Package: ProcData (via r-universe)

September 2, 2024

Type Package

Title Process Data Analysis

Version 0.4.0

Date 2021-05-29

Description Provides tools for exploratory process data analysis. Process data refers to the data describing participants' problem-solving processes in computer-based assessments. It is often recorded in computer log files. This package provides functions to read, process, and write process data. It also implements two feature extraction methods to compress the information stored in process data into standard numerical vectors. This package also provides recurrent neural network based models that relate response processes with other binary or scale variables of interest. The functions that involve training and evaluating neural networks are wrappers of functions in 'keras'.

BugReports https://github.com/xytangtang/ProcData/issues

License GPL (>= 2)

Depends R (>= 3.5)

Imports Rcpp (>= 0.12.16), keras (>= 2.2.4)

LinkingTo Rcpp

SystemRequirements Python (>= 2.7), Keras (>= 2.0), TensorFlow (>= 1.13)

RoxygenNote 7.1.0

LazyData true

NeedsCompilation yes

Repository https://xytangtang.r-universe.dev

RemoteUrl https://github.com/xytangtang/procdata

RemoteRef HEAD

RemoteSha 27233583440f0c98e0d9c0e5db39ebd6b0b6790e

Contents

ProcData-package *ProcData: A package for process data analysis*

Description

General tools for exploratory process data analysis. Process data refers to the data describing participants' problem solving processes in computer-based assessments. It is often recorded in computer log files. This package a process dataset and functions for reading processes from a csv file, process manipulation, action sequence generators. It also implements two automatic feature extraction methods that compress the information stored in process data, which often has a nonstandard format, into standard numerical vectors. This package also provides recurrent neural network based models that relate response processes with other binary or scale variables of interest. The functions that involve training and evaluating neural networks are based on functions in keras.

Data structure

`ProcData` organizes response processes as an object of class `proc`. Some functions are provided for summarizing and manipulating `proc` objects.

- `summary.proc` calculates summary statistics of a `proc` object.
- `remove_action` removes actions and the corresponding timestamps
- `replace_action` replaces an action by another action
- `combine_actions` combines consecutive action into one action.

Read sequences

- `read.seqs` reads response processes from a csv file.

Sequence generators

- `seq_gen` generates action sequences of an imaginery simulation-based item.
- `seq_gen2` generates action sequences according to a given probability transition matrix.
- `seq_gen3` generates action sequences according to a recurrent neural network.

Feature extraction methods

- `seq2feature_mds` extracts features from response processes by multidimensional scaling.
- `seq2feature_seq2seq` extracts features from response processes by autoencoder.
- `seq2feature_ngram` extracts ngram features from response processes.

Sequence models

- `seqm` fits a neural network model that relates response processes with a response variable.
- `predict.seqm` makes predictions from the models fitted by `seqm`.

Author(s)

Maintainer: Xueying Tang <xueyingtang1989@gmail.com>

Authors:

- Susu Zhang <susu.zhang1992@gmail.com>
- Zhi Wang <zhiwpku@gmail.com>
- Jingchen Liu <jcliu@stat.columbia.edu>
- Zhiliang Ying <zying@stat.columbia.edu>

See Also

Useful links:

- Report bugs at https://github.com/xytangtang/ProcData/issues

---

`action2entropy` *Step 1 of Subtask Analysis: obtaining entropy sequences of action sequences*

Description

`action2entropy` fit a recurrent-neural-network-based action prediction model to a set of action sequences `action_seqs`.

Usage

```
action2entropy(action_seqs, rnn_dim = 20, n_epoch = 50,
  step_size = 0.001, batch_size = 1, optimizer_name = "rmsprop",
  index_valid = 0.2, verbose = FALSE)
```

Arguments

`action_seqs` a list of action sequences

`rnn_dim` latent dimension of RNN

`n_epoch` the number of training epochs.

`step_size` the learning rate of optimizer.

`batch_size` the batch size used in training.

`optimizer_name` a character string specifying the optimizer to be used for training. Availabel options are `"sgd"`, `"rmsprop"`, `"adadelta"`, and `"adam"`.

`index_valid` proportion of sequences used as the validation set or a vector of indices specifying the validation set.

`verbose` logical. If TRUE, training progress is printed.

Value

action2entropy returns a list containing

entropy_seqs — a list of entropy sequences. The length of each entropy sequence is one less than that of the corresponding action sequence.

loss_history — a n_epoch by 2 matrix. The two columns contain the loss at the end of each epoch on the training set and the validation set, respectively.

rnn_dim — the latent dimension of the recurrent neural network

model_fit — a vector of class "raw". It is the serialized version of the trained keras model.

actions — a vector of the actions in action_seqs.

max_len — maximum length of the action sequences.

References

Wang, Z., Tang, X., Liu, J., and Ying, Z. (2020) Subtask analysis of process data through a predictive model. https://arxiv.org/abs/2009.00717

See Also

entropy2segment and segment2subtask for steps 2 and 3 of the subtask analysis procedure; subtask_analysis for the complete procedure.

---

action_seqs_summary *Summarize action sequences*

Description

Summarize action sequences

Usage

```
action_seqs_summary(action_seqs)
```

Arguments

action_seqs — a list of action sequences.

Value

a list containing the following objects:

n_seq — the number of action sequences

n_action — the number of distinct actions

action — the action set

seq_length — sequence lengths

action_freq	action counts
action_seqfreq	the number of sequences that each action appears
trans_count	a length(action) by length(action) matrix whose element in the i-th row and j-th column is the counts of transition from action[i] to action[j].

See Also

time_seqs_summary for summarizing timestamp sequences.

---

aseq2feature_seq2seq *Feature Extraction by action sequence autoencoder*

Description

aseq2feature_seq2seq extract features from action sequences by action sequence autoencoder.

Usage

```
aseq2feature_seq2seq(aseqs, K, rnn_type = "lstm", n_epoch = 50,
  method = "last", step_size = 1e-04, optimizer_name = "adam",
  samples_train, samples_valid, samples_test = NULL, pca = TRUE,
  verbose = TRUE, return_theta = TRUE)
```

Arguments

aseqs	a list of n action sequences. Each element is an action sequence in the form of a vector of actions.
K	the number of features to be extracted.
rnn_type	the type of recurrent unit to be used for modeling response processes. "lstm" for the long-short term memory unit. "gru" for the gated recurrent unit.
n_epoch	the number of training epochs for the autoencoder.
method	the method for computing features from the output of an recurrent neural network in the encoder. Available options are "last" and "avg".
step_size	the learning rate of optimizer.
optimizer_name	a character string specifying the optimizer to be used for training. Availabel options are "sgd", "rmsprop", "adadelta", and "adam".
samples_train	vectors of indices specifying the training, validation and test sets for training autoencoder.
samples_valid	vectors of indices specifying the training, validation and test sets for training autoencoder.
samples_test	vectors of indices specifying the training, validation and test sets for training autoencoder.
pca	logical. If TRUE, the principal components of features are returned. Default is TRUE.
verbose	logical. If TRUE, training progress is printed.
return_theta	logical. If TRUE, extracted features are returned.

Details

This function trains a sequence-to-sequence autoencoder using keras. The encoder of the autoencoder consists of an embedding layer and a recurrent neural network. The decoder consists of another recurrent neural network and a fully connect layer with softmax activation. The outputs of the encoder are the extracted features.

The output of the encoder is a function of the encoder recurrent neural network. It is the last output of the encoder recurrent neural network if `method="last"` and the average of the encoder recurrent nenural network if `method="avg"`.

Value

`aseq2feature_seq2seq` returns a list containing

`theta` a matrix containing `K` features or principal features. Each column is a feature.

`train_loss` a vector of length `n_epoch` recording the trace of training losses.

`valid_loss` a vector of length `n_epoch` recording the trace of validation losses.

`test_loss` a vector of length `n_epoch` recording the trace of test losses. Exists only if `samples_test` is not `NULL`.

See Also

`chooseK_seq2seq` for choosing `K` through cross-validation.

Other feature extraction methods: `atseq2feature_seq2seq`, `seq2feature_mds_large`, `seq2feature_mds`, `seq2feature_ngram`, `seq2feature_seq2seq`, `tseq2feature_seq2seq`

Examples

```
if (!system("python -c 'import tensorflow as tf'", ignore.stdout = TRUE, ignore.stderr= TRUE)) {
  n <- 50
  seqs <- seq_gen(n)
  seq2seq_res <- aseq2feature_seq2seq(seqs$action_seqs, 5, rnn_type="lstm", n_epoch=5,
                                   samples_train=1:40, samples_valid=41:50)
  features <- seq2seq_res$theta
  plot(seq2seq_res$train_loss, col="blue", type="l")
  lines(seq2seq_res$valid_loss, col="red")
}
```

`atseq2feature_seq2seq` *Feature Extraction by action and time sequence autoencoder*

Description

`atseq2feature_seq2seq` extract features from action and timestamp sequences by a sequence autoencoder.

Usage

```
atseq2feature_seq2seq(atseqs, K, weights = c(1, 0.5),
  cumulative = FALSE, log = TRUE, rnn_type = "lstm", n_epoch = 50,
  method = "last", step_size = 1e-04, optimizer_name = "rmsprop",
  samples_train, samples_valid, samples_test = NULL, pca = TRUE,
  verbose = TRUE, return_theta = TRUE)
```

Arguments

`atseqs` a list of two elements, first element is the list of `n` action sequences, Each element is an action sequence in the form of a vector of actions. The second element is the list of `n` timestamp sequences corresponding to the action sequences. Each element is a numeric sequence in the form of a vector of timestamps associated with actions, with the timestamp of the first event (e.g., "start") of 0.

`K` the number of features to be extracted.

`weights` a vector of 2 elements for the weight of the loss of action sequences (categorical_crossentropy) and time sequences (mean squared error), respectively. The total loss is calculated as the weighted sum of the two losses.

`cumulative` logical. If TRUE, the sequence of cumulative time up to each event is used as input to the neural network. If FALSE, the sequence of inter-arrival time (gap time between an event and the previous event) will be used as input to the neural network. Default is FALSE.

`log` logical. If TRUE, for the timestamp sequences, input of the neural net is the base-10 log of the original sequence of times plus 1 (i.e., log10(t+1)). If FALSE, the original sequence of times is used.

`rnn_type` the type of recurrent unit to be used for modeling response processes. `"lstm"` for the long-short term memory unit. `"gru"` for the gated recurrent unit.

`n_epoch` the number of training epochs for the autoencoder.

`method` the method for computing features from the output of an recurrent neural network in the encoder. Available options are `"last"` and `"avg"`.

`step_size` the learning rate of optimizer.

`optimizer_name` a character string specifying the optimizer to be used for training. Availabel options are `"sgd"`, `"rmsprop"`, `"adadelta"`, and `"adam"`.

`samples_train` vectors of indices specifying the training, validation and test sets for training autoencoder.

`samples_valid` vectors of indices specifying the training, validation and test sets for training autoencoder.

`samples_test` vectors of indices specifying the training, validation and test sets for training autoencoder.

`pca` logical. If TRUE, the principal components of features are returned. Default is TRUE.

`verbose` logical. If TRUE, training progress is printed.

`return_theta` logical. If TRUE, extracted features are returned.

Details

This function trains a sequence-to-sequence autoencoder using keras. The encoder of the autoencoder consists of a recurrent neural network. The decoder consists of another recurrent neural network followed by a fully connected layer with softmax activation for actions and another fully connected layer with ReLU activation for times. The outputs of the encoder are the extracted features.

The output of the encoder is a function of the encoder recurrent neural network. It is the last latent state of the encoder recurrent neural network if `method="last"` and the average of the encoder recurrent neural network latent states if `method="avg"`.

Value

`tseq2feature_seq2seq` returns a list containing

`theta` a matrix containing `K` features or principal features. Each column is a feature.

`train_loss` a vector of length `n_epoch` recording the trace of training losses.

`valid_loss` a vector of length `n_epoch` recording the trace of validation losses.

`test_loss` a vector of length `n_epoch` recording the trace of test losses. Exists only if `samples_test` is not `NULL`.

See Also

`chooseK_seq2seq` for choosing `K` through cross-validation.

Other feature extraction methods: `aseq2feature_seq2seq`, `seq2feature_mds_large`, `seq2feature_mds`, `seq2feature_ngram`, `seq2feature_seq2seq`, `tseq2feature_seq2seq`

Examples

```
if (!system("python -c 'import tensorflow as tf'", ignore.stdout = TRUE, ignore.stderr= TRUE)) {
  n <- 50
  data(cc_data)
  samples <- sample(1:length(cc_data$seqs$time_seqs), n)
  atseqs <- sub_seqs(cc_data$seqs, samples)
 action_and_time_seq2seq_res <- atseq2feature_seq2seq(atseqs, 5, rnn_type="lstm", n_epoch=5,
                                   samples_train=1:40, samples_valid=41:50)
  features <- action_and_time_seq2seq_res$theta
  plot(action_and_time_seq2seq_res$train_loss, col="blue", type="l",
       ylim = range(c(action_and_time_seq2seq_res$train_loss,
                      action_and_time_seq2seq_res$valid_loss)))
  lines(action_and_time_seq2seq_res$valid_loss, col="red", type = 'l')
}
```

`calculate_dist_cpp` *Calculate "oss_action" dissimilarity matrix through Rcpp*

Description

Calculate "oss_action" dissimilarity matrix through Rcpp

Usage

```
calculate_dist_cpp(seqs)
```

Arguments

`seqs`	a list of action sequences

Value

`calculate_dist_cpp` returns the "oss_action" dissimilarity matrix of the action sequences in `seqs`.

`cc_data` *Data of item CP025Q01 (climate control item 1) in PISA 2012*

Description

A dataset containing the response processes and binary response outcomes of 16763 respondents.

Usage

```
cc_data
```

Format

A list with two elements.

seqs An object of class "`proc`" containing the action sequences and the time sequences of the respondents.

responses Binary responses of 16763 respondents. The order of the respondents matches that in `seqs`.

Source

item interface: `http://www.oecd.org/pisa/test-2012/testquestions/question3/`

chooseK_mds *Choose the number of multidimensional scaling features*

Description

chooseK_mds choose the number of multidimensional scaling features to be extracted by cross-validation.

Usage

```
chooseK_mds(seqs = NULL, K_cand, dist_type = "oss_action",
  n_fold = 5, max_epoch = 100, step_size = 0.01, tot = 1e-06,
  return_dist = FALSE, L_set = 1:3)
```

Arguments

seqs a "proc" object or a square matrix. If a squared matrix is provided, it is treated as the dissimilary matrix of a group of response processes.

K_cand the candidates of the number of features.

dist_type a character string specifies the dissimilarity measure for two response processes. See 'Details'.

n_fold the number of folds for cross-validation.

max_epoch the maximum number of epochs for stochastic gradient descent.

step_size the step size of stochastic gradient descent.

tot the accuracy tolerance for determining convergence.

return_dist logical. If TRUE, the dissimilarity matrix will be returned. Default is FALSE.

L_set length of ngrams considered

Value

chooseK_mds returns a list containing

K the value in K_cand producing the smallest cross-validation loss.

K_cand the candidates of the number of features.

cv_loss the cross-validation loss for each candidate in K_cand.

dist_mat the dissimilary matrix. This element exists only if return_dist=TRUE.

References

Gomez-Alonso, C. and Valls, A. (2008). A similarity measure for sequences of categorical data based on the ordering of common elements. In V. Torra & Y. Narukawa (Eds.) *Modeling Decisions for Artificial Intelligence*, (pp. 134-145). Springer Berlin Heidelberg.

See Also

seq2feature_mds for feature extraction after choosing the number of features.

Examples

```
n <- 50
set.seed(12345)
seqs <- seq_gen(n)
K_res <- chooseK_mds(seqs, 5:10, return_dist=TRUE)
theta <- seq2feature_mds(K_res$dist_mat, K_res$K)$theta
```

chooseK_seq2seq *Choose the number of autoencoder features*

Description

chooseK_seq2seq chooses the number of features to be extracted by cross-validation.

Usage

```
chooseK_seq2seq(seqs, ae_type, K_cand, rnn_type = "lstm", n_epoch = 50,
  method = "last", step_size = 1e-04, optimizer_name = "adam",
  n_fold = 5, cumulative = FALSE, log = TRUE, weights = c(1, 0.5),
  valid_prop = 0.1, verbose = TRUE)
```

Arguments

seqs	an object of class "proc".
ae_type	a string specifies the type of autoencoder. The autoencoder can be an action sequence autoencoder ("action"), a time sequence autoencoder ("time"), or an action-time sequence autoencoder ("both").
K_cand	the candidates of the number of features.
rnn_type	the type of recurrent unit to be used for modeling response processes. "lstm" for the long-short term memory unit. "gru" for the gated recurrent unit.
n_epoch	the number of training epochs for the autoencoder.
method	the method for computing features from the output of an recurrent neural network in the encoder. Available options are "last" and "avg".
step_size	the learning rate of optimizer.
optimizer_name	a character string specifying the optimizer to be used for training. Availabel options are "sgd", "rmsprop", "adadelta", and "adam".
n_fold	the number of folds for cross-validation.
cumulative	logical. If TRUE, the sequence of cumulative time up to each event is used as input to the neural network. If FALSE, the sequence of inter-arrival time (gap time between an event and the previous event) will be used as input to the neural network. Default is FALSE.

`log` logical. If TRUE, for the timestamp sequences, input of the neural net is the base-10 log of the original sequence of times plus 1 (i.e., log10(t+1)). If FALSE, the original sequence of times is used.

`weights` a vector of 2 elements for the weight of the loss of action sequences (categorical_crossentropy) and time sequences (mean squared error), respectively. The total loss is calculated as the weighted sum of the two losses.

`valid_prop` the proportion of validation samples in each fold.

`verbose` logical. If TRUE, training progress is printed.

Value

`chooseK_seq2seq` returns a list containing

`K` the candidate in `K_cand` producing the smallest cross-validation loss.

`K_cand` the candidates of number of features.

`cv_loss` the cross-validation loss for each candidate in `K_cand`.

See Also

`seq2feature_seq2seq` for feature extraction given the number of features.

`combine_actions` *Combine consecutive actions into a single action*

Description

Combine the action pattern described in `old_actions` into a single action `new_action`. The timestamp of the combined action can be the timestamp of the first action in the action pattern, the timestamp of the last action in the action pattern, or the average of the two timestamps.

Usage

```
combine_actions(seqs, old_actions, new_action, timestamp = "first")
```

Arguments

`seqs` an object of class "`proc`"

`old_actions` a character vector giving consecutive actions to be replaced.

`new_action` a string giving the combined action

`timestamp` "first", "last", or "avg", specifying how the timestamp of the combined action should be derived.

Value

an object of class "`proc`"

Examples

```
seqs <- seq_gen(100)
new_seqs <- combine_actions(seqs,
                            old_actions=c("OPT1_3", "OPT2_2", "RUN"),
                            new_action="KEY_ACTION")
```

count_actions *Count action appearances*

Description

This function counts the appearances of each action in actions in action sequence x.

Usage

```
count_actions(x, actions)
```

Arguments

x	an action sequence.
actions	a set of actions whose number of appearances will be count.

Value

an integer vector of counts.

entropy2segment *Step 2 of Subtask Analysis: Segmenting Entropy Sequences*

Description

entropy2segment segments the entropy sequences in entropy_seqs using segment_function.

Usage

```
entropy2segment(entropy_seqs, lambda = 0.3, verbose = FALSE)
```

Arguments

entropy_seqs	a list of entropy sequences
lambda	a number between 0 and 1
verbose	print progress if TRUE. default is FALSE

Value

a list containg the segment boundaries of each entropy sequence.

References

Wang, Z., Tang, X., Liu, J., and Ying, Z. (2020) Subtask analysis of process data through a predictive model. https://arxiv.org/abs/2009.00717

See Also

`action2entropy` and `segment2subtask` for steps 1 and 3 of the subtask analysis procedure; `subtask_analysis` for the complete procedure.

`plot.subtask` *Plot an subtask Object*

Description

Plot the subtask analysis results for either the entire dataset or individual sequences.

Usage

```
## S3 method for class 'subtask'
plot(object, type = "all", index = NULL,
  max_len = 5, col.subtask = 1:length(object$subtasks),
  cex.action = 0.5, lty = 1, pch = 16, srt = -90,
  plot_legend = TRUE, legend_pos = "topright", ...)
```

Arguments

`object`	an object of class `"subtask"`
`type`	`"all"` or `"individual"`
`index`	a vector of indices of sequences to plot
`max_len`	maximum length of plotted subtasks
`col.subtask`	a vector of colors for subtasks
`lty`	line types
`pch`	point characters
`plot_legend`	a logical value. If `TRUE` (default), plot the legend of subtasks.
`legend_pos`	a character string or the coordinates to be used to position the legend.
`...`	other arguments passed to `legend`

Value

this function does not return values

See Also

`plot_subtask_seq`, `plot_subtask_seqs`.

plot_subtask_seq *Plot Subtask Analysis Results for One Sequence*

Description

Plot Subtask Analysis Results for One Sequence

Usage

```
plot_subtask_seq(action_seq, entropy_seq, subtask_seq, subtasks,
  col.subtask = 1:length(subtasks), cex.action = 0.5, lty = 1,
  pch = 16, srt = -90, plot_legend = TRUE, legend_pos = "topleft",
  ...)
```

Arguments

`action_seq`	an action sequence
`entropy_seq`	an entropy sequence
`subtask_seq`	a subtask sequence
`subtasks`	a vector of all subtasks
`col.subtask`	a vector of colors for subtasks
`lty`	line types
`pch`	point characters
`plot_legend`	a logical value. If `TRUE` (default), plot the legend of subtasks.
`legend_pos`	a character string or the coordinates to be used to position the legend.
`...`	other arguments passed to `legend`

Value

this function does not return values

See Also

`plot_subtask_seqs` for plotting results for all sequences. `plot.subtask` for the plot method of `"subtask"` object.

plot_subtask_seqs *Plot Subtask Analysis Results for Entire Dataset*

Description

Plot Subtask Analysis Results for Entire Dataset

Usage

```
plot_subtask_seqs(subtask_seqs, subtasks, max_len = 5,
  col.subtask = 1:length(subtasks), plot_legend = TRUE,
  legend_pos = "topright", ...)
```

Arguments

`subtask_seqs`	a list of subtask sequences
`subtasks`	a vector of all subtasks
`max_len`	maximum length of plotted subtasks
`col.subtask`	a vector of colors for subtasks
`plot_legend`	a logical value. If `TRUE` (default), plot the legend of subtasks.
`legend_pos`	a character string or the coordinates to be used to position the legend.
`...`	other arguments passed to `legend`

Value

this function does not return values

See Also

`plot_subtask_seq` for ploting results for one sequence. `plot.subtask` for ploting an object of class `"subtask"`

predict.seqm *Predict method for sequence models*

Description

Obtains predictions from a fitted sequence model object.

Usage

```
## S3 method for class 'seqm'
predict(object, new_seqs, new_covariates = NULL,
  type = "response", ...)
```

Arguments

`object`	a fitted object of class `"seqm"` from `seqm`.
`new_seqs`	an object of class `"proc"` with which to predict.
`new_covariates`	a new covariate matrix with which to predict.
`type`	a string specifying whether to predict responses (`"response"`) or features (`"feature"`) or both (`"both"`).
`...`	further arguments to be passed to `predict.keras.engine.training.Model`.

Details

It unserialize `object$model_fit` to obtain a keras model of class `"keras.engin.training.Model"` and then calls `predict` to obtain predictions.

Value

If `type="response"`, a vector of predictions. The vector gives the probabilities of the response variable being one if `response_type="binary"`. If `type="feature"`, a matrix of rnn outputs. If `type="both"`, a list containing both the vector of response variable prediction and the rnn output matrix.

See Also

`seqm` for fitting sequence models.

`print.proc` *Print method for class* `"proc"`

Description

Print method for class `"proc"`

Usage

```
## S3 method for class 'proc'
print(x, n = 5, index = NULL, quote = FALSE, ...)
```

Arguments

`x`	an object of class `"proc"`
`n`	number of processes to be printed.
`index`	indice of processes to be printed.
`quote`	logical, indicating whether or not strings should be printed with surrounding quotes.
`...`	not used.

Value

print.proc invisibly returns the "proc" object it prints.

print.summary.proc *Print method for class "summary.proc"*

Description

Print method for class "summary.proc"

Usage

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

Arguments

x	an object of class "proc"
...	not used.

Value

No return value.

proc *Class "proc" constructor*

Description

Create a "proc" object from given action sequences and timestamp sequences

Usage

```
proc(action_seqs, time_seqs, ids = NULL)
```

Arguments

action_seqs	a list of action sequences.
time_seqs	a list of timestamp sequences.
ids	ids identifiers of response processes.

Details

An object of class "proc" is a list containing the following components:

- action_seqsa list of action sequences.
- time_seqsa list of timestamp sequences.

The names of the elements in `seqs$action_seqs` and `seqs$time_seqs` are process identifiers.

Value

an object of class "`proc`" containing the provided action and timestamp sequences.

`read.seqs` *Reading response processes from csv files*

Description

Reads a csv file and creates response process data.

Usage

```
read.seqs(file, style, id_var = NULL, action_var = NULL,
  time_var = NULL, step_sep = ",", ...)
```

Arguments

`file`	the name of the csv file from which the response processes are to be read.
`style`	the style that the response processes are stored. See 'Details'.
`id_var`	a string giving the name of the variable storing the process identifier.
`action_var`	a string giving the name of the variable storing action sequences.
`time_var`	a string giving the name of the variable storing timestamp sequences.
`step_sep`	the step separator characters. It is only used if `style="single"`.
`...`	further arguments to be passed to `read.csv`.

Details

`read.seqs` calls `read.csv` to read process data stored in a csv file into `R`. The csv file to be read should at least include an identifier of distinct response processes, and action sequences. It can also include timestamp sequences.

The response processes (action sequences and timestamp sequences) stored in csv files can be in one of the two styles, "single" and "multiple". In "single" style, each response process occupies a single line. Actions and timestamps at different steps are separated by `step_sep`. In "multiple" style, each response process occupies multiple lines with each step taking up one line.

Value

`read.seqs` returns an object of class "`proc`".

remove_action *Remove actions from response processes*

Description

Remove actions in `actions` and the corresponding timestamps in response processes `seqs`.

Usage

```
remove_action(seqs, actions)
```

Arguments

`seqs`	an object of class "`proc`"
`actions`	a character vector. Each element is an action to be removed.

Value

an object of class "`proc`" with actions in `actions` and the corresponding timestamps removed.

Examples

```
seqs <- seq_gen(10)
new_seqs <- remove_action(seqs, c("RUN", "Start"))
```

remove_repeat *Remove repeated actions*

Description

Remove repeated actions

Usage

```
remove_repeat(seqs, ignore = NULL)
```

Arguments

`seqs`	an object of class "`proc`"
`ignore`	repeated actions in ignore will not be deleted.

Value

an object of class "`proc`"

replace_action *Replace actions in response processes*

Description

Replace old_action with new_action in seqs. Timestamp sequences are not affected.

Usage

```
replace_action(seqs, old_action, new_action)
```

Arguments

seqs	an object of class "proc"
old_action	a string giving the action to be replaced.
new_action	a string giving the action replacing old_action

Value

an object of class "proc"

Examples

```
seqs <- seq_gen(10)
new_seqs <- replace_action(seqs, "Start", "Begin")
```

segment2subtask *Step 3 of Subtask Analysis: Grouping Segments*

Description

segment2subtask clustering action sequence segments according to their action frequency profiles. Each cluster forms a subtask.

Usage

```
segment2subtask(action_seqs, seg_seqs, n_subtask, actions,
  verbose = FALSE, ...)
```

Arguments

action_seqs	a list of action sequences
seg_seqs	a list of segment locations
n_subtask	the desired number of subtasks or a vector of candidate number of subtasks
actions	a set of actions
verbose	logical. If TRUE, training progress is printed.
...	additional arguments passed to kmeans

Value

a list containing

n_subtask the number of subtasks

subtasks a vector of subtasks

subtask_seqs a list of subtask sequences

tot.withinss a vector of total within cluster sum of squares

relative_cluster_profiles
a n_subtask by length(actions) matrix. Each row gives the action frequency profile of each subtask relative to the overall action frequency profile

References

Wang, Z., Tang, X., Liu, J., and Ying, Z. (2020) Subtask analysis of process data through a predictive model. https://arxiv.org/abs/2009.00717

See Also

action2entropy and segment2subtask for steps 1 and 3 of the subtask analysis procedure; subtask_analysis for the complete procedure.

segment_function *Segment an entropy sequence*

Description

segment_function segments the entropy sequence entropy_seq by identifying deep U-shaped curves in it.

Usage

```
segment_function(entropy_seq, lambda)
```

Arguments

entropy_seq a vector of entropies

lambda a number between 0 and 1.

Value

a vector of segment boundaries

seq2feature_mds *Feature extraction via multidimensional scaling*

Description

seq2feature_mds extracts K features from response processes by multidimensional scaling.

Usage

```
seq2feature_mds(seqs = NULL, K = 2, method = "auto",
  dist_type = "oss_action", pca = TRUE, subset_size = 100,
  subset_method = "random", n_cand = 10, return_dist = FALSE,
  L_set = 1:3)
```

Arguments

seqs	a "proc" object or a square matrix. If a squared matrix is provided, it is treated as the dissimilary matrix of a group of response processes.
K	the number of features to be extracted.
method	a character string specifies the algorithm used for performing MDS. See 'Details'.
dist_type	a character string specifies the dissimilarity measure for two response processes. See 'Details'.
pca	logical. If TRUE (default), the principal components of the extracted features are returned.
subset_size, n_cand	two parameters used in the large data algorithm. See 'Details' and seq2feature_mds_large.
subset_method	a character string specifying the method for choosing the subset in the large data algorithm. See 'Details' and seq2feature_mds_large.
return_dist	logical. If TRUE, the dissimilarity matrix will be returned. Default is FALSE.
L_set	length of ngrams considered

Details

Since the classical MDS has a computational complexity of order n^3 where n is the number of response processes, it is computational expensive to perform classical MDS when a large number of response processes is considered. In addition, storing an $n \times n$ dissimilarity matrix when n is large require a large amount of memory. In seq2feature_mds, the algorithm proposed in Paradis (2018) is implemented to obtain MDS for large datasets. method specifies the algorithm to be used for obtaining MDS features. If method = "small", classical MDS is used by calling cmdscale. If method = "large", the algorithm for large datasets will be used. If method = "auto" (default), seq2feature_mds selects the algorithm automatically based on the sample size.

dist_type specifies the dissimilarity to be used for measuring the discrepancy between two response processes. If dist_type = "oss_action", the order-based sequence similarity (oss) proposed in Gomez-Alonso and Valls (2008) is used for action sequences. If dist_type = "oss_both", both action sequences and timestamp sequences are used to compute a time-weighted oss.

The number of features to be extracted K can be selected by cross-validation using chooseK_mds.

Value

seq2feature_mds returns a list containing

theta	a numeric matrix giving the K extracted features or principal features. Each column is a feature.
dist_mat	the dissimilary matrix. This element exists only if return_dist=TRUE.

References

Gomez-Alonso, C. and Valls, A. (2008). A similarity measure for sequences of categorical data based on the ordering of common elements. In V. Torra & Y. Narukawa (Eds.) *Modeling Decisions for Artificial Intelligence*, (pp. 134-145). Springer Berlin Heidelberg.

Paradis, E. (2018). Multidimensional scaling with very large datasets. *Journal of Computational and Graphical Statistics*, 27(4), 935-939.

Tang, X., Wang, Z., He, Q., Liu, J., and Ying, Z. (2020) Latent Feature Extraction for Process Data via Multidimensional Scaling. *Psychometrika*, 85, 378-397.

See Also

chooseK_mds for choosing K.

Other feature extraction methods: aseq2feature_seq2seq, atseq2feature_seq2seq, seq2feature_mds_large, seq2feature_ngram, seq2feature_seq2seq, tseq2feature_seq2seq

Examples

```
n <- 50
set.seed(12345)
seqs <- seq_gen(n)
theta <- seq2feature_mds(seqs, 5)$theta
```

seq2feature_mds_large *Feature Extraction by MDS for Large Dataset*

Description

seq2feature_mds_large extracts MDS features from a large number of response processes. The algorithm proposed in Paradis (2018) is implemented with minor variations to perform MDS. The algorithm first selects a relatively small subset of response processes to perform the classical MDS. Then the coordinate of each of the other response processes are obtained by minimizing the loss function related to the target response processes and the those in the subset through BFGS.

Usage

```
seq2feature_mds_large(seqs, K, dist_type = "oss_action", subset_size,
  subset_method = "random", n_cand = 10, pca = TRUE, L_set = 1:3)
```

Arguments

`seqs`	an object of class "`proc`"
`K`	the number of features to be extracted.
`dist_type`	a character string specifies the dissimilarity measure for two response processes. See 'Details'.
`subset_size`	the size of the subset on which classical MDS is performed.
`subset_method`	a character string specifying the method for choosing the subset. It must be one of `"random"`, `"sample_avgmax"`, `"sample_minmax"`, `"full_avgmax"`, and `"full_minmax"`.
`n_cand`	The size of the candidate set when selecting the subset. It is only used when `subset_method` is `"sample_avgmax"` or `"sample_minmax"`.
`pca`	logical. If `TRUE` (default), the principal components of the extracted features are returned.
`L_set`	length of ngrams considered

Value

`seq2feature_mds_large` returns an $n \times K$ matrix of extracted features.

References

Paradis, E. (2018). Multidimensional Scaling with Very Large Datasets. *Journal of Computational and Graphical Statistics*, 27, 935–939.

See Also

Other feature extraction methods: `aseq2feature_seq2seq`, `atseq2feature_seq2seq`, `seq2feature_mds`, `seq2feature_ngram`, `seq2feature_seq2seq`, `tseq2feature_seq2seq`

`seq2feature_mds_stochastic` *Feature extraction by stochastic mds*

Description

Feature extraction by stochastic mds

Usage

```
seq2feature_mds_stochastic(seqs = NULL, K = 2,
  dist_type = "oss_action", max_epoch = 100, step_size = 0.01,
  pca = TRUE, tot = 1e-06, return_dist = FALSE, L_set = 1:3)
```

Arguments

`seqs` a "proc" object or a square matrix. If a squared matrix is provided, it is treated as the dissimilary matrix of a group of response processes.

`K` the number of features to be extracted.

`dist_type` a character string specifies the dissimilarity measure for two response processes. See 'Details'.

`max_epoch` the maximum number of epochs for stochastic gradient descent.

`step_size` the step size of stochastic gradient descent.

`pca` a logical scalar. If `TRUE`, the principal components of the extracted features are returned.

`tot` the accuracy tolerance for determining convergence.

`return_dist` logical. If `TRUE`, the dissimilarity matrix will be returned. Default is `FALSE`.

`L_set` length of ngrams considered.

Value

`seq2feature_mds_stochastic` returns a list containing

`theta` a numeric matrix giving the `K` extracted features or principal features. Each column is a feature.

`loss` the value of the multidimensional scaling objective function.

`dist_mat` the dissimilary matrix. This element exists only if `return_dist=TRUE`.

`seq2feature_ngram` *ngram feature extraction*

Description

`seq2feature_ngram` extracts ngram features from response processes.

Usage

```
seq2feature_ngram(seqs, level = 2, type = "binary", sep = "\t")
```

Arguments

`seqs` an object of class "proc"

`level` an integer specifying the max length of ngrams

`type` a character string (`"binary"`, `"freq"`, or `"weighted"`) specifying the type of ngram features.

`sep` action seperator within ngram.

Details

Three types of ngram features can be extracted. `type = "binary"` gives binary ngram features indicating whether an ngram appears in a response process. `type = "freq"` gives ngram frequency features. Each feature is the count of the corresponding ngram in a response process. `type = "weighted"` gives the weighted ngram features proposed in He and von Davier (2015).

Value

a matrix of ngram features

References

He Q., von Davier M. (2015). Identifying Feature Sequences from Process Data in Problem-Solving Items with N-Grams. In: van der Ark L., Bolt D., Wang WC., Douglas J., Chow SM. (eds) *Quantitative Psychology Research*. Springer Proceedings in Mathematics & Statistics, vol 140. Springer, Cham.

See Also

Other feature extraction methods: `aseq2feature_seq2seq`, `atseq2feature_seq2seq`, `seq2feature_mds_large`, `seq2feature_mds`, `seq2feature_seq2seq`, `tseq2feature_seq2seq`

Examples

```
seqs <- seq_gen(100)
theta <- seq2feature_ngram(seqs)
```

`seq2feature_seq2seq` *Feature Extraction by autoencoder*

Description

`seq2feature_seq2seq` extract features from response processes by autoencoder.

Usage

```
seq2feature_seq2seq(seqs, ae_type = "action", K, rnn_type = "lstm",
  n_epoch = 50, method = "last", step_size = 1e-04,
  optimizer_name = "adam", cumulative = FALSE, log = TRUE,
  weights = c(1, 0.5), samples_train, samples_valid,
  samples_test = NULL, pca = TRUE, verbose = TRUE,
  return_theta = TRUE)
```

Arguments

seqs an object of class "proc".

ae_type a string specifies the type of autoencoder. The autoencoder can be an action sequence autoencoder ("action"), a time sequence autoencoder ("time"), or an action-time sequence autoencoder ("both").

K the number of features to be extracted.

rnn_type the type of recurrent unit to be used for modeling response processes. "lstm" for the long-short term memory unit. "gru" for the gated recurrent unit.

n_epoch the number of training epochs for the autoencoder.

method the method for computing features from the output of an recurrent neural network in the encoder. Available options are "last" and "avg".

step_size the learning rate of optimizer.

optimizer_name a character string specifying the optimizer to be used for training. Availabel options are "sgd", "rmsprop", "adadelta", and "adam".

cumulative logical. If TRUE, the sequence of cumulative time up to each event is used as input to the neural network. If FALSE, the sequence of inter-arrival time (gap time between an event and the previous event) will be used as input to the neural network. Default is FALSE.

log logical. If TRUE, for the timestamp sequences, input of the neural net is the base-10 log of the original sequence of times plus 1 (i.e., log10(t+1)). If FALSE, the original sequence of times is used.

weights a vector of 2 elements for the weight of the loss of action sequences (categorical_crossentropy) and time sequences (mean squared error), respectively. The total loss is calculated as the weighted sum of the two losses.

samples_train, samples_valid, samples_test
vectors of indices specifying the training, validation and test sets for training autoencoder.

pca logical. If TRUE, the principal components of features are returned. Default is TRUE.

verbose logical. If TRUE, training progress is printed.

return_theta logical. If TRUE, extracted features are returned.

Details

This function wraps aseq2feature_seq2seq, tseq2feature_seq2seq, and atseq2feature_seq2seq.

Value

seq2feature_seq2seq returns a list containing

theta a matrix containing K features or principal features. Each column is a feature.

train_loss a vector of length n_epoch recording the trace of training losses.

valid_loss a vector of length n_epoch recording the trace of validation losses.

test_loss a vector of length n_epoch recording the trace of test losses. Exists only if samples_test is not NULL.

References

Tang, X., Wang, Z., Liu, J., and Ying, Z. (2020) An exploratory analysis of the latent structure of process data via action sequence autoencoders. *British Journal of Mathematical and Statistical Psychology*. 74(1), 1-33.

See Also

chooseK_seq2seq for choosing K through cross-validation.

Other feature extraction methods: aseq2feature_seq2seq, atseq2feature_seq2seq, seq2feature_mds_large, seq2feature_mds, seq2feature_ngram, tseq2feature_seq2seq

Examples

```
if (!system("python -c 'import tensorflow as tf'", ignore.stdout = TRUE, ignore.stderr= TRUE)) {
  n <- 50
  data(cc_data)
  samples <- sample(1:length(cc_data$seqs$time_seqs), n)
  seqs <- sub_seqs(cc_data$seqs, samples)

  # action sequence autoencoder
  K_res <- chooseK_seq2seq(seqs=seqs, ae_type="action", K_cand=c(5, 10),
                           n_epoch=5, n_fold=2, valid_prop=0.2)
  seq2seq_res <- seq2feature_seq2seq(seqs=seqs, ae_type="action", K=K_res$K,
                         n_epoch=5, samples_train=1:40, samples_valid=41:50)
  theta <- seq2seq_res$theta

  # time sequence autoencoder
  K_res <- chooseK_seq2seq(seqs=seqs, ae_type="time", K_cand=c(5, 10),
                           n_epoch=5, n_fold=2, valid_prop=0.2)
  seq2seq_res <- seq2feature_seq2seq(seqs=seqs, ae_type="time", K=K_res$K,
                         n_epoch=5, samples_train=1:40, samples_valid=41:50)
  theta <- seq2seq_res$theta

  # action and time sequence autoencoder
  K_res <- chooseK_seq2seq(seqs=seqs, ae_type="both", K_cand=c(5, 10),
                           n_epoch=5, n_fold=2, valid_prop=0.2)
  seq2seq_res <- seq2feature_seq2seq(seqs=seqs, ae_type="both", K=K_res$K,
                         n_epoch=5, samples_train=1:40, samples_valid=41:50)
  theta <- seq2seq_res$theta
  plot(seq2seq_res$train_loss, col="blue", type="l")
  lines(seq2seq_res$valid_loss, col="red")
}
```

seqm *Fitting sequence models*

Description

seqm is used to fit a neural network model relating a response process with a variable.

Usage

```
seqm(seqs, response, covariates = NULL, response_type,
  actions = unique(unlist(seqs$action_seqs)), rnn_type = "lstm",
  include_time = FALSE, time_interval = TRUE, log_time = TRUE,
  K_emb = 20, K_rnn = 20, n_hidden = 0, K_hidden = NULL,
  index_valid = 0.2, verbose = FALSE, max_len = NULL, n_epoch = 20,
  batch_size = 16, optimizer_name = "rmsprop", step_size = 0.001)
```

Arguments

`seqs` an object of class "`proc`".

`response` response variable.

`covariates` covariate matrix.

`response_type` "binary" or "scale".

`actions` a character vector gives all possible actions. It is will be expanded to include all actions appear in `seqs` if necessary.

`rnn_type` the type of recurrent unit to be used for modeling response processes. `"lstm"` for the long-short term memory unit. `"gru"` for the gated recurrent unit.

`include_time` logical. If the timestamp sequence should be included in the model.

`time_interval` logical. If the timestamp sequence is included as a sequence of inter-arrival time.

`log_time` logical. If take the logarithm of the time sequence.

`K_emb` the latent dimension of the embedding layer.

`K_rnn` the latent dimension of the recurrent neural network.

`n_hidden` the number of hidden fully-connected layers.

`K_hidden` a vector of length `n_hidden` specifying the number of nodes in each hidden layer.

`index_valid` proportion of sequences used as the validation set or a vector of indices specifying the validation set.

`verbose` logical. If TRUE, training progress is printed.

`max_len` the maximum length of response processes.

`n_epoch` the number of training epochs.

`batch_size` the batch size used in training.

`optimizer_name` a character string specifying the optimizer to be used for training. Availabel options are `"sgd"`, `"rmsprop"`, `"adadelta"`, and `"adam"`.

`step_size` the learning rate of optimizer.

Details

The model consists of an embedding layer, a recurrent layer and one or more fully connected layers. The embedding layer takes an action sequence and output a sequences of `K` dimensional numeric vectors to the recurrent layer. If `include_time = TRUE`, the embedding sequence is combined with the timestamp sequence in the response process as the input the recurrent layer. The last output of the recurrent layer and the covariates specified in covariates are used as the input of the subsequent fully connected layer. If response_type="binary", the last layer uses the sigmoid activation to produce the probability of the response being one. If response_type="scale", the last layer uses the linear activation. The dimension of the output of other fully connected layers (if any) is specified by K_hidden.

The action sequences are re-coded into integer sequences and are padded with zeros to length max_len before feeding into the model. If the provided max_len is smaller than the length of the longest sequence in seqs, it will be overridden.

Value

seqm returns an object of class "seqm", which is a list containing

structure a string describing the neural network structure.

coefficients a list of fitted coefficients. The length of the list is 6 + 2 * n_hidden. The first element gives the action embedding. Elements 2-4 are parameters in the recurrent unit. The rest of the elements are for the fully connected layers. Elements 4 + (2 * i - 1) and 4 + 2 * i give the parameters for the i-th fully connected layer.

model_fit a vector of class "raw". It is the serialized version of the trained keras model.

feature_model a vector of class "raw". It is the serialized version of the keras model for obtaining the rnn outputs.

include_time if the timestamp sequence is included in the model.

time_interval if inter-arrival time is used.

log_time if the logarithm time is used.

actions all possible actions.

max_len the maximum length of action sequences.

history a n_epoch by 2 matrix giving the training and validation losses at the end of each epoch.

See Also

predict.seqm for the predict method for seqm objects.

Examples

```
if (!system("python -c 'import tensorflow as tf'", ignore.stdout = TRUE, ignore.stderr= TRUE)) {
  n <- 100
  data(cc_data)
  samples <- sample(1:length(cc_data$responses), n)
  seqs <- sub_seqs(cc_data$seqs, samples)

  y <- cc_data$responses[samples]
  x <- matrix(rnorm(n*2), ncol=2)

  index_test <- 91:100
  index_train <- 1:90
  seqs_train <- sub_seqs(seqs, index_train)
  seqs_test <- sub_seqs(seqs, index_test)

  actions <- unique(unlist(seqs$action_seqs))

  ## no covariate is used
  res1 <- seqm(seqs = seqs_train, response = y[index_train],
               response_type = "binary", actions=actions, K_emb = 5, K_rnn = 5,
               n_epoch = 5)
  pred_res1 <- predict(res1, new_seqs = seqs_test)

  mean(as.numeric(pred_res1 > 0.5) == y[index_test])

  ## add more fully connected layers after the recurrent layer.
  res2 <- seqm(seqs = seqs_train, response = y[index_train],
               response_type = "binary", actions=actions, K_emb = 5, K_rnn = 5,
               n_hidden=2, K_hidden=c(10,5), n_epoch = 5)
  pred_res2 <- predict(res2, new_seqs = seqs_test)
  mean(as.numeric(pred_res2 > 0.5) == y[index_test])

  ## add covariates
  res3 <- seqm(seqs = seqs_train, response = y[index_train],
               covariates = x[index_train, ],
               response_type = "binary", actions=actions,
               K_emb = 5, K_rnn = 5, n_epoch = 5)
  pred_res3 <- predict(res3, new_seqs = seqs_test,
                       new_covariates=x[index_test, ])

  ## include time sequences
  res4 <- seqm(seqs = seqs_train, response = y[index_train],
               response_type = "binary", actions=actions,
               include_time=TRUE, K_emb=5, K_rnn=5, n_epoch=5)
  pred_res4 <- predict(res4, new_seqs = seqs_test)
}
```

seq_gen *Action sequence generator*

Description

seq_gen generates action sequences of an imaginary simulation-based item.

Usage

```
seq_gen(n, action_set1 = c("OPT1_1", "OPT1_2", "OPT1_3"),
  action_set2 = c("OPT2_1", "OPT2_2"), answer_set = c("CHECK_A",
  "CHECK_B", "CHECK_C", "CHECK_D"), p1 = rep(1, length(action_set1)),
  p2 = rep(1, length(action_set2)), p_answer = rep(1,
  length(answer_set)), p_continue = 0.5, p_choose = 0.5,
  include_time = FALSE, time_intv_dist = list("exp", 1))
```

Arguments

n An integer. The number of action sequences to be generated.

action_set1, action_set2
Character vectors giving the choices for the first and the second conditions.

answer_set A character vector giving the choices for the answer.

p1, p2 Nonnegative numeric vectors. They are the weights for sampling from action_set1 and action_set2.

p_answer A nonnegative numeric vector giving the weights for sampling from answer_set.

p_continue Probability of running an/another experiment.

p_choose Probability of choosing an answer.

include_time logical. Indicate if timestamp sequences should be generated. Default is FALSE.

time_intv_dist A list specifying the distribution of the inter-arrival time.

Details

The format of the generated sequences resembles that of the response processes of simulation-based items. In these items, participants are asked to answer a question by running simulated experiments in which two conditions can be controlled. A simulated experiment can be run by setting the two conditions at one of the given choices and click "Run" button.

The possible actions are "Start", "End", "Run", and the elements in action_set1, action_set2, and answer_set. The generated sequences begin with "Start" and continue with groups of three actions. Each group of three actions, representing one experiment, consists of an action chosen from action_set1 according to p1, an action chosen from action_set2 according to p2, and "Run". The probability of performing an experiment after "Start" or one experiment is p_continue. After the experiment process, with probability p_choose, an answer will be chosen. The chosen answer is randomly sampled from answer_set according to p_answer. All generated sequences end with "End".

Value

An object of class "proc" with time_seqs = NULL.

See Also

Other sequence generators: seq_gen2, seq_gen3

seq_gen2 *Markov action sequence generator*

Description

seq_gen2 generates action sequences according to a given probability transition matrix.

Usage

```
seq_gen2(n, Pmat = NULL, events = letters, start_index = 1,
  end_index = length(events), max_len = 200, include_time = FALSE,
  time_intv_dist = list("exp", 1))
```

Arguments

`n`	An integer. The number of action sequences to be generated.
`Pmat`	An `N` by `N` probability transition matrix.
`events`	A character vector specifying the set of `N` possible actions. Default is `letters`.
`start_index`	Index of the action indicating the start of an item in `events`.
`end_index`	Index of the action indicating the end of an item in `events`.
`max_len`	Maximum length of generated sequences.
`include_time`	logical. Indicate if timestamp sequences should be generated. Default is FALSE.
`time_intv_dist`	A list specifying the distribution of the inter-arrival time.

Details

This function generates `n` action sequences according `Pmat`. The set of possible actions is `events`. All generated sequences start with `events[start_index]` and end with `events[end_index]`. If `Pmat` is not supplied, actions is uniformly drawn from `events[-start_index]` until `events[end_index]` appears.

Value

An object of class "`proc`" with `time_seqs` = `NULL`.

See Also

Other sequence generators: `seq_gen3`, `seq_gen`

`seq_gen3` *RNN action sequence generator*

Description

`seq_gen3` generates action sequences according to a recurrent neural network

Usage

```
seq_gen3(n, events = letters, rnn_type = "lstm", K = 10,
  weights = NULL, max_len = 100, initial_state = NULL,
  start_index = 1, end_index = length(events), include_time = FALSE,
  time_intv_dist = list("exp", 1))
```

Arguments

`n` An integer. The number of action sequences to be generated.

`events` A character vector specifying the set of `N` possible actions. Default is `letters`.

`rnn_type` the type of recurrent unit to be used for generating sequences. `"lstm"` for the long-short term memory unit. `"gru"` for the gated recurrent unit.

`K` the latent dimension of the recurrent unit.

`weights` a list containing the weights in the embedding layer, the recurrent unit, the fully connected layer. If not (properly) specified, randomly generated weights are used.

`max_len` Maximum length of generated sequences.

`initial_state` a list containing the initial state of the recurrent neural network. If `rnn_type="lstm"`, it contains two 1 by `K` matrices. If `rnn_type="gru"`, it contains one 1 by `K` matrix. If not specified, all the elements are set to zero.

`start_index` Index of the action indicating the start of an item in `events`.

`end_index` Index of the action indicating the end of an item in `events`.

`include_time` logical. Indicate if timestamp sequences should be generated. Default is FALSE.

`time_intv_dist` A list specifying the distribution of the inter-arrival time.

Value

A list containing the following elements

`seqs` an object of class `"proc"` with `time_seqs=NULL`.

`weights` a list containing the weights used for generating sequences.

See Also

Other sequence generators: `seq_gen2`, `seq_gen`

`subtask_analysis` *Subtask Analysis*

Description

`subtask_analysis` performs subtask identification procedure.

Usage

```
subtask_analysis(action_seqs, lambda = 0.3, n_subtask, rnn_dim = 20,
  n_epoch = 20, step_size = 0.001, batch_size = 1,
  optimizer_name = "rmsprop", index_valid = 0.2, verbose = FALSE,
  ...)
```

Arguments

`action_seqs` a list of action sequences

`lambda` a number between 0 and 1

`n_subtask` the desired number of subtasks or a vector of candidate number of subtasks

`rnn_dim` latent dimension of RNN

`n_epoch` the number of training epochs.

`step_size` the learning rate of optimizer.

`batch_size` the batch size used in training.

`optimizer_name` a character string specifying the optimizer to be used for training. Availabel options are `"sgd"`, `"rmsprop"`, `"adadelta"`, and `"adam"`.

`index_valid` proportion of sequences used as the validation set or a vector of indices specifying the validation set.

`verbose` logical. If TRUE, training progress is printed.

`...` additional arguments passed to `kmeans`

Value

an object of class `"subtask"`. It is a list containing

`action_seqs` a list of action sequences

`entropy_seqs` a list of entropy sequences

`seg_seqs` a list of segment boundaries

`subtask_seqs` a list of subtask sequences

`subtasks` a vector of subtasks

`n_subtask` the number of subtasks

`tot.withinss` a vector of total within cluster sum of squares

`relative_cluster_profiles`
a `n_subtask` by `length(actions)` matrix. Each row gives the action frequency profile of each subtask relative to the overall action frequency profile

`loss_history` a `n_epoch` by 2 matrix. The two columns contain the loss at the end of each epoch on the training set and the validation set, respectively.

`rnn_dim` the latent dimension of the recurrent neural network

`model_fit` a vector of class `"raw"`. It is the serialized version of the trained action prediction model.

`actions` a vector of the actions in `action_seqs`.

`max_len` maximum length of the action sequences.

References

Wang, Z., Tang, X., Liu, J., and Ying, Z. (2020) Subtask analysis of process data through a predictive model. https://arxiv.org/abs/2009.00717

See Also

action2entropy, entropy2segment, and segment2subtask for the three steps of subtask analysis.

sub_seqs *Subset response processes*

Description

Subset response processes

Usage

```
sub_seqs(seqs, ids)
```

Arguments

seqs	an object of class "proc"
ids	a vector of indices

Value

an object of class "proc"

Examples

```
data(cc_data)
seqs <- sub_seqs(cc_data$seqs, 1:10)
```

summary.proc *Summary method for class* "proc"

Description

The summary of a "proc" object combines the summary of the action sequences and the summary of the timestamp sequences.

Usage

```
## S3 method for class 'proc'
summary(object, ...)
```

Arguments

object	an object of class "proc".
...	not used.

Value

a list. Its components are the components returned by action_seqs_summary and time_seqs_summary.

See Also

action_seqs_summary and time_seqs_summary

---

time_seqs_summary *Summarize timestamp sequences*

Description

Summarize timestamp sequences

Usage

```
time_seqs_summary(time_seqs)
```

Arguments

time_seqs	a list of timestamp sequences

Value

a list containing the following objects

total_time	total response time of n_seq response processes
mean_react_time	mean reaction time of n_seq response processes

---

tseq2feature_seq2seq *Feature Extraction by time sequence autoencoder*

Description

tseq2feature_seq2seq extract features from timestamps of action sequences by a sequence autoencoder.

Usage

```
tseq2feature_seq2seq(tseqs, K, cumulative = FALSE, log = TRUE,
  rnn_type = "lstm", n_epoch = 50, method = "last",
  step_size = 1e-04, optimizer_name = "rmsprop", samples_train,
  samples_valid, samples_test = NULL, pca = TRUE, verbose = TRUE,
  return_theta = TRUE)
```

Arguments

`tseqs` a list of `n` timestamp sequences. Each element is a numeric sequence in the form of a vector of timestamps associated with actions, with the timestamp of the first event (e.g., "start") of 0.

`K` the number of features to be extracted.

`cumulative` logical. If TRUE, the sequence of cumulative time up to each event is used as input to the neural network. If FALSE, the sequence of inter-arrival time (gap time between an event and the previous event) will be used as input to the neural network. Default is FALSE.

`log` logical. If TRUE, for the timestamp sequences, input of the neural net is the base-10 log of the original sequence of times plus 1 (i.e., log10(t+1)). If FALSE, the original sequence of times is used.

`rnn_type` the type of recurrent unit to be used for modeling response processes. `"lstm"` for the long-short term memory unit. `"gru"` for the gated recurrent unit.

`n_epoch` the number of training epochs for the autoencoder.

`method` the method for computing features from the output of an recurrent neural network in the encoder. Available options are `"last"` and `"avg"`.

`step_size` the learning rate of optimizer.

`optimizer_name` a character string specifying the optimizer to be used for training. Availabel options are `"sgd"`, `"rmsprop"`, `"adadelta"`, and `"adam"`.

`samples_train` vectors of indices specifying the training, validation and test sets for training autoencoder.

`samples_valid` vectors of indices specifying the training, validation and test sets for training autoencoder.

`samples_test` vectors of indices specifying the training, validation and test sets for training autoencoder.

`pca` logical. If TRUE, the principal components of features are returned. Default is TRUE.

`verbose` logical. If TRUE, training progress is printed.

`return_theta` logical. If TRUE, extracted features are returned.

Details

This function trains a sequence-to-sequence autoencoder using keras. The encoder of the autoencoder consists of a recurrent neural network. The decoder consists of another recurrent neural network and a fully connected layer with ReLU activation. The outputs of the encoder are the extracted features.

The output of the encoder is a function of the encoder recurrent neural network. It is the last latent state of the encoder recurrent neural network if `method="last"` and the average of the encoder recurrent neural network latent states if `method="avg"`.

Value

tseq2feature_seq2seq returns a list containing

theta	a matrix containing K features or principal features. Each column is a feature.
train_loss	a vector of length n_epoch recording the trace of training losses.
valid_loss	a vector of length n_epoch recording the trace of validation losses.
test_loss	a vector of length n_epoch recording the trace of test losses. Exists only if samples_test is not NULL.

See Also

chooseK_seq2seq for choosing K through cross-validation.

Other feature extraction methods: aseq2feature_seq2seq, atseq2feature_seq2seq, seq2feature_mds_large, seq2feature_mds, seq2feature_ngram, seq2feature_seq2seq

Examples

```
if (!system("python -c 'import tensorflow as tf'", ignore.stdout = TRUE, ignore.stderr= TRUE)) {
  n <- 50
  data(cc_data)
  samples <- sample(1:length(cc_data$seqs$time_seqs), n)
  tseqs <- cc_data$seqs$time_seqs[samples]
  time_seq2seq_res <- tseq2feature_seq2seq(tseqs, 5, rnn_type="lstm", n_epoch=5,
                                 samples_train=1:40, samples_valid=41:50)
  features <- time_seq2seq_res$theta
  plot(time_seq2seq_res$train_loss, col="blue", type="l",
     ylim = range(c(time_seq2seq_res$train_loss, time_seq2seq_res$valid_loss)))
  lines(time_seq2seq_res$valid_loss, col="red", type = 'l')
}
```

tseq2interval *Transform a timestamp sequence into a inter-arrival time sequence*

Description

Transform a timestamp sequence into a inter-arrival time sequence

Usage

```
tseq2interval(x)
```

Arguments

x	a timestamp sequence

Value

a numeric vector of the same length as x. The first element in the returned vector is 0. The t-th returned element is x[t] - x[t-1].

---

write.seqs *Write process data to csv files*

Description

Write process data to csv files

Usage

```
write.seqs(seqs, file, style, id_var = "ID", action_var = "Event",
  time_var = "Time", step_sep = ",", ...)
```

Arguments

seqs	an object of class "proc" to written in the csv file.
file	the name of the csv file from which the response processes are to be read.
style	the style that the response processes are stored. See 'Details'.
id_var	a string giving the name of the variable storing the process identifier.
action_var	a string giving the name of the variable storing action sequences.
time_var	a string giving the name of the variable storing timestamp sequences.
step_sep	the step separator characters. It is only used if style="single".
...	further arguments to be passed to write.csv

Value

No return value.

Index