

Package: sigminer.prediction (via r-universe)

July 21, 2024

Title Train and Predict Cancer Subtype with Keras Model based on Mutational Signatures

Version 0.2.0

Description Mutational signatures represent mutational processes occurred in cancer evolution, thus are stable and genetic resources for subtyping. This tool provides functions for training neural network models to predict the subtype a sample belongs to based on 'keras' and 'sigminer' packages.

License Apache License (≥ 2.0)

Depends keras, R (≥ 3.6)

Imports caret, sigminer (≥ 1.0), dplyr

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

Repository <https://shixiangwang.r-universe.dev>

RemoteUrl <https://github.com/ShixiangWang/sigminer.prediction>

RemoteRef HEAD

RemoteSha 0572bfc65fd9eb43bf0bc73b108881953b07787f

Contents

batch_modeling_and_fitting	2
copy_model	3
list_trained_models	3
load_trained_model	4
modeling_and_fitting	4
prepare_data	7
tidy	8

Index	9
--------------	----------

batch_modeling_and_fitting

Construct A Batch of Keras Models

Description

Construct A Batch of Keras Models

Usage

```
batch_modeling_and_fitting(data_list, param_combination, ...)
```

Arguments

`data_list` A list containing predictor and label matrix of training data and test data. Please use [prepare_data](#) to generate this.

`param_combination` A parameter matrix/data.frame with each row representing the parameters for run Keras model once. Column names should indicate parameter names and should be same as in modeling function. `base::expand.grid()` may be very useful to generate it.

... Other arguments passing to [modeling_and_fitting](#).

Value

a tibble.

Examples

```
load(system.file("extdata", "wang2020-input.RData",
  package = "sigminer.prediction", mustWork = TRUE
))
dat_list <- prepare_data(expo_all,
  col_to_vars = c(paste0("Sig", 1:5), paste0("AbsSig", 1:5)),
  col_to_label = "enrich_sig",
  label_names = paste0("Sig", 1:5)
)
pc <- expand.grid(
  c(10, 20, 50, 100),
  c(0, 0.1, 0.2, 0.3, 0.4, 0.5),
  c(10, 20, 50, 100),
  c(0, 0.1, 0.2, 0.3, 0.4, 0.5)
)
colnames(pc) <- c(
  "first_layer_unit", "second_layer_drop_rate",
  "third_layer_unit", "fourth_layer_drop_rate"
)
```

```
# Just use 2 rows for illustration
batch_res <- batch_modeling_and_fitting(dat_list, param_combination = pc %>% head(2))
batch_res

tidy(batch_res)
```

copy_model

Copy Model File

Description

It is usefully when your result model file is stored in temp directory and you want to keep it.

Usage

```
copy_model(model_file, dest)
```

Arguments

model_file A file path to the model file.
dest The destination file path.

Value

Nothing

list_trained_models

List Current Available Trained Keras Models

Description

List Current Available Trained Keras Models

Usage

```
list_trained_models()
```

Value

A tibble containing summary models.

Examples

```
list_trained_models()
```

load_trained_model *Load Trained Models*

Description

Load Trained Models

Usage

```
load_trained_model(x)
```

Arguments

x A subset from [list_trained_models](#).

Value

A (list of) Keras model.

Examples

```
z <- list_trained_models() %>%  
  head(1) %>%  
  load_trained_model()  
z
```

modeling_and_fitting *Create 5-layer Keras Model and Fitting Datasets*

Description

Create 5-layer Keras Model and Fitting Datasets

Usage

```
modeling_and_fitting(  
  data_list,  
  first_layer_unit,  
  second_layer_drop_rate,  
  third_layer_unit,  
  fourth_layer_drop_rate,  
  epochs = 30,  
  batch_size = 16,  
  validation_split = 0.2,  
  validation_data = NULL,  
  test_split = NULL,
```

```

first_layer_activation = "relu",
third_layer_activation = "relu",
fifth_layer_activation = "softmax",
loss = "categorical_crossentropy",
optimizer = optimizer_rmsprop(),
metrics = c("accuracy"),
model_file = tempfile(pattern = "keras_model", tmpdir = file.path(tempdir(),
  "sigminer.pred"), fileext = ".h5"),
test_mode = FALSE
)

```

Arguments

data_list A list containing predictor and label matrix of training data and test data. Please use [prepare_data](#) to generate this.

first_layer_unit Positive integer, dimensionality of the output space for the first layer.

second_layer_drop_rate Float between 0 and 1. Fraction of the input units to drop for the second layer.

third_layer_unit Positive integer, dimensionality of the output space for the third layer.

fourth_layer_drop_rate Float between 0 and 1. Fraction of the input units to drop for the fourth layer.

epochs Number of epochs to train the model, default is 30.

batch_size Integer or NULL. Number of samples per gradient update. If unspecified, `batch_size` will default to 16.

validation_split Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x and y data provided, before shuffling.

validation_data Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This could be a list (x_val, y_val) or a list (x_val, y_val, val_sample_weights). `validation_data` will override `validation_split`.

test_split Float between 0 and 1. Fraction of the all data to be used as test data. If not set, it will be auto-calculated from input data. This value is used for calculating total accuracy.

first_layer_activation activation function for the first layer, default is "relu".

third_layer_activation activation function for the third layer, default is "relu".

fifth_layer_activation activation function for the fifth layer, default is "softmax".

loss	String (name of objective function), objective function or a <code>keras\$losses\$Loss</code> subclass instance. An objective function is any callable with the signature <code>loss = fn(y_true, y_pred)</code> , where <code>y_true</code> = ground truth values with shape = <code>[batch_size, d0, .. dN]</code> , except sparse loss functions such as sparse categorical crossentropy where shape = <code>[batch_size, d0, .. dN-1]</code> . <code>y_pred</code> = predicted values with shape = <code>[batch_size, d0, .. dN]</code> . It returns a weighted loss float tensor. If a custom Loss instance is used and reduction is set to NULL, return value has the shape <code>[batch_size, d0, .. dN-1]</code> i.e. per-sample or per-timestep loss values; otherwise, it is a scalar. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses, unless <code>loss_weights</code> is specified.
optimizer	String (name of optimizer) or optimizer instance. For most models, this defaults to "rmsprop"
metrics	List of metrics to be evaluated by the model during training and testing. Each of this can be a string (name of a built-in function), function or a <code>keras\$metrics\$Metric</code> class instance. See <code>?tf\$keras\$metrics</code> . Typically you will use <code>metrics=list('accuracy')</code> . A function is any callable with the signature <code>result = fn(y_true, y_pred)</code> . To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as <code>metrics=list(output_a = 'accuracy', output_b = c('accuracy', 'mse'))</code> . You can also pass a list to specify a metric or a list of metrics for each output, such as <code>metrics=list(list('accuracy'), list('accuracy', 'mse'))</code> or <code>metrics=list('accuracy', c('accuracy', 'mse'))</code> . When you pass the strings 'accuracy' or 'acc', this is converted to one of <code>tf.keras.metrics.BinaryAccuracy</code> , <code>tf.keras.metrics.CategoricalAccuracy</code> , <code>tf.keras.metrics.SparseCategoricalAccuracy</code> based on the loss function used and the model output shape. A similar conversion is done for the strings 'crossentropy' and 'ce'.
model_file	file path to save the model file in hdf5 format. Default use a temp file path, the path will be stored in returned data. You can load the model with <code>keras::load_model_hdf5()</code> .
test_mode	Default is FALSE, if TRUE, print the input parameters from the user and exit.

Value

a tibble.

Examples

```
load(system.file("extdata", "wang2020-input.RData",
  package = "sigminer.prediction", mustWork = TRUE
))
dat_list <- prepare_data(expo_all,
  col_to_vars = c(paste0("Sig", 1:5), paste0("AbsSig", 1:5)),
  col_to_label = "enrich_sig",
  label_names = paste0("Sig", 1:5)
)
res <- modeling_and_fitting(dat_list, 20, 0, 20, 0.1)
res$history[[1]] %>% plot()
```

```
## Load model and predict
model <- load_model_hdf5(res$model_file)

model %>% predict_classes(dat_list$x_train[1, , drop = FALSE])
model %>% predict_proba(dat_list$x_train[1, , drop = FALSE])
```

```
prepare_data
```

Prepare Training and Test Dataset

Description

Prepare Training and Test Dataset

Usage

```
prepare_data(
  data,
  col_to_vars,
  col_to_label,
  label_names,
  seed = 1234,
  test_split = 0.2
)
```

Arguments

<code>data</code>	A data.frame.
<code>col_to_vars</code>	A character vector specifying the predictive columns.
<code>col_to_label</code>	A column indicating the labels/classes.
<code>label_names</code>	Label/class names. The order is important. For example, "a", "b", "c" will be transformed to 0, 1, 2.
<code>seed</code>	Random seed, default is 1234.
<code>test_split</code>	A fraction of samples to treated as test dataset, default is 0.2.

Value

a list containing `x_train`, `y_train`, `x_test`, `y_test` datasets.

Examples

```
load(system.file("extdata", "wang2020-input.RData",
  package = "sigminer.prediction", mustWork = TRUE
))
dat_list <- prepare_data(expo_all,
  col_to_vars = c(paste0("Sig", 1:5), paste0("AbsSig", 1:5)),
  col_to_label = "enrich_sig",
  label_names = paste0("Sig", 1:5)
)
str(dat_list)
```

tidy	<i>Tidy Modeling Result</i>
------	-----------------------------

Description

Tidy Modeling Result

Usage

```
tidy(x)
```

Arguments

x A result tibble from either [modeling_and_fitting](#) or [batch_modeling_and_fitting](#).

Value

a tibble

Index

`base::expand.grid()`, 2
`batch_modeling_and_fitting`, 2, 8
`copy_model`, 3
`keras::load_model_hdf5()`, 6
`list_trained_models`, 3, 4
`load_trained_model`, 4
`modeling_and_fitting`, 2, 4, 8
`prepare_data`, 2, 5, 7
`tidy`, 8