumbel with location (identity), scale (exp)
- "half_cauchy": half cauchy with location (identity), scale (exp)
- "half_normal": half normal with scale (exp)
- "horseshoe": horseshoe with scale (exp)
- "inverse_gamma": inverse gamma with concentration (exp) and rate (exp)
- "inverse_gamma_ls": inverse gamma with location (exp) and variance (1/exp)
- "inverse_gaussian": inverse Gaussian with location (exp) and concentration (exp)
- "laplace": Laplace with location (identity) and scale (exp)
- "log_normal": Log-normal with location (identity) and scale (exp) of underlying normal distribution
- "logistic": logistic with location (identity) and scale (exp)
- "negbinom": neg. binomial with count (exp) and prob (sigmoid)
- "negbinom_ls": neg. binomial with mean (exp) and clutter factor (exp)
- "pareto": Pareto with concentration (exp) and scale (exp)
- "pareto_ls": Pareto location scale version with mean (exp) and scale (exp), which corresponds to a Pareto distribution with parameters scale = mean and concentration = 1/sigma, where sigma is the scale in the pareto_ls version.
- "poisson": poisson with rate (exp)
- "poisson_lograte": poisson with lograte (identity))
- "student_t": Student’s t with df (exp)
- "student_t_ls": Student’s t with df (exp), location (identity) and scale (exp)
- "uniform": uniform with upper and lower (both identity)
- "zib": Zero-inflated negative binomial with mean (exp), variance (exp) and prob (sigmoid)
- "zip": Zero-inflated poisson distribution with mean (exp) and prob (sigmoid)
names_families  

*Returns the parameter names for a given family*

**Description**

Returns the parameter names for a given family

**Usage**

`names_families(family)`

**Arguments**

- `family`  character specifying the family as defined by `deepregression`

**Value**

vector of parameter names

---

**orthog_control**  

*Options for orthogonalization*

**Description**

Options for orthogonalization

**Usage**

```r
orthog_control(
    split_fun = split_model,
    orthog_type = c("tf", "manual"),
    orthogonalize = options()$orthogonalize,
    identify_intercept = options()$identify_intercept,
    deep_top = NULL
)
```

**Arguments**

- `split_fun`  a function separating the deep neural network in two parts so that the orthogonalization can be applied to the first part before applying the second network part; per default, the function `split_model` is used which assumes a dense layer as penultimate layer and separates the network into a first part without this last layer and a second part only consisting of a single dense layer that is fed into the output layer
**orthog_type**

one of two options; If "manual", the QR decomposition is calculated before model fitting, otherwise ("tf") a QR is calculated in each batch iteration via TF. The first only works well for larger batch sizes or ideally batch_size == NROW(y).

**orthogonalize**

logical; if set to TRUE, automatic orthogonalization is activated

**identify_intercept**

whether to orthogonalize the deep network w.r.t. the intercept to make the intercept identifiable

**deep_top**

function; optional function to put on top of the deep network instead of splitting the function using split_fun

---

**Value**

Returns a list with options

---

**Description**

Options for penalty setup in the pre-processing

**Usage**

```r
penalty_control(
  defaultSmoothing = NULL,
  df = 10,
  null_space_penalty = FALSE,
  absorb_cons = FALSE,
  anisotropic = TRUE,
  zero_constraint_for_smooths = TRUE,
  hat1 = FALSE,
  sp_scale = function(x) 1/NROW(x)
)
```

**Arguments**

- **defaultSmoothing**
  function applied to all s-terms, per default (NULL) the minimum df of all possible terms is used. Must be a function the smooth term from mgcv’s smoothCon and an argument df.

- **df**
  degrees of freedom for all non-linear structural terms (default = 7); either one common value or a list of the same length as number of parameters; if different df values need to be assigned to different smooth terms, use df as an argument for s(), te() or ti()
null_space_penalty

logical value; if TRUE, the null space will also be penalized for smooth effects. Per default, this is equal to the value give in variational.

absorb_cons

logical; adds identifiability constraint to the basisi. See ?mgcv::smoothCon for more details.

anisotropic

whether or not use anisotropic smoothing (default is TRUE)

zero_constraint_for_smooths

logical; the same as absorb_cons, but done explicitly. If true a constraint is put on each smooth to have zero mean. Can be a vector of length(list_of_formulas) for each distribution parameter.

hat1

logical; if TRUE, the smoothing parameter is defined by the trace of the hat matrix sum(diag(H)), else sum(diag(2*H-HH))

sp_scale

function of response; for scaling the penalty (1/n per default)

Value

Returns a list with options

---

plot.deepregression  Generic functions for deepregression models

Description

Generic functions for deepregression models
Predict based on a deepregression object
Function to extract fitted distribution
Fit a deepregression model (pendant to fit for keras)
Extract layer weights / coefficients from model
Print function for deepregression model
Cross-validation for deepgression objects
mean of model fit
Standard deviation of fit distribution
Calculate the distribution quantiles

Usage

## S3 method for class 'deepregression'
plot(
x,
which = NULL,
which_param = 1,
only_data = FALSE,
grid_length = 40,
plot.deepregression

```r

## S3 method for class 'deepregression'
predict(
  object,
  newdata = NULL,
  batch_size = NULL,
  apply_fun = tfd_mean,
  convert_fun = as.matrix,
  ...
)

## S3 method for class 'deepregression'
fitted(object, apply_fun = tfd_mean, ...)

## S3 method for class 'deepregression'
fit(
  object,
  batch_size = 32,
  epochs = 10,
  early_stopping = FALSE,
  early_stopping_metric = "val_loss",
  verbose = TRUE,
  view_metrics = FALSE,
  patience = 20,
  save_weights = FALSE,
  validation_data = NULL,
  validation_split = ifelse(is.null(validation_data), 0.1, 0),
  callbacks = list(),
  convertfun = function(x) tf$constant(x, dtype = "float32"),
  ...
)

## S3 method for class 'deepregression'
coef(object, which_param = 1, type = NULL, ...)

## S3 method for class 'deepregression'
print(x, ...)
```

---

```r

# S3 method for class 'deepregression'
coef(object, which_param = 1, type = NULL, ...)
```
plot.deepregression

cv_folds = 5,
stop_if_nan = TRUE,
mylapply = lapply,
save_weights = FALSE,
callbacks = list(),
save_fun = NULL,
...
)

## S3 method for class 'deepregression'
mean(x, data = NULL, ...)

## S3 method for class 'deepregression'
stddev(x, data = NULL, ...)

## S3 method for class 'deepregression'
quant(x, data = NULL, probs, ...)

Arguments

x a deepregression object
which character vector or number(s) identifying the effect to plot; default plots all effects
which_param integer, indicating for which distribution parameter coefficients should be returned (default is first parameter)
only_data logical, if TRUE, only the data for plotting is returned
grid_length the length of an equidistant grid at which a two-dimensional function is evaluated for plotting.
type either NULL (all types of coefficients are returned), "linear" for linear coefficients or "smooth" for coefficients of smooth terms
... arguments passed to the predict function
object a deepregression model
newdata optional new data, either data.frame or list
batch_size integer, the batch size used for mini-batch training
apply_fun function applied to fitted distribution, per default tfd_mean
convert_fun how should the resulting tensor be converted, per default as.matrix
epochs integer, the number of epochs to fit the model
early_stopping logical, whether early stopping should be used.
early_stopping_metric character, based on which metric should early stopping be triggered (default: "val_loss")
verbose whether to print training in each fold
view_metrics logical, whether to trigger the Viewer in RStudio / Browser.
patience number of patience for early stopping
save_weights logical, whether to save weights in each epoch.
validation_data optional specified validation data
validation_split float in [0,1] defining the amount of data used for validation
callbacks a list of callbacks used for fitting
convertfun function to convert R into Tensor object
plot whether to plot the resulting losses in each fold
print_folds whether to print the current fold
cv_folds an integer if list with train and test data sets
stop_if_nan logical; whether to stop CV if NaN values occur
mylapply lapply function to be used; defaults to lapply
save_fun function applied to the model in each fold to be stored in the final result
data either NULL or a new data set
probs the quantile value(s)

Value

Returns an object drCV, a list, one list element for each fold containing the model fit and the weighthistory.

plot_cv

Plot CV results from deepregression

Description

Plot CV results from deepregression

Usage

plot_cv(x, what = c("loss", "weight"), ...)

Arguments

x drCV object returned by cv.deepregression
what character indicating what to plot (currently supported 'loss' or 'weights')
... further arguments passed to matplot
prepare_data

*Function to prepare data based on parsed formulas*

**Description**

Function to prepare data based on parsed formulas

**Usage**

```r
prepare_data(pfc)
```

**Arguments**

- `pfc`: list of processor transformed formulas

**Value**

list of matrices or arrays

prepare_newdata

*Function to prepare new data based on parsed formulas*

**Description**

Function to prepare new data based on parsed formulas

**Usage**

```r
prepare_newdata(pfc, newdata)
```

**Arguments**

- `pfc`: list of processor transformed formulas
- `newdata`: list in the same format as the original data

**Value**

list of matrices or arrays
**Description**

Control function to define the processor for terms in the formula

**Usage**

```r
processor(
  form,
  data,
  controls,
  output_dim,
  param_nr,
  specials_to_oz = c(),
  automatic_oz_check = TRUE,
  identify_intercept = FALSE,
  ...
)
```

**Arguments**

- `form` the formula to be processed
- `data` the data for the terms in the formula
- `controls` controls for gam terms
- `output_dim` the output dimension of the response
- `param_nr` integer; identifier for the distribution parameter
- `specials_to_oz` specials that should be automatically checked for
- `automatic_oz_check` logical; whether to automatically check for DNNs to be orthogonalized
- `identify_intercept` logical; whether to make the intercept automatically identifiable
- `...` further processors

**Value**

returns a processor function
**quant**

*Generic quantile function*

**Description**

Generic quantile function

**Usage**

`quant(x, ...)`

**Arguments**

- `x` object
- `...` further arguments passed to the class-specific function

---

**separate_define_relation**

*Function to define orthogonalization connections in the formula*

**Description**

Function to define orthogonalization connections in the formula

**Usage**

```r
separate_define_relation(
  form,
  specials,
  specials_to_oz,
  automatic_oz_check = TRUE,
  identify_intercept = FALSE
)
```

**Arguments**

- `form` a formula for one distribution parameter
- `specials` specials in formula to handle separately
- `specials_to_oz` parts of the formula to orthogonalize
- `automatic_oz_check` logical; automatically check if terms must be orthogonalized
- `identify_intercept` logical; whether to make the intercept identifiable

**Value**

Returns a list of formula components with ids and assignments for orthogonalization
**stddev**

*Generic sd function*

**Description**

Generic sd function

**Usage**

```r
stddev(x, ...)
```

**Arguments**

- `x` object
- `...` further arguments passed to the class-specific function

---

**stop_iter_cv_result**

*Function to get the stoppting iteration from CV*

**Description**

Function to get the stoppting iteration from CV

**Usage**

```r
stop_iter_cv_result(
  res,
  thisFUN = mean,
  loss = "validloss",
  whichFUN = which.min
)
```

**Arguments**

- `res` result of cv call
- `thisFUN` aggregating function applied over folds
- `loss` which loss to use for decision
- `whichFUN` which function to use for decision
subnetwork_init  Initializes a Subnetwork based on the Processed Additive Predictor

Description

Initializes a Subnetwork based on the Processed Additive Predictor

Usage

subnetwork_init(
  pp,
  deep_top = NULL,
  orthog_fun = orthog_tf,
  split_fun = split_model,
  param_nr = 1
)

Arguments

- **pp**: processed predictor list from processor
- **deep_top**: keras layer if the top part of the deep network after orthogonalization is different to the one extracted from the provided network
- **orthog_fun**: function used for orthogonalization
- **split_fun**: function to split the network to extract head
- **param_nr**: integer number for the distribution parameter

Value

returns a list of input and output for this additive predictor

tfd_zinb  Implementation of a zero-inflated negbinom distribution for TFP

Description

Implementation of a zero-inflated negbinom distribution for TFP

Usage

tfd_zinb(mu, r, probs)

Arguments

- **mu**, **r**: parameter of the negbin_ls distribution
- **probs**: vector of probabilites of length 2 (probability for poisson and probability for 0s)
**tfd_zip**

Implementation of a zero-inflated poisson distribution for TFP

**Description**

Implementation of a zero-inflated poisson distribution for TFP

**Usage**

```r
tfd_zip(lambda, probs)
```

**Arguments**

- **lambda**: scalar value for rate of poisson distribution
- **probs**: vector of probabilities of length 2 (probability for poisson and probability for 0s)

---

**tf_stride_cols**

Function to index tensors columns

**Description**

Function to index tensors columns

**Usage**

```r
tf_stride_cols(A, start, end = NULL)
```

**Arguments**

- **A**: tensor
- **start**: first index
- **end**: last index (equals start index if NULL)

**Value**

sliced tensor
Index

check_and_install, 3
coef.deepregression
  (plot.deepregression), 22
cv, 4
cv.deepregression
  (plot.deepregression), 22
depregression, 4
distfun_to_dist, 6
extractval, 7
family_to_tfd, 8
family_to_trafo, 8
fit, 9
fit.deepregression
  (plot.deepregression), 22
fitted.deepregression
  (plot.deepregression), 22
from_dist_to_loss, 9
from_preds_to_dist, 10
get_distribution, 11
get_partial_effect, 11
get_type_pfc, 12
get_weight_by_name, 12
handle_gam_term, 13
keras_dr, 13
layer_add_identity, 14
layer_concatenate_identity
  (layer_add_identity), 14
log_score, 15
loop_through_pfc_and_call_trafo, 15
make_folds, 16
make_generator, 17
make_generator_from_matrix, 17
make_tfd_dist, 5, 8, 18
makeInputs, 16
mean.deepregression
  (plot.deepregression), 22
names_families, 20
orthog_control, 5, 20
penalty_control, 5, 21
plot.deepregression, 22
plot_cv, 25
predict.deepregression
  (plot.deepregression), 22
prepare_data, 26
prepare_newdata, 26
print.deepregression
  (plot.deepregression), 22
processor, 27
quant, 28
quant.deepregression
  (plot.deepregression), 22
separate_define_relation, 28
stddev, 29
stddev.deepregression
  (plot.deepregression), 22
stop_iter_cv_result, 29
subnetwork_init, 30
tf_stride_cols, 31
tfd_zinb, 30
tfd_zip, 31