Package 'ML2Pvae'

October 12, 2022

Type Package

Title Variational Autoencoder Models for IRT Parameter Estimation

Version 1.0.0.1

Maintainer Geoffrey Converse <converseg@gmail.com>

Description Based on the work of Curi, Converse, Hajewski, and Oliveira (2019) <doi:10.1109/IJCNN.2019.8852333>. This package provides easy-to-use functions which create a variational autoencoder (VAE) to be used for parameter estimation in Item Response Theory (IRT) - namely the Multidimensional Logistic 2-Parameter (ML2P) model. To use a neural network as such, nontrivial modifications to the architecture must be made, such as restricting the nonzero weights in the decoder according to some binary matrix Q. The functions in this package allow for straight-forward construction, training, and evaluation so that minimal knowledge of 'tensorflow' or 'keras' is required.

Note The developer version of 'keras' should be used, rather than the CRAN version. The latter will cause tests to fail on an initial run, but work on subsequent tries. To avoid this, use devtools::install_github(``rstudio/keras''). The user also must have an installation of 'Python 3'.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports keras (>= 2.3.0), reticulate (>= 1.0), tensorflow (>= 2.2.0), tfprobability (>= 0.11.0)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, R.rsp

VignetteBuilder R.rsp

Depends R (>= 3.6)

URL https://converseg.github.io

SystemRequirements TensorFlow (https://www.tensorflow.org), Keras (https://keras.io), TensorFlow Probability (https://www.tensorflow.org/probability)

Config/reticulate list(packages = list(list(package = ``keras'', pip = TRUE), list(package = ``tensorflow'', pip = TRUE), list(package = ``tensorflow-probability'', pip = TRUE)))

NeedsCompilation no

Author Geoffrey Converse [aut, cre, cph],
Suely Oliveira [ctb, ths],
Mariana Curi [ctb]

Repository CRAN

Date/Publication 2022-05-23 08:02:16 UTC

R topics documented:

`.onLoad` *Display a message upon loading package*

Description

Display a message upon loading package

Usage

```
.onLoad(libnam, pkgname)
```

Arguments

`libnam`	the library name
`pkgname`	the package name

`build_hidden_encoder` *Build the encoder for a VAE*

Description

Build the encoder for a VAE

Usage

```
build_hidden_encoder(
  input_size,
  layers,
  activations = rep("sigmoid", length(layers))
)
```

Arguments

`input_size`	an integer representing the number of items
`layers`	a list of integers giving the size of each hidden layer
`activations`	a list of strings, the same length as layers

Value

two tensors: the input layer to the VAE and the last hidden layer of the encoder

---

`build_vae_correlated` *Build a VAE that fits to a normal, full covariance N(m,S) latent distribution*

Description

Build a VAE that fits to a normal, full covariance N(m,S) latent distribution

Usage

```
build_vae_correlated(
  num_items,
  num_skills,
  Q_matrix,
  mean_vector = rep(0, num_skills),
  covariance_matrix = diag(num_skills),
  model_type = 2,
  enc_hid_arch = c(ceiling((num_items + num_skills)/2)),
  hid_enc_activations = rep("sigmoid", length(enc_hid_arch)),
  output_activation = "sigmoid",
  kl_weight = 1,
  learning_rate = 0.001
)
```

Arguments

`num_items`	an integer giving the number of items on the assessment; also the number of nodes in the input/output layers of the VAE
`num_skills`	an integer giving the number of skills being evaluated; also the dimensionality of the distribution learned by the VAE
`Q_matrix`	a binary, `num_skills` by `num_items` matrix relating the assessment items with skills
`mean_vector`	a vector of length `num_skills` specifying the mean of each latent trait; the default of `rep(0, num_skills)` should almost always be used
`covariance_matrix`	a symmetric, positive definite, `num_skills` by `num_skills` matrix giving the covariance of the latent traits
`model_type`	either 1 or 2, specifying a 1 parameter (1PL) or 2 parameter (2PL) model; if 1PL, then all decoder weights are fixed to be equal to one
`enc_hid_arch`	a vector detailing the size of hidden layers in the encoder; the number of hidden layers is determined by the length of this vector
`hid_enc_activations`	a vector specifying the activation function in each hidden layer in the encoder; must be the same length as `enc_hid_arch`
`output_activation`	a string specifying the activation function in the output of the decoder; the ML2P model always used 'sigmoid'
`kl_weight`	an optional weight for the KL divergence term in the loss function
`learning_rate`	an optional parameter for the adam optimizer

Value

returns three keras models: the encoder, decoder, and vae

Examples

```
Q <- matrix(c(1,0,1,1,0,1,1,0), nrow = 2, ncol = 4)
cov <- matrix(c(.7,.3,.3,1), nrow = 2, ncol = 2)
models <- build_vae_correlated(4, 2, Q,
          mean_vector = c(-0.5, 0), covariance_matrix = cov,
          enc_hid_arch = c(6, 3), hid_enc_activation = c('sigmoid', 'relu'),
          output_activation = 'tanh',
          kl_weight = 0.1)
vae <- models[[3]]
```

`build_vae_independent` *Build a VAE that fits to a standard N(0,I) latent distribution with independent latent traits*

Description

Build a VAE that fits to a standard N(0,I) latent distribution with independent latent traits

Usage

```
build_vae_independent(
  num_items,
  num_skills,
  Q_matrix,
  model_type = 2,
  enc_hid_arch = c(ceiling((num_items + num_skills)/2)),
  hid_enc_activations = rep("sigmoid", length(enc_hid_arch)),
  output_activation = "sigmoid",
  kl_weight = 1,
  learning_rate = 0.001
)
```

Arguments

`num_items`	an integer giving the number of items on the assessment; also the number of nodes in the input/output layers of the VAE
`num_skills`	an integer giving the number of skills being evaluated; also the dimensionality of the distribution learned by the VAE
`Q_matrix`	a binary, `num_skills` by `num_items` matrix relating the assessment items with skills
`model_type`	either 1 or 2, specifying a 1 parameter (1PL) or 2 parameter (2PL) model; if 1PL, then all decoder weights are fixed to be equal to one
`enc_hid_arch`	a vector detailing the size of hidden layers in the encoder; the number of hidden layers is determined by the length of this vector

hid_enc_activations
: a vector specifying the activation function in each hidden layer in the encoder; must be the same length as enc_hid_arch

output_activation
: a string specifying the activation function in the output of the decoder; the ML2P model always uses 'sigmoid'

kl_weight
: an optional weight for the KL divergence term in the loss function

learning_rate
: an optional parameter for the adam optimizer

Value

returns three keras models: the encoder, decoder, and vae.

Examples

```
Q <- matrix(c(1,0,1,1,0,1,1,0), nrow = 2, ncol = 4)
models <- build_vae_independent(4, 2, Q,
          enc_hid_arch = c(6, 3), hid_enc_activation = c('sigmoid', 'relu'),
          output_activation = 'tanh', kl_weight = 0.1)
models <- build_vae_independent(4, 2, Q)
vae <- models[[3]]
```

---

correlation_matrix *Simulated latent abilities correlation matrix*

Description

A symmetric positive definite matrix detailing the correlations among three latent traits.

Usage

```
correlation_matrix
```

Format

A data frame with 3 rows and 3 columns

Source

Generated using the python package SciPy

`diff_true` *Simulated difficulty parameters*

Description

Difficulty parameters for an exam with 30 items.

Usage

```
diff_true
```

Format

A data frame with 30 rows and one column. Each entry corresponds to the true value of a particular difficulty parameter.

Source

Each entry is sampled uniformly from `[-3,3]`.

`disc_true` *Simulated discrimination parameters*

Description

Difficulty parameters for an exam of 30 items assessing 3 latent abilities.

Usage

```
disc_true
```

Format

A data frame with 3 rows and 30 columns. Entry `[k,i]` represents the discrimination parameter between item `i` and ability `k`.

Source

Each entry is sampled uniformly from `[0.25,1.75]`. If an entry in `q_matrix.rda` is 0, then so is the corresponding entry in `disc_true.rda`.

get_ability_parameter_estimates

Feed forward response sets through the encoder, which outputs student ability estimates

Description

Feed forward response sets through the encoder, which outputs student ability estimates

Usage

```
get_ability_parameter_estimates(encoder, responses)
```

Arguments

encoder	a trained keras model; should be the encoder returned from either `build_vae_independent()` or `build_vae_correlated`
responses	a `num_students` by `num_items` matrix of binary responses, as used in training

Value

a list where the first entry contains student ability estimates and the second entry holds the variance (or covariance matrix) of those estimates

Examples

```
data <- matrix(c(1,1,0,0,1,0,1,1,0,1,1,0), nrow = 3, ncol = 4)
Q <- matrix(c(1,0,1,1,0,1,1,0), nrow = 2, ncol = 4)
models <- build_vae_independent(4, 2, Q, model_type = 2)
encoder <- models[[1]]
ability_parameter_estimates_variances <- get_ability_parameter_estimates(encoder, data)
student_ability_est <- ability_parameter_estimates_variances[[1]]
```

get_item_parameter_estimates

Get trainable variables from the decoder, which serve as item parameter estimates.

Description

Get trainable variables from the decoder, which serve as item parameter estimates.

Usage

```
get_item_parameter_estimates(decoder, model_type = 2)
```

Arguments

decoder	a trained keras model; can either be the decoder or vae returned from `build_vae_independent()` or `build_vae_correlated`
model_type	either 1 or 2, specifying a 1 parameter (1PL) or 2 parameter (2PL) model; if 1PL, then only the difficulty parameter estimates (output layer bias) will be returned; if 2PL, then the discrimination parameter estimates (output layer weights) will also be returned

Value

a list which contains item parameter estimates; the length of this list is equal to model_type - the first entry in the list holds the difficulty parameter estimates, and the second entry (if 2PL) contains discrimination parameter estimates

Examples

```
Q <- matrix(c(1,0,1,1,0,1,1,0), nrow = 2, ncol = 4)
models <- build_vae_independent(4, 2, Q, model_type = 2)
decoder <- models[[2]]
item_parameter_estimates <- get_item_parameter_estimates(decoder, model_type = 2)
difficulty_est <- item_parameter_estimates[[1]]
discrimination_est <- item_parameter_estimates[[2]]
```

ML2Pvae *ML2Pvae: A package for creating a VAE whose decoder recovers the parameters of the ML2P model. The encoder can be used to predict the latent skills based on assessment scores.*

Description

The ML2Pvae package includes functions which build a VAE with the desired architecture, and fits the latent skills to either a standard normal (independent) distrubution, or a multivariate normal distribution with a full covariance matrix. Based on the work "Interpretable Variational Autoencdoers for Cognitive Models" by Curi, M., Converse, G., Hajewski, J., and Oliveira, S. Found in International Joint Conference on Neural Networks, 2019.

`q_1pl_constraint` *A custom kernel constraint function that forces nonzero weights to be equal to one, so the VAE will estimate the 1-parameter logistic model. Nonzero weights are determined by the Q matrix.*

Description

A custom kernel constraint function that forces nonzero weights to be equal to one, so the VAE will estimate the 1-parameter logistic model. Nonzero weights are determined by the Q matrix.

Usage

```
q_1pl_constraint(Q)
```

Arguments

`Q`	a binary matrix of size `num_skills` by `num_items`

Value

returns a function whose parameters match keras kernel constraint format

`q_constraint` *A custom kernel constraint function that restricts weights between the learned distribution and output. Nonzero weights are determined by the Q matrix.*

Description

A custom kernel constraint function that restricts weights between the learned distribution and output. Nonzero weights are determined by the Q matrix.

Usage

```
q_constraint(Q)
```

Arguments

`Q`	a binary matrix of size `num_skills` by `num_items`

Value

returns a function whose parameters match keras kernel constraint format

q_matrix *Simulated Q-matrix*

Description

The Q-matrix determines the relation between items and abilities.

Usage

```
q_matrix
```

Format

A data frame with 3 rows and 30 columns. If entry `[k,i] = 1`, then item `i` requires skill `k`.

Source

Generated by sampling each entry from `Bernoulli(0.35)`, but ensures each item assess at least one latent ability

responses *Response data*

Description

Simulated response sets for 5000 students on an exam with 30 items.

Usage

```
responses
```

Format

A data frame with 30 columns and 5000 rows. Entry `[j,i]` is 1 if student `j` answers item `i` correctly, and 0 otherwise.

Source

Generated by sampling from the probability of student success on a given item according to the ML2P model. Model parameters can be found in `diff_true.rda`, `disc_true.rda`, and `theta_true.rda`.

`sampling_correlated`	*A reparameterization in order to sample from the learned multivariate normal distribution of the VAE*

Description

A reparameterization in order to sample from the learned multivariate normal distribution of the VAE

Usage

```
sampling_correlated(arg)
```

Arguments

`arg`	a layer of tensors representing the mean and log cholesky transform of the covariance matrix

`sampling_independent`	*A reparameterization in order to sample from the learned standard normal distribution of the VAE*

Description

A reparameterization in order to sample from the learned standard normal distribution of the VAE

Usage

```
sampling_independent(arg)
```

Arguments

`arg`	a layer of tensors representing the mean and variance

theta_true — *Simulated ability parameters*

Description

Three correlated ability parameters for 5000 students.

Usage

```
theta_true
```

Format

A data frame with 5000 rows and 3 columns. Each row represents a particular student's three latent abilities.

Source

Generated by sampling from a 3-dimensional multivariate Gaussian distribution with mean 0 and covariance matrix `correlation_matrix.rda`.

train_model — *Trains a VAE or autoencoder model. This acts as a wrapper for keras::fit().*

Description

Trains a VAE or autoencoder model. This acts as a wrapper for keras::fit().

Usage

```
train_model(
  model,
  train_data,
  num_epochs = 10,
  batch_size = 1,
  validation_split = 0.15,
  shuffle = FALSE,
  verbose = 1
)
```

Arguments

`model` the keras model to be trained; this should be the vae returned from `build_vae_independent()` or `build_vae_correlated`

`train_data` training data; this should be a binary `num_students` by `num_items` matrix of student responses to an assessment

`num_epochs` number of epochs to train for

`batch_size` batch size for mini-batch stochastic gradient descent; default is 1, detailing pure SGD; if a larger batch size is used (e.g. 32), then a larger number of epochs should be set (e.g. 50)

`validation_split` split percentage to use as validation data

`shuffle` whether or not to shuffle data

`verbose` verbosity levels; 0 = silent; 1 = progress bar and epoch message; 2 = epoch message

Value

a list containing training history; this holds the loss from each epoch which can be plotted

Examples

```
data <- matrix(c(1,1,0,0,1,0,1,1,0,1,1,0), nrow = 3, ncol = 4)
Q <- matrix(c(1,0,1,1,0,1,1,0), nrow = 2, ncol = 4)
models <- build_vae_independent(4, 2, Q)
vae <- models[[3]]
history <- train_model(vae, data, num_epochs = 3, validation_split = 0, verbose = 0)
plot(history)
```

`vae_loss_correlated` *A custom loss function for a VAE learning a multivariate normal distribution with a full covariance matrix*

Description

A custom loss function for a VAE learning a multivariate normal distribution with a full covariance matrix

Usage

```
vae_loss_correlated(
  encoder,
  inv_skill_cov,
  det_skill_cov,
  skill_mean,
```

```
  kl_weight,
  rec_dim
)
```

Arguments

encoder	the encoder model of the VAE, used to obtain z_mean and z_log_cholesky from inputs
inv_skill_cov	a constant tensor matrix of the inverse of the covariance matrix being learned
det_skill_cov	a constant tensor scalar representing the determinant of the covariance matrix being learned
skill_mean	a constant tensor vector representing the means of the latent skills being learned
kl_weight	weight for the KL divergence term
rec_dim	the number of nodes in the input/output of the VAE

Value

returns a function whose parameters match keras loss format

---

vae_loss_independent — *A custom loss function for a VAE learning a standard normal distribution*

Description

A custom loss function for a VAE learning a standard normal distribution

Usage

```
vae_loss_independent(encoder, kl_weight, rec_dim)
```

Arguments

encoder	the encoder model of the VAE, used to obtain z_mean and z_log_var from inputs
kl_weight	weight for the KL divergence term
rec_dim	the number of nodes in the input/output of the VAE

Value

returns a function whose parameters match keras loss format

`validate_inputs` *Give error messages for invalid inputs in exported functions.*

Description

Give error messages for invalid inputs in exported functions.

Usage

```
validate_inputs(
  num_items,
  num_skills,
  Q_matrix,
  model_type = 2,
  mean_vector = rep(0, num_skills),
  covariance_matrix = diag(num_skills),
  enc_hid_arch = c(ceiling((num_items + num_skills)/2)),
  hid_enc_activations = rep("sigmoid", length(enc_hid_arch)),
  output_activation = "sigmoid",
  kl_weight = 1,
  learning_rate = 0.001
)
```

Arguments

`num_items`	the number of items on the assessment; also the number of nodes in the input/output layers of the VAE
`num_skills`	the number of skills being evaluated; also the size of the distribution learned by the VAE
`Q_matrix`	a binary, `num_skills` by `num_items` matrix relating the assessment items with skills
`model_type`	either 1 or 2, specifying a 1 parameter (1PL) or 2 parameter (2PL) model
`mean_vector`	a vector of length `num_skills` specifying the mean of each latent trait
`covariance_matrix`	a symmetric, positive definite, `num_skills` by `num_skills`, matrix giving the covariance of the latent traits
`enc_hid_arch`	a vector detailing the number an size of hidden layers in the encoder
`hid_enc_activations`	a vector specifying the activation function in each hidden layer in the encoder; must be the same length as `enc_hid_arch`
`output_activation`	a string specifying the activation function in the output of the decoder; the ML2P model alsways used 'sigmoid'
`kl_weight`	an optional weight for the KL divergence term in the loss function
`learning_rate`	an optional parameter for the adam optimizer

Index