Elliot

Release Elliot

Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Tommaso Di Noia

Mar 01, 2021

#ELLIOT

«Every hacker has her fixation. You hack people, I hack time.»

![PyPI - Python Version](https://img.shields.io/pypi/pyversions/scikit-daisy) [![Version](https://img.shields.io/badge/version-v1.1.2-orange)](https://github.com/sisinflab/elliot) ![GitHub repo size](https://img.shields.io/github/repo-size/sisinflab/elliot) ![GitHub](https://img.shields.io/github/license/sisinflab/elliot)

Elliot is a comprehensive recommendation framework that analyzes the recommendation problem from the researcher's perspective. It conducts a whole experiment, from dataset loading to results gathering. The core idea is to feed the system with a simple and straightforward configuration file that drives the framework through the experimental setting choices. Elliot untangles the complexity of combining splitting strategies, hyperparameter model optimization, model training, and the generation of reports of the experimental results.

The framework loads, filters, and splits the data considering a vast set of strategies (splitting methods and filtering approaches, from temporal training-test splitting to nested K-folds Cross-Validation). Elliot optimizes hyperparameters for several recommendation algorithms, selects the best models, compares them with the baselines providing intra-model statistics, computes metrics spanning from accuracy to beyond-accuracy, bias, and fairness, and conducts statistical analysis (Wilcoxon and Paired t-test).

Elliot aims to keep the entire experiment reproducible and put the user in control of the framework.

## Installation Elliot works with the following operating systems:

- Linux
- Windows 10
- macOS X

Elliot requires Python version 3.6 or later.

RecBole requires tensorflow version 2.4.1 or later. If you want to use Elliot with GPU, please ensure that CUDA or cudatoolkit version is XXX.XXX or later. This requires NVIDIA driver version >= XXX.XXX (for Linux) or >= XXX.XXX (for Windows10).

### Install from source

#### CONDA `` `bash git clone https://github.com//sisinflab/elliot.git && cd elliot conda create --name elliot_env python=3.8 conda activate pip install -e . --verbose ` ``

#### VIRTUALENV `` `bash git clone https://github.com//sisinflab/elliot.git && cd elliot python3 -m venv ./venv source venv/bin/activate pip install -e . --verbose ` ``

## Quick Start

Elliot's entry point is the function *run_experiment*, which accepts a configuration file that drives the whole experiment. In the following, a sample configuration file is shown to demonstrate how a sample and explicit structure can generate a rigorous experiment.

~~~
```python from elliot.run import run_experiment

run_experiment(“configuration/file/path”) ```
~~~

The following file is a simple configuration for an experimental setup. It contains all the instructions to get the MovieLens-1M catalog from a specific path and perform a train test split in a random sample way with a ratio of 20%.

This experiment provides a hyperparameter optimization with a grid search strategy for an Item-KNN model. Indeed, it is seen that the possible values of neighbors are closed in squared brackets. It indicates that two different models equipped with two different neighbors' values will be trained and compared to select the best configuration. Moreover, this configuration obliges Elliot to save the recommendation lists with at most 10 items per user as suggest by top_k property.

In this basic experiment, only a simple metric is considered in the final evaluation study. The candidate metric is nDCG for a cutoff equal to top_k, unless otherwise noted.

~~~
```yaml experiment:

      dataset: movielens_1m data_config:

            strategy: dataset dataset_path: ../data/movielens_1m/dataset.tsv splitting:

                  test_splitting:
                      strategy: random_subsampling test_ratio: 0.2

            models:

                ItemKNN:

                    meta:
                        hyper_opt_alg: grid save_recs: True
                    neighbors: [50, 100] similarity: cosine

            evaluation:
                simple_metrics: [nDCG]

            top_k: 10

```
~~~

## Contributing

There are many ways to contribute to Elliot! You can contribute code, make improvements to the documentation, report or investigate [bugs and issues](https://github.com/sisinflab/elliot/issues)

We welcome all contributions from bug fixes to new features and extensions.

Feel free to share with us your custom configuration files. We are creating a vault of reproducible experiments, and we would be glad of mentioning your contribution.

Reference Elliot in your blogs, papers, and articles.

Talk about Elliot on social media with the hashtag **#elliotrs**.

## The Team Elliot is developed by * Vito Walter Anelli<sup id=”a1”>[*](#f1)</sup> (vitowalter.anelli@poliba.it) * Alejandro Bellogín (alejandro.bellogin@uam.es) * Antonio Ferrara (antonio.ferrara@poliba.it) * Daniele Malitesta (daniele.malitesta@poliba.it) * Felice Antonio Merra (felice.merra@poliba.it) * Claudio Pomo<sup id=”a1”>[*](#f1)</sup> (claudio.pomo@poliba.it) * Tommaso Di Noia (tommaso.dinoia@poliba.it)

It is maintained by [SisInfLab Group](http://sisinflab.poliba.it/) and [Information Retrieval Group](http://ir.ii.uam.es/).

<b id=”f1”><sup>*</sup></b> Corresponding authors ## License ELLIOT uses [APACHE2 License](./LICENSE).

## Acknowledgements

We refer to the following repositories to improve our code:

- SliM and KNN-CF parts with [RecSys2019_DeepLearning_Evaluation](https://github.com/MaurizioFD/RecSys2019_DeepLearning_Ev

Chapter 1

elliot package

1.1 Subpackages

1.1.1 elliot.dataset package

Subpackages

elliot.dataset.dataloader package

Submodules

elliot.dataset.dataloader.knowledge_aware_chains module

Module description:

class `elliot.dataset.dataloader.knowledge_aware_chains.`**`KnowledgeChainsDataObject`** (*config, data_tuple, side_information_data, *args, **kwargs*)

Bases: `object`

Load train and test dataset

`build_dict` (*dataframe, users*)

`build_sparse` ( )

`build_sparse_ratings` ( )

`dataframe_to_dict` (*data*)

`get_test` ( )

`get_validation` ( )

class `elliot.dataset.dataloader.knowledge_aware_chains.`**`KnowledgeChainsLoader`** (*config, *args, **kwargs*)

Bases: `object`

Load train and test dataset

`check_timestamp` (*d: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

`generate_dataobjects` ( ) → List[object]

`generate_dataobjects_mock` ( ) → List[object]

`load_attribute_file` (*attribute_file, separator='\t'*)

load_dataset_dataframe (*file_ratings, separator='\t', attribute_file=None, feature_file=None, properties_file=None, column_names=['userId', 'itemId', 'rating', 'timestamp'], additive=True, threshold=10*)

load_dataset_dict (*file_ratings, separator='\t', attribute_file=None, feature_file=None, properties_file=None, additive=True, threshold=10*)

load_feature_names (*infile, separator='\t'*)

load_item_set (*ratings_file, separator='\t', itemPosition=1*)

load_properties (*properties_file*)

read_splitting (*folder_path*)

reduce_attribute_map_property_selection (*map, items, feature_names, properties, additive, threshold=10*)

reduce_dataset_by_item_list (*ratings_file, items, separator='\t'*)

elliot.dataset.dataloader.visual_dataloader module

Module description:

class `elliot.dataset.dataloader.visual_dataloader.`**VisualDataObject** (*config, data_tuple, side_information_data, *args, **kwargs*)

Bases: `object`

Load train and test dataset

build_dict (*dataframe, users*)

build_sparse ( )

build_sparse_ratings ( )

dataframe_to_dict (*data*)

get_test ( )

get_validation ( )

read_images (*images_folder, image_set, size_tuple*)

read_images_multiprocessing (*images_folder, image_set, size_tuple*)

static read_single_image (*images_folder, image_set, size_tuple, image_path*)

class `elliot.dataset.dataloader.visual_dataloader.`**VisualLoader** (*config, *args, **kwargs*)

Bases: `object`

Load train and test dataset

check_timestamp (*d: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

generate_dataobjects ( ) → List[object]

generate_dataobjects_mock ( ) → List[object]

load_dataset_dataframe (*file_ratings, separator='\t', visual_feature_set=None, column_names=['userId', 'itemId', 'rating', 'timestamp']*)

read_splitting (*folder_path*)

reduce_dataset_by_item_list (*ratings_file, items, separator='\t'*)

Module contents

elliot.dataset.samplers package

Submodules

elliot.dataset.samplers.custom_pointwise_sparse_sampler module

Module description:

class `elliot.dataset.samplers.custom_pointwise_sparse_sampler.`**Sampler** (*indexed_ratings, sp_i_train*)

Bases: `object`

step (*events: int, batch_size: int*)

elliot.dataset.samplers.custom_sampler module

Module description:

class `elliot.dataset.samplers.custom_sampler.`**Sampler** (*indexed_ratings*)

Bases: `object`

step (*events: int, batch_size: int*)

elliot.dataset.samplers.custom_sparse_sampler module

Module description:

class `elliot.dataset.samplers.custom_sparse_sampler.`**Sampler** (*indexed_ratings, sp_i_train*)

Bases: `object`

step (*events: int, batch_size: int*)

elliot.dataset.samplers.pairwise_sampler module

Module description:

class `elliot.dataset.samplers.pairwise_sampler.`**Sampler** (*ratings, users, items*)

Bases: `object`

step (*events: int*)

elliot.dataset.samplers.pipeline_sampler module

Module description:

class `elliot.dataset.samplers.pipeline_sampler.`**Sampler** (*indexed_ratings, item_indices, images_path, output_image_size, epochs*)

Bases: `object`

pipeline (*num_users, batch_size*)

pipeline_eval (*batch_size*)

read_image (*item*)

read_images_triple (*user, pos, neg*)

step (*events: int, batch_size: int*)

elliot.dataset.samplers.pointwise_cfgan_sampler module

Module description:

class `elliot.dataset.samplers.pointwise_cfgan_sampler.`**`Sampler`** (*indexed_ratings, sp_i_train, s_zr, s_pm*)

Bases: `object`

`step` (*events: int, batch_size: int*)

elliot.dataset.samplers.pointwise_pos_neg_ratings_sampler module

Module description:

class `elliot.dataset.samplers.pointwise_pos_neg_ratings_sampler.`**`Sampler`** (*indexed_ratings, sparse_i_ratings*)

Bases: `object`

`step` (*events: int, batch_size: int*)

elliot.dataset.samplers.pointwise_pos_neg_ratio_ratings_sampler module

Module description:

class `elliot.dataset.samplers.pointwise_pos_neg_ratio_ratings_sampler.`**`Sampler`** (*indexed_ratings, sparse_i_ratings, neg_ratio*)

Bases: `object`

`step` (*events: int, batch_size: int*)

elliot.dataset.samplers.pointwise_pos_neg_sampler module

Module description:

class `elliot.dataset.samplers.pointwise_pos_neg_sampler.`**`Sampler`** (*indexed_ratings*)

Bases: `object`

`step` (*events: int, batch_size: int*)

elliot.dataset.samplers.pointwise_wide_and_deep_sampler module

Module description:

class `elliot.dataset.samplers.pointwise_wide_and_deep_sampler.`**`Sampler`** (*data*)

Bases: `object`

`step` (*events: int, batch_size: int*)

elliot.dataset.samplers.sparse_sampler module

Module description:

class `elliot.dataset.samplers.sparse_sampler.`**`Sampler`** (*sp_i_train*)

Bases: `object`

`step` (*users: int, batch_size: int*)

Module contents

Module description:

Submodules

elliot.dataset.abstract_dataset module

class `elliot.dataset.abstract_dataset.`**`AbstractDataset`** (**args, **kwargs*)

Bases: `object`

abstract **`build_dict`** ( )

abstract **`build_sparse`** (**args*)

abstract get_test (**args*)

required_attributes = ['config', 'args', 'kwargs', 'users', 'items', 'num_users', 'num_items', 'private_users', 'public_users', 'private_items', 'public_items', 'transactions', 'train_dict', 'i_train_dict', 'sp_i_train', 'test_dict']

class `elliot.dataset.abstract_dataset.`**`ForceRequiredAttributeDefinitionMeta`**

Bases: `type`

`check_required_attributes` (*class_object*)

elliot.dataset.dataset module

Module description:

class `elliot.dataset.dataset.`**`DataSet`** (**args, **kwargs*)

Bases: `elliot.dataset.abstract_dataset.AbstractDataset`

Load train and test dataset

`build_dict` (*dataframe, users*)

`build_sparse` ( )

`build_sparse_ratings` ( )

`dataframe_to_dict` (*data*)

`get_test` ( )

`get_validation` ( )

class `elliot.dataset.dataset.`**`DataSetLoader`** (*config, *args, **kwargs*)

Bases: `object`

Load train and test dataset

`check_timestamp` (*d: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

`generate_dataobjects` ( ) → List[object]

`generate_dataobjects_mock` ( ) → List[object]

`read_splitting` (*folder_path*)

Module contents

Module description:

1.1.2 elliot.evaluation package

Subpackages

elliot.evaluation.metrics package

Subpackages

elliot.evaluation.metrics.accuracy package

Subpackages

elliot.evaluation.metrics.accuracy.AUC package

Submodules

elliot.evaluation.metrics.accuracy.AUC.auc module

This is the implementation of the global AUC metric. It proceeds from a system-wise computation.

class `elliot.evaluation.metrics.accuracy.AUC.auc.`**`AUC`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the global AUC recommendation metric. Passing 'AUC' to the metrics list will enable the computation of the metric.

Note:

> This metric does not calculate group-based AUC which considers the AUC scores averaged across users. It is also not limited to k. Instead, it calculates the scores on the entire prediction results regardless the users.

\mathrm {AUC} = \frac{\sum\limits_{i=1}^M rank_{i} - \frac {{M} \times {(M+1)}}{2}} {{{M} \times {N}}}

`M` is the number of positive samples. `N` is the number of negative samples. `rank_i` is the ascending rank of the ith positive sample.

`eval` ( )

Evaluation function :return: the overall value of AUC

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

static **`needs_full_recommendations`** ( )

elliot.evaluation.metrics.accuracy.AUC.gauc module

This is the implementation of the GroupAUC metric. It proceeds from a user-wise computation, and average the AUC values over the users.

class `elliot.evaluation.metrics.accuracy.AUC.gauc.`**`GAUC`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the GroupAUC recommendation metric. Passing 'GAUC' to the metrics list will enable the computation of the metric.

Note:

> It calculates the AUC score of each user, and finally obtains GAUC by weighting the user AUC. It is also not limited to k. Due to our padding for *scores_tensor* in *RankEvaluator* with *-np.inf*, the padding value will influence the ranks of origin items. Therefore, we use descending sort here and make an identity transformation to the formula of *AUC*, which is shown in *auc_* function. For readability, we didn't do simplification in the code.

\mathrm {GAUC} = \frac {{{M} \times {(M+N+1)} - \frac{M \times (M+1)}{2}} - \sum\limits_{i=1}^M rank_{i}} {{M} \times {N}}

`M` is the number of positive samples. `N` is the number of negative samples. `rank_i` is the descending rank of the ith positive sample.

`eval` ( )

Evaluation function :return: the overall averaged value of AUC

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of AUC per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

static **`needs_full_recommendations`** ( )

elliot.evaluation.metrics.accuracy.AUC.lauc module

This is the implementation of the Limited AUC metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.AUC.lauc.`**`LAUC`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Limited AUC recommendation metric. Passing 'LAUC' to the metrics list will enable the computation of the metric.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of LAUC per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.DSC package

Submodules

elliot.evaluation.metrics.accuracy.DSC.dsc module

This is the implementation of the Sørensen–Dice coefficient metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.DSC.dsc.`**`DSC`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Sørensen–Dice coefficient recommendation metric. Passing 'DSC' to the metrics list will enable the computation of the metric.

. math::

mathrm {F1@K} = frac{1+beta^{2}}{frac{1}{text { precision@k }}+frac{beta^{2}}{text { recall@k }}}

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of Sørensen–Dice coefficient per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.f1 package

Submodules

elliot.evaluation.metrics.accuracy.f1.extended_f1 module

This is the implementation of the F-score metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.f1.extended_f1.`**`ExtendedF1`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the F-score recommendation metric. Passing 'ExtendedF1' to the metrics list will enable the computation of the metric.

`eval_user_metric` ( )

`get` ( )

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

`process` ( )

Evaluation function :return: the overall value of Bias Disparity

elliot.evaluation.metrics.accuracy.f1.f1 module

This is the implementation of the F-score metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.f1.f1.`**`F1`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the F-score recommendation metric. Passing 'F1' to the metrics list will enable the computation of the metric.

\mathrm {F1@K} = \frac{1+\beta^{2}}{\frac{1}{\text { precision@k }}+\frac{\beta^{2}}{\text { recall@k }}}

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of F-score

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.hit_rate package

Submodules

elliot.evaluation.metrics.accuracy.hit_rate.hit_rate module

This is the implementation of the Hit Rate metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.hit_rate.hit_rate.`**`HR`** (*recommendations: Dict[int, List[Tuple[int, float]]], config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Hit Rate recommendation metric. Passing 'HR' to the metrics list will enable the computation of the metric.

\mathrm {HR@K} =\frac{Number \space of \space Hits @K}{|GT|}

`HR` is the number of users with a positive sample in the recommendation list. `GT` is the total number of samples in the test set.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of Hit Rate per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.map package

Submodules

elliot.evaluation.metrics.accuracy.map.map module

This is the implementation of the Mean Average Precision metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.map.map.`**`MAP`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Mean Average Precision recommendation metric. Passing 'MAP' to the metrics list will enable the computation of the metric.

Note:

In this case the normalization factor used is `\frac{1}{\min (m,N)}`, which prevents your AP score from being unfairly suppressed when your number of recommendations couldn't possibly capture all the correct ones.

\begin{align*} \mathrm{AP@N} &= \frac{1}{\mathrm{min}(m,N)}\sum_{k=1}^N P(k) \cdot rel(k) \\ \mathrm{MAP@N}& = \frac{1}{|U|}\sum_{u=1}^{|U|}(\mathrm{AP@N})_u \end{align*}

eval_user_metric ( )

Evaluation function :return: the overall averaged value of Mean Average Precision per user

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.mar package

Submodules

elliot.evaluation.metrics.accuracy.mar.mar module

This is the implementation of the Mean Average Recall metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.mar.mar.`**MAR** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Mean Average Recall recommendation metric. Passing 'MAR' to the metrics list will enable the computation of the metric.

\begin{align*} \mathrm{Recall@N} &= \frac{1}{\mathrm{min}(m,|rel(k)|)}\sum_{k=1}^N P(k) \cdot rel(k) \\ \mathrm{MAR@N}& = \frac{1}{|U|}\sum_{u=1}^{|U|}(\mathrm{Recall@N})_u \end{align*}

eval_user_metric ( )

Evaluation function :return: the overall averaged value of Mean Average Recall per user

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.mrr package

Submodules

elliot.evaluation.metrics.accuracy.mrr.mrr module

This is the implementation of the Mean Reciprocal Rank metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.mrr.mrr.`**MRR** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Mean Reciprocal Rank recommendation metric. Passing 'MRR' to the metrics list will enable the computation of the metric.

\mathrm {MRR} = \frac{1}{|{U}|} \sum_{i=1}^{|{U}|} \frac{1}{rank_i}

`U` is the number of users, `rank_i` is the rank of the first item in the recommendation list in the test set results for user `i`.

eval_user_metric ( )

Evaluation function :return: the overall averaged value of Mean Reciprocal Rank per user

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.accuracy.ndcg package

Submodules

elliot.evaluation.metrics.accuracy.ndcg.ndcg module

This is the implementation of the normalized Discounted Cumulative Gain metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.ndcg.ndcg.`**`NDCG`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the nDCG recommendation metric. Passing 'nDCG' to the metrics list will enable the computation of the metric.

\begin{gather} \mathrm {DCG@K}=\sum_{i=1}^{K} \frac{2^{rel_i}-1}{\log_{2}{(i+1)}}\\ \mathrm {IDCG@K}=\sum_{i=1}^{K}\frac{1}{\log_{2}{(i+1)}}\\ \mathrm {NDCG_u@K}=\frac{DCG_u@K}{IDCG_u@K}\\ \mathrm {NDCG@K}=\frac{\sum \nolimits_{u \in u^{te}NDCG_u@K}}{|u^{te}|} \end{gather}

`K` stands for recommending `K` items. And the `rel_i` is the relevance of the item in position `i` in the recommendation list. `2^{rel_i}` equals to 1 if the item hits otherwise 0. `U^{te}` is for all users in the test set.

`compute_idcg` (*user*, *cutoff: int*) → float

Method to compute Ideal Discounted Cumulative Gain :param gain_map: :param cutoff: :return:

`compute_user_ndcg` (*user_recommendations: List*, *user*, *cutoff: int*) → float

Method to compute normalized Discounted Cumulative Gain :param sorted_item_predictions: :param gain_map: :param cutoff: :return:

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of normalized Discounted Cumulative Gain per user

static `name` ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

This is the nDCG metric module.

This module contains and expose the recommendation metric.

elliot.evaluation.metrics.accuracy.precision package

Submodules

elliot.evaluation.metrics.accuracy.precision.precision module

This is the implementation of the Precision metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.precision.precision.`**`Precision`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Precision recommendation metric. Passing 'Precision' to the metrics list will enable the computation of the metric.

\mathrm {Precision@K} = \frac{|Rel_u \cap Rec_u|}{Rec_u}

`Rel_u` is the set of items relevant to user `U`, `Rec_u` is the top K items recommended to users. We obtain the result by calculating the average `Precision@K` of each user.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of Precision

static `name` ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

This is the Precision metric module.

This module contains and expose the recommendation metric.

elliot.evaluation.metrics.accuracy.recall package

Submodules

elliot.evaluation.metrics.accuracy.recall.recall module

This is the implementation of the Recall metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.accuracy.recall.recall.`**`Recall`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Recall recommendation metric. Passing 'Recall' to the metrics list will enable the computation of the metric.

. _Recall: https://en.wikipedia.org/wiki/Precision_and_recall#Recall

\mathrm {Recall@K} = \frac{|Rel_u\cap Rec_u|}{Rel_u}

`Rel_u` is the set of items relevant to user `U`, `Rec_u` is the top K items recommended to users. We obtain the result by calculating the average `Recall@K` of each user.

`eval_user_metric` ( )

Evaluation Function :return: the overall averaged value of Recall per user

static name ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

This is the Recall metric implementation.

This module contains and expose the recommendation metric.

Module contents

elliot.evaluation.metrics.bias package

Subpackages

elliot.evaluation.metrics.bias.aclt package

Submodules

elliot.evaluation.metrics.bias.aclt.aclt module

This is the implementation of the Average coverage of long tail items metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.aclt.aclt.`**`ACLT`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Average coverage of long tail items recommendation metric. Passing 'ACLT' to the metrics list will enable the computation of the metric.

Himan Abdollahpouri, Robin Burke, Bamshad Mobasher Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference, 2019

\mathrm {ACLT}=\frac{1}{\left|U_{t}\right|} \sum_{u \in U_{f}} \sum_{i \in L_{u}} 1(i \in \Gamma)

`U_{t}` is the number of users in the test set. `L_{u}` is the recommended list of items for user u. `1(i \in \Gamma)` is an indicator function and it equals to 1 when i is in Gamma.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of ACLT

static name ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.bias.aplt package

Submodules

elliot.evaluation.metrics.bias.aplt.aplt module

This is the implementation of the Average percentage of long tail items metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.aplt.aplt.`**`APLT`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Average percentage of long tail items recommendation metric. Passing 'APLT' to the metrics list will enable the computation of the metric.

Abdollahpouri, H.; Burke, R.; and Mobasher Proceedings of the Eleventh ACM Conference on Recommender Systems, 2017

\mathrm {ACLT}=\frac{1}{\left|U_{t}\right|} \sum_{u \in U_{t}} \frac{|\{i, i \in(L(u) \cap \sim \Phi)\}|}{|L(u)|}

`U_{t}` is the number of users in the test set. `L_{u}` is the recommended list of items for user u. `\sim \Phi` medium-tail items.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of APLT

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.bias.arp package

Submodules

elliot.evaluation.metrics.bias.arp.arp module

This is the implementation of the Average Recommendation Popularity metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.arp.arp.`**`ARP`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Average Recommendation Popularity recommendation metric. Passing 'ARP' to the metrics list will enable the computation of the metric.

Yin, H.; Cui, B.; Li, J.; Yao, J.; and Chen, C. 2012. Challenging the long tail recommendation. Proceedings of the VLDB Endowment

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of ARP

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.bias.pop_reo package

Submodules

elliot.evaluation.metrics.bias.pop_reo.extended_pop_reo module

This is the implementation of the Popularity-based Ranking-based Equal Opportunity (REO) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.pop_reo.extended_pop_reo.`**`ExtendedPopREO`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Popularity-based Ranking-based Equal Opportunity (REO) recommendation metric. Passing 'ExtendedPopREO' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Jianling Wang, and James Caverlee. "Measuring and Mitigating Item Under-Recommendation Bias in Personalized Ranking Systems." Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.

eval ( )

Evaluation function :return: the overall averaged value of PopREO

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.bias.pop_reo.pop_reo module

This is the implementation of the Popularity-based Ranking-based Equal Opportunity (REO) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.pop_reo.pop_reo.`**`PopREO`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Popularity-based Ranking-based Equal Opportunity (REO) recommendation metric. Passing 'PopREO' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Jianling Wang, and James Caverlee. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.

\mathrm {REO}=\frac{{std}\left(P\left(R @ k \mid g=g_{1}, y=1\right) \ldots P\left(R(a) k=g_{A}, y=1\right)\right)} {{mean}\left(P\left(R @ k \mid g=g_{1}, y=1\right) \ldots P\left(R @ k \mid g=g_{A}, y=1\right)\right)}

`P\left(R @ k \mid g=g_{a}, y=1\right)` is *frac{sum_{u=1}^{N} sum_{i=1}^{k} G_{g_{a}}left(R_{u, i}right) Yleft(u, R_{u, i}right)} {sum_{u=1}^{N} sum_{i in I backslash I_{u}^{+}} G_{g_{a}}(i) Y(u, i)}*

`Y\left(u, R_{u, i}\right)` identifies the ground-truth label of a user-item pair *left(u, R_{u, i}right)*, if item *R_{u, i}* is liked by user ?, returns 1, otherwise 0

`\sum_{i=1}^{k} G_{g_{a}}\left(R_{u, i}\right) Y\left(u, R_{u, i}\right)` counts how many items in test set from group *{g_a}* are ranked in top-? for user u

`\sum_{i \in I \backslash I_{u}^{+}} G_{g_{a}}(i) Y(u, i)` counts the total number of items from group *{g_a}* ? in test set for user u

eval ( )

Evaluation function :return: the overall averaged value of PopREO

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.bias.pop_rsp package

Submodules

elliot.evaluation.metrics.bias.pop_rsp.extended_pop_rsp module

This is the implementation of the Popularity-based Ranking-based Statistical Parity (RSP) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.pop_rsp.extended_pop_rsp.`**`ExtendedPopRSP`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Popularity-based Ranking-based Statistical Parity (RSP) recommendation metric. Passing 'ExtendedPopRSP' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Jianling Wang, and James Caverlee. "Measuring and Mitigating Item Under-Recommendation Bias in Personalized Ranking Systems." Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.

eval ( )

Evaluation function :return: the overall averaged value of PopRSP

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.bias.pop_rsp.pop_rsp module

This is the implementation of the Popularity-based Ranking-based Statistical Parity (RSP) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.bias.pop_rsp.pop_rsp.`**`PopRSP`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Popularity-based Ranking-based Statistical Parity (RSP) recommendation metric. Passing 'PopRSP' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Jianling Wang, and James Caverlee. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.

```
\mathrm {RSP}=\frac{{std}\left(P\left(R @ k \mid g=g_{1}\right), \ldots,
P\left(R @ k \mid g=g_{A}\right)\right)} {{mean}\left(P\left(R @ k \mid
g=g_{1}\right), \ldots, P\left(R @ k \mid g=g_{A}\right)\right)}
```

`P\left(R @ k \mid g=g_{a}\right)}` is *frac{sum_{u=1}^{N} sum_{i=1}^{k} G_{g_{a}}left-(R_{u, i}right)} {sum_{u=1}^{N} sum_{i in I backslash I_{u}^{+}} G_{g_{a}}(i)}*

`\sum_{i=1}^{k} G_{g_{a}}\left(R_{u, i}\right)` calculates how many un-interacted items from group *{g_a}* are ranked in top-? for user u.

`\sum_{i \in I \backslash I_{u}^{+}} G_{g_{a}}(i)` calculates how many un-interacted items belong to group *{g_a}* for u

eval ( )

Evaluation function :return: the overall averaged value of PopRSP

static **name** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

Module contents

elliot.evaluation.metrics.coverage package

Subpackages

elliot.evaluation.metrics.coverage.item_coverage package

Submodules

elliot.evaluation.metrics.coverage.item_coverage.item_coverage module

This is the implementation of the Item Coverage metric. It directly proceeds from a system-wise computation, and it considers all the users at the same time.

class `elliot.evaluation.metrics.coverage.item_coverage.item_coverage.`**`ItemCoverage`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Item Coverage recommendation metric. Passing 'ItemCoverage' to the metrics list will enable the computation of the metric.

Ricci F, Rokach L, Shapira B, Kantor P. 2015

Note:

The simplest measure of catalog coverage is the percentage of all items that can ever be recommended. This measure can be computed in many cases directly given the algorithm and the input data set.

`eval` ( )
Evaluation function :return: the overall averaged value of Item Coverage

static `name` ( )
Metric Name Getter :return: returns the public name of the metric

Module contents

This is the Item Coverage metric module.

This module contains and expose the recommendation metric.

elliot.evaluation.metrics.coverage.num_retrieved package

Submodules

elliot.evaluation.metrics.coverage.num_retrieved.num_retrieved module

This is the implementation of the NumRetrieved metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.coverage.num_retrieved.num_retrieved.`**`NumRetrieved`** (*recommendations*, *config*, *params*, *eval_objects*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the NumRetrieved recommendation metric. Passing 'NumRetrieved' to the metrics list will enable the computation of the metric.

`eval_user_metric` ( )
Evaluation function :return: the overall averaged value of NumRetrieved

static `name` ( )
Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.coverage.user_coverage package

Submodules

elliot.evaluation.metrics.coverage.user_coverage.user_coverage module

This is the implementation of the User Coverage metric. It directly proceeds from a system-wise computation, and it considers all the users at the same time.

class `elliot.evaluation.metrics.coverage.user_coverage.user_coverage.`**`UserCoverage`** (*recommendations*, *config*, *params*, *eval_objects*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the User Coverage recommendation metric. Passing 'UserCoverage' to the metrics list will enable the computation of the metric.

Ricci F, Rokach L, Shapira B, Kantor P. 2015

Note:
The proportion of users or user interactions for which the system can recommend items. In many applications the recommender may not provide recommendations for some users due to, e.g. low confidence in the accuracy of predictions for that user.

`eval` ( )
Evaluation function :return: the overall averaged value of User Coverage

static `name` ( )
Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.coverage.user_coverage.user_coverage_at_n module

This is the implementation of the User Coverage metric. It directly proceeds from a system-wise computation, and it considers all the users at the same time.

class `elliot.evaluation.metrics.coverage.user_coverage.user_coverage_at_n.`**`UserCoverageAtN`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the User Coverage recommendation metric. Passing 'UserCoverageAtN' to the metrics list will enable the computation of the metric.

`eval` ( )

Evaluation function :return: the overall averaged value of User Coverage

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

Module contents

elliot.evaluation.metrics.diversity package

Subpackages

elliot.evaluation.metrics.diversity.SRecall package

Submodules

elliot.evaluation.metrics.diversity.SRecall.srecall module

This is the implementation of the SRecall metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.diversity.SRecall.srecall.`**`SRecall`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the SRecall recommendation metric. Passing 'SRecall' to the metrics list will enable the computation of the metric.

3. 24. Zhai, W. W. Cohen, and J. Lafferty, 2003

\mathrm {SRecall}=\frac{\left | \cup_{i=1}^{K} {subtopics}\left(d_{i}\right)\right | }{n_{A}}

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of SRecall

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.diversity.gini_index package

Submodules

elliot.evaluation.metrics.diversity.gini_index.gini_index module

This is the implementation of the Gini Index metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.diversity.gini_index.gini_index.`**`GiniIndex`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Gini Index recommendation metric. Passing 'Gini' to the metrics list will enable the computation of the metric.

Ricci F, Rokach L, Shapira B, Kantor P. 2015

\mathrm {GiniIndex}=\frac{1}{n-1} \sum_{j=1}^{n}(2 j-n-1) p\left(i_{j}\right)

`i_{j}` is the list of items ordered according to increasing *p(i)*

eval ( )

Evaluation function :return: the overall averaged value of Gini Index

static name ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.diversity.shannon_entropy package

Submodules

elliot.evaluation.metrics.diversity.shannon_entropy.shannon_entropy module

This is the implementation of the Shannon Entropy metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.diversity.shannon_entropy.shannon_entropy.`**`ShannonEntropy`** (*recommendations*, *config*, *params*, *eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Shannon Entropy recommendation metric. Passing 'SEntropy' to the metrics list will enable the computation of the metric.

Ricci F, Rokach L, Shapira B, Kantor P. 2015

\mathrm {ShannonEntropy}=-\sum_{i=1}^{n} p(i) \log p(i)

eval ( )

Evaluation function :return: the overall value of Shannon Entropy

static name ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

Module contents

elliot.evaluation.metrics.fairness package

Subpackages

elliot.evaluation.metrics.fairness.BiasDisparity package

Submodules

elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBD module

This is the implementation of the Bias Disparity metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBD.`**`BiasDisparityBD`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Bias Disparity recommendation metric. Passing 'BiasDisparityBD' to the metrics list will enable the computation of the metric.

Tsintzou, Virginia, Evaggelia Pitoura, and Panayiotis Tsaparas. Proceedings of the Workshop on Recommendation in Multi-stakeholder Environments co-located with the 13th {ACM} Conference on Recommender Systems (RecSys 2019)

\mathrm {BD(G, C)}=\frac{B_{R}(G, C)-B_{S}(G, C)}{B_{S}(G, C)}

eval ( )

get ( )

name ( )

Metric Name Getter :return: returns the public name of the metric

process ( )
Evaluation function :return: the overall value of Bias Disparity

elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBR module

This is the implementation of the Bias Disparity - Bias Recommendations metric. It proceeds from a user-wise computation, and average the values over the users.

class
`elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBR.`**`BiasDisparityBR`**
(*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Bias Disparity - Bias Recommendations recommendation metric. Passing 'BiasDisparityBR' to the metrics list will enable the computation of the metric.

Tsintzou, Virginia, Evaggelia Pitoura, and Panayiotis Tsaparas. Proceedings of the Workshop on Recommendation in Multi-stakeholder Environments co-located with the 13th {ACM} Conference on Recommender Systems (RecSys 2019)

```
\mathrm {BD(G, C)}=
```

rac{B_{R}(G, C)-B_{S}(G, C)}{B_{S}(G, C)}

eval ( )

get ( )

get_BR ( )

name ( )
Metric Name Getter :return: returns the public name of the metric

process ( )
Evaluation function :return: the overall value of Bias Disparity - Bias Recommendations

elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBS module

This is the implementation of the Bias Disparity - Bias Source metric. It proceeds from a user-wise computation, and average the values over the users.

class
`elliot.evaluation.metrics.fairness.BiasDisparity.BiasDisparityBS.`**`BiasDisparityBS`**
(*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Bias Disparity - Bias Source recommendation metric. Passing 'BiasDisparityBS' to the metrics list will enable the computation of the metric.

Tsintzou, Virginia, Evaggelia Pitoura, and Panayiotis Tsaparas. Proceedings of the Workshop on Recommendation in Multi-stakeholder Environments co-located with the 13th {ACM} Conference on Recommender Systems (RecSys 2019)

```
\mathrm {B_{S}(G, C)}=
```

rac{P R_{S}(G, C)}{P(C)}

eval ( )

get ( )

get_BS ( )

name ( )
Metric Name Getter :return: returns the public name of the metric

process ( )
Evaluation function :return: the overall value of Bias Disparity - Bias Source

Module contents

elliot.evaluation.metrics.fairness.MAD package

Submodules

elliot.evaluation.metrics.fairness.MAD.ItemMADranking module

This is the implementation of the Item MAD ranking metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.MAD.ItemMADranking.`**`ItemMADranking`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Item MAD ranking recommendation metric. Passing 'ItemMADranking' to the metrics list will enable the computation of the metric.

> Deldjoo, Yashar, Vito Walter Anelli, Hamed Zamani, Alejandro Bellogin, and Tommaso Di Noia. User Modeling and User-Adapted Interaction (2020): 1-47.
>
> ```
> \mathrm {MAD}={avg}_{i, j}({MAD}(R^{(i)}, R^{(j)}))
> ```

Math {MAD}={avg}_{i, j}({MAD}left(R^{(i)}, R^{(j)}))

eval ( )
Evaluation function :return: the overall averaged value of Item MAD ranking

get ( )

name ( )
Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.fairness.MAD.ItemMADrating module

This is the implementation of the Item MAD rating metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.MAD.ItemMADrating.`**`ItemMADrating`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Item MAD rating recommendation metric. Passing 'ItemMADrating' to the metrics list will enable the computation of the metric.
Zhu, Ziwei, Xia Hu, and James Caverlee. Proceedings of the 27th ACM International Conference on Information and Knowledge Management. 2018.
\mathrm {MAD}={avg}_{i, j}({MAD}(R^{(i)}, R^{(j)}))

Math {MAD}={avg}_{i, j}({MAD}left(R^{(i)}, R^{(j)}))

eval ( )
Evaluation function :return: the overall averaged value of Item MAD rating

get ( )

name ( )
Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.fairness.MAD.UserMADranking module

This is the implementation of the User MAD ranking metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.MAD.UserMADranking.`**`UserMADranking`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the User MAD ranking recommendation metric. Passing 'UserMADranking' to the metrics list will enable the computation of the metric.

```
\mathrm {MAD}={avg}_{i, j}({MAD}(R^{(i)}, R^{(j)}))
```

Math {MAD}={avg}_{i, j}({MAD}left(R^{(i)}, R^{(j)}))

`compute_idcg` (*user: int*, *cutoff: int*) → float

Method to compute Ideal Discounted Cumulative Gain :param gain_map: :param cutoff: :return:

`compute_user_ndcg` (*user_recommendations: List*, *user: int*, *cutoff: int*) → float

Method to compute normalized Discounted Cumulative Gain :param sorted_item_predictions: :param gain_map: :param cutoff: :return:

`eval` ( )

Evaluation function :return: the overall averaged value of User MAD ranking

`get` ( )

`name` ( )

Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.fairness.MAD.UserMADrating module

This is the implementation of the User MAD rating metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.MAD.UserMADrating.`**`UserMADrating`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the User MAD rating recommendation metric. Passing 'UserMADrating' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Xia Hu, and James Caverlee. Proceedings of the 27th ACM International Conference on Information and Knowledge Management. 2018.

\mathrm {MAD}={avg}_{i, j}({MAD}(R^{(i)}, R^{(j)}))

Math {MAD}={avg}_{i, j}({MAD}left(R^{(i)}, R^{(j)}))

`eval` ( )

Evaluation function :return: the overall averaged value of User MAD rating

`get` ( )

`name` ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

This is the Precision metric module.

This module contains and expose the recommendation metric.

elliot.evaluation.metrics.fairness.reo package

Submodules

elliot.evaluation.metrics.fairness.reo.reo module

This is the implementation of the Ranking-based Equal Opportunity (REO) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.reo.reo.`**`REO`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Ranking-based Equal Opportunity (REO) recommendation metric. Passing 'REO' to the metrics list will enable the computation of the metric.

Zhu, Ziwei, Jianling Wang, and James Caverlee. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.

\mathrm {REO}=\frac{{std}\left(P\left(R @ k \mid g=g_{1}, y=1\right) \ldots P\left(R(a) k=g_{A}, y=1\right)\right)} {{mean}\left(P\left(R @ k \mid g=g_{1}, y=1\right) \ldots P\left(R @ k \mid g=g_{A}, y=1\right)\right)}

`P\left(R @ k \mid g=g_{a}, y=1\right)` is *frac{sum_{u=1}^{N} sum_{i=1}^{k} G_{g_{a}}left(R_{u, i}right) Yleft(u, R_{u, i}right)} {sum_{u=1}^{N} sum_{i in I backslash I_{u}^{+}} G_{g_{a}}(i) Y(u, i)}*

`Y\left(u, R_{u, i}\right)` identifies the ground-truth label of a user-item pair *left(u, R_{u, i}right)*, if item *R_{u, i}* is liked by user ?, returns 1, otherwise 0

`\sum_{i=1}^{k} G_{g_{a}}\left(R_{u, i}\right) Y\left(u, R_{u, i}\right)` counts how many items in test set from group *{g_a}* are ranked in top-? for user u

`\sum_{i \in I \backslash I_{u}^{+}} G_{g_{a}}(i) Y(u, i)` counts the total number of items from group *{g_a}* ? in test set for user u

`eval` ( )

`get` ( )

`name` ( )

Metric Name Getter :return: returns the public name of the metric

`process` ( )

Evaluation function :return: the overall value of Ranking-based Equal Opportunity (REO)

Module contents

elliot.evaluation.metrics.fairness.rsp package

Submodules

elliot.evaluation.metrics.fairness.rsp.rsp module

This is the implementation of the Ranking-based Statistical Parity (RSP) metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.fairness.rsp.rsp.`**`RSP`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

> This class represents the implementation of the Ranking-based Statistical Parity (RSP) recommendation metric. Passing 'RSP' to the metrics list will enable the computation of the metric.
>
> Zhu, Ziwei, Jianling Wang, and James Caverlee. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020.
>
> `\mathrm {RSP}=`

rac{{std}(P(R @ k mid g=g_{1}), ldots, P(R @ k mid g=g_{A}))}

{{mean}(P(R @ k mid g=g_{1}), ldots, P(R @ k mid g=g_{A}))}

`P(R @ k \mid g=g_{a})` is`

rac{sum_{u=1}^{N} sum_{i=1}^{k} G_{g_{a}}(R_{u, i})}

{sum_{u=1}^{N} sum_{i in I ?ackslash I_{u}^{+}} G_{g_{a}}(i)}`

`\sum_{i=1}^{k} G_{g_{a}}(R_{u, i})` calculates how many un-interacted items from group *{g_a}* are ranked in top-? for user u.

`\sum_{i \in I ?ackslash I_{u}^{+}} G_{g_{a}}(i)` calculates how many un-interacted items belong to group *{g_a}* for u

eval ( )

get ( )

name ( )
Metric Name Getter :return: returns the public name of the metric

process ( )
Evaluation function :return: the overall value of Ranking-based Statistical Parity (RSP)

Module contents

Module contents

elliot.evaluation.metrics.novelty package

Subpackages

elliot.evaluation.metrics.novelty.EFD package

Submodules

elliot.evaluation.metrics.novelty.EFD.efd module

This is the implementation of the Expected Free Discovery metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.novelty.EFD.efd.`**`EFD`** (*recommendations*, *config*, *params*, *eval_objects*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Expected Free Discovery recommendation metric. Passing 'EFD' to the metrics list will enable the computation of the metric.

Note:
EFD can be read as the expected ICF of seen recommended items

```
\mathrm {EFD}=C \sum_{i_{k} \in R} {disc}(k) p({rel} \mid i_{k}, u)(
-\log _{2} p(i \mid {seen},        heta))
```

eval_user_metric ( )
Evaluation function :return: the overall averaged value of Expected Free Discovery per user

static name ( )
Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.novelty.EFD.extended_efd module

This is the implementation of the Expected Free Discovery metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.novelty.EFD.extended_efd.`**`ExtendedEFD`** (*recommendations*, *config*, *params*, *eval_objects*, *additional_data*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`
This class represents the implementation of the Expected Free Discovery recommendation metric. Passing 'ExtendedEFD' to the metrics list will enable the computation of the metric.

eval_user_metric ( )
Evaluation function :return: the overall averaged value of Expected Free Discovery per user

static name ( )
Metric Name Getter :return: returns the public name of the metric

Module contents

elliot.evaluation.metrics.novelty.EPC package

Submodules

elliot.evaluation.metrics.novelty.EPC.epc module

This is the implementation of the Expected Popularity Complement metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.novelty.EPC.epc.`**`EPC`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Expected Popularity Complement recommendation metric. Passing 'EPC' to the metrics list will enable the computation of the metric.

> S. Vargas and P. Castells Proceedings of RecSys 2011
>
> **Note:**
>
> EPC can be read as the expected number of seen relevant recommended items not previously seen

```
\mathrm {EPC}=C \sum_{i_{k} \in R} {disc}(k) p\left({rel} \mid i_{k},
u\right)\left(1-p\left(\{seen} \mid i_{k}\right)\right)
```

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of Expected Popularity Complement per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

elliot.evaluation.metrics.novelty.EPC.extended_epc module

This is the implementation of the Expected Popularity Complement metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.novelty.EPC.extended_epc.`**`ExtendedEPC`** (*recommendations, config, params, eval_objects, additional_data*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Expected Popularity Complement recommendation metric. Passing 'ExtendedEPC' to the metrics list will enable the computation of the metric.

`eval_user_metric` ( )

Evaluation function :return: the overall averaged value of Expected Popularity Complement per user

static **`name`** ( )

Metric Name Getter :return: returns the public name of the metric

Module contents

Module contents

elliot.evaluation.metrics.rating package

Subpackages

elliot.evaluation.metrics.rating.mae package

Submodules

elliot.evaluation.metrics.rating.mae.mae module

This is the implementation of the Mean Absolute Error metric. It proceeds from a system-wise computation.

class `elliot.evaluation.metrics.rating.mae.mae.`**`MAE`** (*recommendations, config, params, eval_objects*)

Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Mean Absolute Error recommendation metric. Passing 'MAE' to the metrics list will enable the computation of the metric.

\mathrm{MAE}=\frac{1}{|{T}|} \sum_{(u, i) \in {T}}\left|\hat{r}_{u i}-r_{u i}\right|

`T` is the test set, `\hat{r}_{u i}` is the score predicted by the model, and `r_{u i}` the actual score of the test set.

`eval` ( )
Evaluation function :return: the overall averaged value of Mean Absolute Error

`eval_user_metric` ( )
Evaluation function :return: the overall averaged value of Mean Absolute Error per user

static `name` ( )
Metric Name Getter :return: returns the public name of the metric

static `needs_full_recommendations` ( )

Module contents

elliot.evaluation.metrics.rating.mse package

Submodules

elliot.evaluation.metrics.rating.mse.mse module

This is the implementation of the Mean Squared Error metric. It proceeds from a system-wise computation.

class `elliot.evaluation.metrics.rating.mse.mse.`**`MSE`** (*recommendations*, *config*, *params*, *eval_objects*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

This class represents the implementation of the Mean Squared Error recommendation metric. Passing ‘MSE’ to the metrics list will enable the computation of the metric.

`\mathrm{MSE} =`

rac{1}{|{T}|} sum_{(u, i) in {T}}(hat{r}_{u i}-r_{u i})^{2}

`T` is the test set, `\hat{r}_{u i}` is the score predicted by the model, and `r_{u i}` the actual score of the test set.

`eval` ( )
Evaluation function :return: the overall averaged value of Mean Squared Error

`eval_user_metric` ( )
Evaluation function :return: the overall averaged value of Mean Squared Error per user

static `name` ( )
Metric Name Getter :return: returns the public name of the metric

static `needs_full_recommendations` ( )

Module contents

elliot.evaluation.metrics.rating.rmse package

Submodules

elliot.evaluation.metrics.rating.rmse.rmse module

This is the implementation of the Root Mean Squared Error metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.rating.rmse.rmse.`**`RMSE`** (*recommendations*, *config*, *params*, *eval_objects*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`
This class represents the implementation of the Root Mean Squared Error recommendation metric. Passing ‘RMSE’ to the metrics list will enable the computation of the metric.
\mathrm{RMSE} = \sqrt{\frac{1}{|{T}|} \sum_{(u, i) \in {T}}(\hat{r}_{u i}-r_{u i})^{2}}
`T` is the test set, `\hat{r}_{u i}` is the score predicted by the model, and `r_{u i}` the actual

score of the test set.

eval ( )
Evaluation function :return: the overall averaged value of Root Mean Squared Error

eval_user_metric ( )
Evaluation function :return: the overall averaged value of Root Mean Squared Error

static **name** ( )
Metric Name Getter :return: returns the public name of the metric

static **needs_full_recommendations** ( )

Module contents

Module contents

Submodules

elliot.evaluation.metrics.base_metric module

This is the implementation of the Precision metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.base_metric.`**BaseMetric** (*recommendations*, *config*, *params*, *evaluation_objects*, *additional_data=None*)
Bases: `abc.ABC`
This class represents the implementation of the Precision recommendation metric. Passing 'Precision' to the metrics list will enable the computation of the metric.

eval ( )

get ( )

abstract **name** ( )

static **needs_full_recommendations** ( )

elliot.evaluation.metrics.metrics_utils module

class `elliot.evaluation.metrics.metrics_utils.`**ProxyMetric** (*name='ProxyMetric'*, *val=0*, *needs_full_recommendations=False*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

eval ( )

name ( )

needs_full_recommendations ( )

class `elliot.evaluation.metrics.metrics_utils.`**ProxyStatisticalMetric** (*name='ProxyMetric'*, *val=0*, *user_val=0*, *needs_full_recommendations=False*)
Bases: `elliot.evaluation.metrics.base_metric.BaseMetric`

eval ( )

eval_user_metric ( )

name ( )

needs_full_recommendations ( )

elliot.evaluation.metrics.statistical_array_metric module

This is the implementation of the Precision metric. It proceeds from a user-wise computation, and average the values over the users.

class `elliot.evaluation.metrics.statistical_array_metric.`**`StatisticalMetric`**

Bases: `object`

This class represents the implementation of the Precision recommendation metric. Passing 'Precision' to the metrics list will enable the computation of the metric.

abstract **`eval_user_metric`** ( )

Module contents

This is the metrics' module.

This module contains and expose the recommendation metrics. Each metric is encapsulated in a specific package.

See the implementation of Precision metric for creating new per-user metrics. See the implementation of Item Coverage for creating new cross-user metrics.

`elliot.evaluation.metrics.`**`parse_metric`** (*metric*)

`elliot.evaluation.metrics.`**`parse_metrics`** (*metrics*)

elliot.evaluation.popularity_utils package

Submodules

elliot.evaluation.popularity_utils.popularity module

Module description: This module provides a popularity class based on number of users who have experienced an item (user-item repetitions in the dataset are counted once)

class `elliot.evaluation.popularity_utils.popularity.`**`Popularity`** (*data, pop_ratio=0.8*)

Bases: `object`

`get_custom_pop_obj` (*pop_ratio=0.8*)

`get_long_tail` ( )

`get_pop_items` ( )

`get_short_head` ( )

`get_sorted_pop_items` ( )

Module contents

elliot.evaluation.relevance package

Submodules

elliot.evaluation.relevance.relevance module

Module description:

class `elliot.evaluation.relevance.relevance.`**`AbstractRelevanceSingleton`**

Bases: `abc.ABC`

abstract **`get_rel`** (*user, item*)

static **`logarithmic_ranking_discount`** (*k: int*) → float

Method to compute logarithmic discount :param k: :return:

class `elliot.evaluation.relevance.relevance.`**`BinaryRelevance`** (*test, rel_threshold*)

Bases: `elliot.evaluation.relevance.relevance.AbstractRelevanceSingleton`

`get_rel` (*user, item*)

`get_user_rel` (*user*)

`get_user_rel_gains` (*user*)

class `elliot.evaluation.relevance.relevance.`**`DiscountedRelevance`** (*test*, *rel_threshold*)

Bases: `elliot.evaluation.relevance.relevance.AbstractRelevanceSingleton`

`get_rel` (*user*, *item*)

`get_user_rel` (*user*)

`get_user_rel_gains` (*user*)

class `elliot.evaluation.relevance.relevance.`**`Relevance`** (*test*, *rel_threshold*)

Bases: `object`

property **`binary_relevance`**

property **`discounted_relevance`**

`get_test` ( )

Module contents

Module description:

Submodules

elliot.evaluation.evaluator module

Module description:

class `elliot.evaluation.evaluator.`**`Evaluator`** (*data: elliot.dataset.dataset.DataSet*, *params: types.SimpleNamespace*)

Bases: `object`

`eval` (*recommendations*)

Runtime Evaluation of Accuracy Performance (top-k) :return:

`eval_at_k` (*recommendations*, *k*)

`get_needed_recommendations` ( )

elliot.evaluation.statistical_significance module

Module description:

class `elliot.evaluation.statistical_significance.`**`PairedTTest`**

Bases: `object`

static **`common_users`** (*arr_0: Dict[int, float]*, *arr_1: Dict[int, float]*)

static **`compare`** (*arr_0: Dict[int, float]*, *arr_1: Dict[int, float]*, *users: List[int]*)

class `elliot.evaluation.statistical_significance.`**`WilcoxonTest`**

Bases: `object`

static **`common_users`** (*arr_0: Dict[int, float]*, *arr_1: Dict[int, float]*)

static **`compare`** (*arr_0: Dict[int, float]*, *arr_1: Dict[int, float]*, *users: List[int]*)

Module contents

Module description:

1.1.3 elliot.hyperoptimization package

Submodules

elliot.hyperoptimization.model_coordinator module

Module description:

class `elliot.hyperoptimization.model_coordinator.`**`ModelCoordinator`** (*data_objs*, *base: types.SimpleNamespace*, *params*, *model_class: ClassVar*)

Bases: `object`

This class handles the selection of hyperparameters for the hyperparameter tuning realized with HyperOpt.

`objective` (*args*)

This function respect the signature, and the return format required for HyperOpt optimization :param args: a Dictionary that contains the new hyper-parameter values that will be used in the current run :return: it returns a Dictionary with loss, and status being required by HyperOpt, and params, and results being required by the framework

`single` ( )

This function respect the signature, and the return format required for HyperOpt optimization :param args: a Dictionary that contains the new hyper-parameter values that will be used in the current run :return: it returns a Dictionary with loss, and status being required by HyperOpt, and params, and results being required by the framework

Module contents

Module description:

`elliot.hyperoptimization.`**`parse_algorithms`** (*opt_alg*)

`elliot.hyperoptimization.`**`suggest`** (*new_ids*, *domain*, *trials*, *seed*, *nbMaxSucessiveFailures=1000*)

1.1.4 elliot.namespace package

Submodules

elliot.namespace.namespace_model module

Module description:

class `elliot.namespace.namespace_model.`**`NameSpaceModel`** (*config_path*, *base_folder_path_elliot*, *base_folder_path_config*)

Bases: `object`

`fill_base` ( )

`fill_model` ( )

elliot.namespace.namespace_model_builder module

Module description:

class `elliot.namespace.namespace_model_builder.`**`Builder`**

Bases: `abc.ABC`

The Builder interface specifies methods for creating the different parts of the Product objects.

abstract property **`base`**

abstract **`models`** ( ) → None

class `elliot.namespace.namespace_model_builder.`**`NameSpaceBuilder`** (*config_path*, *base_folder_path_elliot*, *base_folder_path_config*)

Bases: `elliot.namespace.namespace_model_builder.Builder`

property **`base`**

`models` ( ) → tuple

Module contents

Module description:

1.1.5 elliot.prefiltering package

Submodules

elliot.prefiltering.standard_prefilters module

class `elliot.prefiltering.standard_prefilters.`**`PreFilter`**

Bases: `object`

static **`filter`** (*d: pandas.core.frame.DataFrame*, *ns: types.SimpleNamespace*) → pandas.core.frame.DataFrame

static **`filter_items_by_popularity`** (*d: pandas.core.frame.DataFrame*, *threshold*) → pandas.core.frame.DataFrame

static **`filter_iterative_k_core`** (*d: pandas.core.frame.DataFrame*, *threshold*) → pandas.core.frame.DataFrame

static **`filter_ratings_by_global_average`** (*d: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

static **`filter_ratings_by_threshold`** (*d: pandas.core.frame.DataFrame*, *threshold*) → pandas.core.frame.DataFrame

static **`filter_ratings_by_user_average`** (*d: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

static **`filter_retain_cold_users`** (*d: pandas.core.frame.DataFrame*, *threshold*) → pandas.core.frame.DataFrame

static **`filter_rounds_k_core`** (*d: pandas.core.frame.DataFrame*, *threshold*, *n_rounds*) → pandas.core.frame.DataFrame

static **`filter_users_by_profile_size`** (*d: pandas.core.frame.DataFrame*, *threshold*) → pandas.core.frame.DataFrame

Module contents

Module description:

1.1.6 elliot.recommender package

Subpackages

elliot.recommender.NN package

Subpackages

elliot.recommender.NN.attribute_item_knn package

Submodules

elliot.recommender.NN.attribute_item_knn.attribute_item_knn module

Module description:

class `elliot.recommender.NN.attribute_item_knn.attribute_item_knn.`**`AttributeItemKNN`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`build_feature_sparse` ( )

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.NN.attribute_item_knn.attribute_item_knn_similarity module

class `elliot.recommender.NN.attribute_item_knn.attribute_item_knn_similarity.`**`Similarity`** (*data, attribute_matrix, num_neighbors, similarity*)

Bases: `object`

Simple kNN class

`compute_cosine` (*i_index, j_index*)

`compute_neighbors` ( )

`get_item_neighbors` (*item*)

`get_model_state` ( )

`get_transactions` ( )

`get_user_recs` (*u, k*)

`initialize` ( )

This function initialize the data model

`process_similarity` (*similarity*)

static `score_item` (*neighs, user_items*)

`set_model_state` (*saving_dict*)

Module contents

elliot.recommender.NN.attribute_user_knn package

Submodules

elliot.recommender.NN.attribute_user_knn.attribute_user_knn module

Module description:

class `elliot.recommender.NN.attribute_user_knn.attribute_user_knn.`**`AttributeUserKNN`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`build_feature_sparse` ( )

`build_feature_sparse_values` ( )

`compute_binary_profile` (*user_items_dict: Dict*)

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.NN.attribute_user_knn.attribute_user_knn_similarity module

class `elliot.recommender.NN.attribute_user_knn.attribute_user_knn_similarity.`**`Similarity`** (*data, attribute_matrix, num_neighbors, similarity*)

Bases: `object`

Simple kNN class

`compute_neighbors` ( )

`get_model_state` ( )

`get_transactions` ( )

`get_user_neighbors` (*item*)

`get_user_recs` (*u, k*)

`initialize` ( )

This function initialize the data model

`process_similarity` (*similarity*)

static `score_item` (*neighs, user_neighs_items*)

`set_model_state` (*saving_dict*)

elliot.recommender.NN.attribute_user_knn.tfidf_utils module

class `elliot.recommender.NN.attribute_user_knn.tfidf_utils.`**`TFIDF`** (*map: Dict[int, List[int]]*)

Bases: `object`

`get_profiles` (*ratings: Dict[int, Dict[int, float]]*)

`tfidf` ( )

Module contents

elliot.recommender.NN.item_knn package

Submodules

elliot.recommender.NN.item_knn.aiolli_ferrari module

Created on 23/10/17 @author: Maurizio Ferrari Dacrema

class `elliot.recommender.NN.item_knn.aiolli_ferrari.`**`AiolliSimilarity`** (*data, maxk=40, shrink=100, similarity='cosine', normalize=True*)

Bases: `object`

`get_user_recs` (*user, k=100*)

`initialize` ( )

`predict` (*u, i*)

class `elliot.recommender.NN.item_knn.aiolli_ferrari.`**`Compute_Similarity`** (*dataMatrix, topK=100, shrink=0, normalize=True, asymmetric_alpha=0.5, tversky_alpha=1.0, tversky_beta=1.0, similarity='cosine', row_weights=None*)

Bases: `object`

`applyAdjustedCosine` ( )

Remove from every data point the average for the corresponding row :return:

`applyPearsonCorrelation` ( )

Remove from every data point the average for the corresponding column :return:

`compute_similarity` (*start_col=None, end_col=None, block_size=100*)

Compute the similarity for the given dataset :param self: :param start_col: column to begin with :param end_col: column to stop before, end_col is excluded :return:

`useOnlyBooleanInteractions` ( )

`elliot.recommender.NN.item_knn.aiolli_ferrari.`**`check_matrix`** (*X, format='csc', dtype=<class 'numpy.float32'>*)

This function takes a matrix as input and transforms it into the specified format. The matrix in input can be either sparse or ndarray. If the matrix in input has already the desired format, it is returned as-is the dtype parameter is always applied and the default is np.float32 :param X: :param format: :param dtype: :return:

elliot.recommender.NN.item_knn.item_knn module

Module description:

class `elliot.recommender.NN.item_knn.item_knn.`**`ItemKNN`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.NN.item_knn.item_knn_similarity module

class `elliot.recommender.NN.item_knn.item_knn_similarity.`**`Similarity`** (*data, num_neighbors, similarity*)

Bases: `object`

Simple kNN class

`compute_cosine` (*i_index, j_index*)

`compute_neighbors` ( )

`get_item_neighbors` (*item*)

`get_model_state` ( )

`get_transactions` ( )

`get_user_recs` (*u, k*)

`initialize` ( )
This function initialize the data model

`process_cosine` ( )

`process_similarity` (*similarity*)

static **`score_item`** (*neighs, user_items*)

`set_model_state` (*saving_dict*)

Module contents

elliot.recommender.NN.user_knn package

Submodules

elliot.recommender.NN.user_knn.aiolli_ferrari module

Created on 23/10/17 @author: Maurizio Ferrari Dacrema

class `elliot.recommender.NN.user_knn.aiolli_ferrari.`**`AiolliSimilarity`** (*data, maxk=40, shrink=100, similarity='cosine', normalize=True*)
Bases: `object`

`get_user_recs` (*user, k=100*)

`initialize` ( )

`predict` (*u, i*)

class `elliot.recommender.NN.user_knn.aiolli_ferrari.`**`Compute_Similarity`** (*dataMatrix, topK=100, shrink=0, normalize=True, asymmetric_alpha=0.5, tversky_alpha=1.0, tversky_beta=1.0, similarity='cosine', row_weights=None*)
Bases: `object`

`applyAdjustedCosine` ( )
Remove from every data point the average for the corresponding row :return:

`applyPearsonCorrelation` ( )
Remove from every data point the average for the corresponding column :return:

`compute_similarity` (*start_col=None, end_col=None, block_size=100*)
Compute the similarity for the given dataset :param self: :param start_col: column to begin with :param end_col: column to stop before, end_col is excluded :return:

`useOnlyBooleanInteractions` ( )

`elliot.recommender.NN.user_knn.aiolli_ferrari.`**`check_matrix`** (*X, format='csc', dtype=<class 'numpy.float32'>*)
This function takes a matrix as input and transforms it into the specified format. The matrix in

input can be either sparse or ndarray. If the matrix in input has already the desired format, it is returned as-is the dtype parameter is always applied and the default is np.float32 :param X: :param format: :param dtype: :return:

elliot.recommender.NN.user_knn.user_knn module

Module description:

class `elliot.recommender.NN.user_knn.user_knn.`**`UserKNN`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.NN.user_knn.user_knn_similarity module

class `elliot.recommender.NN.user_knn.user_knn_similarity.`**`Similarity`** (*data, num_neighbors, similarity*)

Bases: `object`

Simple kNN class

`compute_neighbors` ( )

`get_model_state` ( )

`get_transactions` ( )

`get_user_neighbors` (*item*)

`get_user_recs` (*u, k*)

`initialize` ( )

This function initialize the data model

`process_similarity` (*similarity*)

static `score_item` (*neighs, user_neighs_items*)

`set_model_state` (*saving_dict*)

Module contents

Module contents

elliot.recommender.adversarial package

Subpackages

elliot.recommender.adversarial.AMF package

Submodules

elliot.recommender.adversarial.AMF.AMF module

Module description:

class `elliot.recommender.adversarial.AMF.AMF.`**`AMF`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.adversarial.AMF.AMF_model module

Module description:

class `elliot.recommender.adversarial.AMF.AMF_model.`**`AMF_model`** (**args*, ***kwargs*)
Bases: `tensorflow.python.keras.engine.training.Model`

`build_perturbation` (*batch*)
Evaluate Adversarial Perturbation with FGSM-like Approach

`call` (*inputs*, *training=None*)
Calls the model on new inputs.
In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be
either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`get_config` ( )
Returns the config of the layer.
A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.
The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_positions` (*predictions*, *train_mask*, *items*, *inner_test_user_true_mask*)

`get_top_k` (*predictions*, *train_mask*, *k=100*)

`predict` (*start*, *stop*, ***kwargs*)
Generates output predictions for the input samples.
Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using *__call__* is recommended for faster execution, e.g., *model(x)*, or *model(x, training=False)* if you have layers such as *tf.keras.layers.BatchNormalization* that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments:

x: Input samples. It could be:

- A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).
- A *tf.data* dataset.

- A generator or *keras.utils.Sequence* instance.

A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the *Unpacking behavior for iterator-like inputs* section of *Model.fit*.

batch_size: Integer or *None*.
Number of samples per batch. If unspecified, *batch_size* will default to 32. Do not specify the *batch_size* if your data is in the form of dataset, generators, or *keras.utils.Sequence* instances (since they generate batches).

verbose: Verbosity mode, 0 or 1. steps: Total number of steps (batches of samples)

before declaring the prediction round finished. Ignored with the default value of *None*. If x is a *tf.data* dataset and *steps* is None, *predict* will run until the input dataset is exhausted.

callbacks: List of *keras.callbacks.Callback* instances.
List of callbacks to apply during prediction. See [callbacks](/api_docs/python/tf/keras/callbacks).

max_queue_size: Integer. Used for generator or *keras.utils.Sequence*
input only. Maximum size for the generator queue. If unspecified, *max_queue_size* will default to 10.

workers: Integer. Used for generator or *keras.utils.Sequence* input
only. Maximum number of processes to spin up when using process-based threading. If unspecified, *workers* will default to 1. If 0, will execute the generator on the main thread.

use_multiprocessing: Boolean. Used for generator or
keras.utils.Sequence input only. If *True*, use process-based threading. If unspecified, *use_multiprocessing* will default to *False*. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of *Unpacking behavior for iterator-like inputs* for *Model.fit*. Note that Model.predict uses the same interpretation rules as *Model.fit* and *Model.evaluate*, so inputs must be unambiguous for all three methods.

Returns:
Numpy array(s) of predictions.

Raises:
RuntimeError: If *model.predict* is wrapped in *tf.function*. ValueError: In case of mismatch between the provided

input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`train_step` (*batch*, *user_adv_train=False*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:
data: A nested structure of `Tensor`s.

Returns:
A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s

metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

Module contents

Module description:

elliot.recommender.adversarial.AMR package

Submodules

elliot.recommender.adversarial.AMR.AMR module

Module description:

class `elliot.recommender.adversarial.AMR.AMR.`**`AMR`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.adversarial.AMR.AMR_model module

Module description:

class `elliot.recommender.adversarial.AMR.AMR_model.`**`AMR_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`build_perturbation` (*batch*)

Evaluate Adversarial Perturbation with FGSM-like Approach

`call` (*inputs, training=None, mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*start, stop*)

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using *__call__* is recommended for faster execution, e.g., *model(x)*, or *model(x, training=False)* if you have layers such as *tf.keras.layers.BatchNormalization* that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments:

x: Input samples. It could be:

- A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).
- A *tf.data* dataset.
- A generator or *keras.utils.Sequence* instance.

A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the *Unpacking behavior for iterator-like inputs* section of *Model.fit*.

batch_size: Integer or *None*.
Number of samples per batch. If unspecified, *batch_size* will default to 32. Do not specify the *batch_size* if your data is in the form of dataset, generators, or *keras.utils.Sequence* instances (since they generate batches).

verbose: Verbosity mode, 0 or 1. steps: Total number of steps (batches of samples)

before declaring the prediction round finished. Ignored with the default value of *None*. If x is a *tf.data* dataset and *steps* is None, *predict* will run until the input dataset is exhausted.

callbacks: List of *keras.callbacks.Callback* instances.
List of callbacks to apply during prediction. See [callbacks](/api_docs/python/tf/keras/callbacks).

max_queue_size: Integer. Used for generator or *keras.utils.Sequence*
input only. Maximum size for the generator queue. If unspecified, *max_queue_size* will default to 10.

workers: Integer. Used for generator or *keras.utils.Sequence* input
only. Maximum number of processes to spin up when using process-based threading. If unspecified, *workers* will default to 1. If 0, will execute the generator on the main thread.

use_multiprocessing: Boolean. Used for generator or
keras.utils.Sequence input only. If *True*, use process-based threading. If unspecified, *use_multiprocessing* will default to *False*. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of *Unpacking behavior for iterator-like inputs* for *Model.fit*. Note that Model.predict uses the same interpretation rules as *Model.fit* and *Model.evaluate*, so inputs must be unambiguous for all three methods.

Returns:
Numpy array(s) of predictions.

Raises:
RuntimeError: If *model.predict* is wrapped in *tf.function*. ValueError: In case of mismatch between the provided

input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`train_step` (*batch*, *user_adv_train=False*)
The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

Module contents

Module description:

Module contents

elliot.recommender.algebric package

Subpackages

elliot.recommender.algebric.slope_one package

Submodules

elliot.recommender.algebric.slope_one.slope_one module

Module description: Lemire, Daniel, and Anna Maclachlan. "Slope one predictors for online rating-based collaborative filtering." Proceedings of the 2005 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics

class `elliot.recommender.algebric.slope_one.slope_one.`**`SlopeOne`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.algebric.slope_one.slope_one_model module

Lemire, Daniel, and Anna Maclachlan. "Slope one predictors for online rating-based collaborative filtering." Proceedings of the 2005 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics

class `elliot.recommender.algebric.slope_one.slope_one_model.`**`SlopeOneModel`** (*data*)

Bases: `object`

`get_model_state` ( )

`get_user_recs` (*u*, *k*)

`initialize` ( )

`predict` (*user*, *item*)

`set_model_state` (*saving_dict*)

Module contents

Module contents

elliot.recommender.attentive package

Subpackages

elliot.recommender.attentive.afm package

Submodules

elliot.recommender.attentive.afm.afm module

Module description:

class `elliot.recommender.attentive.afm.afm.`**`AFM`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

property name

`train` ( )

elliot.recommender.attentive.afm.afm_model module

Module description:

class `elliot.recommender.attentive.afm.afm_model.`**`AFMModel`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

Module contents

Module description:

elliot.recommender.autoencoders package

Subpackages

elliot.recommender.autoencoders.dae package

Submodules

elliot.recommender.autoencoders.dae.multi_dae module

Module description:

class `elliot.recommender.autoencoders.dae.multi_dae.`**`MultiDAE`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

property name

`train` ( )

elliot.recommender.autoencoders.dae.multi_dae_model module

Module description:

class `elliot.recommender.autoencoders.dae.multi_dae_model.`**`Decoder`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Converts z, the encoded vector, back into a uaser interaction vector.

`call` (*inputs*, ***kwargs*)

class `elliot.recommender.autoencoders.dae.multi_dae_model.`**`DenoisingAutoEncoder`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

Combines the encoder and decoder into an end-to-end model for training.

`call` (*inputs*, *training=None*, ***kwargs*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

class `elliot.recommender.autoencoders.dae.multi_dae_model.`**`Encoder`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Maps user-item interactions to a triplet (z_mean, z_log_var, z).

`call` (*inputs*, *training=None*)

Module contents

Module description:

elliot.recommender.autoencoders.vae package

Submodules

elliot.recommender.autoencoders.vae.multi_vae module

Module description:

class `elliot.recommender.autoencoders.vae.multi_vae.`**`MultiVAE`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

property **`name`**

train ( )

elliot.recommender.autoencoders.vae.multi_vae_model module

Module description:

class `elliot.recommender.autoencoders.vae.multi_vae_model.`**`Decoder`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Converts z, the encoded digit vector, back into a readable digit.

call (*inputs, **kwargs*)

class `elliot.recommender.autoencoders.vae.multi_vae_model.`**`Encoder`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Maps MNIST digits to a triplet (z_mean, z_log_var, z).

call (*inputs, training=None*)

class `elliot.recommender.autoencoders.vae.multi_vae_model.`**`Sampling`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Uses (z_mean, z_log_var) to sample z, the vector encoding a digit.

call (*inputs*)

class `elliot.recommender.autoencoders.vae.multi_vae_model.`**`VariationalAutoEncoder`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

Combines the encoder and decoder into an end-to-end model for training.

call (*inputs, training=None, **kwargs*)

get_config ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

get_top_k (*preds, train_mask, k=100*)

predict (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

train_step (*batch, anneal_ph=0.0, **kwargs*)

Module contents

Module description:

Module contents

Module description:

elliot.recommender.content_based package

Subpackages

elliot.recommender.content_based.VSM package

Submodules

elliot.recommender.content_based.VSM.tfidf_utils module

class `elliot.recommender.content_based.VSM.tfidf_utils.`**`TFIDF`** (*map: Dict[int, List[int]]*)

Bases: `object`

`get_profiles` (*ratings: Dict[int, Dict[int, float]]*)

`tfidf` ( )

elliot.recommender.content_based.VSM.vector_space_model module

Module description:

class `elliot.recommender.content_based.VSM.vector_space_model.`**`VSM`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`build_feature_sparse` (*feature_dict, num_entities*)

`build_feature_sparse_values` (*feature_dict, num_entities*)

`compute_binary_profile` (*user_items_dict: Dict*)

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.content_based.VSM.vector_space_model_similarity module

class `elliot.recommender.content_based.VSM.vector_space_model_similarity.`**`Similarity`** (*data, user_profile_matrix, item_attribute_matrix, similarity*)

Bases: `object`

Simple kNN class

`get_model_state` ( )

`get_transactions` ( )

`get_user_recs` (*u, k*)

`initialize` ( )

This function initialize the data model

`process_similarity` (*similarity*)

`set_model_state` (*saving_dict*)

Module contents

Module contents

elliot.recommender.gan package

Subpackages

elliot.recommender.gan.CFGAN package

Submodules

elliot.recommender.gan.CFGAN.cfgan module

Module description:

class `elliot.recommender.gan.CFGAN.cfgan.`**`CFGAN`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.gan.CFGAN.cfgan_model module

Module description:

class `elliot.recommender.gan.CFGAN.cfgan_model.`**`CFGAN_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*predictions, train_mask, k=100*)

`predict` (*start, stop, **kwargs*)

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using *__call__* is recommended for faster execution, e.g., *model(x)*, or *model(x, training=False)* if you have layers such as *tf.keras.layers.BatchNormalization* that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments:

x: Input samples. It could be:

- A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).
- A *tf.data* dataset.
- A generator or *keras.utils.Sequence* instance.

A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the *Unpacking behavior for iterator-like inputs* section of *Model.fit*.

batch_size: Integer or *None*.
Number of samples per batch. If unspecified, *batch_size* will default to 32. Do not specify the *batch_size* if your data is in the form of dataset, generators, or *keras.utils.Sequence* instances (since they generate batches).

verbose: Verbosity mode, 0 or 1. steps: Total number of steps (batches of samples)

before declaring the prediction round finished. Ignored with the default value of *None*. If x is a *tf.data* dataset and *steps* is None, *predict* will run until the input dataset is exhausted.

callbacks: List of *keras.callbacks.Callback* instances.
List of callbacks to apply during prediction. See [callbacks](/api_docs/python/tf/keras/callbacks).

max_queue_size: Integer. Used for generator or *keras.utils.Sequence*
input only. Maximum size for the generator queue. If unspecified, *max_queue_size* will default to 10.

workers: Integer. Used for generator or *keras.utils.Sequence* input
only. Maximum number of processes to spin up when using process-based threading. If unspecified, *workers* will default to 1. If 0, will execute the generator on the main thread.

use_multiprocessing: Boolean. Used for generator or
keras.utils.Sequence input only. If *True*, use process-based threading. If unspecified, *use_multiprocessing* will default to *False*. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of *Unpacking behavior for iterator-like inputs* for *Model.fit*. Note that Model.predict uses the same interpretation rules as *Model.fit* and *Model.evaluate*, so inputs must be unambiguous for all three methods.

Returns:
Numpy array(s) of predictions.

Raises:
RuntimeError: If *model.predict* is wrapped in *tf.function*. ValueError: In case of mismatch between the provided

input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`train_step` (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:
data: A nested structure of `Tensor`s.

Returns:
A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

class `elliot.recommender.gan.CFGAN.cfgan_model.`**`Discriminator`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`discriminate_fake_data` (*X*)

train_step (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

class `elliot.recommender.gan.CFGAN.cfgan_model.`**Generator** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

generate_fake_data (*mask*, *C_u*)

infer (*C_u*)

train_step (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

Module contents

elliot.recommender.gan.IRGAN package

Submodules

elliot.recommender.gan.IRGAN.irgan module

Module description:

class `elliot.recommender.gan.IRGAN.irgan.`**IRGAN** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

train ( )

elliot.recommender.gan.IRGAN.irgan_model module

Module description:

class `elliot.recommender.gan.IRGAN.irgan_model.`**`Discriminator`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`train_step` (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

class `elliot.recommender.gan.IRGAN.irgan_model.`**`Generator`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`train_step` (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

`train_step_with_reward` (*batch*)

class `elliot.recommender.gan.IRGAN.irgan_model.`**`IRGAN_model`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_positions` (*predictions*, *train_mask*, *items*, *inner_test_user_true_mask*)

`get_top_k` (*predictions*, *train_mask*, *k=100*)

`pre_train_discriminator` ( )

`pre_train_generator` ( )

`predict` (*start*, *stop*, ***kwargs*)

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using *__call__* is recommended for faster execution, e.g., *model(x)*, or *model(x, training=False)* if you have layers such as *tf.keras.layers.BatchNormalization* that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments:

x: Input samples. It could be:

- A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).

- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).
- A *tf.data* dataset.
- A generator or *keras.utils.Sequence* instance.

A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the *Unpacking behavior for iterator-like inputs* section of *Model.fit*.

batch_size: Integer or *None*.
Number of samples per batch. If unspecified, *batch_size* will default to 32. Do not specify the *batch_size* if your data is in the form of dataset, generators, or *keras.utils.Sequence* instances (since they generate batches).

verbose: Verbosity mode, 0 or 1. steps: Total number of steps (batches of samples)

before declaring the prediction round finished. Ignored with the default value of *None*. If x is a *tf.data* dataset and *steps* is None, *predict* will run until the input dataset is exhausted.

callbacks: List of *keras.callbacks.Callback* instances.
List of callbacks to apply during prediction. See [callbacks](/api_docs/python/tf/keras/callbacks).

max_queue_size: Integer. Used for generator or *keras.utils.Sequence*
input only. Maximum size for the generator queue. If unspecified, *max_queue_size* will default to 10.

workers: Integer. Used for generator or *keras.utils.Sequence* input
only. Maximum number of processes to spin up when using process-based threading. If unspecified, *workers* will default to 1. If 0, will execute the generator on the main thread.

use_multiprocessing: Boolean. Used for generator or
keras.utils.Sequence input only. If *True*, use process-based threading. If unspecified, *use_multiprocessing* will default to *False*. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of *Unpacking behavior for iterator-like inputs* for *Model.fit*. Note that Model.predict uses the same interpretation rules as *Model.fit* and *Model.evaluate*, so inputs must be unambiguous for all three methods.

Returns:
Numpy array(s) of predictions.

Raises:
RuntimeError: If *model.predict* is wrapped in *tf.function*. ValueError: In case of mismatch between the provided

input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`train_step` ( )

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:
data: A nested structure of `Tensor`s.

Returns:
A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

Module contents

Module contents

elliot.recommender.graph_based package

Subpackages

elliot.recommender.graph_based.lightgcn package

Submodules

elliot.recommender.graph_based.lightgcn.LightGCN module

Module description:

class `elliot.recommender.graph_based.lightgcn.LightGCN.`**`LightGCN`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property **`name`**

`train` ( )

elliot.recommender.graph_based.lightgcn.LightGCN_model module

Module description:

class `elliot.recommender.graph_based.lightgcn.LightGCN_model.`**`LightGCNModel`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, ***kwargs*)

Generates prediction for passed users and items indices

Args:
inputs: user, item (batch) the *Network* in training mode or inference mode.

Returns:
prediction and extracted model parameters

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:
Python dictionary.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*start*, *stop*, ***kwargs*)

`train_step` (*batch*)

Apply a single training step on one batch.

Args:
batch: batch used for the current train step

Returns:
loss value at the current batch

Module contents

elliot.recommender.graph_based.ngcf package

Submodules

elliot.recommender.graph_based.ngcf.NGCF module

Module description:

class `elliot.recommender.graph_based.ngcf.NGCF.`**`NGCF`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.graph_based.ngcf.NGCF_model module

Module description:

class `elliot.recommender.graph_based.ngcf.NGCF_model.`**`NGCFModel`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, ***kwargs*)

Generates prediction for passed users and items indices

Args:
inputs: user, item (batch) the *Network* in training mode or inference mode.

Returns:
prediction and extracted model parameters

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:
Python dictionary.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*start*, *stop*, ***kwargs*)

`train_step` (*batch*)

Apply a single training step on one batch.

Args:
batch: batch used for the current train step

Returns:
loss value at the current batch

Module contents

Module contents

elliot.recommender.knowledge_aware package

Subpackages

elliot.recommender.knowledge_aware.kaHFM package

Submodules

elliot.recommender.knowledge_aware.kaHFM.ka_hfm module

class `elliot.recommender.knowledge_aware.kaHFM.ka_hfm.`**`KaHFM`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int, i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

`restore_weights` ( )

`train` ( )

`train_step` ( )

`update_factors` (*u: int, i: int, j: int*)

class `elliot.recommender.knowledge_aware.kaHFM.ka_hfm.`**`MF`** (*ratings: Dict, map: Dict, tfidf: Dict, user_profiles: Dict, random: Any, *args*)

Bases: `object`

Simple Matrix Factorization class

`get_factors` ( )

`get_item_bias` (*item: int*)

`get_item_factors` (*item: int*)

`get_model_state` ( )

`get_transactions` ( )

`get_user_bias` (*user: int*)

`get_user_factors` (*user: int*)

`get_user_recs` (*user: int, k: int*)

`get_user_recs_argpartition` (*user: int, k: int*)

`initialize` (*loc: float = 0, scale: float = 0.1*)

This function initialize the data model :param loc: :param scale: :return:

property name

`predict` (*user: int, item: int*)

`set_item_bias` (*item: int, v: float*)

set_item_factors (*item: int, v: float*)

set_model_state (*saving_dict*)

set_user_bias (*user: int, v: float*)

set_user_factors (*user: int, v: float*)

elliot.recommender.knowledge_aware.kaHFM.tfidf_utils module

class `elliot.recommender.knowledge_aware.kaHFM.tfidf_utils.`**TFIDF** (*map: Dict[int, List[int]]*)

Bases: `object`

get_profiles (*ratings: Dict[int, Dict[int, float]]*)

tfidf ( )

Module contents

elliot.recommender.knowledge_aware.kaHFM_batch package

Submodules

elliot.recommender.knowledge_aware.kaHFM_batch.kahfm_batch module

Module description:

class `elliot.recommender.knowledge_aware.kaHFM_batch.kahfm_batch.`**KaHFMBatch** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

train ( )

elliot.recommender.knowledge_aware.kaHFM_batch.kahfm_batch_model module

Module description:

class `elliot.recommender.knowledge_aware.kaHFM_batch.kahfm_batch_model.`**KaHFM_model** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, **kwargs*)

get_config ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

get_top_k (*preds, train_mask, k=100*)

predict (*inputs, training=False, **kwargs*)

predict_all ( )

`predict_batch` (*start*, *stop*)

`train_step` (*batch*)

elliot.recommender.knowledge_aware.kaHFM_batch.tfidf_utils module

class `elliot.recommender.knowledge_aware.kaHFM_batch.tfidf_utils.`**`TFIDF`** (*map: Dict[int, List[int]]*)

Bases: `object`

`get_profiles` (*ratings: Dict[int, Dict[int, float]]*)

`tfidf` ( )

Module contents

elliot.recommender.knowledge_aware.kahfm_embeddings package

Submodules

elliot.recommender.knowledge_aware.kahfm_embeddings.kahfm_embeddings module

Module description:

class `elliot.recommender.knowledge_aware.kahfm_embeddings.kahfm_embeddings.`**`KaHFMEmbeddings`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.knowledge_aware.kahfm_embeddings.kahfm_embeddings_model module

Module description:

class `elliot.recommender.knowledge_aware.kahfm_embeddings.kahfm_embeddings_model.`**`KaHFMEmbeddi`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*, ***kwargs*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*inputs*, *training=False*, ***kwargs*)

`predict_batch` (*start*, *stop*)

`train_step` (*batch*)

elliot.recommender.knowledge_aware.kahfm_embeddings.tfidf_utils module

class `elliot.recommender.knowledge_aware.kahfm_embeddings.tfidf_utils.`**`TFIDF`** (*map: Dict[int, List[int]]*)

Bases: `object`

`get_profiles` (*ratings: Dict[int, Dict[int, float]]*)

`tfidf` ( )

Module contents

Module contents

elliot.recommender.latent_factor_models package

Subpackages

elliot.recommender.latent_factor_models.BPRMF package

Submodules

elliot.recommender.latent_factor_models.BPRMF.BPRMF module

Module description:

class `elliot.recommender.latent_factor_models.BPRMF.BPRMF.`**`BPRMF`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int, i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

`restore_weights` ( )

`train` ( )

`train_step` ( )

`update_factors` (*u: int, i: int, j: int*)

class `elliot.recommender.latent_factor_models.BPRMF.BPRMF.`**`MF`** (*F, ratings, random, *args*)

Bases: `object`

Simple Matrix Factorization class

`get_item_bias` (*item: int*)

`get_item_factors` (*item: int*)

`get_model_state` ( )

`get_transactions` ( )

`get_user_bias` (*user: int*)

`get_user_factors` (*user: int*)

`get_user_recs` (*user, k*)

get_user_recs_argpartition (*user: int, k: int*)

initialize (*loc: float = 0, scale: float = 0.1*)

This function initialize the data model :param loc: :param scale: :return:

property name

predict (*user, item*)

set_item_bias (*item: int, v: float*)

set_item_factors (*item: int, v: float*)

set_model_state (*saving_dict*)

set_user_bias (*user: int, v: float*)

set_user_factors (*user: int, v: float*)

Module contents

Module description:

elliot.recommender.latent_factor_models.BPRMF_batch package

Submodules

elliot.recommender.latent_factor_models.BPRMF_batch.BPRMF_batch module

Module description:

class `elliot.recommender.latent_factor_models.BPRMF_batch.BPRMF_batch.`**`BPRMF_batch`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.BPRMF_batch.BPRMF_batch_model module

Module description:

class `elliot.recommender.latent_factor_models.BPRMF_batch.BPRMF_batch_model.`**`BPRMF_batch_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None*)

get_config ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

get_positions (*predictions, train_mask, items, inner_test_user_true_mask*)

get_top_k (*predictions, train_mask, k=100*)

predict (*start, stop, **kwargs*)

train_step (*batch*)

Module contents

Module description:

elliot.recommender.latent_factor_models.BPRSlim package

Submodules

elliot.recommender.latent_factor_models.BPRSlim.bprslim module

Module description:

class `elliot.recommender.latent_factor_models.BPRSlim.bprslim.`**BPRSlim** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.BPRSlim.bprslim_model module

Module description:

class `elliot.recommender.latent_factor_models.BPRSlim.bprslim_model.`**BPRSlimModel** (*data, num_users, num_items, lr, lj_reg, li_reg, sampler, random_seed=42*)

Bases: `object`

get_model_state ( )

get_user_recs (*user, k=100*)

predict (*u, i*)

set_model_state (*saving_dict*)

train_step (*batch*)

Module contents

Module description:

elliot.recommender.latent_factor_models.CML package

Submodules

elliot.recommender.latent_factor_models.CML.CML module

Module description:

class `elliot.recommender.latent_factor_models.CML.CML.`**`CML`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.latent_factor_models.CML.CML_model module

Module description:

class `elliot.recommender.latent_factor_models.CML.CML_model.`**`CML_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_positions` (*predictions, train_mask, items, inner_test_user_true_mask*)

`get_top_k` (*predictions, train_mask, k=100*)

`predict` (*start, stop, **kwargs*)

`train_step` (*batch*)

class `elliot.recommender.latent_factor_models.CML.CML_model.`**`LatentFactor`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.layers.embeddings.Embedding`

`censor` (*censor_id*)

Module contents

Module description:

elliot.recommender.latent_factor_models.FFM package

Submodules

elliot.recommender.latent_factor_models.FFM.field_aware_factorization_machine module

Module description:

class `elliot.recommender.latent_factor_models.FFM.field_aware_factorization_machine.`**`FFM`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.FFM.field_aware_factorization_machine_model module

Module description:

class elliot.recommender.latent_factor_models.FFM.field_aware_factorization_machine_model.**Fie**
(**args, **kwargs*)

Bases: tensorflow.python.keras.engine.training.Model

call (*inputs, training=None, mask=None*)

get_recs (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

get_top_k (*preds, train_mask, k=100*)

predict (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.latent_factor_models.FISM package

Submodules

elliot.recommender.latent_factor_models.FISM.FISM module

Module description:

class elliot.recommender.latent_factor_models.FISM.FISM.**FISM** (*data, config, params, *args, **kwargs*)

Bases: elliot.recommender.recommender_utils_mixin.RecMixin, elliot.recommender.base_recommender_model.BaseRecommenderModel

get_recommendations (*k: int = 100, auc_compute: bool = False*)

property name

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.FISM.FISM_model module

Module description:

class elliot.recommender.latent_factor_models.FISM.FISM_model.**FISM_model** (**args, **kwargs*)

Bases: tensorflow.python.keras.engine.training.Model

batch_predict (user_start, user_stop, **kwargs)

call (*inputs, training=None*)

create_history_item_matrix ( )

get_config ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:
Python dictionary.

get_positions (*predictions, train_mask, items, inner_test_user_true_mask*)

get_top_k (*predictions, train_mask, k=100*)

predict (user, **kwargs)

train_step (*batch*)

class elliot.recommender.latent_factor_models.FISM.FISM_model.**LatentFactor** (*args, **kwargs)

Bases: `tensorflow.python.keras.layers.embeddings.Embedding`

censor (*censor_id*)

Module contents

Module description:

elliot.recommender.latent_factor_models.FM package

Submodules

elliot.recommender.latent_factor_models.FM.factorization_machine module

Module description:

class elliot.recommender.latent_factor_models.FM.factorization_machine.**FM** (*data, config, params*, *args, **kwargs)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.FM.factorization_machine_model module

Module description:

class
elliot.recommender.latent_factor_models.FM.factorization_machine_model.**FactorizationMac**
(*args, **kwargs)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`train_step` (*batch*)

Module contents

elliot.recommender.latent_factor_models.FunkSVD package

Submodules

elliot.recommender.latent_factor_models.FunkSVD.funk_svd module

Module description:

class `elliot.recommender.latent_factor_models.FunkSVD.funk_svd.`**`FunkSVD`** (*data, config, params, *args, **kwargs*)
Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int, i: int*)

`restore_weights` ( )

`train` ( )

elliot.recommender.latent_factor_models.FunkSVD.funk_svd_model module

Module description:

class `elliot.recommender.latent_factor_models.FunkSVD.funk_svd_model.`**`FunkSVDModel`** (**args, **kwargs*)
Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.latent_factor_models.LogisticMF package

Submodules

elliot.recommender.latent_factor_models.LogisticMF.logistic_matrix_factorization module

Module description:

class `elliot.recommender.latent_factor_models.LogisticMF.logistic_matrix_factorization.`**`Logist`**
(*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.LogisticMF.logistic_matrix_factorization_model module

Module description:

class `elliot.recommender.latent_factor_models.LogisticMF.logistic_matrix_factorization_model.`
(**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, mask=None*)

get_top_k (*preds, train_mask, k=100*)

predict_batch (*start, stop, **kwargs*)

set_update_user (*update_user*)

train_step (*batch*)

Module contents

elliot.recommender.latent_factor_models.MF package

Submodules

elliot.recommender.latent_factor_models.MF.matrix_factorization module

Module description:

class `elliot.recommender.latent_factor_models.MF.matrix_factorization.`**`MF`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.MF.matrix_factorization_model module

Module description:

class
elliot.recommender.latent_factor_models.MF.matrix_factorization_model.**MatrixFactorizati**
(**args*, ***kwargs*)

Bases: tensorflow.python.keras.engine.training.Model

call (*inputs*, *training=None*, *mask=None*)

get_recs (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

get_top_k (*preds*, *train_mask*, *k=100*)

predict (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.latent_factor_models.NonNegMF package

Submodules

elliot.recommender.latent_factor_models.NonNegMF.non_negative_matrix_factorization module

Module description:

class
elliot.recommender.latent_factor_models.NonNegMF.non_negative_matrix_factorization.**NonN**
(*data*, *config*, *params*, **args*, ***kwargs*)

Bases: elliot.recommender.recommender_utils_mixin.RecMixin, elliot.recommender.base_recommender_model.BaseRecommenderModel

get_recommendations (*k: int = 100*)

property name

predict (*u: int*, *i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.NonNegMF.non_negative_matrix_factorization_model module

Module description:

class
`elliot.recommender.latent_factor_models.NonNegMF.non_negative_matrix_factorization_mode`
(*data, num_users, num_items, global_mean, embed_mf_size, lambda_weights, learning_rate=0.01, random_seed=42*)

Bases: `object`

`get_model_state` ( )

`get_user_recs` (*user, k=100*)

`predict` (*user, item*)

`set_model_state` (*saving_dict*)

`train_step` ( )

Module contents

Module description:

elliot.recommender.latent_factor_models.PMF package

Submodules

elliot.recommender.latent_factor_models.PMF.probabilistic_matrix_factorization module

Module description:

Mnih, Andriy, and Russ R. Salakhutdinov. “Probabilistic matrix factorization.” Advances in neural information processing systems 20 (2007)

class
`elliot.recommender.latent_factor_models.PMF.probabilistic_matrix_factorization.`**`PMF`**
(*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int, i: int*)

`restore_weights` ( )

`train` ( )

elliot.recommender.latent_factor_models.PMF.probabilistic_matrix_factorization_model module

Module description:

Mnih, Andriy, and Russ R. Salakhutdinov. “Probabilistic matrix factorization.” Advances in neural information processing systems 20 (2007)

class
`elliot.recommender.latent_factor_models.PMF.probabilistic_matrix_factorization_model.`**`Pr`**
(**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`dot_prod` (*layer_0, layer_1*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

get_top_k (*preds*, *train_mask*, *k=100*)

predict (*inputs*, *training=False*, ***kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.latent_factor_models.PureSVD package

Submodules

elliot.recommender.latent_factor_models.PureSVD.pure_svd module

Module description:

class elliot.recommender.latent_factor_models.PureSVD.pure_svd.**PureSVD** (*data*, *config*, *params*, **args*, ***kwargs*)
Bases: elliot.recommender.recommender_utils_mixin.RecMixin, elliot.recommender.base_recommender_model.BaseRecommenderModel

get_recommendations (*k: int = 100*)

property name

predict (*u: int*, *i: int*)
Get prediction on the user item pair.

Returns:
A single float vaue.

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.PureSVD.pure_svd_model module

Module description:

class
elliot.recommender.latent_factor_models.PureSVD.pure_svd_model.**PureSVDModel** (*factors*, *data*, *random_seed*)
Bases: object
Simple Matrix Factorization class

get_model_state ( )

get_user_recs (*user*, *k=100*)

predict (*user*, *item*)

set_model_state (*saving_dict*)

train_step ( )

Module contents

elliot.recommender.latent_factor_models.SVDpp package

Submodules

elliot.recommender.latent_factor_models.SVDpp.svdpp module

Module description:

class `elliot.recommender.latent_factor_models.SVDpp.svdpp.`**`SVDpp`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int*, *i: int*)

`train` ( )

elliot.recommender.latent_factor_models.SVDpp.svdpp_model module

Module description:

class `elliot.recommender.latent_factor_models.SVDpp.svdpp_model.`**`SVDppModel`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*, *mask=None*)

`get_recs` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

elliot.recommender.latent_factor_models.Slim package

Submodules

elliot.recommender.latent_factor_models.Slim.slim module

Module description:

class `elliot.recommender.latent_factor_models.Slim.slim.`**`Slim`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int*, *i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.Slim.slim_model module

Module description:

class `elliot.recommender.latent_factor_models.Slim.slim_model.`**SlimModel** (*data, num_users, num_items, l1_ratio, alpha, epochs*)

Bases: `object`

get_model_state ( )

get_user_recs (*user, k=100*)

predict (*u, i*)

set_model_state (*saving_dict*)

train (*verbose*)

Module contents

Module description:

elliot.recommender.latent_factor_models.WRMF package

Submodules

elliot.recommender.latent_factor_models.WRMF.wrmf module

Module description:

class `elliot.recommender.latent_factor_models.WRMF.wrmf.`**WRMF** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

predict (*u: int, i: int*)

Get prediction on the user item pair.

Returns:

A single float vaue.

restore_weights ( )

train ( )

elliot.recommender.latent_factor_models.WRMF.wrmf_model module

Module description:

class `elliot.recommender.latent_factor_models.WRMF.wrmf_model.`**WRMFModel** (*factors, data, random, alpha, reg*)

Bases: `object`

Simple Matrix Factorization class

get_model_state ( )

get_user_recs (*user, k=100*)

predict (*user, item*)

`set_model_state` (*saving_dict*)

`train_step` ( )

Module contents

Module contents

Module description:

elliot.recommender.neural package

Subpackages

elliot.recommender.neural.ConvMF package

Submodules

elliot.recommender.neural.ConvMF.convolutional_matrix_factorization module

Module description:

class `elliot.recommender.neural.ConvMF.convolutional_matrix_factorization.`**`ConvMF`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.neural.ConvMF.convolutional_matrix_factorization_model module

Module description:

class `elliot.recommender.neural.ConvMF.convolutional_matrix_factorization_model.`**`ConvMatrixFac`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=False*, ***kwargs*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`get_recs` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:

data: A nested structure of `Tensor`s.

Returns:

A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

class `elliot.recommender.neural.ConvMF.convolutional_matrix_factorization_model.`**`Convolutional`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, ***kwargs*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

class `elliot.recommender.neural.ConvMF.convolutional_matrix_factorization_model.`**`MLPComponent`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=False*, ***kwargs*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:
A tensor if there is a single output, or a list of tensors if there are more than one outputs.

Module contents

elliot.recommender.neural.ConvNeuMF package

Submodules

elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization module

Module description:

class
`elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization.`**`ConvNeuMF`**
(*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`restore_weights` ( )

`train` ( )

elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization_model module

Module description:

class
`elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization_model.`**`Con`**
(**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=False*, ***kwargs*)

`get_recs` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict` (*inputs*, *training=False*, ***kwargs*)

Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`train_step` (*batch*)

class
`elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization_model.`**`Con`**
(**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, ***kwargs*)

class
`elliot.recommender.neural.ConvNeuMF.convolutional_neural_matrix_factorization_model.`**`MLP`**
(**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=False, **kwargs*)

Module contents

elliot.recommender.neural.DMF package

Submodules

elliot.recommender.neural.DMF.deep_matrix_factorization module

Module description:

class `elliot.recommender.neural.DMF.deep_matrix_factorization.`**DMF** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

train ( )

elliot.recommender.neural.DMF.deep_matrix_factorization_model module

Module description:

class `elliot.recommender.neural.DMF.deep_matrix_factorization_model.`**DeepMatrixFactorizationMo** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, mask=None*)

cosine (*layer_0, layer_1*)

dot_prod (*layer_0, layer_1*)

get_recs (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

get_top_k (*preds, train_mask, k=100*)

predict (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.neural.DeepFM package

Submodules

elliot.recommender.neural.DeepFM.deep_fm module

Module description:

class `elliot.recommender.neural.DeepFM.deep_fm.`**DeepFM** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property **name**

predict (*u: int, i: int*)

restore_weights ( )

train ( )

elliot.recommender.neural.DeepFM.deep_fm_model module

Module description:

class `elliot.recommender.neural.DeepFM.deep_fm_model.`**DeepFMModel** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, mask=None*)

get_recs (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

get_top_k (*preds, train_mask, k=100*)

predict (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

train_step (*batch*)

Module contents

elliot.recommender.neural.GeneralizedMF package

Submodules

elliot.recommender.neural.GeneralizedMF.generalized_matrix_factorization module

Module description:

class
`elliot.recommender.neural.GeneralizedMF.generalized_matrix_factorization.`**GMF**
(*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property **name**

predict (*u: int, i: int*)

train ( )

elliot.recommender.neural.GeneralizedMF.generalized_matrix_factorization_model module

Module description:

class
`elliot.recommender.neural.GeneralizedMF.generalized_matrix_factorization_model.`**Generali**
(**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, mask=None*)

get_recs (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

get_top_k (*preds, train_mask, k=100*)

train_step (*batch*)

Module contents

elliot.recommender.neural.ItemAutoRec package

Submodules

elliot.recommender.neural.ItemAutoRec.itemautorec module

Module description:

class `elliot.recommender.neural.ItemAutoRec.itemautorec.`**ItemAutoRec** (*data, config, params, *args, **kwargs*)
Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

get_recommendations (*k: int = 100*)

property name

train ( )

elliot.recommender.neural.ItemAutoRec.itemautorec_model module

Module description:

class `elliot.recommender.neural.ItemAutoRec.itemautorec_model.`**Decoder** (**args, **kwargs*)
Bases: `tensorflow.python.keras.engine.base_layer.Layer`

call (*inputs, **kwargs*)

class `elliot.recommender.neural.ItemAutoRec.itemautorec_model.`**Encoder** (**args, **kwargs*)
Bases: `tensorflow.python.keras.engine.base_layer.Layer`

call (*inputs, training=None*)

class `elliot.recommender.neural.ItemAutoRec.itemautorec_model.`**ItemAutoRecModel** (**args, **kwargs*)
Bases: `tensorflow.python.keras.engine.training.Model`

call (*inputs, training=None, **kwargs*)

get_config ( )
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:
Python dictionary.

`get_recs` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix. [Is Inverted]

Returns:
The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)
Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`train_step` (*batch*)

Module contents

Module description:

elliot.recommender.neural.NAIS package

Submodules

elliot.recommender.neural.NAIS.nais module

Module description:

class `elliot.recommender.neural.NAIS.nais.`**`NAIS`** (*data, config, params, *args, **kwargs*)
Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100, auc_compute: bool = False*)

property name

`train` ( )

elliot.recommender.neural.NAIS.nais_model module

Module description:

class `elliot.recommender.neural.NAIS.nais_model.`**`LatentFactor`** (**args, **kwargs*)
Bases: `tensorflow.python.keras.layers.embeddings.Embedding`

`censor` (*censor_id*)

class `elliot.recommender.neural.NAIS.nais_model.`**`NAIS_model`** (**args, **kwargs*)
Bases: `tensorflow.python.keras.engine.training.Model`

`attention` (*user_history, target*)

`batch_attention` (*user_history, target*)

`batch_predict` (*user_start, user_stop*)

`batch_softmax` (*logits, item_num, similarity, user_bias, item_bias*)

`call` (*inputs, training=None*)

`create_history_item_matrix` ( )

`get_config` ( )
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:
Python dictionary.

`get_positions` (*predictions, train_mask, items, inner_test_user_true_mask*)

`get_top_k` (*predictions, train_mask, k=100*)

`predict` (*user, **kwargs*)

`softmax` (*logits, item_num, similarity, user_bias, item_bias, batch_mask_mat=None*)

`train_step` (*batch*)

Module contents

elliot.recommender.neural.NFM package

Submodules

elliot.recommender.neural.NFM.neural_fm module

Module description:

class `elliot.recommender.neural.NFM.neural_fm.`**`NFM`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`predict` (*u: int, i: int*)

`restore_weights` ( )

`train` ( )

elliot.recommender.neural.NFM.neural_fm_model module

Module description:

class `elliot.recommender.neural.NFM.neural_fm_model.`**`NeuralFactorizationMachineModel`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:
The matrix of predicted values.

`train_step` (*batch*)

Module contents

elliot.recommender.neural.NPR package

Submodules

elliot.recommender.neural.NPR.neural_personalized_ranking module

Module description:

class `elliot.recommender.neural.NPR.neural_personalized_ranking.`**`NPR`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.neural.NPR.neural_personalized_ranking_model module

Module description:

class `elliot.recommender.neural.NPR.neural_personalized_ranking_model.`**`NPRModel`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

elliot.recommender.neural.NeuMF package

Submodules

elliot.recommender.neural.NeuMF.neural_matrix_factorization module

Module description:

class `elliot.recommender.neural.NeuMF.neural_matrix_factorization.`**`NeuMF`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.neural.NeuMF.neural_matrix_factorization_model module

Module description:

class
`elliot.recommender.neural.NeuMF.neural_matrix_factorization_model.`**`NeuralMatrixFactoriza`**
(**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

elliot.recommender.neural.UserAutoRec package

Submodules

elliot.recommender.neural.UserAutoRec.userautorec module

Module description:

class `elliot.recommender.neural.UserAutoRec.userautorec.`**`UserAutoRec`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

property name

`train` ( )

elliot.recommender.neural.UserAutoRec.userautorec_model module

Module description:

class `elliot.recommender.neural.UserAutoRec.userautorec_model.`**`Decoder`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

`call` (*inputs, **kwargs*)

class `elliot.recommender.neural.UserAutoRec.userautorec_model.`**`Encoder`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

`call` (*inputs, training=None*)

class `elliot.recommender.neural.UserAutoRec.userautorec_model.`**`UserAutoRecModel`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, **kwargs*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

Module description:

elliot.recommender.neural.WideAndDeep package

Submodules

elliot.recommender.neural.WideAndDeep.wide_and_deep module

Module description:

class `elliot.recommender.neural.WideAndDeep.wide_and_deep.`**`WideAndDeep`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

`elliot.recommender.neural.WideAndDeep.wide_and_deep.`**`build_sparse_features`** (*data*)

elliot.recommender.neural.WideAndDeep.wide_and_deep_model module

Module description:

class `elliot.recommender.neural.WideAndDeep.wide_and_deep_model.`**`WideAndDeepModel`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=False, **kwargs*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

`get_sparse` (*u*, *i*)

`get_top_k` (*preds*, *train_mask*, *k=100*)

`get_user_recs` (*user*, *k=100*)

`predict` (*user*, ***kwargs*)

Generates output predictions for the input samples.

Computation is done in batches. This method is designed for performance in large scale inputs. For small amount of inputs that fit in one batch, directly using *__call__* is recommended for faster execution, e.g., *model(x)*, or *model(x, training=False)* if you have layers such as *tf.keras.layers.BatchNormalization* that behaves differently during inference. Also, note the fact that test loss is not affected by regularization layers like noise and dropout.

Arguments:

x: Input samples. It could be:

- A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
- A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).
- A *tf.data* dataset.
- A generator or *keras.utils.Sequence* instance.

A more detailed description of unpacking behavior for iterator types (Dataset, generator, Sequence) is given in the *Unpacking behavior for iterator-like inputs* section of *Model.fit*.

batch_size: Integer or *None*.

Number of samples per batch. If unspecified, *batch_size* will default to 32. Do not specify the *batch_size* if your data is in the form of dataset, generators, or *keras.utils.Sequence* instances (since they generate batches).

verbose: Verbosity mode, 0 or 1. steps: Total number of steps (batches of samples)

before declaring the prediction round finished. Ignored with the default value of *None*. If x is a *tf.data* dataset and *steps* is None, *predict* will run until the input dataset is exhausted.

callbacks: List of *keras.callbacks.Callback* instances.

List of callbacks to apply during prediction. See [callbacks](/api_docs/python/tf/keras/callbacks).

max_queue_size: Integer. Used for generator or *keras.utils.Sequence*

input only. Maximum size for the generator queue. If unspecified, *max_queue_size* will default to 10.

workers: Integer. Used for generator or *keras.utils.Sequence* input

only. Maximum number of processes to spin up when using process-based threading. If unspecified, *workers* will default to 1. If 0, will execute the generator on the main thread.

use_multiprocessing: Boolean. Used for generator or
keras.utils.Sequence input only. If *True*, use process-based threading. If unspecified, *use_multiprocessing* will default to *False*. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

See the discussion of *Unpacking behavior for iterator-like inputs* for *Model.fit*. Note that Model.predict uses the same interpretation rules as *Model.fit* and *Model.evaluate*, so inputs must be unambiguous for all three methods.

Returns:
Numpy array(s) of predictions.

Raises:
RuntimeError: If *model.predict* is wrapped in *tf.function*. ValueError: In case of mismatch between the provided

input data and the model's expectations, or in case a stateful model receives a number of samples that is not a multiple of the batch size.

`train_step` (*batch*)

The logic for one training step.

This method can be overridden to support custom training logic. This method is called by *Model.make_train_function*.

This method should contain the mathematical logic for one step of training. This typically includes the forward pass, loss calculation, backpropagation, and metric updates.

Configuration details for *how* this logic is run (e.g. *tf.function* and *tf.distribute.Strategy* settings), should be left to *Model.make_train_function*, which can also be overridden.

Arguments:
data: A nested structure of `Tensor`s.

Returns:
A *dict* containing values that will be passed to *tf.keras.callbacks.CallbackList.on_train_batch_end*. Typically, the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

Module contents

Module contents

elliot.recommender.unpersonalized package

Subpackages

elliot.recommender.unpersonalized.most_popular package

Submodules

elliot.recommender.unpersonalized.most_popular.most_popular module

Created on April 4, 2020 Tensorflow 2.1.0 implementation of APR. @author Anonymized

class `elliot.recommender.unpersonalized.most_popular.most_popular.`**`MostPop`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*top_k*)

property name

`train` ( )

Module contents

elliot.recommender.unpersonalized.random_recommender package

Submodules

elliot.recommender.unpersonalized.random_recommender.Random module

Created on April 4, 2020 Tensorflow 2.1.0 implementation of APR. @author Anonymized

class `elliot.recommender.unpersonalized.random_recommender.Random.`**`Random`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*top_k*)

property **`name`**

`train` ( )

Module contents

Module contents

elliot.recommender.visual_recommenders package

Subpackages

elliot.recommender.visual_recommenders.ACF package

Submodules

elliot.recommender.visual_recommenders.ACF.ACF module

Module description:

class `elliot.recommender.visual_recommenders.ACF.ACF.`**`ACF`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

property **`name`**

`train` ( )

elliot.recommender.visual_recommenders.ACF.ACF_model module

Module description:

class `elliot.recommender.visual_recommenders.ACF.ACF_model.`**`ACF_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*start, stop*)

`train_step` (*batch*)

Module contents

Module description:

elliot.recommender.visual_recommenders.DVBPR package

Submodules

elliot.recommender.visual_recommenders.DVBPR.DVBPR module

Module description:

class `elliot.recommender.visual_recommenders.DVBPR.DVBPR.`**`DVBPR`** (*data*, *config*, *params*, **args*, ***kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property name

`train` ( )

elliot.recommender.visual_recommenders.DVBPR.DVBPR_model module

Module description:

class `elliot.recommender.visual_recommenders.DVBPR.DVBPR_model.`**`DVBPR_model`** (**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs*, *training=None*, *mask=None*)

`get_config` ( )

`get_top_k` (*preds*, *train_mask*, *k=100*)

`predict_batch` (*start*, *stop*, *phi*)

`train_step` (*batch*)

elliot.recommender.visual_recommenders.DVBPR.FeatureExtractor module

class
`elliot.recommender.visual_recommenders.DVBPR.FeatureExtractor.`**`FeatureExtractor`**
(**args*, ***kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`, `abc.ABC`

`call` (*inputs*, *training=None*, *mask=None*)

Calls the model on new inputs.

In this case *call* just reapplies all ops in the graph to the new inputs (e.g. build a new computational graph from the provided inputs).

Arguments:

inputs: A tensor or list of tensors. training: Boolean or boolean scalar tensor, indicating whether to run

the *Network* in training mode or inference mode.

mask: A mask or list of masks. A mask can be

either a tensor or None (no mask).

Returns:

A tensor if there is a single output, or a list of tensors if there are more than one outputs.

Module contents

Module description:

elliot.recommender.visual_recommenders.DeepStyle package

Submodules

elliot.recommender.visual_recommenders.DeepStyle.DeepStyle module

Module description:

class `elliot.recommender.visual_recommenders.DeepStyle.DeepStyle.`**`DeepStyle`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property **`name`**

`train` ( )

elliot.recommender.visual_recommenders.DeepStyle.DeepStyle_model module

Module description:

class `elliot.recommender.visual_recommenders.DeepStyle.DeepStyle_model.`**`DeepStyle_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None*)

`get_config` ( )

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*start, stop, training=False*)

`train_step` (*batch*)

Module contents

Module description:

elliot.recommender.visual_recommenders.VBPR package

Submodules

elliot.recommender.visual_recommenders.VBPR.VBPR module

Module description:

class `elliot.recommender.visual_recommenders.VBPR.VBPR.`**`VBPR`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property **`name`**

`train` ( )

elliot.recommender.visual_recommenders.VBPR.VBPR_model module

Module description:

class `elliot.recommender.visual_recommenders.VBPR.VBPR_model.`**`VBPR_model`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None*)

`get_config` ( )

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns:

Python dictionary.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*start, stop*)

`train_step` (*batch*)

Module contents

Module description:

elliot.recommender.visual_recommenders.VNPR package

Submodules

elliot.recommender.visual_recommenders.VNPR.visual_neural_personalized_ranking module

Module description:

class `elliot.recommender.visual_recommenders.VNPR.visual_neural_personalized_ranking.`**`VNPR`** (*data, config, params, *args, **kwargs*)

Bases: `elliot.recommender.recommender_utils_mixin.RecMixin`, `elliot.recommender.base_recommender_model.BaseRecommenderModel`

`get_recommendations` (*k: int = 100*)

property `name`

`train` ( )

elliot.recommender.visual_recommenders.VNPR.visual_neural_personalized_ranking_model module

Module description:

class `elliot.recommender.visual_recommenders.VNPR.visual_neural_personalized_ranking_model.`**`VN`** (**args, **kwargs*)

Bases: `tensorflow.python.keras.engine.training.Model`

`call` (*inputs, training=None, mask=None*)

`get_recs` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`get_top_k` (*preds, train_mask, k=100*)

`predict` (*inputs, training=False, **kwargs*)

Get full predictions on the whole users/items matrix.

Returns:

The matrix of predicted values.

`train_step` (*batch*)

Module contents

Module contents

Module description:

Submodules

elliot.recommender.base_recommender_model module

Module description:

class `elliot.recommender.base_recommender_model.`**`BaseRecommenderModel`** (*data, config, params, *args, **kwargs*)

Bases: `abc.ABC`

`autoset_params` ( )

Define Parameters as tuples: (variable_name, public_name, shortcut, default, reading_function, printing_function) Example:

self._params_list = [

(“_similarity”, “similarity”, “sim”, “cosine”, None, None), (“_user_profile_type”, “user_profile”, “up”, “tfidf”, None, None), (“_item_profile_type”, “item_profile”, “ip”, “tfidf”, None, None), (“_mlpunits”, “mlp_units”, “mlpunits”, “(1,2,3)”, lambda x: list(make_tuple(x)), lambda x: str(x).replace(“,”, “-“)),

]

abstract `get_loss` ( )

abstract `get_params` ( )

`get_params_shortcut` ( )

abstract `get_recommendations` (**args*)

abstract `get_results` ( )

abstract `train` ( )

`elliot.recommender.base_recommender_model.`**`init_charger`** (*init*)

elliot.recommender.recommender_utils_mixin module

class `elliot.recommender.recommender_utils_mixin.`**`RecMixin`**

Bases: `object`

`get_loss` ( )

`get_params` ( )

`get_recommendations` (*k: int = 100*)

`get_results` ( )

`get_train_mask` (*start, stop*)

`restore_weights` ( )

`train` ( )

Module contents

Module description:

1.1.7 elliot.result_handler package

Submodules

elliot.result_handler.result_handler module

Module description:

class `elliot.result_handler.result_handler.`**`HyperParameterStudy`** (*rel_threshold=1*)
Bases: `object`

`add_trials` (*obj*)

`save_trials` (*output='../results/'*)

`save_trials_as_triplets` (*output='../results/'*)

class `elliot.result_handler.result_handler.`**`ResultHandler`** (*rel_threshold=1*)
Bases: `object`

`add_oneshot_recommender` (***kwargs*)

`save_best_models` (*output='../results/'*)

`save_best_results` (*output='../results/'*)

`save_best_results_as_triplets` (*output='../results/'*)

`save_best_statistical_results` (*stat_test, output='../results/'*)

class `elliot.result_handler.result_handler.`**`StatTest`** (*value*)
Bases: `enum.Enum`
An enumeration.

`PairedTTest` **= [<class 'elliot.evaluation.statistical_significance.PairedTTest'>, 'paired_ttest']**

`WilcoxonTest` **= [<class 'elliot.evaluation.statistical_significance.WilcoxonTest'>, 'wilcoxon_test']**

Module contents

Module description:

1.1.8 elliot.splitter package

Submodules

elliot.splitter.base_splitter module

class `elliot.splitter.base_splitter.`**`Splitter`** (*data: pandas.core.frame.DataFrame, splitting_ns: types.SimpleNamespace*)
Bases: `object`

`fold_list_generator` (*length, folds=5*)

`generic_split_function` (*data: pandas.core.frame.DataFrame, **kwargs*) → List[Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]]

handle_hierarchy (*data: pandas.core.frame.DataFrame, valtest_splitting_ns: types.SimpleNamespace*) → List[Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]]

process_splitting ( )

read_folder (*folder_path*)

rearrange_data (*train_test: List[Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]], train_val: List[List[Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]]]*)

splitting_best_timestamp (*d: pandas.core.frame.DataFrame, min_below=1, min_over=1*)

splitting_kfolds (*data: pandas.core.frame.DataFrame, folds=5*)

splitting_passed_timestamp (*d: pandas.core.frame.DataFrame, timestamp=1*)

splitting_randomsubsampling_kfolds (*d: pandas.core.frame.DataFrame, folds=5, ratio=0.2*)

splitting_randomsubsampling_kfolds_leavenout (*d: pandas.core.frame.DataFrame, folds=5, n=1*)

splitting_temporal_holdout (*d: pandas.core.frame.DataFrame, ratio=0.2*)

splitting_temporal_leavenout (*d: pandas.core.frame.DataFrame, n=1*)

store_splitting (*tuple_list*)

subsampling_leavenout_list_generator (*length, n=1*)

subsampling_list_generator (*length, ratio=0.2*)

Module contents

1.1.9 elliot.utils package

Submodules

elliot.utils.folder module

Module description:

`elliot.utils.folder.`**build_log_folder** (*path_log_folder*)

`elliot.utils.folder.`**build_model_folder** (*path_output_rec_weight, model*)

`elliot.utils.folder.`**create_folder_by_index** (*path, index*)

`elliot.utils.folder.`**manage_directories** (*path_output_rec_result, path_output_rec_weight, path_output_rec_performance*)

elliot.utils.logger_util module

class `elliot.utils.logger_util.`**QueueListenerHandler** (*handlers, respect_handler_level=False, auto_run=True, queue=<queue.Queue object>*)

Bases: `logging.handlers.QueueHandler`

emit (*record*)

Emit a record.

Writes the LogRecord to the queue, preparing it for pickling first.

start ( )

stop ( )

elliot.utils.logging module

class `elliot.utils.logging.`**`TimeFilter`** (*name=''*)

Bases: `logging.Filter`

`filter` (*record: logging.LogRecord*) → bool

Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. If deemed appropriate, the record may be modified in-place.

`elliot.utils.logging.`**`get_logger`** (*name, log_level=10*)

`elliot.utils.logging.`**`init`** (*path_config, folder_log, log_level=30*)

`elliot.utils.logging.`**`prepare_logger`** (*name, path, log_level=10*)

elliot.utils.read module

Module description:

`elliot.utils.read.`**`find_checkpoint`** (*dir, restore_epochs, epochs, rec, best=0*)

Parameters

- **`dir`** – directory of the model where we start from the reading.
- **`restore_epochs`** – epoch from which we start from.
- **`epochs`** – epochs from which we restore (0 means that we have best)
- **`rec`** – recommender model
- **`best`** – 0 No Best - 1 Search for the Best

Returns

`elliot.utils.read.`**`load_obj`** (*name*)

Load the pkl object by name :param name: name of file :return:

`elliot.utils.read.`**`read_config`** (*sections_fields*)

Args:

sections_fields (list): list of fields to retrieve from configuration file

Return:

A list of configuration values.

`elliot.utils.read.`**`read_csv`** (*filename*)

Args:

filename (str): csv file path

Return:

A pandas dataframe.

`elliot.utils.read.`**`read_imagenet_classes_txt`** (*filename*)

Args:

filename (str): txt file path

Return:

A list with 1000 imagenet classes as strings.

`elliot.utils.read.`**`read_multi_config`** ( )

It reads a config file that contains the configuration parameters for the recommendation systems.

Return:

A list of configuration settings.

`elliot.utils.read.`**`read_np`** (*filename*)

Args:

filename (str): filename of numpy to load

Return:

The loaded numpy.

elliot.utils.write module

Module description:

`elliot.utils.write.`**`save_np`** (*npy, filename*)

Store numpy to memory. Args:

npy: numpy to save filename (str): filename

`elliot.utils.write.`**`save_obj`** (*obj, name*)

Store the object in a pkl file :param obj: python object to be stored :param name: file name (Not insert .pkl) :return:

`elliot.utils.write.`**`store_recommendation`** (*recommendations, path=''*)

Store recommendation list (top-k) :return:

Module contents

Module description:

1.2 Submodules

1.3 elliot.run module

Module description:

`elliot.run.`**`config_test`** (*builder, base*)

`elliot.run.`**`run_experiment`** (*config_path: str = './config/config.yml'*)

1.4 Module contents

Module description:

- genindex
- modindex
- search

Python Module Index

Index