# Package: DeepPINCS (via r-universe)

October 26, 2026

**Type** Package

**Title** Protein Interactions and Networks with Compounds based on Sequences using Deep Learning

**Description** The identification of novel compound-protein interaction (CPI) is important in drug discovery. Revealing unknown compound-protein interactions is useful to design a new drug for a target protein by screening candidate compounds. The accurate CPI prediction assists in effective drug discovery process. To identify potential CPI effectively, prediction methods based on machine learning and deep learning have been developed. Data for sequences are provided as discrete symbolic data. In the data, compounds are represented as SMILES (simplified molecular-input line-entry system) strings and proteins are sequences in which the characters are amino acids. The outcome is defined as a variable that indicates how strong two molecules interact with each other or whether there is an interaction between them. In this package, a deep-learning based model that takes only sequence information of both compounds and proteins as input and the outcome as output is used to predict CPI. The model is implemented by using compound and protein encoders with useful features. The CPI model also supports other modeling tasks, including protein-protein interaction (PPI), chemical-chemical interaction (CCI), or single compounds and proteins. Although the model is designed for proteins, DNA and RNA can be used if they are represented as sequences.

**Version** 1.20.0

**Date** 2023-07-06

**LazyData** TRUE

**LazyDataCompression** xz

**Depends** keras, R (>= 4.1)

**Imports** tensorflow, CatEncoders, matlab, rcdk, stringdist, tokenizers, webchem, purrr, ttgsea, PRROC, reticulate, stats

**Suggests** knitr, testthat, rmarkdown

**License** Artistic-2.0

**biocViews** Software, Network, GraphAndNetwork, NeuralNetwork

**NeedsCompilation** no

**VignetteBuilder** knitr

**Config/pak/sysreqs** cmake libglpk-dev make default-jdk libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 libx11-dev

**Repository** Bioconductor 3.23

**Date/Publication** 2026-04-28 12:55:56 UTC

**RemoteUrl** https://github.com/bioc/DeepPINCS

**RemoteRef** RELEASE_3_23

**RemoteSha** d3ecf5cc2e2ae6a338d4fb8902382045c2fad7c4

# Contents

antiviral_drug . . . 3
cpi_model . . . 3
encoder_in_out . . . 9
example_bioassay . . . 11
example_cci . . . 11
example_chem . . . 12
example_cpi . . . 12
example_pd . . . 13
example_ppi . . . 14
example_prot . . . 14
get_canonical_smiles . . . 15
get_fingerprint . . . 16
get_graph_structure_node_feature . . . 17
get_seq_encode_pad . . . 18
metric_concordance_index . . . 19
metric_f1_score . . . 20
multiple_sampling_generator . . . 21
SARS_CoV2_3CL_Protease . . . 22
seq_check . . . 23
seq_preprocessing . . . 24

**Index** **26**

## antiviral_drug *List of antiviral drugs with SMILES strings*

### Description

81 antiviral drugs with SMILES strings

### Usage

```
antiviral_drug
```

### Value

SMILES string

### Author(s)

Dongmin Jung

### Source

Huang, K., Fu, T., Glass, L. M., Zitnik, M., Xiao, C., & Sun, J. (2020). DeepPurpose: A Deep Learning Library for Drug-Target Interaction Prediction. Bioinformatics.

## cpi_model *Deep learning model fitting and prediction for compound-protein interactions*

### Description

The model for compound-protein interactions (CPI) takes the pair of SMILES strings of compounds and amino acid sequences (one letter amino acid code) of proteins as input. They are fed into the compound and protein encoders, respectively, and then these encoders are concatenated. Due to the combination of compound and protein encoders, there are many kinds of CPI models. However, the graph neural network such as the graph concolutional network (GCN) is only available for compounds. We need to select one of types of compounds. For graph and fingerprint, the SMILES sequences are not used for encoders, because the information of graph or fingerprint is extracted from the SMILES sequenes and then it is fed into encoders. For sequence, the unigram is used as default, but the n-gram is available only for proteins. Since the CPI model needs some arguments of encoders, we may have to match the names of such arguments.

## Usage

```
fit_cpi(smiles = NULL, AAseq = NULL, outcome,
        convert_canonical_smiles = TRUE,
        compound_type = NULL, compound_max_atoms,
        compound_length_seq, protein_length_seq,
        compound_embedding_dim, protein_embedding_dim,
        protein_ngram_max = 1, protein_ngram_min = 1,
        smiles_val = NULL, AAseq_val = NULL, outcome_val = NULL,
        net_args = list(
            compound,
            compound_args,
            protein,
            protein_args,
            fc_units = c(1),
            fc_activation = c("linear"), ...),
        net_names = list(
            name_compound_max_atoms = NULL,
            name_compound_feature_dim = NULL,
            name_compound_fingerprint_size = NULL,
            name_compound_embedding_layer = NULL,
            name_compound_length_seq = NULL,
            name_compound_num_tokens = NULL,
            name_compound_embedding_dim = NULL,
            name_protein_length_seq = NULL,
            name_protein_num_tokens = NULL,
            name_protein_embedding_dim = NULL),
        preprocessor_only = FALSE,
        preprocessing = list(
            outcome = NULL,
            outcome_val = NULL,
            convert_canonical_smiles = NULL,
            canonical_smiles = NULL,
            compound_type = NULL,
            compound_max_atoms = NULL,
            compound_A_pad = NULL,
            compound_X_pad = NULL,
            compound_A_pad_val = NULL,
            compound_X_pad_val = NULL,
            compound_fingerprint = NULL,
            compound_fingerprint_val = NULL,
            smiles_encode_pad = NULL,
            smiles_val_encode_pad = NULL,
            compound_lenc = NULL,
            compound_length_seq = NULL,
            compound_num_tokens = NULL,
            compound_embedding_dim = NULL,
            AAseq_encode_pad = NULL,
            AAseq_val_encode_pad = NULL,
```

```
            protein_lenc = NULL,
            protein_length_seq = NULL,
            protein_num_tokens = NULL,
            protein_embedding_dim = NULL,
            protein_ngram_max = NULL,
            protein_ngram_min = NULL),
        batch_size, use_generator = FALSE,
        validation_split = 0, ...)

predict_cpi(modelRes, smiles = NULL, AAseq = NULL,
            preprocessing = list(
                canonical_smiles = NULL,
                compound_A_pad = NULL,
                compound_X_pad = NULL,
                compound_fingerprint = NULL,
                smiles_encode_pad = NULL,
                AAseq_encode_pad = NULL),
            use_generator = FALSE,
            batch_size = NULL)
```

## Arguments

| | |
|---|---|
| smiles | SMILES strings, each column for the element of a pair (default: NULL) |
| AAseq | amino acid sequences, each column for the element of a pair (default: NULL) |
| outcome | a variable that indicates how strong two molecules interact with each other or whether there is an interaction between them |
| convert_canonical_smiles | SMILES strings are converted to canonical SMILES strings if TRUE (default: TRUE) |
| compound_type | "graph", "fingerprint" or "sequence" |
| compound_max_atoms | maximum number of atoms for compounds |
| compound_length_seq | length of compound sequence |
| protein_length_seq | length of protein sequence |
| compound_embedding_dim | dimension of the dense embedding for compounds |
| protein_embedding_dim | dimension of the dense embedding for proteins |
| protein_ngram_max | maximum size of an n-gram for protein sequences (default: 1) |
| protein_ngram_min | minimum size of an n-gram for protein sequences (default: 1) |
| smiles_val | SMILES strings for validation (default: NULL) |
| AAseq_val | amino acid sequences for validation (default: NULL) |

`outcome_val` outcome for validation (default: NULL)

`net_args` list of arguments for compound and protein encoder networks and for fully connected layer

- compound : encoder network for compounds
- compound_args : arguments of compound encoder
- protein : encoder network for proteins
- protein_args : arguments of protein encoder
- fc_units : dimensionality of the output space in the fully connected layer (default: 1)
- fc_activation : activation of the fully connected layer (default: "linear")
- ... : arguments of "keras::compile" but for object

`net_names` list of names of arguments used in both the CPI model and encoder networks, names are set to NULL as default

- name_compound_max_atoms : corresponding name for the maximum number of atoms in the compound encoder, "max_atoms" if NULL
- name_compound_feature_dim : corresponding name for the dimension of node features in the compound encoder, "feature_dim" if NULL
- name_compound_fingerprint_size : corresponding name for the length of a fingerprint in the compound encoder, "fingerprint_size" if NULL
- name_compound_embedding_layer : corresponding name for the use of the embedding layer in the compound encoder, "embedding_layer" if NULL
- name_compound_length_seq : corresponding name for the length of sequences in the compound encoder, "length_seq" if NULL
- name_compound_num_tokens : corresponding name for the total number of distinct strings in the compound encoder, "num_tokens" if NULL
- name_compound_embedding_dim : corresponding name for dimension of the dense embedding in the compound encoder, "embedding_dim" if NULL
- name_protein_length_seq : corresponding name for the length of sequences in the protein encoder, "length_seq" if NULL
- name_protein_num_tokens : corresponding name for the total number of distinct strings in the protein encoder, "num_tokens" if NULL
- name_protein_embedding_dim : corresponding name for dimension of the dense embedding in the protein encoder, "embedding_dim" if NULL

`preprocessor_only`
model is not fitted after preprocessing if TRUE (default: FALSE)

`preprocessing` list of preprocessed results for "fit_cpi" or "predict_cpi", they are set to NULL as default

- outcome : outcome variable
- outcome_val : outcome variable for validation
- convert_canonical_smiles : canonical representation used for preprocessing if TRUE
- canonical_smiles : canonical representation of SMILES
- compound_type : "graph", "fingerprint" or "sequence"

- compound_max_atoms : maximum number of atoms for compounds
- compound_A_pad : padded or turncated adjacency matrix of compounds
- compound_X_pad : padded or turncated node features of compounds
- compound_A_pad_val : padded or turncated adjacency matrix for validation
- compound_X_pad_val : padded or turncated node features for validation
- compound_fingerprint : fingerprint of compounds
- compound_fingerprint_val : fingerprint for validation
- smiles_encode_pad : encoded SMILES sequence which is padded or truncated
- smiles_val_encode_pad : encoded SMILES sequence for validation
- compound_lenc : encoded labels for characters of SMILES strings
- compound_length_seq : length of compound sequence
- compound_num_tokens : total number of characters of compounds
- compound_embedding_dim : dimension of the dense embedding for compounds
- AAseq_encode_pad : encoded amino acid sequence which is padded or truncated
- AAseq_val_encode_pad : encoded amino acid sequence for validation
- protein_lenc : encoded labels for characters of amino acid sequenes
- protein_length_seq : length of protein sequence
- protein_num_tokens : total number of characters of proteins
- protein_embedding_dim : dimension of the dense embedding for proteins
- protein_ngram_max : maximum size of an n-gram for protein sequences
- protein_ngram_min : minimum size of an n-gram for protein sequences
- removed_smiles : index for removed smiles while checking
- removed_AAseq : index for removed AAseq while checking
- removed_smiles_val : index for removed smiles of validation
- removed_AAseq_val : index for removed AAseq of validation

`batch_size` batch size

`use_generator` use data generator if TRUE (default: FALSE)

`validation_split` proportion of validation data, it is ignored when there is a validation set (default: 0)

`modelRes` result of the "fit_cpi"

`...` additional parameters for the "keras::fit" or "keras::fit_generator"

## Value

model

## Author(s)

Dongmin Jung

## See Also

keras::compile, keras::fit, keras::fit_generator, keras::layer_dense, keras::keras_model, purrr::pluck, webchem::is.smiles

## Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    compound_max_atoms <- 50
    protein_embedding_dim <- 16
    protein_length_seq <- 100
    gcn_cnn_cpi <- fit_cpi(
        smiles = example_cpi[1:100, 1],
        AAseq = example_cpi[1:100, 2],
        outcome = example_cpi[1:100, 3],
        compound_type = "graph",
        compound_max_atoms = compound_max_atoms,
        protein_length_seq = protein_length_seq,
        protein_embedding_dim = protein_embedding_dim,
        net_args = list(
            compound = "gcn_in_out",
            compound_args = list(
                gcn_units = c(128, 64),
                gcn_activation = c("relu", "relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
            protein = "cnn_in_out",
            protein_args = list(
                cnn_filters = c(32),
                cnn_kernel_size = c(3),
                cnn_activation = c("relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
            fc_units = c(1),
            fc_activation = c("sigmoid"),
            loss = "binary_crossentropy",
            optimizer = keras::optimizer_adam(),
            metrics = "accuracy"),
        epochs = 2, batch_size = 16)
    pred <- predict_cpi(gcn_cnn_cpi, example_cpi[101:110, 1], example_cpi[101:110, 2])

    gcn_cnn_cpi2 <- fit_cpi(
        preprocessing = gcn_cnn_cpi$preprocessing,
        net_args = list(
            compound = "gcn_in_out",
            compound_args = list(
                gcn_units = c(128, 64),
                gcn_activation = c("relu", "relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
            protein = "cnn_in_out",
            protein_args = list(
                cnn_filters = c(32),
```

```
                cnn_kernel_size = c(3),
                cnn_activation = c("relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
            fc_units = c(1),
            fc_activation = c("sigmoid"),
            loss = "binary_crossentropy",
            optimizer = keras::optimizer_adam(),
            metrics = "accuracy"),
        epochs = 2, batch_size = 16)
    pred <- predict_cpi(gcn_cnn_cpi2, preprocessing = pred$preprocessing)
}
```

---

encoder_in_out *Input and output tensors of encoders*

---

## Description

The graph convolutional network (GCN), recurrent neural network (RNN), convolutional neural network (CNN), and multilayer perceptron (MLP) are used as encoders. The last layer of the encoders is the fully connected layer. The units and activation can be vectors and the length of the vectors represents the number of layers.

## Usage

```
gcn_in_out(max_atoms, feature_dim, gcn_units, gcn_activation,
    fc_units, fc_activation)

rnn_in_out(length_seq, fingerprint_size, embedding_layer = TRUE,
    num_tokens, embedding_dim, rnn_type, rnn_bidirectional,
    rnn_units, rnn_activation, fc_units, fc_activation)

cnn_in_out(length_seq, fingerprint_size, embedding_layer = TRUE,
    num_tokens, embedding_dim, cnn_filters, cnn_kernel_size, cnn_activation,
    fc_units, fc_activation)

mlp_in_out(length_seq, fingerprint_size, embedding_layer = TRUE,
    num_tokens, embedding_dim, fc_units, fc_activation)
```

## Arguments

| | |
|---|---|
| max_atoms | maximum number of atoms for gcn |
| feature_dim | dimension of atom features for gcn |
| gcn_units | dimensionality of the output space in the gcn layer |
| gcn_activation | activation of the gcn layer |
| fingerprint_size | the length of a fingerprint |

| | |
|---|---|
| embedding_layer | use the embedding layer if TRUE (default: TRUE) |
| embedding_dim | a non-negative integer for dimension of the dense embedding |
| length_seq | length of input sequences |
| num_tokens | total number of distinct strings |
| cnn_filters | dimensionality of the output space in the cnn layer |
| cnn_kernel_size | length of the 1D convolution window in the cnn layer |
| cnn_activation | activation of the cnn layer |
| rnn_type | "lstm" or "gru" |
| rnn_bidirectional | use the bidirectional wrapper for rnn if TRUE |
| rnn_units | dimensionality of the output space in the rnn layer |
| rnn_activation | activation of the rnn layer |
| fc_units | dimensionality of the output space in the fully connected layer |
| fc_activation | activation of the fully connected layer |

## Value

input and output tensors of encoders

## Author(s)

Dongmin Jung

## See Also

keras::layer_activation, keras::bidirectional, keras::layer_conv_1d, keras::layer_dense, keras::layer_dot, keras::layer_embedding, keras::layer_global_average_pooling_1d, keras::layer_input, keras::layer_lstm, keras::layer_gru, keras::layer_flatten

## Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    gcn_in_out(max_atoms = 50,
        feature_dim = 50,
        gcn_units = c(128, 64),
        gcn_activation = c("relu", "relu"),
        fc_units = c(10),
        fc_activation = c("relu"))
}
```

## example_bioassay *Example Data for PubChem AID1706 bioassay*

### Description

This is a compound-protein interaction data set retrieved from PubChem AID1706 bioassay. The data is balanced and a randomly selected subset of a dataset of size 5000. The label is 1 if the score is greater than or equal to 15, otherwise it is 0.

### Usage

```
example_bioassay
```

### Value

compound-protein interaction data

### Author(s)

Dongmin Jung

### Source

Huang, K., Fu, T., Glass, L. M., Zitnik, M., Xiao, C., & Sun, J. (2020). DeepPurpose: A Deep Learning Library for Drug-Target Interaction Prediction. Bioinformatics.

## example_cci *Example Data for Chemical-Chemical Interactions*

### Description

The data is a randomly selected subset with size 1000 for chemical-chemical interactions. The two SMILES strings are for compound pairs and the label is for their interactions.

### Usage

```
example_cci
```

### Value

chemical-chemical interaction data

### Author(s)

Dongmin Jung

## Source

Huang, K., Xiao, C., Hoang, T., Glass, L., & Sun, J. (2020). CASTER: Predicting drug interactions with chemical substructure representation. AAAI.

---

`example_chem` *Example Data for Compounds*

---

## Description

Blood-Brain-Barrier (BBB) is a permeability barrier for maintaining homeostasis of Central Nervous System (CNS). The data is a curated compound dataset with known BBB permeability. Compounds are divided into two groups according to whether the brain to blood concentration ratio was greater or less than 0.1. The row name labels each row with the compound name.

## Usage

```
example_chem
```

## Value

compound data

## Author(s)

Dongmin Jung

## Source

Gao, Z., Chen, Y., Cai, X., & Xu, R. (2017). Predict drug permeability to blood-brain-barrier from clinical phenotypes: drug side effects and drug indications. Bioinformatics, 33(6), 901-908.

---

`example_cpi` *Example Data for Compound-Protein Interactions*

---

## Description

The data consist of compound-protein pairs and their interactions of human. The SMILES and amino acid sequences are used for compounds and proteins, respectively. The binary outcome label is whether or not they interact each other.

## Usage

```
example_cpi
```

## Value

compound-protein interaction data

## Author(s)

Dongmin Jung

## Source

Tsubaki, M., Tomii, K., & Sese, J. (2019). Compound-protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. Bioinformatics, 35(2), 309-318.

---

# example_pd *Example Data for Primer-Dimer*

---

## Description

This is a primer-primer interaction data set with size 319. The two sequences are for primer pairs and the label is for their interactions.

## Usage

```
example_pd
```

## Value

primer sequences and dimer formation data

## Author(s)

Dongmin Jung

## Source

Johnston, A. D., Lu, J., Ru, K. L., Korbie, D., & Trau, M. (2019). PrimerROC: accurate condition-independent dimer prediction using ROC analysis. Scientific reports.

## example_ppi *Example Data for Protein-Protein Interactions*

### Description

The data is a randomly selected subset with size 5000 for protein-protein interactions of yeast. The two amino acid sequences are for protein pairs and the label is for their interactions.

### Usage

```
example_ppi
```

### Value

protein-protein interaction data

### Author(s)

Dongmin Jung

### Source

Chen, M., et al. (2019). Multifaceted protein-protein interaction prediction based on siamese residual rcnn. Bioinformatics, 35(14), i305-i314.

## example_prot *Example Data for Proteins*

### Description

This is a protein data set retrieved from Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank (PDB).The data consist of amino acid sequences with three classes. The row name labels each row with the PDB identification code.

### Usage

```
example_prot
```

### Value

protein data

### Author(s)

Dongmin Jung

## Source

Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank (PDB) and https://www.kaggle.com/shahir/p
data-set

---

`get_canonical_smiles` *Convert SMILES strings to canonical SMILES strings*

---

## Description

There may be many different ways to construct the SMILES string for a given molecule. A canonical representation is a unique ordering of the atoms for a given molecular graph.

## Usage

```
get_canonical_smiles(smiles)
```

## Arguments

| | |
|---|---|
| `smiles` | SMILES strings |

## Value

canonical representation of SMILES

## Author(s)

Dongmin Jung

## References

Leach, A. R., & Gillet, V. J. (2007). An introduction to chemoinformatics. Springer.

## See Also

rcdk::parse.smile, rcdk::get.smiles, rcdk::smiles.flavors

## Examples

```
get_canonical_smiles(example_cpi[1, 1])
```

`get_fingerprint` *Molecular fingerprint of compounds from SMILES strings*

## Description

A molecular fingerprint is a way of encoding the structural features of a molecule. The most common type of fingerprint is a sequence of ones and zeros. Fingerprints are special kinds of descriptors that characterize a molecule and its properties as a binary bit vector that represents the presence or absence of particular substructure in the molecule. For such a fingerprint, the Chemistry Development Kit (CDK) is used as a cheminformatics tool.

## Usage

```
get_fingerprint(smiles, ...)
```

## Arguments

| | |
|---|---|
| `smiles` | SMILES strings |
| `...` | arguments for "rcdk::get.fingerprint" but for molecule |

## Value

a fingerprint of a compound

## Author(s)

Dongmin Jung

## References

Balakin, K. V. (2009). Pharmaceutical data mining: approaches and applications for drug discovery. Wiley.

## See Also

rcdk::get.fingerprint, rcdk::parse.smiles

## Examples

```
get_fingerprint(example_cpi[1, 1])
```

`get_graph_structure_node_feature` *Graph structure and node features from SMILES strings*

## Description

In molecular graph representations, nodes represent atoms and edges represent bonds. For molecular features, the Chemistry Development Kit (CDK) is used as a cheminformatics tool. The degree of an atom in the graph representation and the atomic symbol and implicit hydrogen count for an atom are used as molecular features.

## Usage

```
get_graph_structure_node_feature(smiles, max_atoms,
    element_list = c(
        "C", "N", "O", "S", "F", "Si", "P", "Cl",
        "Br", "Mg", "Na", "Ca", "Fe",  "Al", "I",
        "B", "K", "Se", "Zn", "H", "Cu", "Mn"))
```

## Arguments

| | |
|---|---|
| `smiles` | SMILES strings |
| `max_atoms` | maximum number of atoms |
| `element_list` | list of atom symbols |

## Value

| | |
|---|---|
| `A_pad` | a padded or turncated adjacency matrix for each SMILES string |
| `X_pad` | a padded or turncated node features for each SMILES string |
| `feature_dim` | dimension of node features |
| `element_list` | list of atom symbols |

## Author(s)

Dongmin Jung

## References

Balakin, K. V. (2009). Pharmaceutical data mining: approaches and applications for drug discovery. Wiley.

## See Also

matlab::padarray, purrr::chuck, rcdk::get.adjacency.matrix, rcdk::get.atoms, rcdk::get.hydrogen.count, rcdk::get.symbol rcdk::parse.smiles

## Examples

```
get_graph_structure_node_feature(example_cpi[1, 1], 10)
```

---

`get_seq_encode_pad` *Vectorization of characters of strings*

---

## Description

A vectorization of characters of strings is necessary. Vectorized characters are padded or truncated.

## Usage

```
get_seq_encode_pad(sequences, length_seq, ngram_max = 1, ngram_min = 1,
    lenc = NULL)
```

## Arguments

| | |
|---|---|
| `sequences` | SMILE strings or amino acid sequences |
| `length_seq` | length of input sequences |
| `ngram_max` | maximum size of an n-gram (default: 1) |
| `ngram_min` | minimum size of an n-gram (default: 1) |
| `lenc` | encoded labels for characters, LableEncoder object fitted by "CatEncoders::LabelEncoder.fit" (default: NULL) |

## Value

| | |
|---|---|
| `sequences_encode_pad` | for each SMILES string, an encoded sequence which is padded or truncated |
| `lenc` | encoded labels for characters |
| `num_token` | total number of characters |

## Author(s)

Dongmin Jung

## See Also

CatEncoders::LabelEncoder.fit, CatEncoders::transform, keras::pad_sequences, stringdist::qgrams, tokenizers::tokenize_ngrams

## Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    get_seq_encode_pad(example_cpi[1, 2], 10)
}
```

`metric_concordance_index` *Concordance index*

## Description

The concordance index or c-index can be seen as one of the model performance metrics. It represents a good fit of the model.

## Author(s)

Dongmin Jung

## References

Kose, U., & Alzubi, J. (2020). Deep learning for cancer diagnosis. Springer.

## See Also

keras::k_cast, keras::k_equal, keras::k_sum, tensorflow::tf

## Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    compound_length_seq <- 50
    compound_embedding_dim <- 16
    protein_embedding_dim <- 16
    protein_length_seq <- 100

    mlp_cnn_cpi <- fit_cpi(
        smiles = example_cpi[1:100, 1],
        AAseq = example_cpi[1:100, 2],
        outcome = example_cpi[1:100, 3],
        compound_type = "sequence",
        compound_length_seq = compound_length_seq,
        compound_embedding_dim = compound_embedding_dim,
        protein_length_seq = protein_length_seq,
        protein_embedding_dim = protein_embedding_dim,
        net_args = list(
        compound = "mlp_in_out",
        compound_args = list(
                fc_units = c(10),
                fc_activation = c("relu")),
            protein = "cnn_in_out",
            protein_args = list(
                cnn_filters = c(32),
                cnn_kernel_size = c(3),
                cnn_activation = c("relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
```

```
            fc_units = c(1),
            fc_activation = c("sigmoid"),
            loss = "binary_crossentropy",
            optimizer = keras::optimizer_adam(),
            metrics = custom_metric("concordance_index",
                metric_concordance_index)),
        epochs = 2,
        batch_size = 16)
}
```

---

`metric_f1_score` *F1-score*

---

## Description

The F1-score is a metric combining precision and recall. It is typically used instead of accuracy in the case of severe class imbalance in the dataset. The higher the values of F1-score, the better the validation of the model.

## Author(s)

Dongmin Jung

## References

Kubben, P., Dumontier, M., & Dekker, A. (2019). Fundamentals of clinical data science. Springer.

Mishra, A., Suseendran, G., & Phung, T. N. (Eds.). (2020). Soft Computing Applications and Techniques in Healthcare. CRC Press.

## See Also

keras::k_equal, keras::k_sum, tensorflow::tf

## Examples

```
if (keras::is_keras_available() & reticulate::py_available()) {
    compound_length_seq <- 50
    compound_embedding_dim <- 16
    protein_embedding_dim <- 16
    protein_length_seq <- 100

    mlp_cnn_cpi <- fit_cpi(
        smiles = example_cpi[1:100, 1],
        AAseq = example_cpi[1:100, 2],
        outcome = example_cpi[1:100, 3],
        compound_type = "sequence",
        compound_length_seq = compound_length_seq,
        compound_embedding_dim = compound_embedding_dim,
        protein_length_seq = protein_length_seq,
```

```
        protein_embedding_dim = protein_embedding_dim,
        net_args = list(
        compound = "mlp_in_out",
        compound_args = list(
                fc_units = c(10),
                fc_activation = c("relu")),
            protein = "cnn_in_out",
            protein_args = list(
                cnn_filters = c(32),
                cnn_kernel_size = c(3),
                cnn_activation = c("relu"),
                fc_units = c(10),
                fc_activation = c("relu")),
            fc_units = c(1),
            fc_activation = c("sigmoid"),
            loss = "binary_crossentropy",
            optimizer = keras::optimizer_adam(),
            metrics = custom_metric("F1_score",
                metric_f1_score)),
        epochs = 2,
        batch_size = 16)
}
```

---

`multiple_sampling_generator` *Generator function for multiple inputs*

---

## Description

This is a generator function that yields batches of data with multiple inputs.

## Usage

```
multiple_sampling_generator(X_data, Y_data = NULL, batch_size,
    shuffle = TRUE)
```

## Arguments

| | |
|---|---|
| `X_data` | list of multiple inputs |
| `Y_data` | targets (default: NULL) |
| `batch_size` | batch size |
| `shuffle` | whether to shuffle the data or not (default: TRUE) |

## Value

generator for "keras::fit" or "keras::predict"

## Author(s)

Dongmin Jung

## Examples

```
X_data <- c(list(matrix(rnorm(200), ncol = 2)),
    list(matrix(rnorm(200), ncol = 2)))
Y_data <- matrix(rnorm(100), ncol = 1)
multiple_sampling_generator(X_data, Y_data, 32)
```

---

# SARS_CoV2_3CL_Protease

*Amino Acid Sequence for the SARS coronavirus 3C-like Protease*

---

## Description

306 amino acid residues of the SARS coronavirus 3C-like Protease

## Usage

```
SARS_CoV2_3CL_Protease
```

## Value

amino acid sequence

## Author(s)

Dongmin Jung

## Source

Huang, K., Fu, T., Glass, L. M., Zitnik, M., Xiao, C., & Sun, J. (2020). DeepPurpose: A Deep Learning Library for Drug-Target Interaction Prediction. Bioinformatics.

seq_check *Check SMILES strings and amino acid sequences*

## Description

In real-world cases, most of the data are not complete and contains incorrect values, missing values, and so on. Thus, there may be invalid sequences in the data. This function can find such sequences and remove them from the data. For SMILES strings, the function "webchem::is.smiles" is used. A valid amino acid sequence means a string that only contains capital letters of an alphabet.

## Usage

```
seq_check(smiles = NULL, AAseq = NULL, outcome = NULL)
```

## Arguments

| | |
|---|---|
| smiles | SMILES strings (default: NULL) |
| AAseq | amino acid sequences (default: NULL) |
| outcome | a variable that indicates how strong two molecules interact with each other or whether there is an interaction between them (default: NULL) |

## Value

valid sequences

## Author(s)

Dongmin Jung

## References

Dey, N., Wagh, S., Mahalle, P. N., & Pathan, M. S. (Eds.). (2019). Applied machine learning for smart data analysis. CRC Press.

## See Also

webchem::is.smiles

## Examples

```
seq_check(smiles = example_cpi[1, 1], outcome = example_cpi[1, 3])
```

`seq_preprocessing` *Preprocessing for SMILES strings and amino acid sequences*

## Description

Preprocessing helps make the data suitable for the model depending on the type of data the preprocessing works upon. Preprocessing is more time consuming for text data. The adjacency matrix and node feature, fingerprint, or string data are preprocessed from sequences.

## Usage

```
seq_preprocessing(smiles = NULL,
    AAseq = NULL,
    type,
    convert_canonical_smiles,
    max_atoms,
    length_seq,
    lenc = NULL,
    ngram_max = 1,
    ngram_min = 1)
```

## Arguments

| | |
|---|---|
| `smiles` | SMILES strings (default: NULL) |
| `AAseq` | amino acid sequences (default: NULL) |
| `type` | "graph", "fingerprint" or "sequence" |
| `convert_canonical_smiles` | SMILES strings are converted to canonical SMILES strings if TRUE |
| `max_atoms` | maximum number of atoms for compounds |
| `length_seq` | length of compound or protein sequence |
| `lenc` | encoded labels for characters of SMILES strings or amino acid sequenes (default: NULL) |
| `ngram_max` | maximum size of an n-gram for protein sequences (default: 1) |
| `ngram_min` | minimum size of an n-gram for protein sequences (default: 1) |

## Value

| | |
|---|---|
| `canonical_smiles` | canonical representation of SMILES |
| `convert_canonical_smiles` | canonical representation is used or not |
| `A_pad` | padded or turncated adjacency matrix of compounds if type is "graph" |
| `X_pad` | padded or turncated node features of compounds if type is "graph" |

| | |
|---|---|
| `fp` | fingerprint of compounds if type is "fingerprint" |
| `sequences_encode_pad` | encoded sequences which are padded or truncated |
| `lenc` | encoded labels for characters of SMILES strings or amino acid sequenes |
| `length_seq` | length of compound or protein sequence |
| `num_tokens` | total number of characters of compounds or proteins |

## Author(s)

Dongmin Jung

## References

Dey, N., Wagh, S., Mahalle, P. N., & Pathan, M. S. (Eds.). (2019). Applied machine learning for smart data analysis. CRC Press.

## Examples

```
seq_preprocessing(smiles = cbind(example_cpi[1, 1]),
    type = "fingerprint",
    convert_canonical_smiles = TRUE)
```

# Index

antiviral_drug, 3

cnn_in_out *(*encoder_in_out*)*, 9
cpi_model, 3

encoder_in_out, 9
example_bioassay, 11
example_cci, 11
example_chem, 12
example_cpi, 12
example_pd, 13
example_ppi, 14
example_prot, 14

fit_cpi *(*cpi_model*)*, 3

gcn_in_out *(*encoder_in_out*)*, 9
get_canonical_smiles, 15
get_fingerprint, 16
get_graph_structure_node_feature, 17
get_seq_encode_pad, 18

metric_concordance_index, 19
metric_f1_score, 20
mlp_in_out *(*encoder_in_out*)*, 9
multiple_sampling_generator, 21

predict_cpi *(*cpi_model*)*, 3

rnn_in_out *(*encoder_in_out*)*, 9

SARS_CoV2_3CL_Protease, 22
seq_check, 23
seq_preprocessing, 24