D-SCRIPT

Release v2.0

Samuel Sledzieski, Rohit Singh

Jun 25, 2022

CONTENTS

- D-SCRIPT Home Page
- Quick Start

D-SCRIPT is a deep learning method for predicting a physical interaction between two proteins given just their sequences. It generalizes well to new species and is robust to limitations in training data size. Its design reflects the intuition that for two proteins to physically interact, a subset of amino acids from each protein should be in contact with the other. The intermediate stages of D-SCRIPT directly implement this intuition, with the penultimate stage in D-SCRIPT being a rough estimate of the inter-protein contact map of the protein dimer. This structurally-motivated design enhances the interpretability of the results and, since structure is more conserved evolutionarily than sequence, improves generalizability across species.

If you use D-SCRIPT, please cite "D-SCRIPT translates genome to phenome with sequence-based, structure-aware, genome-scale predictions of protein-protein interactions" by Sam Sledzieski, Rohit Singh, Lenore Cowen, and Bonnie Berger.

If you use Topsy-Turvy, please cite "Topsy-Turvy: integrating a global view into sequence-based PPI prediction" by Kapil Devkota, Rohit Singh, Sam Sledzieski, Bonnie Berger, and Lenore Cowen.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

1.1.1 Requirements

- python 3.7
- pytorch 1.5
- h5py
- matplotlib
- numpy
- pandas
- scikit-learn
- scipy
- seaborn
- setuptools
- tqdm

Optional GPU support: CUDA Toolkit, cuDNN

1.1.2 Set up environment

```
$ git clone https://github.com/samsledje/D-SCRIPT.git

$ cd D-SCRIPT

$ conda env create --file environment.yml # Edit this file to change CUDA version if␣
↪necessary

$ conda activate dscript
```

1.1.3 Install from pip

```
pip install dscript
```

1.1.4 Build from source

```
$ git clone https://github.com/samsledje/D-SCRIPT.git

$ cd D-SCRIPT

$ python setup.py build; python setup.py install
```

1.2 Usage

1.2.1 Quick Start

Predict a new network using a trained model

Pre-trained models can be downloaded from here. Candidate pairs should be in tab-separated (`.tsv`) format with no header, and columns for [protein name 1], [protein name 2]. Optionally, a third column with [label] can be provided, so predictions can be made using training or test data files (but the label will not affect the predictions).

```
dscript predict --pairs [input data] --seqs [sequences, .fasta format] --model [model␣
↪file]
```

Embed sequences with language model

Sequences should be in `.fasta` format.

```
dscript embed --seqs [sequences] --outfile [embedding file]
```

Train and save a model

Training and validation data should be in tab-separated (`.tsv`) format with no header, and columns for [protein name 1], [protein name 2], [label].

```
dscript train --train [training data] --val [validation data] --embedding [embedding␣
↪file] --save-prefix [prefix]
```

Evaluate a trained model

```
dscript evaluate --model [model file] --test [test data] --embedding [embedding file]␣
↪--outfile [result file]
```

1.2.2 Prediction

```
usage: dscript predict [-h] --pairs PAIRS --model MODEL [--seqs SEQS]
                       [--embeddings EMBEDDINGS] [-o OUTFILE] [-d DEVICE]
                       [--thresh THRESH]

Make new predictions with a pre-trained model. One of --seqs and --embeddings is␣
↪required.

optional arguments:
-h, --help            show this help message and exit
--pairs PAIRS         Candidate protein pairs to predict
--model MODEL         Pretrained Model
--seqs SEQS           Protein sequences in .fasta format
--embeddings EMBEDDINGS
                        h5 file with embedded sequences
-o OUTFILE, --outfile OUTFILE
                        File for predictions
-d DEVICE, --device DEVICE
                        Compute device to use
--thresh THRESH       Positive prediction threshold - used to store contact
                        maps and predictions in a separate file. [default:
                        0.5]
```

1.2.3 Embedding

```
usage: dscript embed [-h] --seqs SEQS --outfile OUTFILE [-d DEVICE]

Generate new embeddings using pre-trained language model

optional arguments:
-h, --help            show this help message and exit
--seqs SEQS           Sequences to be embedded
--outfile OUTFILE     h5 file to write results
-d DEVICE, --device DEVICE
                        Compute device to use
```

1.2.4 Training

```
usage: dscript train [-h] --train TRAIN --test TEST --embedding EMBEDDING
                     [--no-augment] [--input-dim INPUT_DIM]
                     [--projection-dim PROJECTION_DIM] [--dropout-p DROPOUT_P]
                     [--hidden-dim HIDDEN_DIM] [--kernel-width KERNEL_WIDTH]
                     [--no-w] [--no-sigmoid] [--do-pool]
                     [--pool-width POOL_WIDTH] [--num-epochs NUM_EPOCHS]
                     [--batch-size BATCH_SIZE] [--weight-decay WEIGHT_DECAY]
```

(continues on next page)

(continued from previous page)

```
                [--lr LR] [--lambda INTERACTION_WEIGHT] [--topsy-turvy]
                [--glider-weight GLIDER_WEIGHT]
                [--glider-thresh GLIDER_THRESH] [-o OUTFILE]
                [--save-prefix SAVE_PREFIX] [-d DEVICE]
                [--checkpoint CHECKPOINT]

Train a new model.

optional arguments:
  -h, --help            show this help message and exit

Data:
  --train TRAIN         list of training pairs
  --test TEST           list of validation/testing pairs
  --embedding EMBEDDING
                        h5py path containing embedded sequences
  --no-augment          data is automatically augmented by adding (B A) for
                        all pairs (A B). Set this flag to not augment data

Projection Module:
  --input-dim INPUT_DIM
                        dimension of input language model embedding (per amino
                        acid) (default: 6165)
  --projection-dim PROJECTION_DIM
                        dimension of embedding projection layer (default: 100)
  --dropout-p DROPOUT_P
                        parameter p for embedding dropout layer (default: 0.5)

Contact Module:
  --hidden-dim HIDDEN_DIM
                        number of hidden units for comparison layer in contact
                        prediction (default: 50)
  --kernel-width KERNEL_WIDTH
                        width of convolutional filter for contact prediction
                        (default: 7)

Interaction Module:
  --no-w                don't use weight matrix in interaction prediction
                        model
  --no-sigmoid          don't use sigmoid activation at end of interaction
                        model
  --do-pool             use max pool layer in interaction prediction model
  --pool-width POOL_WIDTH
                        size of max-pool in interaction model (default: 9)

Training:
  --num-epochs NUM_EPOCHS
                        number of epochs (default: 10)
  --batch-size BATCH_SIZE
                        minibatch size (default: 25)
  --weight-decay WEIGHT_DECAY
                        L2 regularization (default: 0)
  --lr LR               learning rate (default: 0.001)
  --lambda INTERACTION_WEIGHT
                        weight on the similarity objective (default: 0.35)
  --topsy-turvy         run in Topsy-Turvy mode -- use top-down GLIDER scoring
                        to guide training (reference TBD)
```

(continues on next page)

(continued from previous page)

```
  --glider-weight GLIDER_WEIGHT
                        weight on the GLIDER accuracy objective (default: 0.2)
  --glider-thresh GLIDER_THRESH
                        proportion of GLIDER scores treated as positive edges
                        (0 < gt < 1) (default: 0.925)

Output and Device:
  -o OUTPUT, --output OUTPUT
                        output file path (default: stdout)
  --save-prefix SAVE_PREFIX
                        path prefix for saving models
  -d DEVICE, --device DEVICE
                        compute device to use
  --checkpoint CHECKPOINT
                        checkpoint model to start training from
```

1.2.5 Evaluation

```
usage: dscript eval [-h] --model MODEL --test TEST --embedding EMBEDDING
                    [-o OUTFILE] [-d DEVICE]

Evaluate a trained model

optional arguments:
-h, --help            show this help message and exit
--model MODEL         Trained prediction model
--test TEST           Test Data
--embedding EMBEDDING
                        h5 file with embedded sequences
-o OUTFILE, --outfile OUTFILE
                        Output file to write results
-d DEVICE, --device DEVICE
                        Compute device to use
```

1.3 Data

1.3.1 Trained Models

- Bepler & Berger language model
- Human data trained model

1.3.2 Sample Data

Sequences

- Human
- Mouse
- Fly
- Yeast
- Worm
- E.coli

Interactions

- Human Train
- Human Test
- Mouse Test
- Fly Test
- Yeast Test
- Worm Test
- E. coli Test

1.4 API

1.4.1 dscript.commands

dscript.commands.predict

See Prediction for full usage details.

dscript.commands.embed

See Embedding for full usage details.

Generate new embeddings using pre-trained language model.

dscript.commands.train

See Training for full usage details.

dscript.commands.evaluate

See Evaluation for full usage details.

Evaluate a trained model.

dscript.commands.evaluate.**plot_eval_predictions**(*labels*, *predictions*, *path='figure'*)

Plot histogram of positive and negative predictions, precision-recall curve, and receiver operating characteristic curve.

Parameters

- **y** (*np.ndarray*) – Labels
- **phat** (*np.ndarray*) – Predicted probabilities
- **path** (*str*) – File prefix for plots to be saved to [default: figure]

1.4.2 dscript.models

dscript.models.embedding

class dscript.models.embedding.**FullyConnectedEmbed**(*nin*, *nout*, *dropout=0.5*, *activation=ReLU()*)

Bases: torch.nn.modules.module.Module

Protein Projection Module. Takes embedding from language model and outputs low-dimensional interaction aware projection.

Parameters

- **nin** (*int*) – Size of language model output
- **nout** (*int*) – Dimension of projection
- **dropout** (*float*) – Proportion of weights to drop out [default: 0.5]
- **activation** (*torch.nn.Module*) – Activation for linear projection model

forward(*x*)

Parameters **x** (*torch.Tensor*) – Input language model embedding ($b \times N \times d_0$)

Returns Low dimensional projection of embedding

Return type torch.Tensor

class dscript.models.embedding.**IdentityEmbed**

Bases: torch.nn.modules.module.Module

Does not reduce the dimension of the language model embeddings, just passes them through to the contact model.

forward(*x*)

Parameters **x** (*torch.Tensor*) – Input language model embedding ($b \times N \times d_0$)

Returns Same embedding

Return type torch.Tensor

class dscript.models.embedding.**LSTMEmbed**(*nout*, *activation='ReLU'*, *sparse=False*, *p=0.5*)

Bases: torch.nn.modules.module.Module

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

long_embed(*x*)

class dscript.models.embedding.**SkipLSTM**(*nin=21*, *nout=100*, *hidden_dim=1024*, *num_layers=3*, *dropout=0*, *bidirectional=True*)

Bases: torch.nn.modules.module.Module

Language model from Bepler & Berger.

Loaded with pre-trained weights in embedding function.

Parameters

- **nin** (*int*) – Input dimension of amino acid one-hot [default: 21]
- **nout** (*int*) – Output dimension of final layer [default: 100]
- **hidden_dim** (*int*) – Size of hidden dimension [default: 1024]
- **num_layers** (*int*) – Number of stacked LSTM models [default: 3]
- **dropout** (*float*) – Proportion of weights to drop out [default: 0]
- **bidirectional** (*bool*) – Whether to use biLSTM vs. LSTM

to_one_hot(*x*)

Transform numeric encoded amino acid vector to one-hot encoded vector

Parameters **x** (*torch.Tensor*) – Input numeric amino acid encoding (N)

Returns One-hot encoding vector $(N \times n_{in})$

Return type torch.Tensor

transform(*x*)

Parameters **x** (*torch.Tensor*) – Input numeric amino acid encoding (N)

Returns Concatenation of all hidden layers $(N \times (n_{in} + 2 \times$

dscript.models.contact

class dscript.models.contact.**ContactCNN**(*embed_dim=100*, *hidden_dim=50*, *width=7*, *activation=Sigmoid()*)

Bases: torch.nn.modules.module.Module

Residue Contact Prediction Module. Takes embeddings from Projection module and produces contact map, output of Contact module.

Parameters

- **embed_dim** (*int*) – Output dimension of *dscript.models.embedding* model d [default: 100]
- **hidden_dim** (*int*) – Hidden dimension h [default: 50]
- **width** (*int*) – Width of convolutional filter $2w + 1$ [default: 7]
- **activation** (*torch.nn.Module*) – Activation function for final contact map [default: torch.nn.Sigmoid()]

cmap(*z0*, *z1*)

Calls *dscript.models.contact.FullyConnected*.

Parameters

- **z0** (*torch.Tensor*) – Projection module embedding $(b \times N \times d)$
- **z1** (*torch.Tensor*) – Projection module embedding $(b \times M \times d)$

Returns Predicted contact broadcast tensor $(b \times N \times M \times h)$

Return type torch.Tensor

forward(*z0*, *z1*)

Parameters

- **z0** (*torch.Tensor*) – Projection module embedding $(b \times N \times d)$
- **z1** (*torch.Tensor*) – Projection module embedding $(b \times M \times d)$

Returns Predicted contact map $(b \times N \times M)$

Return type torch.Tensor

predict(*C*)

Predict contact map from broadcast tensor.

Parameters **B** (*torch.Tensor*) – Predicted contact broadcast $(b \times N \times M \times h)$

Returns Predicted contact map $(b \times N \times M)$

Return type torch.Tensor

class dscript.models.contact.**FullyConnected**(*embed_dim*, *hidden_dim*, *activation=ReLU()*)

Bases: torch.nn.modules.module.Module

Performs part 1 of Contact Prediction Module. Takes embeddings from Projection module and produces broadcast tensor.

Input embeddings of dimension d are combined into a $2d$ length MLP input z_{cat}, where $z_{cat} = [z_0 \ominus z_1 | z_0 \odot z_1]$

Parameters

- **embed_dim** (*int*) – Output dimension of *dscript.models.embedding* model d [default: 100]

- **hidden_dim** (*int*) – Hidden dimension h [default: 50]
- **activation** (*torch.nn.Module*) – Activation function for broadcast tensor [default: torch.nn.ReLU()]

forward(*z0*, *z1*)

Parameters

- **z0** (*torch.Tensor*) – Projection module embedding $(b \times N \times d)$
- **z1** (*torch.Tensor*) – Projection module embedding $(b \times M \times d)$

Returns Predicted broadcast tensor $(b \times N \times M \times h)$

Return type torch.Tensor

dscript.models.interaction

class dscript.models.interaction.**LogisticActivation**(*x0=0*, *k=1*, *train=False*)

Bases: torch.nn.modules.module.Module

Implementation of Generalized Sigmoid Applies the element-wise function:

$\sigma(x) = \frac{1}{1+\exp(-k(x-x_0))}$

Parameters

- **x0** (*float*) – The value of the sigmoid midpoint
- **k** (*float*) – The slope of the sigmoid - trainable - $k \geq 0$
- **train** (*bool*) – Whether k is a trainable parameter

forward(*x*)

Applies the function to the input elementwise

Parameters **x** (*torch.Tensor*) – $(N \times *)$ where $*$ means, any number of additional dimensions

Returns $(N \times *)$, same shape as the input

Return type torch.Tensor

class dscript.models.interaction.**ModelInteraction**(*embedding*, *contact*, *use_cuda=True*, *do_w=True*, *do_sigmoid=True*, *do_pool=False*, *pool_size=9*, *theta_init=1*, *lambda_init=0*, *gamma_init=0*)

Bases: torch.nn.modules.module.Module

cpred(*z0*, *z1*)

Project down input language model embeddings into low dimension using projection module

Parameters

- **z0** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$
- **z1** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$

Returns Predicted contact map $(b \times N \times M)$

Return type torch.Tensor

embed(*x*)

Project down input language model embeddings into low dimension using projection module

Parameters **z** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$

Returns D-SCRIPT projection $(b \times N \times d)$

Return type torch.Tensor

map_predict(*z0*, *z1*)
Project down input language model embeddings into low dimension using projection module

Parameters

- **z0** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$
- **z1** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$

Returns Predicted contact map, predicted probability of interaction $(b \times N \times d_0), (1)$

Return type torch.Tensor, torch.Tensor

predict(*z0*, *z1*)
Project down input language model embeddings into low dimension using projection module

Parameters

- **z0** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$
- **z1** (*torch.Tensor*) – Language model embedding $(b \times N \times d_0)$

Returns Predicted probability of interaction

Return type torch.Tensor, torch.Tensor

1.4.3 dscript.alphabets

class dscript.alphabets.**Alphabet**(*chars*, *encoding=None*, *mask=False*, *missing=255*)
Bases: object

From Bepler & Berger.

Parameters

- **chars** (*byte str*) – List of characters in alphabet
- **encoding** (*np.ndarray*) – Mapping of characters to numbers [default: encoding]
- **mask** (*bool*) – Set encoding mask [default: False]
- **missing** (*int*) – Number to use for a value outside the alphabet [default: 255]

decode(*x*)
Decode numeric encoding to byte string of this alphabet

Parameters **x** (*np.ndarray*) – Numeric encoding

Returns Amino acid string

Return type byte str

encode(*x*)
Encode a byte string into alphabet indices

Parameters **x** (*byte str*) – Amino acid string

Returns Numeric encoding

Return type np.ndarray

get_kmer(*h*, *k*)
retrieve byte string of length k decoded from integer h

unpack(*h*, *k*)
unpack integer h into array of this alphabet with length k

class dscript.alphabets.**Uniprot21**(*mask=False*)
Bases: *dscript.alphabets.Alphabet*

Uniprot 21 Amino Acid Encoding.

From Bepler & Berger.

1.4.4 dscript.fasta

dscript.fasta.**count_bins**(*array*, *bins*)

dscript.fasta.**parse**(*f*, *comment='#'*)

dscript.fasta.**parse_directory**(*directory*, *extension='.seq'*)

dscript.fasta.**write**(*nam*, *seq*, *f*)

1.4.5 dscript.language_model

dscript.language_model.**embed_from_fasta**(*fastaPath*, *outputPath*, *device=0*, *verbose=False*)
Embed sequences using pre-trained language model from Bepler & Berger.

Parameters

- **fastaPath** (*str*) – Input sequence file (`.fasta` format)
- **outputPath** (*str*) – Output embedding file (`.h5` format)
- **device** (*int*) – Compute device to use for embeddings [default: 0]
- **verbose** (*bool*) – Print embedding progress

dscript.language_model.**lm_embed**(*sequence*, *use_cuda=False*, *verbose=True*)
Embed a single sequence using pre-trained language model from Bepler & Berger.

Parameters

- **sequence** (*str*) – Input sequence to be embedded
- **use_cuda** (*bool*) – Whether to generate embeddings using GPU device [default: False]

Returns Embedded sequence

Return type torch.Tensor

1.4.6 dscript.pretrained

dscript.pretrained.**get_pretrained**(*version='human_v1'*, *verbose=True*)
Get pre-trained model object.

See the documentation for most up-to-date list.

- lm_v1 - Language model from Bepler & Berger.
- human_v1 - Human trained model from D-SCRIPT manuscript.

Default: human_v1

Parameters **version** (*str*) – Version of pre-trained model to get

Returns Pre-trained model

Return type dscript.models.*

dscript.pretrained.**get_state_dict**(*version='human_v1'*, *verbose=True*)
Download a pre-trained model if not already exists on local device.

Parameters

- **version** (*str*) – Version of trained model to download [default: human_1]
- **verbose** (*bool*) – Print model download status on stdout [default: True]

Returns Path to state dictionary for pre-trained language model

Return type str

1.4.7 dscript.glider

dscript.glider.**compute_X_normalized**(*A*, *D*, *t=- 1*, *lm=1*, *is_normalized=True*)

dscript.glider.**compute_cw_score**(*p*, *q*, *edgedict*, *ndict*, *params=None*)
Computes the common weighted score between p and q.

Parameters

- **p** – A node of the graph
- **q** – Another node in the graph
- **edgedict** (*dict*) – A dictionary with key *(p, q)* and value *w*.
- **ndict** (*dict*) – A dictionary with key *p* and the value a set *{p1, p2, . . . }*
- **params** (*None*) – Should always be none here

Returns A real value representing the score

Return type float

dscript.glider.**compute_cw_score_normalized**(*p*, *q*, *edgedict*, *ndict*, *params=None*)
Computes the common weighted normalized score between p and q.

Parameters

- **p** – A node of the graph
- **q** – Another node in the graph
- **edgedict** (*dict*) – A dictionary with key *(p, q)* and value *w*.
- **ndict** (*dict*) – A dictionary with key *p* and the value a set *{p1, p2, . . . }*

- **params** (*None*) – Should always be none here

Returns A real value representing the score

Return type float

dscript.glider.**compute_degree_vec**(*edgelist*)

dscript.glider.**compute_l3_score_mat**(*p*, *q*, *edgedict*, *ndict*, *params=None*)

dscript.glider.**compute_l3_unweighted_mat**(*A*)

dscript.glider.**compute_l3_weighted_mat**(*A*)

dscript.glider.**compute_pinverse_diagonal**(*D*)

dscript.glider.**create_edge_dict**(*edgelist*)
Creates an edge dictionary with the edge *(p, q)* as the key, and weight *w* as the value.

Parameters **edgelist** (*list*) – list with elements of form *(p, q, w)*

Returns A dictionary with key *(p, q)* and value *w*.

Return type dict

dscript.glider.**create_neighborhood_dict**(*edgelist*)
Create a dictionary with nodes as key and a list of neighborhood nodes as the value

Parameters **edgelist** (*list*) – A list with elements of form *(p, q, w)*

Returns neighborhood_dict -> A dictionary with key *p* and value, a set *{p1, p2, p3, . . . }*

Return type dict

dscript.glider.**densify**(*edgelist*, *dim=None*, *directed=False*)
Given an adjacency list for the graph, computes the adjacency matrix.

Parameters

- **edgelist** (*list*) – Graph adjacency list
- **dim** (*int*) – Number of nodes in the graph
- **directed** (*bool*) – Whether the graph should be treated as directed

Returns Graph as an adjacency matrix

Return type np.ndarray

dscript.glider.**get_dim**(*edgelist*)
Given an adjacency list for a graph, returns the number of nodes in the graph.

Parameters **edgelist** (*list*) – Graph adjacency list

Returns Number of nodes in the graph

Return type int

dscript.glider.**glide_compute_map**(*pos_df*, *thres_p=0.9*, *params={}*)
Return glide_mat and glide_map.

Parameters

- **pos_df** (*pd.DataFrame*) – Dataframe of weighted edges
- **thres_p** (*float*) – Threshold to treat an edge as positive
- **params** (*dict*) – Parameters for GLIDE

Returns glide_matrix and corresponding glide_map

Return type tuple(np.ndarray, dict)

dscript.glider.**glide_predict_links**(*edgelist*, *X*, *params={}*, *thres_p=0.9*)

Predicts the most likely links in a graph given an embedding X of a graph. Returns a ranked list of (edges, distances) sorted from closest to furthest.

Parameters

- **edgelist** – A list with elements of type *(p, q, wt)*
- **X** – A nxk embedding matrix
- **params** – A dictionary with entries

{ alpha => real number beta => real number delta => real number loc => String, can be *cw* for common weighted, *l3* for l3 local scoring

To enable ctypes, the following entries should be there

ctypes_on => True # This key should only be added if ctypes is on (dont add this # if ctypes is not added)

so_location => String location of the .so dynamic library

}

dscript.glider.**glider_score**(*p*, *q*, *glider_map*, *glider_mat*)

1.4.8 dscript.utils

dscript.utils.**RBF**(*D*, *sigma=None*)

Convert distance matrix into similarity matrix using Radial Basis Function (RBF) Kernel.

$RBF(x, x') = \exp \frac{-(x-x')^2}{2\sigma^2}$

Parameters

- **D** (*np.ndarray*) – Distance matrix
- **sigma** (*float*) – Bandwith of RBF Kernel [default: $\sqrt{\max(D)}$]

Returns Similarity matrix

Return type np.ndarray

dscript.utils.**augment_data**(*df*)

For all pairs (A B), also add pairs (B A) :param df: Data frame with 3 columns - pair1, pair2, label :type df: pd.DataFrame :return: Augmented data frame :rtype: pd.DataFrame

dscript.utils.**config_logger**(*file*, *fmt*, *level=2*, *use_stdout=True*)

dscript.utils.**get_local_or_download**(*destination: str*, *source: Optional[str] = None*)

Return file path *destination*, and if it does not exist download from *source*.

Parameters

- **destination** (*str*) – Destination path for downloaded file
- **source** (*str*) – URL to download file from

Returns Path of local file

Return type str

dscript.utils.**load_hdf5_parallel**(*file_path*, *keys*, *n_jobs=- 1*)

Load keys from hdf5 file into memory

Parameters

- **file_path** (*str*) – Path to hdf5 file
- **keys** (*list[str]*) – List of keys to get

Returns Dictionary with keys and records in memory

Return type dict

dscript.utils.**plot_eval_predictions**(*labels*, *predictions*, *path='figure'*)

Plot histogram of positive and negative predictions, precision-recall curve, and receiver operating characteristic curve.

Parameters

- **y** (*np.ndarray*) – Labels
- **phat** (*np.ndarray*) – Predicted probabilities
- **path** (*str*) – File prefix for plots to be saved to [default: figure]

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

INDEX

I

L

M

P

R

S

T

U

W