Package ‘ttgsea’

April 2, 2024

Type Package

Title Tokenizing Text of Gene Set Enrichment Analysis

Description Functional enrichment analysis methods such as gene set enrichment analysis (GSEA) have been widely used for analyzing gene expression data. GSEA is a powerful method to infer results of gene expression data at a level of gene sets by calculating enrichment scores for predefined sets of genes. GSEA depends on the availability and accuracy of gene sets. There are overlaps between terms of gene sets or categories because multiple terms may exist for a single biological process, and it can thus lead to redundancy within enriched terms. In other words, the sets of related terms are overlapping. Using deep learning, this package is aimed to predict enrichment scores for unique tokens or words from text in names of gene sets to resolve this overlapping set issue. Furthermore, we can coin a new term by combining tokens and find its enrichment score by predicting such a combined token.

Version 1.10.0

Date 2021-11-12

LazyData TRUE

Depends keras

Imports tm, text2vec, tokenizers, textstem, stopwords, data.table, purrr, DiagrammeR, stats

Suggests fgsea, knitr, testthat, reticulate, rmarkdown

SystemRequirement tensorflow

License Artistic-2.0

biocViews Software, GeneExpression, GeneSetEnrichment

NeedsCompilation no

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/ttgsea
git_branch RELEASE_3_18
git_last_commit d21c96b
git_last_commit_date 2023-10-24

Repository Bioconductor 3.18
bi_gru

Bidirectional GRU with embedding layer

Description
A predefined function that is used as a model in "ttgsea". This is a simple model, but you can define your own model. The loss function is "mean_squared_error" and the optimizer is "adam". Pearson correlation is used as a metric.

Usage
bi_gru(num_tokens, embedding_dim, length_seq, num_units)

Arguments
- num_tokens: maximum number of tokens
- embedding_dim: a non-negative integer for dimension of the dense embedding
- length_seq: length of input sequences, input length of "layer_embedding"
- num_units: dimensionality of the output space in the GRU layer

Value
model

Author(s)
Dongmin Jung
See Also
keras::keras_model, keras::layer_input, keras::layer_embedding, keras::layer_gru, keras::bidirectional,
keras::layer_dense, keras::compile

Examples
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  num_tokens <- 1000
  length_seq <- 30
  embedding_dim <- 50
  num_units <- 32
  model <- bi_gru(num_tokens, embedding_dim, length_seq, num_units)

  # stacked gru
  num_units_1 <- 32
  num_units_2 <- 16
  stacked_gru <- function(num_tokens, embedding_dim, length_seq,
                           num_units_1, num_units_2)
  {
    model <- keras::keras_model_sequential() %>%
    keras::layer_embedding(input_dim = num_tokens,
      output_dim = embedding_dim,
      input_length = length_seq,
      mask_zero = TRUE) %>%
    keras::layer_gru(units = num_units_1,
      activation = "relu",
      return_sequences = TRUE) %>%
    keras::layer_gru(units = num_units_2,
      activation = "relu") %>%
    keras::layer_dense(1)

    model %>%
    keras::compile(loss = "mean_squared_error",
      optimizer = "adam",
      metrics = custom_metric("pearson_correlation",
                             metric_pearson_correlation))
  }
}

---

**bi_lstm**

Bidirectional LSTM with embedding layer

Description
A predefined function that is used as a model in "ttgsea". This is a simple model, but you can define your own model. The loss function is "mean_squared_error" and the optimizer is "adam". Pearson correlation is used as a metric.
Usage

\texttt{bi_lstm(num\_tokens, embedding\_dim, length\_seq, num\_units)}

Arguments

\begin{itemize}
  \item \texttt{num\_tokens} \hspace{1cm} maximum number of tokens
  \item \texttt{embedding\_dim} \hspace{1cm} a non-negative integer for dimension of the dense embedding
  \item \texttt{length\_seq} \hspace{1cm} length of input sequences, input length of "layer\_embedding"
  \item \texttt{num\_units} \hspace{1cm} dimensionality of the output space in the LSTM layer
\end{itemize}

Value

model

Author(s)

Dongmin Jung

See Also

keras::keras\_model, keras::layer\_input, keras::layer\_embedding, keras::layer\_lstm, keras::bidirectional, keras::layer\_dense, keras::compile

Examples

\begin{verbatim}
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  num\_tokens <- 1000
  length\_seq <- 30
  embedding\_dim <- 50
  num\_units <- 32
  model <- bi\_lstm(num\_tokens, embedding\_dim, length\_seq, num\_units)

  # stacked lstm
  num\_units\_1 <- 32
  num\_units\_2 <- 16
  stacked\_lstm <- function(num\_tokens, embedding\_dim, length\_seq, 
  num\_units\_1, num\_units\_2)
  {
    model <- keras::keras\_model\_sequential() %>%
      keras::layer\_embedding(input\_dim = num\_tokens, 
      output\_dim = embedding\_dim, 
      input\_length = length\_seq, 
      mask\_zero = TRUE) %>%
      keras::layer\_lstm(units = num\_units\_1, 
      activation = "relu", 
      return\_sequences = TRUE) %>%
      keras::layer\_lstm(units = num\_units\_2, 
      activation = "relu")
  }
\end{verbatim}
fit_model

Deep learning model fitting

Description

From the result of GSEA, we can predict enrichment scores for unique tokens or words from text in names of gene sets by using deep learning. The function "text_token" is used for tokenizing text and the function "token_vector" is used for encoding. Then the encoded sequence is fed to the embedding layer of the model.

Usage

fit_model(gseaRes, text, score, model, ngram_min = 1, ngram_max = 2, num_tokens, length_seq, epochs, batch_size, use_generator = TRUE, ...)

Arguments

gseaRes a table with GSEA result having rows for gene sets and columns for text and scores
text column name for text data
score column name for enrichment score
model deep learning model, input dimension and length for the embedding layer must be same to the "num_token" and "length_seq", respectively
ngram_min minimum size of an n-gram (default: 1)
ngram_max maximum size of an n-gram (default: 2)
num_tokens maximum number of tokens, it must be equal to the input dimension of "layer_embedding" in the "model"
length_seq length of input sequences, it must be equal to the input length of "layer_embedding" in the "model"
ePOCHS number of epochs
batch_size batch size
use_generator if "use_generator" is TRUE, the function "sampling_generator" is used for "fit_generator". Otherwise, the "fit" is used without a generator.
... additional parameters for the "fit" or "fit_generator"
**Value**

- model: trained model
- tokens: information for tokens
- token_pred: prediction for every token, each row has a token and its predicted score
- token_gsea: list of the GSEA result only for the corresponding token
- num_tokens: maximum number of tokens
- length_seq: length of input sequences

**Author(s)**

Dongmin Jung

**See Also**

- keras::fit_generator, keras::layer_embedding, keras::pad_sequences, textstem::lemmatize_strings, text2vec::create_vocabulary, text2vec::prune_vocabulary

**Examples**

```r
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  library(fgsea)
  data(examplePathways)
  data(exampleRanks)
  names(examplePathways) <- gsub("_", " ",
                                  substr(names(examplePathways), 9, 1000))
  set.seed(1)
  fgseaRes <- fgsea(examplePathways, exampleRanks)
  num_tokens <- 1000
  length_seq <- 30
  batch_size <- 32
  embedding_dims <- 50
  num_units <- 32
  epochs <- 1

  ttgseaRes <- fit_model(fgseaRes, "pathway", "NES",
                          model = bi_gru(num_tokens,
                                          embeddingDims, length_seq, num_units),
                          num_tokens = num_tokens,
                          length_seq = length_seq,
                          epochs = epochs,
                          batch_size = batch_size,
                          use_generator = FALSE)
}
```
**metric_pearson_correlation**

*Pearson correlation coefficient*

**Description**

Pearson correlation coefficient can be seen as one of the model performance metrics. This is a measure of how close the predicted value is to the true value. If it is close to 1, the model is considered a good fit. If it is close to 0, the model is not good. A value of 0 corresponds to a random prediction.

**Author(s)**

Dongmin Jung

**See Also**

keras::k_mean, keras::sum, keras::k_square, keras::k_sqrt

**Examples**

```r
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  num_tokens <- 1000
  length_seq <- 30
  embedding_dims <- 50
  num_units_1 <- 32
  num_units_2 <- 16

  stacked_gru <- function(num_tokens, embedding_dims, length_seq,
                          num_units_1, num_units_2)
  {
    model <- keras::keras_model_sequential()
    keras::layer_embedding(input_dim = num_tokens,
                           output_dim = embedding_dims,
                           input_length = length_seq)
    keras::layer_gru(units = num_units_1,
                     activation = "relu",
                     return_sequences = TRUE)
    keras::layer_gru(units = num_units_2,
                     activation = "relu")
    keras::layer_dense(1)

    model
  }

  stacked_gru(num_tokens, embedding_dims, length_seq, num_units_1, num_units_2)
}
```
plot_model  

**Description**

You are allowed to create a visualization of your model architecture. This architecture displays the information about the name, input shape, and output shape of layers in a flowchart.

**Usage**

```r
plot_model(x)
```

**Arguments**

- `x`  
  deep learning model

**Value**

plot for the model architecture

**Author(s)**

Dongmin Jung

**See Also**

purrr::map, purrr::map_chr, purrr::pluck, purrr::imap_dfr, DiagrammeR::grViz

**Examples**

```r
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  inputs1 <- layer_input(shape = c(1000))
  inputs2 <- layer_input(shape = c(1000))

  predictions1 <- inputs1 %>%
    layer_dense(units = 128, activation = 'relu') %>%
    layer_dense(units = 64, activation = 'relu') %>%
    layer_dense(units = 32, activation = 'softmax')

  predictions2 <- inputs2 %>%
    layer_dense(units = 128, activation = 'relu') %>%
    layer_dense(units = 64, activation = 'relu') %>%
    layer_dense(units = 32, activation = 'softmax')

  combined <- layer_concatenate(c(predictions1, predictions2)) %>%
    layer_dense(units = 16, activation = 'softmax')
}
```
model <- keras_model(inputs = c(inputs1, inputs2),
                    outputs = combined)
plot_model(model)
```

**Description**

From the result of the function "ttgsea", we can predict enrichment scores. For each new term, lemmatized text, predicted enrichment score, Monte Carlo p-value and adjusted p-value are provided. The function "token_vector" is used for encoding as we did for training. Of course, mapping from tokens to integers should be the same.

**Usage**

```
predict_model(object, new_text, num_simulations = 1000,
               adj_p_method = "fdr")
```

**Arguments**

- `object`: result of "ttgsea"
- `new_text`: new text data
- `num_simulations`: number of simulations for Monte Carlo p-value (default: 1000)
- `adj_p_method`: correction method (default: "fdr")

**Value**

table for lemmatized text, predicted enrichment score, MC p-value and adjusted p-value

**Author(s)**

Dongmin Jung

**See Also**

stats::p.adjust
Examples

```r
library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  library(fgsea)
  data(examplePathways)
  data(exampleRanks)
  names(examplePathways) <- gsub("_", " ",
                                   substr(names(examplePathways), 9, 1000))
  set.seed(1)
  fgseaRes <- fgsea(examplePathways, exampleRanks)

  num_tokens <- 1000
  length_seq <- 30
  batch_size <- 32
  embedding_dims <- 50
  num_units <- 32
  epochs <- 1

  ttgseaRes <- fit_model(fgseaRes, "pathway", "NES",
                          model = bi_gru(num_tokens,
                                         embedding_dims,
                                         length_seq,
                                         num_units),
                          num_tokens = num_tokens,
                          length_seq = length_seq,
                          epochs = epochs,
                          batch_size = batch_size,
                          use_generator = FALSE)

  set.seed(1)
  predict_model(ttgseaRes, "Cell Cycle")
}
```

---

**sampling_generator**  
*Generator function*

**Description**

This is a generator function that yields batches of training data then pass the function to the "fit_generator" function.

**Usage**

`sampling_generator(X_data, Y_data, batch_size)`

**Arguments**

- `X_data` inputs
- `Y_data` targets
- `batch_size` batch size
Value

generator for "fit_generator"

Author(s)

Dongmin Jung

Examples

X_data <- matrix(rnorm(200), ncol = 2)
Y_data <- matrix(rnorm(100), ncol = 1)
sampling_generator(X_data, Y_data, 32)

text_token

Description

An n-gram is used for tokenization. This function can also be used to limit the total number of tokens.

Usage

text_token(text, ngram_min = 1, ngram_max = 1, num_tokens)

Arguments

text text data
ngram_min minimum size of an n-gram (default: 1)
ngram_max maximum size of an n-gram (default: 1)
um_tokens maximum number of tokens

Value

token result of tokenizing text
ngram_min minimum size of an n-gram
ngram_max maximum size of an n-gram

Author(s)

Dongmin Jung

See Also

tm::removeWords, stopwords::stopwords, textstem::lemmatize_strings, text2vec::create_vocabulary, text2vec::prune_vocabulary
Example

```r
library(fgsea)
data(examplePathways)
data(exampleRanks)
names(examplePathways) <- gsub("_", " ",
  substr(names(examplePathways), 9, 1000))
set.seed(1)
fgseaRes <- fgsea(examplePathways, exampleRanks)
tokens <- text_token(data.frame(fgseaRes)[,"pathway",
  num_tokens = 1000])
```

---

**token_vector**

*Vectorization of tokens*

**Description**

A vectorization of words or tokens of text is necessary for machine learning. Vectorized sequences are padded or truncated.

**Usage**

```r
token_vector(text, token, length_seq)
```

**Arguments**

- `text` text data
- `token` result of tokenization (output of "text_token")
- `length_seq` length of input sequences

**Value**

sequences of integers

**Author(s)**

Dongmin Jung

**See Also**

tm::removeWords, stopwords::stopwords, textstem::lemmatize_strings, tokenizers::tokenize_ngrams, keras::pad_sequences
Examples

library(reticulate)
if (keras::is_keras_available() & reticulate::py_available()) {
  library(fgsea)
  data(examplePathways)
  data(exampleRanks)
  names(examplePathways) <- gsub("_", " ",
                                 substr(names(examplePathways), 9, 1000))
  set.seed(1)
  fgseaRes <- fgsea(examplePathways, exampleRanks)
  tokens <- text_token(data.frame(fgseaRes)[,"pathway"],
                        num_tokens = 1000)
  sequences <- token_vector("Cell Cycle", tokens, 10)
}
Index

bi_gru, 2
bi_lstm, 3

fit_model, 5

metric_pearson_correlation, 7

plot_model, 8
predict_model, 9

sampling_generator, 10

text_token, 11
token_vector, 12