DeepRec Documentation

Release 1

fcolaco@lasige.di.fc.ul.pt

Mar 16, 2021

User Guide

CHAPTER 1

Installation

1.1 Using PIP

```
$ pip install drecpy
```

If you can't get the latest version from PyPi:

```
$ pip install git+https://github.com/fabioiuri/DRecPy
```

1.2 Using Git

By directly by cloning the Git repo:

```
$ git clone https://github.com/fabioiuri/DRecPy
$ cd DRecPy
$ python setup.py install
```

1.3 Update Version

If you want to update to the newest DRecPy version, use:

```
$ pip install drecpy --upgrade
```

CHAPTER 2

Getting Started

2.1 Basic usage

2.1.1 Training a model

Recommenders built using the DRecPy framework follow the usual method definitions: *fit()* to fit the model to the provided data, and *predict()*, *rank()* or *recommend()* to provide predictions. Once trained, in order to evaluate a model, one can build custom evaluation processes, or can use the builtin ones, which are defined on the *DRecPy.Evaluation package*.

Here's a quick example of training the CDAE recommender with the MovieLens-100k data set on 100 epochs, and evaluating the ranking performance on 100 test users. Node that a seed parameter is passed through when instantiating the CDAE object, as well as when calling the evaluation process, so that we can have a deterministic pipeline.

Listing 1: From file `examples/cdae.py`

```
from DRecPy.Recommender import CDAE
from DRecPy.Dataset import get_train_dataset
from DRecPy.Dataset import get_test_dataset
from DRecPy.Evaluation.Processes import ranking_evaluation
import time

ds_train = get_train_dataset('ml-100k')
ds_test = get_test_dataset('ml-100k')

start_train = time.time()
cdae = CDAE(hidden_factors=50, corruption_level=0.2, loss='bce', seed=10)
cdae.fit(ds_train, learning_rate=0.001, reg_rate=0.001, epochs=50, batch_size=64, neg_
↪ratio=5)
print("Training took", time.time() - start_train)

print(ranking_evaluation(cdae, ds_test, k=[1, 5, 10], novelty=True, n_test_users=100,␣
↪n_pos_interactions=1,
```

(continues on next page)

(continued from previous page)

```
                            n_neg_interactions=100, generate_negative_pairs=True,␣
↪seed=10, max_concurrent_threads=4,
                            verbose=True))
```

2.2 Data Set usage

To learn more about the public methods offered by the *InteractionDataset* module, please read the respective api documentation. This section is simply a brief introduction on how to import and make use of data sets.

2.2.1 Importing a built-in data set

At the moment, DRecPy provides various builtin data sets, such as: the MovieLens (100k, 1M, 10M and 20M) and the Book Crossing data set. Whenever you're using a builtin data set for the first time, a new folder will be created at your home path called ".DRecPy_data". If you want to provide a custom path for saving these data sets, you can do so by providing the *DATA_FOLDER* environment variable mapping to the intended path.

The example bellow shows how to use a builtin data set and how to manipulate it using the provided methods:

Listing 2: From file `examples/integrated_datasets.py`

```
from DRecPy.Dataset import get_train_dataset
from DRecPy.Dataset import get_test_dataset
from DRecPy.Dataset import get_full_dataset
from DRecPy.Dataset import available_datasets

print('Available datasets', available_datasets())

# Reading the ml-100k full dataset and prebuilt train and test datasets.
print('ml-100k full dataset', get_full_dataset('ml-100k'))
print('ml-100k train dataset', get_train_dataset('ml-100k'))
print('ml-100k test dataset', get_test_dataset('ml-100k'))

# Reading the ml-1m full dataset and generated train and test datasets using out of␣
↪memory storage.
print('ml-1m full dataset', get_full_dataset('ml-1m', force_out_of_memory=True))
print('ml-1m train dataset', get_train_dataset('ml-1m', force_out_of_memory=True))
print('ml-1m test dataset', get_test_dataset('ml-1m', force_out_of_memory=True))

# Showcase some dataset operations
ds_ml = get_full_dataset('ml-100k')
print('Minimum rating value:', ds_ml.min('interaction'))
print('Unique rating values:', ds_ml.unique('interaction').values_list())

ds_ml.apply('interaction', lambda x: x / ds_ml.max('interaction'))  # standardize the␣
↪rating value
print('New values', ds_ml.values_list()[:5])
```

2.2.2 Importing a custom data set

Custom data sets are also supported, and you should provide the path to the csv file as well as the column names and the delimiter.

Listing 3: From file `examples/custom_datasets.py`

```
from DRecPy.Dataset import InteractionDataset
from os import remove

# create file with sample dataset
with open('tmp.csv', 'w') as f:
    f.write('"john","ps4",4.5\n')
    f.write('"patrick","xbox",4.1\n')
    f.write('"anna","brush",3.6\n')
    f.write('"david","tv",2.0\n')

# load dataset into memory
ds_memory = InteractionDataset('tmp.csv', columns=['user', 'item', 'interaction'])
print('all values:', ds_memory.values_list())
print('filtered values:', ds_memory.select('interaction > 3.5').values_list())
ds_memory_scaled = ds_memory.copy()
ds_memory_scaled.apply('interaction', lambda x: x / ds_memory.max('interaction'))
print('all values scaled:', ds_memory_scaled.values_list())

# load dataset out of memory
ds_out_of_memory = InteractionDataset('tmp.csv', columns=['user', 'item', 'interaction
↪'], in_memory=False)
print('all values:', ds_out_of_memory.values_list())
print('filtered values:', ds_out_of_memory.select('interaction > 3.5').values_list())

remove('tmp.csv')  # delete previously created sample dataset file
```

Note that there are **3 required columns**: user, item and interaction.

CHAPTER 3

Creating Novel Recommenders

3.1 From scratch

DRecPy allows you to easily implement novel recommender models without the need to worry about data handling, id conversion, workflow management and weight updates.

3.1.1 Deep Learning-based Recommenders

To create a DL-based recommender you should create a class that extends the *RecommenderABC* class, and implements at least the required abstract methods: *_pre_fit()*, *_sample_batch()*, *_predict_batch()*, *_compute_batch_loss()* and *_predict()*.

The *_pre_fit()* method should:

- Create model structure (e.g. neural network layer structure);
- Initialize model weights;
- Register the model weights as trainable variables (using the *_register_trainable(var)* or *_register_trainables(vars)* method);

The *_sample_batch()* method should sample *N* data points from the training data and return them. This step can also be used if your model makes some custom data preprocessing during the training phase. The return value of this function is passed to the *_predict_batch()* method.

The *_predict_batch()* method should compute predictions for each of the sampled training data points. The return of this method should be a tuple containing a list of predictions and a list of desired values (in this exact order).

The *_compute_batch_loss()* method should compute and return the loss value associated with the given predictions and desired values. This loss must be differentiable with respect to all training variables, so that weight updates can be made.

Finally, the *_predict()* method should compute a single prediction, for the provided user id and item id.

Another method that can be overridden is the *_compute_reg_loss()*, which should compute and return the regularization loss value associated with the trainable variables. Its default implementation is only useful when the registered

trainable variables are of type tf.keras.models.Model or tf.keras.layers.Layer, in which case the registered regularizers (e.g. via kernel_regularizer attribute of a layer) are added to the loss of the recommender for each batch.

If the *_rank()* method is not implemented, and if you call *rank()* on the new model, the *_predict()* method will be used to score each item. Similarly, if the *_recommend()* method is not implemented, and if you call *recommend()*, the *rank()* will also be used to rank all existent items and return the top *N*.

Usually, the *_rank()* and *_recommend()* methods are only implemented when there's an alternative and more efficient way to compute these values, not relying on the *_predict()* method.

Here's a basic example of a deep learning-based recommender, with only one trainable variable:

Listing 1: From file `examples/custom_deep_recommender.py`

```
from DRecPy.Recommender import RecommenderABC
from DRecPy.Sampler import PointSampler
import tensorflow as tf
from DRecPy.Dataset import get_train_dataset


class TestRecommender(RecommenderABC):

    def __init__(self, **kwds):
        super(TestRecommender, self).__init__(**kwds)

    def _pre_fit(self, learning_rate, neg_ratio, reg_rate, **kwds):
        # used to declare variables and the neural network structure of the model, as␣
↪well as register trainable vars
        self._info(f'doing pre-fit with learning_rate={learning_rate}, neg_ratio={neg_
↪ratio}, reg_rate={reg_rate}')
        self._weights = tf.Variable([[0.5], [0.5]])
        self._register_trainable(self._weights)
        self._loss = tf.losses.BinaryCrossentropy()
        self._sampler = PointSampler(self.interaction_dataset, neg_ratio=neg_ratio)

    def _sample_batch(self, batch_size, **kwds):
        self._info(f'doing _sample_batch {batch_size}')
        return self._sampler.sample(batch_size)

    def _predict_batch(self, batch_samples, **kwds):
        # must return predictions from which gradients can be computed in order to␣
↪update the registered trainable vars
        # and the desired values, so that we're able to compute the batch loss
        self._info(f'doing _predict_batch {batch_samples}')
        predictions = [self._predict(u, i) for u, i, _ in batch_samples]
        desired_values = [y for _, _, y in batch_samples]
        self._info(f'predictions = {predictions}, desired_values = {desired_values}')
        return predictions, desired_values

    def _compute_batch_loss(self, predictions, desired_values, **kwds):
        # receives the predictions and desired values computed during the _predict_
↪batch, and should apply a loss
        # function from which gradients can then be computed
        self._info(f'doing _compute_batch_loss: predictions={predictions}, desired_
↪values={desired_values}')
        return self._loss(desired_values, predictions)

    def _compute_reg_loss(self, reg_rate, batch_size, trainable_models, trainable_
↪layers, trainable_weights, **kwds):
```

(continues on next page)

(continued from previous page)

```
        self._info(f'doing _compute_reg_loss: reg_rate={reg_rate}, batch_size={batch_
↪size}')
        return tf.nn.l2_loss(trainable_weights) * reg_rate / batch_size

    def _predict(self, uid, iid, **kwds):
        # predict for a (user, item) pair
        return tf.sigmoid(tf.matmul(tf.convert_to_tensor([[uid, iid]], dtype=tf.
↪float32), self._weights))


ds_train = get_train_dataset('ml-100k', verbose=False)

print('TestRecommender')
recommender = TestRecommender(verbose=True)
recommender.fit(ds_train, epochs=2, batch_size=10)
print(recommender.predict(1, 1))
```

Some other **important notes**:

- Do not forget to call the *__init__()* method of the super class, on the new model *__init__()*;
- The *_pre_fit()* method should register all trainable variables via calls to the *_register_trainable(var)* or *_register_trainables(vars)* methods;
- The *_compute_batch_loss()* return value must be differentiable with respect to all trainable variables;
- If your model depends on custom random processes, such as weight initialization or id sampling, always use random generators that are created using the self.seed attribute. A seeded random generator object is already provided via the **self._rng** attribute. If a seed argument is provided, both self._rng and all tensorflow.random methods are seeded with the given value.

3.1.2 Non-Deep Learning-based Recommenders

To create a non-DL-based recommender you should create a class that extends the *RecommenderABC* class, and implements at least the required abstract methods: *_pre_fit()*, *_sample_batch()*, *_predict_batch()*, *_compute_batch_loss()* and *_predict()*.

Note that in this case, the implementation of the *_sample_batch()*, *_predict_batch()*, *_compute_batch_loss()* and *_compute_reg_loss()* is irrelevant, since they will never be called.

The *_pre_fit()* method should create the required data structures and do the computations to completely fit the model, because no batch training will be applied. In this case, **trainable variables must not be registered**, otherwise batch-training will proceed.

All other methods such as the *_predict()*, *_rank()* and *_recommend()*, follow the same guidelines.

An example of an implemented non-deep learning-based recommender, follows bellow:

Listing 2: From file `examples/custom_non_deep_recommender.py`

```
from DRecPy.Recommender import RecommenderABC
from DRecPy.Dataset import get_train_dataset


class TestRecommenderNonDeepLearning(RecommenderABC):

```

(continues on next page)

(continued from previous page)

```
    def __init__(self, **kwds):
        super(TestRecommenderNonDeepLearning, self).__init__(**kwds)

    def _pre_fit(self, learning_rate, neg_ratio, reg_rate, **kwds):
        # used to declare variables and do the non-deep learning fit process, such as␣
↪computing similarities and
        # neighbours for knn-based models
        self._info(f'doing pre-fit with learning_rate={learning_rate}, neg_ratio={neg_
↪ratio}, reg_rate={reg_rate}')
        pass

    def _sample_batch(self, batch_size, **kwds):
        raise NotImplemented  # since it's non-deep learning based, no need for batch␣
↪training

    def _predict_batch(self, batch_samples, **kwds):
        raise NotImplemented  # since it's non-deep learning based, no need for batch␣
↪training

    def _compute_batch_loss(self, predictions, desired_values, **kwds):
        raise NotImplemented  # since it's non-deep learning based, no need for batch␣
↪training

    def _predict(self, uid, iid, **kwds):
        return 5  # predict for a (user, item) pair


ds_train = get_train_dataset('ml-100k', verbose=False)

print('TestRecommenderNonDeepLearning')
recommender = TestRecommenderNonDeepLearning(verbose=True)
recommender.fit(ds_train, epochs=2, batch_size=10)
print(recommender.predict(1, 1))
```

3.2 Extending existing models

To extend an existent recommender, one should:

- Create a subclass of the original recommender;
- Override the *__init__()* method, by first calling the original recommender *__init__()*, followed by instructions with specific logic for the extended recommender;
- If new weights are introduced by this extension, override the *_pre_fit()* method and call its original, followed by the initialization of the new weights and registering them as trainable variables (via *_register_trainable(var)* or *_register_trainables(vars)* method calls).
- If there are changes to the batch sampling workflow, override the *_sample_batch()* method and call its original (if some of its logic can be reused), and then apply the custom logic;
- When there are changes on the way predictions are made, override the *_predict_batch()* and call its original (if some of its logic can be reused), followed by adding the custom prediction logic to the existent predictions. Sometimes, if the original model has independent logic on the *_predict()* that does not use the *_predict_batch()*, you might need to override it too.

- If there are changes to the way the predictive loss function is computed, override the *_compute_batch_loss()* and adapt it to return the new predictive loss value from the provided predictions and expected values.
- **Overriding the *_compute_reg_loss()* is only necessary in some situations. Make sure that:**
    1. If the new model uses trainable variables of type tf.keras.models.Model or tf.keras.layers.Layer, and you've added their regularization using the regularization parameters of each layer (e.g. kerner_regularizer attribute), the *RecommenderABC._compute_reg_loss()* should always be called.
    2. If the new model uses trainable variables of type tf.Variable, override the *_compute_reg_loss()* to compute their regularization values. Note that if the original model uses trainable variables of type tf.keras.models.Model or tf.keras.layers.Layer, the *RecommenderABC._compute_reg_loss()* should always be called.

A very simple example of extending an existing model is shown bellow, which is a modification of the DMF (Deep Matrix Factorization) recommender:

Listing 3: From file `examples/extending_recommender_dmf.py`

```
from DRecPy.Recommender import DMF
import tensorflow as tf


class ModifiedDMF(DMF):
    def __init__(self, **kwds):
        super(ModifiedDMF, self).__init__(**kwds)

    def _pre_fit(self, learning_rate, neg_ratio, reg_rate, **kwds):
        super(ModifiedDMF, self)._pre_fit(learning_rate, neg_ratio, reg_rate, **kwds)
        self._extra_weight = tf.Variable([1.])
        self._register_trainable(self._extra_weight)

    def _predict_batch(self, batch_samples, **kwds):
        predictions, desired_values = super(ModifiedDMF, self)._predict_batch(batch_
↪samples, **kwds)
        predictions = [(self._extra_weight * pred) for pred in predictions]
        return predictions, desired_values
```

CHAPTER 4

DRecPy.Dataset package

4.1 DRecPy.Dataset.dataset_abc module

class DRecPy.Dataset.dataset_abc.**InteractionDatasetABC**(***kwds*)

Bases: abc.ABC

apply(*column*, *function*)

Modifies the current dataset instance by applying a transformation to a specific column in every row.

Parameters

- **column** – A string that represents the name of the column that will be transformed.
- **function** – The function that will be used to map the current column value in each row to the new one.

Returns None.

assign_internal_ids()

Assigns user and item internal ids. Internal ids are integer consecutive identifiers that represent each user or item uniquely. Two new columns are created on this dataset instance: "uid" and "iid", for user internal id and item internal id, respectively.

Returns None.

copy()

Copies the current dataset instance into a new one.

Returns InteractionDataset instance with the same data values as the current one.

count_unique(*columns=None*)

Count the number of unique values on the provided column combination.

Parameters **columns** – A list containing the columns to take into account. Default: all.

Returns The count of unique values present on the provided column combination.

drop(*record_ids*, *copy=True*, *keep=False*)

Remove (or keep) the provided list of record ids from the current InteractionDataset instance.

Parameters

- **record_ids** – A list of integers representing record ids.
- **copy** – A boolean indicating whether to create a new InteractionDataset instance to remove (or keep) the provided list of record ids, or if the current InteractionDataset instance should be modified accordingly (copy=False). Default: True.
- **keep** – A boolean indicating whether the provided record ids should be kept or removed (keep=False). Default: False.

Returns An instance of the InteractionDataset with (or without) the filtered records.

exists(*query*)

Compute if the provided query handles at least 1 value or not.

Parameters **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".

Returns A boolean indicating if the query handles any results or not.

iid_to_item(*iid*)

Converts a given internal item id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **iid** – The item internal id.

Returns An integer value representing the item raw id, or None if the internal item id provided does not exist.

item_to_iid(*item*)

Converts a given raw item id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **item** – The item raw id.

Returns An integer value representing the item internal id, or None if the raw item id provided does not exist.

max(*column=None*)

Computes the maximum value for the provided column.

Parameters **column** – The name of the column for which the maximum should be computed.

Returns The maximum value present on the whole dataset, for the provided column name.

min(*column=None*)

Computes the minimum value for the provided column.

Parameters **column** – The name of the column for which the minimum should be computed.

Returns The minimum value present on the whole dataset, for the provided column name.

null_interaction_pair_generator(*interaction_threshold=None*, *seed=None*)

Provides a generator that yields negative / null interaction pairs.

Parameters

- **interaction_threshold** – An optional integer that is used as the boundary interaction value between positive and negative interaction pairs. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. If none is provided, positive interactions are the ones present on the dataset, and all the others are considered negative. Default: None.

- **seed** – An optional integer to be used as the seed value for the pseudo-random number generated used to sample null interaction pairs. Default: None.

Returns A generator that yields negative / null interaction pairs, that is, (user internal id, item internal id) tuples.

remove_internal_ids()
Removes user and item internal ids.

Returns None.

save(*path*, *columns=None*, *write_header=False*)
Persists the current dataset instance in the provided path, as a csv file. Note that internal identifiers, such as the row id (rid), user internal id (uid) and item internal id (iid) are never persisted, since they're only useful during runtime.

Parameters

- **path** – A string that represents the path where the current dataset values will be persisted.
- **columns** – An optional list with the names of the columns that should be persisted. Default: all columns.
- **write_header** – A boolean indicating whether to write the csv header on the persisted file. Default: False.

Returns None.

select(*query*, *copy=True*)
Select rows from the InteractionDataset.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".
- **copy** – A boolean indicating whether to create a new InteractionDataset where the rows that satisfy the provided query are put, or if the filtered rows should be removed from the current InteractionDataset (copy=False). Default: True.

Returns An instance of a InteractionDataset containing the rows selected by the provided query.

select_item_interaction_vec(*iid*)
Compute the item interaction vector for the provided item internal id.

Parameters **iid** – The item internal id that references the item that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each user to the provided item, in order) of the provided iid.

select_one(*query*, *columns=None*, *to_list=False*)
Select the first resulting row for the provided query.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".
- **columns** – A list with the column names to be kept on the resulting record. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns:

select_random_generator(*query=None*)
Provides a generator that yields dataset rows.

Parameters **query** – A string representing the query to be run before selecting random rows. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".

Returns A generator that yields dataset rows, where each row is represented as a dict.

select_user_interaction_vec(*uid*)
Compute the user interaction vector for the provided user internal id.

Parameters **uid** – The user internal id that references the user that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each item to the provided user, in order) of the provided uid.

uid_to_user(*uid*)
Converts a given internal user id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **uid** – The user internal id.

Returns An integer value representing the user raw id, or None if the internal user id provided does not exist.

unique(*columns=None*, *copy=True*)
Return a new InteractionDataset instance containing only the unique values on the provided column combination.

Parameters

- **columns** – The column combination to take into account when computing unique values. Default: all.
- **copy** – A boolean indicating whether a copy of the InteractionDataset should be made, or if it should be modified in-place. Default: True.

Returns A InteractionDataset instance containing the unique values on the provided column combination.

user_to_uid(*user*)
Converts a given raw user id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **user** – The user raw id.

Returns An integer value representing the user internal id, or None if the raw user id provided does not exist.

values(*columns=None*, *to_list=False*)
Provides a generator that yields all the records present in the dataset.

Parameters

- **columns** – The list of columns that should be returned for each data point. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns A generator that yields records present in the dataset.

values_list(*columns=None*, *to_list=False*)
Provides list with all the records present in the dataset.

Parameters

- **columns** – The list of columns that should be returned for each data point. Default: None (show all).
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns A list containing all records present in the dataset.

4.2 DRecPy.Dataset.dataset_factory module

class DRecPy.Dataset.dataset_factory.**InteractionsDatasetFactory**
Bases: object

InteractionsDatasetFactory creates InteractionDataset instances.

Parameters

- **path** – A string representing the path to the file where the dataset is located at.
- **columns** – A list with the names of the columns present on the dataset, ordered accordingly to the column order present in the dataset file. Required column names: 'user', 'item', 'interaction'.
- **delimiter** – A string representing the delimiter used on the dataset file. Default: ','.
- **has_header** – A boolean indicating whether the dataset file has a header row or not (skip first row or not?). Default: false.
- **in_memory** – A boolean indicating whether to load the dataset: in memory or out of memory. Default: True.
- **verbose** – A boolean indicating whether to log info messages or not. Default: True.

static read_df(*df*, *user_label='user'*, *item_label='item'*, *interaction_label='interaction'*, ***kwds*)
Convert the provided dataframe into a InteractionDataset instance.

Parameters

- **df** – A dataframe containing the dataset to be imported.
- **user_label** – The name of the column containing the user identifiers. Default: 'user'.
- **item_label** – The name of the column containing the item identifiers. Default: 'item'.
- **interaction_label** – The name of the column containing the interaction values. Default: 'interaction'.

Returns A InteractionDataset instance containing the provided data.

4.3 DRecPy.Dataset.mem_dataset module

class DRecPy.Dataset.mem_dataset.**MemoryInteractionDataset**(*path=''*, *columns=None*, ***kwds*)
Bases: *DRecPy.Dataset.dataset_abc.InteractionDatasetABC*

apply(*column*, *function*)
Modifies the current dataset instance by applying a transformation to a specific column in every row.

Parameters

- **column** – A string that represents the name of the column that will be transformed.
- **function** – The function that will be used to map the current column value in each row to the new one.

Returns None.

assign_internal_ids()
Assigns user and item internal ids. Internal ids are integer consecutive identifiers that represent each user or item uniquely. Two new columns are created on this dataset instance: "uid" and "iid", for user internal id and item internal id, respectively.

Returns None.

copy()
Copies the current dataset instance into a new one.

Returns InteractionDataset instance with the same data values as the current one.

drop(*record_ids*, *copy=True*, *keep=False*)
Remove (or keep) the provided list of record ids from the current InteractionDataset instance.

Parameters

- **record_ids** – A list of integers representing record ids.
- **copy** – A boolean indicating whether to create a new InteractionDataset instance to remove (or keep) the provided list of record ids, or if the current InteractionDataset instance should be modified accordingly (copy=False). Default: True.
- **keep** – A boolean indicating whether the provided record ids should be kept or removed (keep=False). Default: False.

Returns An instance of the InteractionDataset with (or without) the filtered records.

iid_to_item(*iid*)
Converts a given internal item id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **iid** – The item internal id.

Returns An integer value representing the item raw id, or None if the internal item id provided does not exist.

item_to_iid(*item*)
Converts a given raw item id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **item** – The item raw id.

Returns An integer value representing the item internal id, or None if the raw item id provided does not exist.

max(*column=None*)
Computes the maximum value for the provided column.

Parameters **column** – The name of the column for which the maximum should be computed.

Returns The maximum value present on the whole dataset, for the provided column name.

min(*column=None*)
Computes the minimum value for the provided column.

Parameters **column** – The name of the column for which the minimum should be computed.

Returns The minimum value present on the whole dataset, for the provided column name.

null_interaction_pair_generator(*interaction_threshold=None*, *seed=None*)
Provides a generator that yields negative / null interaction pairs.

Parameters

- **interaction_threshold** – An optional integer that is used as the boundary interaction value between positive and negative interaction pairs. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. If none is provided, positive interactions are the ones present on the dataset, and all the others are considered negative. Default: None.
- **seed** – An optional integer to be used as the seed value for the pseudo-random number generated used to sample null interaction pairs. Default: None.

Returns A generator that yields negative / null interaction pairs, that is, (user internal id, item internal id) tuples.

remove_internal_ids()
Removes user and item internal ids.

Returns None.

save(*path=''*, *columns=None*, *write_header=False*)
Persists the current dataset instance in the provided path, as a csv file. Note that internal identifiers, such as the row id (rid), user internal id (uid) and item internal id (iid) are never persisted, since they're only useful during runtime.

Parameters

- **path** – A string that represents the path where the current dataset values will be persisted.
- **columns** – An optional list with the names of the columns that should be persisted. Default: all columns.
- **write_header** – A boolean indicating whether to write the csv header on the persisted file. Default: False.

Returns None.

select(*query*, *copy=True*)
Select rows from the InteractionDataset.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".
- **copy** – A boolean indicating whether to create a new InteractionDataset where the rows that satisfy the provided query are put, or if the filtered rows should be removed from the current InteractionDataset (copy=False). Default: True.

Returns An instance of a InteractionDataset containing the rows selected by the provided query.

select_item_interaction_vec(*iid*)
Compute the item interaction vector for the provided item internal id.

Parameters **iid** – The item internal id that references the item that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each user to the provided item, in order) of the provided iid.

select_one(*query*, *columns=None*, *to_list=False*)
Select the first resulting row for the provided query.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".
- **columns** – A list with the column names to be kept on the resulting record. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns:

select_random_generator(*query=None*, *seed=None*)
Provides a generator that yields dataset rows.

Parameters **query** – A string representing the query to be run before selecting random rows. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".

Returns A generator that yields dataset rows, where each row is represented as a dict.

select_user_interaction_vec(*uid*)
Compute the user interaction vector for the provided user internal id.

Parameters **uid** – The user internal id that references the user that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each item to the provided user, in order) of the provided uid.

uid_to_user(*uid*)
Converts a given internal user id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **uid** – The user internal id.

Returns An integer value representing the user raw id, or None if the internal user id provided does not exist.

unique(*columns=None*, *copy=True*)
Return a new InteractionDataset instance containing only the unique values on the provided column combination.

Parameters

- **columns** – The column combination to take into account when computing unique values. Default: all.
- **copy** – A boolean indicating whether a copy of the InteractionDataset should be made, or if it should be modified in-place. Default: True.

Returns A InteractionDataset instance containing the unique values on the provided column combination.

user_to_uid(*user*)
Converts a given raw user id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **user** – The user raw id.

Returns An integer value representing the user internal id, or None if the raw user id provided does not exist.

values(*columns=None*, *to_list=False*)
Provides a generator that yields all the records present in the dataset.

Parameters

- **columns** – The list of columns that should be returned for each data point. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns A generator that yields records present in the dataset.

4.4 DRecPy.Dataset.db_dataset module

class DRecPy.Dataset.db_dataset.**DatabaseInteractionDataset**(*path=''*, *columns=None*, ***kwds*)
Bases: *DRecPy.Dataset.dataset_abc.InteractionDatasetABC*

apply(*column*, *function*)
Modifies the current dataset instance by applying a transformation to a specific column in every row.

Parameters

- **column** – A string that represents the name of the column that will be transformed.
- **function** – The function that will be used to map the current column value in each row to the new one.

Returns None.

assign_internal_ids()
Assigns user and item internal ids. Internal ids are integer consecutive identifiers that represent each user or item uniquely. Two new columns are created on this dataset instance: "uid" and "iid", for user internal id and item internal id, respectively.

Returns None.

close()
Cleanup method to delete temporary database files when they're not in use anymore.

copy()
Copies the current dataset instance into a new one.

Returns InteractionDataset instance with the same data values as the current one.

count_unique(*columns=None*)
Count the number of unique values on the provided column combination.

Parameters **columns** – A list containing the columns to take into account. Default: all.

Returns The count of unique values present on the provided column combination.

drop(*record_ids*, *copy=True*, *keep=False*)
Remove (or keep) the provided list of record ids from the current InteractionDataset instance.

Parameters

- **record_ids** – A list of integers representing record ids.
- **copy** – A boolean indicating whether to create a new InteractionDataset instance to remove (or keep) the provided list of record ids, or if the current InteractionDataset instance should be modified accordingly (copy=False). Default: True.
- **keep** – A boolean indicating whether the provided record ids should be kept or removed (keep=False). Default: False.

Returns An instance of the InteractionDataset with (or without) the filtered records.

iid_to_item(*iid*)
Converts a given internal item id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **iid** – The item internal id.

Returns An integer value representing the item raw id, or None if the internal item id provided does not exist.

item_to_iid(*item*)
Converts a given raw item id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **item** – The item raw id.

Returns An integer value representing the item internal id, or None if the raw item id provided does not exist.

max(*column=None*)
Computes the maximum value for the provided column.

Parameters **column** – The name of the column for which the maximum should be computed.

Returns The maximum value present on the whole dataset, for the provided column name.

min(*column=None*)
Computes the minimum value for the provided column.

Parameters **column** – The name of the column for which the minimum should be computed.

Returns The minimum value present on the whole dataset, for the provided column name.

null_interaction_pair_generator(*interaction_threshold=None*, *seed=None*)
Provides a generator that yields negative / null interaction pairs.

Parameters

- **interaction_threshold** – An optional integer that is used as the boundary interaction value between positive and negative interaction pairs. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. If none is provided, positive interactions are the ones present on the dataset, and all the others are considered negative. Default: None.
- **seed** – An optional integer to be used as the seed value for the pseudo-random number generated used to sample null interaction pairs. Default: None.

Returns A generator that yields negative / null interaction pairs, that is, (user internal id, item internal id) tuples.

remove_internal_ids()
Removes user and item internal ids.

Returns None.

save(*path=''*, *columns=None*, *write_header=False*)
Persists the current dataset instance in the provided path, as a csv or sqlite file. Note that internal identifiers, such as the row id (rid), user internal id (uid) and item internal id (iid) are never persisted, since they're only useful during runtime.

Parameters

- **path** – A string that represents the path where the current dataset values will be persisted. If it ends in ".sqlite" then a sqlite db file will be persisted in the provided path. Otherwise a csv file will be persisted.
- **columns** – An optional list with the names of the columns that should be persisted. Default: all columns.
- **write_header** – A boolean indicating whether to write the csv header on the persisted file. Default: False.

Returns None.

select(*query*, *copy=True*)
Select rows from the InteractionDataset.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ',. E.g. "user == '123', interaction > 3.5".
- **copy** – A boolean indicating whether to create a new InteractionDataset where the rows that satisfy the provided query are put, or if the filtered rows should be removed from the current InteractionDataset (copy=False). Default: True.

Returns An instance of a InteractionDataset containing the rows selected by the provided query.

select_item_interaction_vec(*iid*)
Compute the item interaction vector for the provided item internal id.

Parameters **iid** – The item internal id that references the item that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each user to the provided item, in order) of the provided iid.

select_one(*query*, *columns=None*, *to_list=False*)
Select the first resulting row for the provided query.

Parameters

- **query** – A string representing the query to be run. The query format should be: "column_name operator value", where extra conditions should be separated by a ',. E.g. "user == '123', interaction > 3.5".
- **columns** – A list with the column names to be kept on the resulting record. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns:

select_random_generator(*query=None*, *seed=None*)
Provides a generator that yields dataset rows.

Parameters **query** – A string representing the query to be run before selecting random rows. The query format should be: "column_name operator value", where extra conditions should be separated by a ','. E.g. "user == '123', interaction > 3.5".

Returns A generator that yields dataset rows, where each row is represented as a dict.

select_user_interaction_vec(*uid*)
Compute the user interaction vector for the provided user internal id.

Parameters **uid** – The user internal id that references the user that should have its interaction vector computed.

Returns The interaction vector (a vector containing the interaction values of each item to the provided user, in order) of the provided uid.

uid_to_user(*uid*)
Converts a given internal user id into its correspondent raw id. Raises exception if no internal ids are assigned.

Parameters **uid** – The user internal id.

Returns An integer value representing the user raw id, or None if the internal user id provided does not exist.

unique(*columns=None*, *copy=True*)
Return a new InteractionDataset instance containing only the unique values on the provided column combination.

Parameters

- **columns** – The column combination to take into account when computing unique values. Default: all.
- **copy** – A boolean indicating whether a copy of the InteractionDataset should be made, or if it should be modified in-place. Default: True.

Returns A InteractionDataset instance containing the unique values on the provided column combination.

user_to_uid(*user*)
Converts a given raw user id into its correspondent internal id. Raises exception if no internal ids are assigned.

Parameters **user** – The user raw id.

Returns An integer value representing the user internal id, or None if the raw user id provided does not exist.

values(*columns=None*, *to_list=False*)
Provides a generator that yields all the records present in the dataset.

Parameters

- **columns** – The list of columns that should be returned for each data point. Default: all.
- **to_list** – A boolean indicating whether each data point should be returned as a dict or as a list. Default: False.

Returns A generator that yields records present in the dataset.

4.5 DRecPy.Dataset.integrated_datasets module

class DRecPy.Dataset.integrated_datasets.**DatasetReadConfig**(*url*, *full_file*, *columns*, *delimiter*, *encoding='utf8'*, *train_file=None*, *test_file=None*, *unzip_folder=None*, *has_header=False*, *unzip=True*)

Bases: object

DRecPy.Dataset.integrated_datasets.**available_datasets**()
Returns a list of the datasets available to download.

DRecPy.Dataset.integrated_datasets.**download_dataset**(*ds_name*)
Download the dataset with name passed as argument.

DRecPy.Dataset.integrated_datasets.**get_dataset**(*ds_name*, *path*, *is_generated=False*, *force_out_of_memory=False*, *verbose=True*, ***kwds*)
Returns an InteractionDataset containing the data present in the path argument, and uses the settings defined for the dataset specified in the ds_name argument. Downloads the dataset if is not already stored.

DRecPy.Dataset.integrated_datasets.**get_full_dataset**(*ds_name*, *force_out_of_memory=False*, *verbose=True*, ***kwds*)
Gets a full dataset. Might download the dataset if it hasn't been downloaded before.

Parameters

- **ds_name** – A string with the name of the requested dataset. This name should be present in the list returned by available_datasets(), otherwise an error will be thrown.
- **force_out_of_memory** – A boolean indicating whether to force dataset loading to out of memory. Default: False.
- **verbose** – A boolean indicating whether to log info messages or not. Default: True.

Returns A InteractionDataset containing the dataset.

DRecPy.Dataset.integrated_datasets.**get_test_dataset**(*ds_name*, *force_out_of_memory=False*, *verbose=True*, ***kwds*)
Gets a test dataset. If the named dataset does not have a specific test file (example: BX dataset), a test InteractionDataset will be created using leave_k_out() from the Evaluation module on the full dataset. The split is deterministic (i.e. has a defined seed value). Might download the dataset if it hasn't been downloaded before.

Parameters

- **ds_name** – A string with the name of the requested dataset. This name should be present in the list returned by available_datasets(), otherwise an error will be thrown.
- **force_out_of_memory** – A boolean indicating whether to force dataset loading to out of memory. Default: False.
- **verbose** – A boolean indicating whether to log info messages or not. Default: True.

Returns A InteractionDataset containing the test dataset.

DRecPy.Dataset.integrated_datasets.**get_train_dataset**(*ds_name*, *force_out_of_memory=False*, *verbose=True*, ***kwds*)

Gets a train dataset. If the named dataset does not have a specific train file (example: BX dataset), a train InteractionDataset will be created using leave_k_out() from the Evaluation module on the full dataset. The split is deterministic (i.e. has a defined seed value). Might download the dataset if it hasn't been downloaded before.

Parameters

- **ds_name** – A string with the name of the requested dataset. This name should be present in the list returned by available_datasets(), otherwise an error will be thrown.
- **force_out_of_memory** – A boolean indicating whether to force dataset loading to out of memory. Default: False.
- **verbose** – A boolean indicating whether to log info messages or not. Default: True.

Returns A InteractionDataset containing the train dataset.

CHAPTER 5

DRecPy.Recommender package

5.1 DRecPy.Recommender.Baseline package

5.1.1 DRecPy.Recommender.Baseline.base_knn module

class DRecPy.Recommender.Baseline.base_knn.**BaseKNN**(*k=20*, *m=5*, *sim_metric='adjusted_cosine'*, *aggregation='weighted_mean'*, *shrinkage=100*, *use_averages=False*, ***kwds*)

Bases: *DRecPy.Recommender.recommender_abc.RecommenderABC*, abc.ABC

Base Collaborative Filtering recommender abstract class.

This class implements the skeleton methods for building a basic neighbour-based CF. The following methods are still required to be implemented: _fit(): should fit the model. _predict_default(): should return the default prediction value that is used when a minimum number of neighbours is not found. Only used when use_averages=True.

k
An integer representing the number of neighbours used to make a prediction. Default: 20.

m
An integer representing the minimum number of co-rated users/items required to validate the similarity value (if not valid, sim. value is set to 0). Default: 5.

sim_metric
Optional string representing the name of the similarity metric to use. Supported: 'adjusted_cosine', 'cosine', 'cosine_cf', 'jaccard', 'msd', 'pearson'. Default: 'adjusted_cosine'.

aggregation
Optional string representing the name of the aggregation approach to use. Supported: 'mean', 'weighted_mean'. Default: 'weighted_mean'.

shrinkage
Optional integer representing the discounting factor for computing the similarity between items / users (discounts less when #co-ratings increases). Default: 100.

use_averages
Optional boolean indicating whether to use item (for UserKNN) or user (for ItemKNN) averages when no neighbours are found. Default: True.

5.1.2 DRecPy.Recommender.Baseline.item_knn module

class DRecPy.Recommender.Baseline.item_knn.**ItemKNN**(***kwds*)
Bases: *DRecPy.Recommender.Baseline.base_knn.BaseKNN*

Item-based KNN Collaborative Filtering recommender model.

Implementation of a basic item neighbour-based CF.

Public Methods: fit(), predict(), recommend(), rank().

Attributes: See parent object BaseKNN obj:*DRecPy.Recommender.Baseline.BaseKNN*

5.1.3 DRecPy.Recommender.Baseline.user_knn module

class DRecPy.Recommender.Baseline.user_knn.**UserKNN**(***kwds*)
Bases: *DRecPy.Recommender.Baseline.base_knn.BaseKNN*

User-based KNN Collaborative Filtering recommender model.

Implementation of a basic user neighbour-based CF.

Public Methods: fit(), predict(), recommend(), rank().

Attributes: See parent object BaseKNN

5.2 DRecPy.Recommender.cdae module

Implementation of the CDAE model (Collaborative Denoising Auto-Encoder). Paper: Wu, Yao, et al. "Collaborative denoising auto-encoders for top-n recommender systems." Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 2016.

Note: gradients are evaluated for all output units (the output unit selection step discussed in the paper is not done).

class DRecPy.Recommender.cdae.**CDAE**(*hidden_factors=50*, *corruption_level=0.2*, *loss='bce'*, ***kwds*)
Bases: *DRecPy.Recommender.recommender_abc.RecommenderABC*

Collaborative Denoising Auto-Encoder (CDAE) recommender model.

Parameters

- **hidden_factors** – An integer defining the number of units for the hidden layer.
- **corruption_level** – A decimal value representing the level of corruption to apply to the given interactions / ratings during training.
- **loss** – A string that represents the loss function used to optimize the model. Supported: mse, bce. Default: bce.

For more arguments, refer to the base class: DRecPy.Recommender.RecommenderABC.

5.3 DRecPy.Recommender.dmf module

Implementation of the DMF model (Deep Matrix Factorization) Paper: Xue, Hong-Jian, et al. "Deep Matrix Factorization Models for Recommender Systems." IJCAI. 2017.

class DRecPy.Recommender.dmf.**DMF**(*user_factors=None*, *item_factors=None*, *use_nce=True*, *l2_norm_vectors=True*, ***kwds*)

Bases: *DRecPy.Recommender.recommender_abc.RecommenderABC*

Deep Matrix Factorization (DMF) recommender model.

Parameters

- **user_factors** – A list containing the number of hidden neurons in each layer of the user NN. Default: [64, 32].
- **item_factors** – A list containing the number of hidden neurons in each layer of the item NN. Default: [64, 32].
- **use_nce** – A boolean indicating whether to use the normalized cross-entropy described in the paper as the loss function or the regular cross-entropy. Default: true.
- **l2_norm_vectors** – A boolean indicating if user and item interaction vectors should be l2 normalized before being used as input for their respective NNs or not. Default: true.

For more arguments, refer to the base class: DRecPy.Recommender.RecommenderABC.

5.4 DRecPy.Recommender.caser module

Implementation of the Caser model. Paper: Tang, Jiaxi, and Ke Wang. "Personalized top-n sequential recommendation via convolutional sequence embedding." Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. 2018.

class DRecPy.Recommender.caser.**Caser**(*L=5*, *T=3*, *d=50*, *n_v=4*, *n_h=16*, *act_h=<function relu>*, *act_mlp=<function relu>*, *dropout_rate=0.5*, *sort_column='timestamp'*, ***kwds*)

Bases: *DRecPy.Recommender.recommender_abc.RecommenderABC*

Caser recommender model.

Parameters

- **L** – An integer representing the sequence length. Default: 5.
- **T** – An integer representing the number of targets. Default: 3.
- **d** – An integer representing the number of latent dimensions. Default: 50.
- **n_v** – An integer representing the number of vertical filters. Default: 4.
- **n_h** – An integer representing the number of horizontal filters. Default: 16.
- **act_h** – The activation function used for the horizontal convolutional layer. Default: tf.nn.relu.
- **act_mlp** – The activation function used for the dense layer. Default: tf.nn.relu.
- **dropout_rate** – The dropout ratio when performing dropout between the convolutional and dense layers. Default: 0.5.

- **sort_column** – An optional string representing the name of the column used to sort the sequence records. If none is provided, the natural order (present in the data set) will be preserved. Default: 'timestamp'.

For more arguments, refer to the base class: `DRecPy.Recommender.RecommenderABC`.

5.5 DRecPy.Recommender.recommender_abc module

class DRecPy.Recommender.recommender_abc.**RecommenderABC**(***kwds*)

Bases: abc.ABC

Base recommender abstract class.

This class implements the skeleton methods required for building a recommender. It provides id-abstraction (handles conversion between raw to internal ids, auto identifier validation (if a given user/item is known or not), automatic progress logging, weight updates, tracking loss per epoch and support for other features such as epoch callbacks and early stopping. It has a structure that allows it to be fully extensible, whilst promoting model specific behavior for improved flexibility. All private methods are called with internal ids only, and all public methods must be called with raw ids only.

The following methods are still required to be implemented: _pre_fit(), _sample_batch(), _predict_batch(), _compute_batch_loss() and _predict(). If there are no trainable variables set during the _pre_fit(), batch training is skipped (useful for non-deep learning models). Trainable variables can be registered via _register_trainable() or _register_trainables().

If the provided trainable variables are of type tf.keras.models.Model or tf.keras.layers.Layer, then the regularizers added when instantiating these variables (e.g. via kernel_regularizer attribute of a layer) will be used to automatically compute the regularization loss. To add a custom regularization loss (or if the implemented model uses tf.Variables), implement the self._compute_reg_loss().

Optionally, these methods can be overridden: _rank() and _recommend().

Parameters

- **verbose** – Optional boolean indicating if the recommender should print progress logs or not. Default: True.
- **log_file** – Optional boolean indicating if a file containing all produced logs should be created or not. It will be created on the current directory, following the pattern: drecpy__DATE_TIME_RECNAME.log. Default: False.
- **interaction_threshold** – An optional integer that is used as the boundary interaction value between positive and negative interaction pairs. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. Default: 0.001.
- **seed** (*max_rating*) – Optional integer representing the seed value for the model pseudo-random number generator. Default: None.

fit(*interaction_dataset*, *epochs=50*, *batch_size=32*, *learning_rate=0.001*, *neg_ratio=5*, *reg_rate=0.001*, *copy_dataset=False*, ***kwds*)

Processes the provided dataframe and builds id-abstraction, infers min. and max. interactions (if not passed through constructor) and calls _fit() to fit the current model.

Parameters

- **interaction_dataset** – A interactionsDataset instance containing the training data.
- **epochs** – Optional number of epochs to train the model. Default: 50.

- **batch_size** – Optional number of data points to use for each epoch to train the model. Default: 32.
- **learning_rate** – Optional decimal representing the learning rate of the model. Default: 0.001.
- **neg_ratio** – Optional integer that represents the number of negative instances for each positive one. Default: 5.
- **reg_rate** – Optional decimal representing the model regularization rate. Default: 0.01.
- **epoch_callback_fn** – Optional function that is called, for each epoch_callback_freq, with the model at its current state. It receives one argument - the model at its current state - and should return a dict mapping each metric's name to the corresponding value. The results will be displayed in a graph at the end of the model fit and during the fit process on the logged progress bar description only if verbose is set to True. Default: None
- **epoch_callback_freq** – Optional integer representing the frequency in which the epoch_callback_fn is called. If epoch_callback_fn is not defined, this parameter is ignored. Default: 5 (called every 5 epochs).
- **early_stopping_rule** – Optional instance of EarlyStoppingRuleABC that will be used to compute the early stopping epoch and according to how the rule works, it might stop the training before achieving the total number of epochs. Since this uses epoch callbacks, the rule will only be used if an epoch_callback_fn is defined. Default: None
- **early_stopping_freq** – Optional integer representing the frequency in which the early_stopping_rule is computed. If early_stopping_rule is not defined, this parameter is ignored. Default: 5 (called every 5 epochs).
- **copy_dataset** – Optional boolean indicating weather a copy of the given dataset should be made. If set to False, the given dataset instance is used. Default: False.
- **optimizer** – Optional instance of a tf/keras optimizer that will be used even if there's a model specific optimizer. Default: Adam optimizer with the learning rate set with the value provided in the learning_rate argument; if there's a model specific optimizer (set during the model's _pre_fit), this default optimizer will not be used.

Returns None.

static load(*load_path*)
Load/import a saved/exported model.

Parameters **load_path** – A string that represents the path to the saved/exported model.

Returns Recommender model.

predict(*user_id*, *item_id*, *skip_errors=False*, ***kwds*)
Performs a prediction using the provided user_id and item_id.

Parameters

- **user_id** – An integer representing the raw user id.
- **item_id** – An integer representing the raw item id.
- **skip_errors** – A boolean that controls if errors should be avoided or if they should be be thrown. Default: False. An example would be calling predict(None, None): If skip_errors is True, then it would return None; else it would throw an error.

Returns A float value representing the predicted interaction for the provided item, user pair. Or None, if an error occurs and skip_errors = True.

rank(*user_id*, *item_ids*, *novelty=True*, *skip_invalid_items=True*, ***kwds*)
Ranks the provided item list for the given user and with the requested characteristics.

Parameters

- **user_id** – A string or integer representing the user id.
- **item_ids** – A list of strings or integers representing the ids of the items to rank.
- **novelty** – Optional boolean indicating if we only novelty recommendations or not. Default: True.
- **n** – Optional integer representing the number of best items to return. Default: len(item_ids).
- **skip_invalid_items** – Optional boolean indicating if invalid items should be skipped. If set to False, will throw an exception when one is found. Default: True.

Returns A ranked item list in the form of (similarity, item) tuples.

recommend(*user_id*, *n=None*, *novelty=True*, *interaction_threshold=None*, ***kwds*)
Computes a recommendation list for the given user and with the requested characteristics.

Parameters

- **user_id** – A string or integer representing the user id.
- **n** – An integer representing the number of recommended items.
- **novelty** – An optional boolean indicating if we only novelty recommendations or not. Default: True.
- **interaction_threshold** – Optional float value that represents the value required to consider an item to be a useful recommendation Default: None.

Returns A list containing recommendations in the form of (similarity, item) tuples.

save(*save_path*)
Save/export the current model.

Parameters **save_path** – A string that represents the path in which the model will be saved.

Returns None.

CHAPTER 6

DRecPy.Sampler package

6.1 DRecPy.Sampler.point_sampler module

class DRecPy.Sampler.point_sampler.**PointSampler**(*interaction_dataset*, *neg_ratio*, *interaction_threshold=None*, *seed=None*)

Bases: object

PointSampler class that abstracts sampling positive and negative (user, item) interaction pairs.

Parameters

- **interaction_dataset** – An instance of the InteractionDataset type, representing the data set where points are being sampled from.
- **neg_ratio** – The number of negative interaction pairs to be sampled for each positive interaction pair.
- **interaction_threshold** – An optional integer that is used as the boundary interaction value between positive and negative interaction pairs. All values above or equal to interaction_threshold are considered positive, and all values bellow are considered negative. If none is provided, positive interactions are the ones present on the data set, and all the others are considered negative. Default: None.
- **seed** – An optional integer to be used as the seed value for the pseudo-random number generated used to sample interaction pairs. Default: None.

sample(*n=16*)

Sample positive and negative interaction pairs according to the set definitions of the PointSampler instance.

Parameters **n** – An integer representing the number of interaction pairs to be sampled. Default: 16.

Returns

A list with n triple entries, each representing either a negative or positive interaction pair. The first element on each triple represents the sampled uid (user), the second the sampled iid (item), and

the third the interaction value.

sample_negative()
Sample one negative interaction pair.

Returns

A triple representing a negative interaction pair. The first element on each triple represents the sampled uid (user), the second the sampled iid (item), and

the third the interaction value (which will always be 0).

sample_one()
Sample one positive or negative interaction pair according to the set definitions of the PointSampler instance.

Returns

A triple representing either a negative or positive interaction pair. The first element on each triple represents the sampled uid (user), the second the sampled iid (item), and

the third the interaction value.

sample_positive()
Sample one positive interaction pair.

Returns

A triple representing a positive interaction pair. The first element on each triple represents the sampled uid (user), the second the sampled iid (item), and

the third the interaction value.

CHAPTER 7

DRecPy.Evaluation package

7.1 DRecPy.Evaluation.Metrics package

7.1.1 DRecPy.Evaluation.Metrics.ranking module

class DRecPy.Evaluation.Metrics.ranking.**AveragePrecision**
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Average Precision at k.

class DRecPy.Evaluation.Metrics.ranking.**DCG**(*strong_relevancy=True*)
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Discounted Cumulative Gain at k.

Parameters **strong_relevancy** – An optional boolean indicating which variant of the DCG is used. If set to True, usually results in smaller values than when it's set to False. Default: True.

class DRecPy.Evaluation.Metrics.ranking.**FScore**(*beta=1*)
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

F-score at k.

Parameters **beta** – An optional integer representing the weight of the recall value on the combined score. Beta > 1 favours recall over precision, while beta < 1 favours precision over recall. Default: 1.

class DRecPy.Evaluation.Metrics.ranking.**HitRatio**
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Hit Ratio at k.

class DRecPy.Evaluation.Metrics.ranking.**NDCG**(*strong_relevancy=True*)
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Discounted Cumulative Gain at k.

Parameters **strong_relevancy** – An optional boolean indicating which variant of the DCG is used. If set to True, usually results in smaller values than when it's set to False. Default: True.

class DRecPy.Evaluation.Metrics.ranking.**Precision**
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Precision at k.

class DRecPy.Evaluation.Metrics.ranking.**RankingMetricABC**
Bases: DRecPy.Evaluation.Metrics.metric_abc.MetricABC

class DRecPy.Evaluation.Metrics.ranking.**Recall**
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Recall at k.

class DRecPy.Evaluation.Metrics.ranking.**ReciprocalRank**
Bases: *DRecPy.Evaluation.Metrics.ranking.RankingMetricABC*

Reciprocal Rank at k.

7.1.2 DRecPy.Evaluation.Metrics.regression module

class DRecPy.Evaluation.Metrics.regression.**MAE**
Bases: *DRecPy.Evaluation.Metrics.regression.PredictiveMetricABC*

Mean absolute Error.

class DRecPy.Evaluation.Metrics.regression.**MSE**
Bases: *DRecPy.Evaluation.Metrics.regression.PredictiveMetricABC*

Mean Squared Error.

class DRecPy.Evaluation.Metrics.regression.**PredictiveMetricABC**
Bases: DRecPy.Evaluation.Metrics.metric_abc.MetricABC

class DRecPy.Evaluation.Metrics.regression.**RMSE**
Bases: *DRecPy.Evaluation.Metrics.regression.PredictiveMetricABC*

Root Mean Squared Error.

7.2 DRecPy.Evaluation.Splits package

7.2.1 DRecPy.Evaluation.Splits.leave_k_out module

DRecPy.Evaluation.Splits.leave_k_out.**leave_k_out**(*interaction_dataset*, *k=1*, *min_user_interactions=0*, *last_timestamps=False*, *timestamp_label='timestamp'*, *seed=0*, *max_concurrent_threads=4*, ***kwds*)

Dataset split method that uses a leave k out strategy. More specifically, for each user with more than k interactions, k interactions are randomly selected and taken out from the train set and put into the test set. This means that there are never users present in the test set that are not present in the train set. Also, users without at least min_user_interactions interactions, will not be sampled in either sets (i.e. they're removed). This function is not thread-safe (i.e. concurrent calls might produce unexpected results). Instead of trying this, increase the max_concurrent_threads argument to speed up the process (if you've the available cores).

Parameters

- **interaction_dataset** – A InteractionDataset instance containing the user-item interactions.
- **k** – Optional integer or float value: if k is an integer, then it represents the number of interactions, per user, to use in the test set (and to remove from the train set); if k is a float value (and between 0 and 1), it represents the percentage of interactions, per user, to use in the test set (and to remove from the train set). Default: 1.
- **min_user_interactions** – Optional integer that represents the minimum number of interactions each user needs to have to be included in the train or test set. Default: 0.
- **last_timestamps** – Optional boolean that indicates whether the test records should be sampled by last timestamps (using the column with the name passed in the timestamp_label argument). Default: False.
- **timestamp_label** – Optional string that corresponds to the name of the timestamp column on the interaction_dataset. This is only used when the last_timestamps argument is set to True. Default: 'timestamp'.
- **max_concurrent_threads** – An optional integer representing the max concurrent threads to use. Default: 4.
- **seed** – An integer that is used as a seed value for the pseudorandom number generator. Default: 0.
- **verbose** – Optional boolean that indicates if a progress bar showing the splitting progress should be displayed or not. Default: True.

Returns the train and test interaction datasets in this order.

Return type Two InteractionDataset instances

7.2.2 DRecPy.Evaluation.Splits.matrix_split module

DRecPy.Evaluation.Splits.matrix_split.**matrix_split**(*interaction_dataset*, *user_test_ratio=0.2*, *item_test_ratio=0.2*, *min_user_interactions=0*, *seed=0*, *max_concurrent_threads=4*, ***kwds*)

Dataset split method that uses a matrix split strategy. More specifically, item_test_ratio items from user_test_ratio users are sampled out of the full dataset and moved to the test set, while the missing items and users make the training set. If all records for a given user are selected to be moved into the test set, the split for that user is skipped, and its records are kept in the train set. This function is not thread-safe (i.e. concurrent calls might produce unexpected results). Instead of trying this, increase the max_concurrent_threads argument to speed up the process (if you've the available cores).

Parameters

- **interaction_dataset** – A InteractionDataset instance containing the user-item interactions.
- **user_test_ratio** – Optional float value that represents the percentage of users to be sampled to the test set.
- **item_test_ratio** – Optional float value that represents the percentage of items to be sampled to the test set.

- **min_user_interactions** – Optional integer that represents the minimum number of interactions each user needs to have to be included in the train or test set. Default: 0.
- **max_concurrent_threads** – An optional integer representing the max concurrent threads to use. Default: 4.
- **seed** – An integer that is used as a seed value for the pseudorandom number generator. Default: 0.
- **verbose** – Optional boolean that indicates if a progress bar showing the splitting progress should be displayed or not. Default: True.

Returns the train and test interaction datasets in this order.

Return type Two InteractionDataset instances

7.2.3 DRecPy.Evaluation.Splits.random_split module

DRecPy.Evaluation.Splits.random_split.**random_split**(*interaction_dataset*, *test_ratio=0.25*, *seed=0*, ***kwds*)

Random split that creates a train set with (100-100*test_ratio)% of the total rows, and a test set with the other (100*test_ratio)% of the rows. No guarantees of users or items existing on both datasets are made, therefore cases like: user X exists on the test set but not on the train set might happen. The use of this split should be directed to models that support these types of behaviour.

Parameters

- **interaction_dataset** – A InteractionDataset instance containing the user-item interactions.
- **test_ratio** – A floating-point value representing the ratio of rows used for the test set. Default: 0.25.
- **seed** – An integer that is used as a seed value for the pseudorandom number generator. If none is given, no seed will be used. Default: 0.
- **verbose** – Optional boolean that indicates if a progress bar showing the splitting progress should be displayed or not. Default: True.

Returns the train and test interaction datasets in this order.

Return type Two InteractionDataset instances

7.3 DRecPy.Evaluation.Processes package

7.3.1 DRecPy.Evaluation.Processes.predictive_evaluation module

DRecPy.Evaluation.Processes.predictive_evaluation.**predictive_evaluation**(*model*, *ds_test=None*, *count_none_predictions=F*, *n_test_predictions=None*, *skip_errors=True*, ***kwds*)

Executes a predictive evaluation process, where the given model will be evaluated under the provided settings.

Parameters

- **model** – An instance of a Recommender to be evaluated.

- **ds_test** – An optional test InteractionDataset. If none is provided, then the test data will be the model training data. Evaluating on train data is not ideal for assessing the model's performance.
- **count_none_predictions** – An optional boolean indicating whether to count none predictions (i.e. when the model predicts None, count it as being a 0) or not (i.e. skip that user-item pair). Default: False.
- **n_test_predictions** – An optional integer representing the number of predictions to evaluate. Default: predict for every (user, item) pair on the test dataset.
- **skip_errors** – A boolean indicating whether to ignore errors produced during the predict calls, or not. Default: False.
- **metrics** – An optional list containing instances of PredictiveMetricABC. Default: [RMSE(), MSE()].
- **verbose** – A boolean indicating whether state logs should be produced or not. Default: true.

Returns A dict containing each metric name mapping to the corresponding metric value.

7.3.2 DRecPy.Evaluation.Processes.ranking_evaluation module

DRecPy.Evaluation.Processes.ranking_evaluation.**ranking_evaluation**(*model*, *ds_test=None*, *n_test_users=None*, *k=10*, *n_pos_interactions=None*, *n_neg_interactions=None*, *generate_negative_pairs=False*, *novelty=False*, *seed=0*, *max_concurrent_threads=4*, ***kwds*)

Executes a ranking evaluation process, where the given model will be evaluated under the provided settings. This function is not thread-safe (i.e. concurrent calls might produce unexpected results). Instead of trying this, increase the max_concurrent_threads argument to speed up the process (if you've the available cores).

Parameters

- **model** – An instance of a Recommender to be evaluated.
- **ds_test** – An optional test InteractionDataset. If none is provided, then the test data will be the model training data. Evaluating on train data is not ideal for assessing the model's performance.
- **n_test_users** – An optional integer representing the number of users to evaluate the produced rankings. Default: Number of unique users of the provided test dataset.
- **k** – An optional integer (or a list of integers) representing the truncation factor (keep the first k elements for each ranked list), which then affects the produced metric evaluation. Default: 10.
- **n_pos_interactions** – The number of positive interactions to sample into the list that is going to be ranked and evaluated for each user. If for a given user, there's less than n_pos_interactions positive interactions, the user's evaluation will be skipped. When this

argument is not provided, all positive interactions on the test set from each user will be sampled. Default: None.

- **n_neg_interactions** – The max. number of negative interactions to sample into the list that is going to be ranked and evaluated for each user. If a float value is provided, then the max. number of sampled negative interactions will be the percentage of positive interactions present on each user's test set. If this argument is not defined, the sampled negative interactions will be all negative interactions present on each user's test set. Default: None.
- **generate_negative_pairs** – An optional boolean that controls whether negative interaction pairs should also be generated (interaction pairs not present on the train or test data sets are also sampled) or not (i.e. only gathered from the test data set, where interaction values are bellow than the interaction_threshold). If this parameter is set to True, then the number of sampled negative interactions for each user will always match the n_neg_interactions. Default: False.
- **interaction_threshold** – The interaction value threshold to consider an interaction value positive or negative. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. Default: model.interaction_threshold.
- **novelty** – A boolean indicating whether only novel recommendations should be taken into account or not. Default: False.
- **metrics** – An optional list containing instances of RankingMetricABC. Default: [Precision(), Recall(), HitRatio(), NDCG()].
- **max_concurrent_threads** – An optional integer representing the max concurrent threads to use. Default: 4.
- **seed** – An optional, integer representing the seed for the random number generator used to sample positive and negative interaction pairs. Default: 0.
- **verbose** – A boolean indicating whether state logs and a final graph should be produced or not. Default: true.
- **block** – A boolean indicating whether the displayed graph block code execution or not. Note that this graph is only displayed when verbose=True. Default: true.

Returns A dict containing each metric name mapping to the corresponding metric value.

7.3.3 DRecPy.Evaluation.Processes.recommendation_evaluation module

DRecPy.Evaluation.Processes.recommendation_evaluation.**recommendation_evaluation**(*model*, *ds_test=None*, *n_test_users=N*, *k=10*, *n_pos_interacti*, *novelty=False*, *ignore_low_predi*, *seed=0*, *max_concurren*, ***kwds*)

Executes a recommendation evaluation process, where the given model will be evaluated under the provided settings. This function is not thread-safe (i.e. concurrent calls might produce unexpected results). Instead

of trying this, increase the max_concurrent_threads argument to speed up the process (if you've the available cores).

Parameters

- **model** – An instance of a Recommender to be evaluated.
- **ds_test** – An optional test InteractionDataset. If none is provided, then the test data will be the model training data. Evaluating on train data is not ideal for assessing the model's performance.
- **n_test_users** – An optional integer representing the number of users to evaluate the produced rankings. Defaults to the number of unique users of the provided test dataset.
- **k** – An optional integer (or a list of integers) representing the truncation factor (keep the first k elements for each ranked list), which then affects the produced metric evaluation. Default: 10.
- **n_pos_interactions** – The number of positive interactions to sample into the list that contains the positive items to be considered when evaluating the model provided recommendations, for each user. If for a given user, there's less than n_pos_interactions positive interactions, the user's evaluation will be skipped. When this argument is not provided, all positive interactions on the test set from each user will be sampled. Default: None.
- **interaction_threshold** – The interaction value threshold to consider an interaction value positive or negative. All values above or equal interaction_threshold are considered positive, and all values bellow are considered negative. Default: model.interaction_threshold.
- **novelty** – A boolean indicating whether only novel recommendations should be taken into account or not. Default: False.
- **ignore_low_predictions_threshold** – An optional value representing the interaction value threshold to consider a model prediction, if the item-prediction is higher or equal than the specified, or to skip it if it is less than the specified, i.e. remove those items with low model prediction from the recommendation list. Default: None.
- **metrics** – An optional list containing instances of RankingMetricABC. Default: [Precision(), Recall(), HitRatio(), NDCG()].
- **max_concurrent_threads** – An optional integer representing the max concurrent threads to use. Default: 4.
- **seed** – An optional, integer representing the seed for the random number generator used to sample positive and negative interaction pairs. Default: 0.
- **verbose** – A boolean indicating whether state logs and a final graph should be produced or not. Default: true.
- **block** – A boolean indicating whether the displayed graph block code execution or not. Note that this graph is only displayed when verbose=True. Default: true.

Returns A dict containing each metric name mapping to the corresponding metric value.

7.4 DRecPy.Evaluation.loss_tracker module

class DRecPy.Evaluation.loss_tracker.**LossTracker**

Bases: object

add_epoch_callback_result(*name*, *result*, *epoch*)
Adds a new epoch callback result.

Parameters

- **name** – A string representing the name of the evaluated metric.
- **result** – A number representing the evaluated value for the current metric.
- **epoch** – A number representing the epoch for which the evaluated metric showed the passed result.

add_epoch_loss(*loss*)
Adds a new epoch loss.

Parameters **loss** – The loss value obtained during the epoch.

display_graph(*model_name=None*, *stopping_epoch=None*, *block=False*)
Displays a graph containing the average batch loss per epoch, as well as the epoch callback results for each metric, if they exist.

Parameters

- **model_name** – A string representing the name of the model. If this is provided, the model name will be displayed on the title of the figure. Default: None.
- **stopping_epoch** – An integer representing the epoch for which early stopping has been applied and therefore all network weights were reverted to.
- **block** – A boolean indicating whether the displayed graph should block code execution or not. Default: False.

get_epoch_avg_loss()
Gets the current average epoch loss.

Returns The current average epoch loss, computed from the provided epoch losses.

reset_epoch_losses()
Resets the stored epoch losses and sets the epoch average loss to 0.

CHAPTER 8

Table of Contents

CHAPTER 9

Introduction

DRecPy is a Python framework that makes building deep learning based recommender systems easier, by making available various tools to develop and test new models.

The main key features DRecPy provides are listed bellow: - Support for **in-memory and out-of-memory data sets**, by using an intermediary data structure called InteractionDataset.

- **Auto Internal to raw id conversion**: a mapping from raw to internal identifiers is automatically built, so that datasets containing string ids or non-contiguous numeric ids are supported by all recommenders.
- **Support for multi-column data sets**, i.e. not being limited to (user, item, rating) triples, but also supporting other columns such as timestamp, session, location, etc.
- Well defined **workflow for model building** for developing deep learning-based recommenders (while also supporting non-deep learning-based recommenders).
- Support for **epoch callbacks** using custom functions, whose results are logged and displayed in a plot at the end of model training.
- **Early stopping** support using custom functions that can make use of previous epoch callback results or model loss values.
- **Data set splitting techniques** adjusted for the distinct nature of data sets dedicated for recommender systems.
- **Sampling techniques** for point based and list based models.
- **Evaluation processes** for predictive models, as well as for learn-to-rank models.
- Automatic **progress logging** and **plot generation for loss values during model training**, as well as test scores during model evaluation.
- **All methods with stochastic factors receive a seed parameter**, in order to allow result reproducibility.

For more information about the framework and its components, please visit the documentation page.

Here's a brief overview of the general call workflow for every recommender:

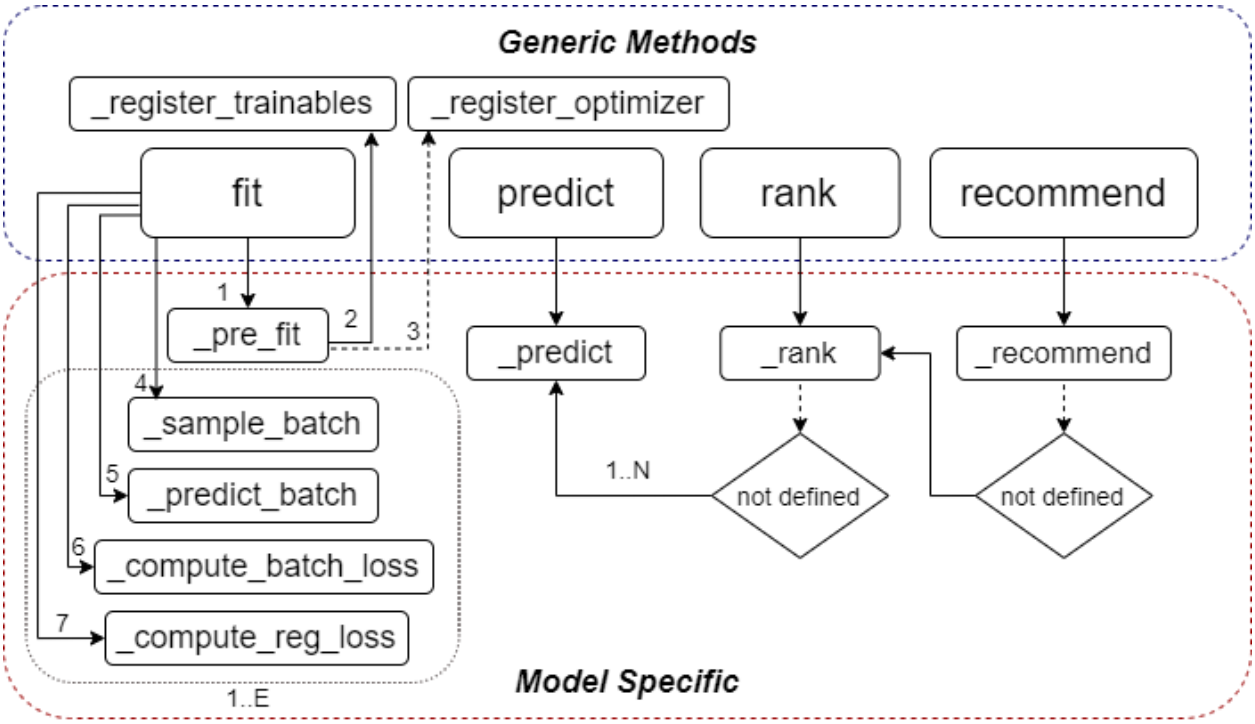

Fig. 1: Call Worlflow

CHAPTER 10

Installation

With pip:

```
$ pip install drecpy
```

If you can't get the latest version from PyPi:

```
$ pip install git+https://github.com/fabioiuri/DRecPy
```

Or directly by cloning the Git repo:

```
$ git clone https://github.com/fabioiuri/DRecPy
$ cd DRecPy
$ python setup.py install
```

CHAPTER 11

Getting Started

Here's an example script using one of the implemented recommenders (CDAE), to train, with a validation set, and evaluate its ranking performance on the MovieLens 100k data set.

```
from DRecPy.Recommender import CDAE
from DRecPy.Recommender.EarlyStopping import MaxValidationValueRule
from DRecPy.Dataset import get_train_dataset
from DRecPy.Dataset import get_test_dataset
from DRecPy.Evaluation.Processes import ranking_evaluation
from DRecPy.Evaluation.Splits import leave_k_out
from DRecPy.Evaluation.Metrics import NDCG
from DRecPy.Evaluation.Metrics import HitRatio
from DRecPy.Evaluation.Metrics import Precision
import time


ds_train = get_train_dataset('ml-100k')
ds_test = get_test_dataset('ml-100k')
ds_train, ds_val = leave_k_out(ds_train, k=1, min_user_interactions=10, seed=0)


def epoch_callback_fn(model):
    return {'val_' + metric: v for metric, v in
            ranking_evaluation(model, ds_val, n_pos_interactions=1, n_neg_
↪interactions=100,
                               generate_negative_pairs=True, k=10, verbose=False,␣
↪seed=10,
                               metrics=[HitRatio(), NDCG()]).items()}


start_train = time.time()
cdae = CDAE(hidden_factors=50, corruption_level=0.2, loss='bce', seed=10)
cdae.fit(ds_train, learning_rate=0.001, reg_rate=0.001, epochs=100, batch_size=64,␣
↪neg_ratio=5,
         epoch_callback_fn=epoch_callback_fn, epoch_callback_freq=10,
```

(continues on next page)

(continued from previous page)

```
        early_stopping_rule=MaxValidationValueRule('val_HitRatio'), early_stopping_
↪freq=10)
print("Training took", time.time() - start_train)

print(ranking_evaluation(cdae, ds_test, k=[1, 5, 10], novelty=True, n_pos_
↪interactions=1,
                         n_neg_interactions=100, generate_negative_pairs=True,␣
↪seed=10,
                         metrics=[HitRatio(), NDCG(), Precision()], max_concurrent_
↪threads=4, verbose=True))
```

Output:

```
Creating user split tasks: 100%|| 943/943 [00:00<00:00, 4704.11it/s]
Splitting dataset: 100%|| 943/943 [00:03<00:00, 296.04it/s]

[2020-09-02 00:13:37,764] (INFO) CDAE_CLOGGER: Max. interaction value: 5
[2020-09-02 00:13:37,764] (INFO) CDAE_CLOGGER: Min. interaction value: 0
[2020-09-02 00:13:37,764] (INFO) CDAE_CLOGGER: Interaction threshold value: 0.001
[2020-09-02 00:13:37,764] (INFO) CDAE_CLOGGER: Number of unique users: 943
[2020-09-02 00:13:37,765] (INFO) CDAE_CLOGGER: Number of unique items: 1680
[2020-09-02 00:13:37,765] (INFO) CDAE_CLOGGER: Number of training points: 89627
[2020-09-02 00:13:37,765] (INFO) CDAE_CLOGGER: Sparsity level: approx. 94.3426%
[2020-09-02 00:13:37,765] (INFO) CDAE_CLOGGER: Creating auxiliary structures...
[2020-09-02 00:13:37,833] (INFO) CDAE_CLOGGER: Number of registered trainable␣
↪variables: 5
Fitting model... Epoch 100 Loss: 0.1882 | val_HitRatio@10: 0.5493 | val_NDCG@10: 0.
↪3137 | MaxValidationValueRule best epoch: 80: 100%|| 100/100 [15:05<00:00, 29.77s/
↪it]
[2020-09-02 00:30:02,831] (INFO) CDAE_CLOGGER: Reverting network weights to epoch 80␣
↪due to the evaluation of the early stopping rule MaxValidationValueRule.
[2020-09-02 00:30:02,833] (INFO) CDAE_CLOGGER: Network weights reverted from epoch␣
↪100 to epoch 80.
[2020-09-02 00:30:02,979] (INFO) CDAE_CLOGGER: Model fitted.

Starting user evaluation tasks: 100%|| 943/943 [00:00<00:00, 2454.84it/s]
Evaluating model ranking performance:  99%|| 929/943 [02:16<00:02,  4.81it/s]

{'HitRatio@1': 0.1198, 'HitRatio@5': 0.3945, 'HitRatio@10': 0.5536, 'NDCG@1': 0.1198,
'NDCG@5': 0.2588, 'NDCG@10': 0.3103, 'Precision@1': 0.1198, 'Precision@5': 0.0789,
↪'Precision@10': 0.0554}
```

Generated Plots:

- Training
- Evaluation

More quick and easy examples are available here.

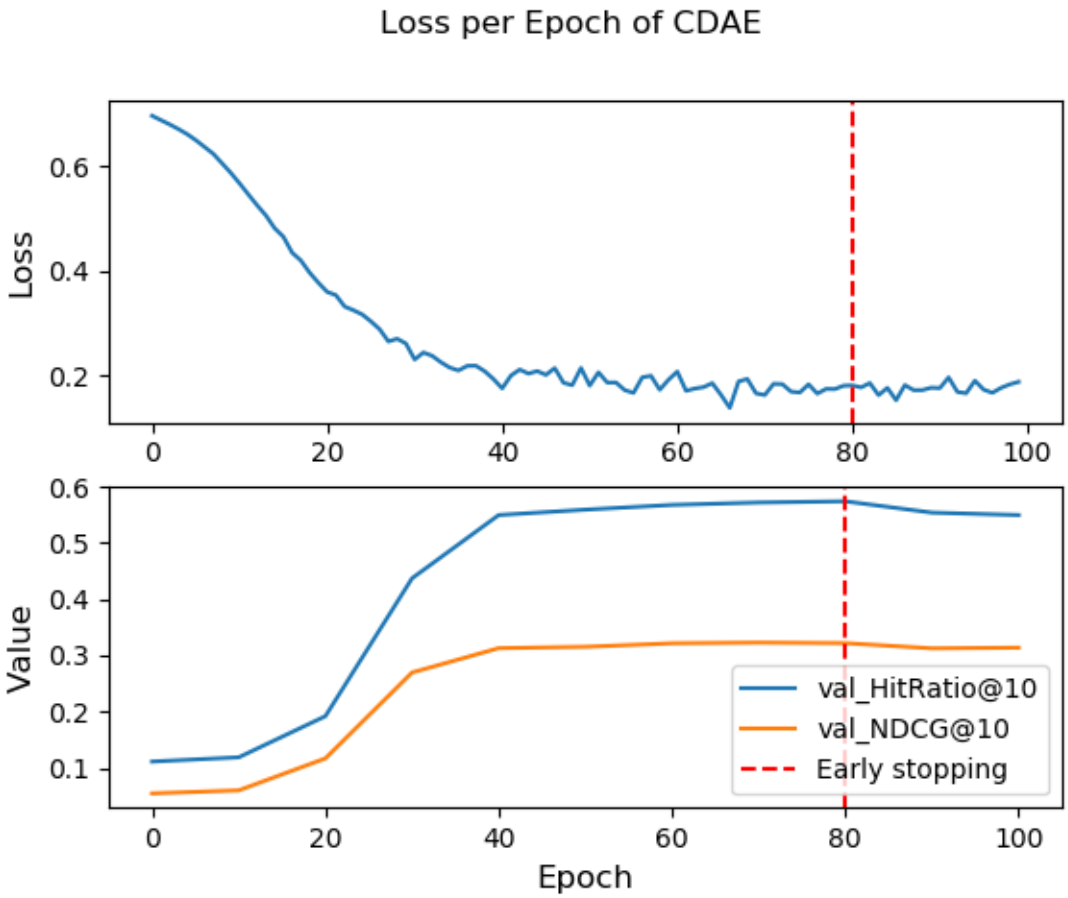

Fig. 1: CDAE Training Performance

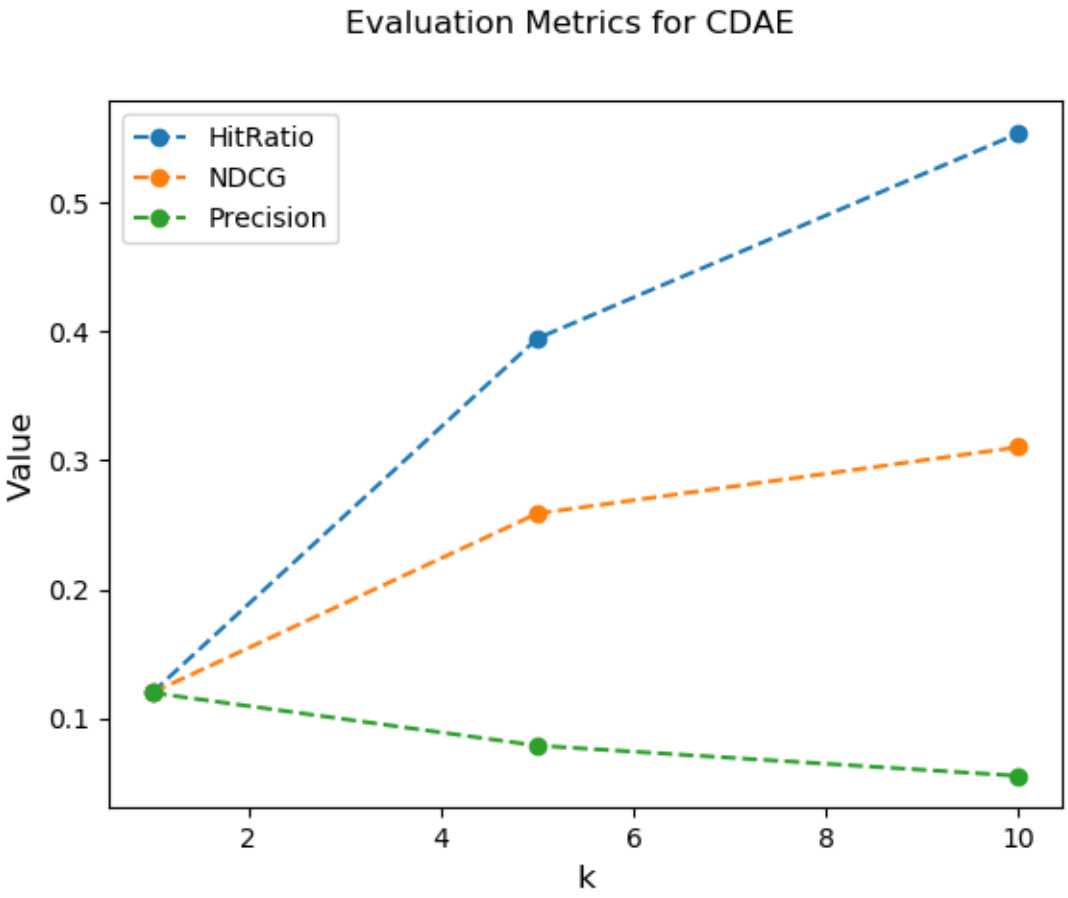

Fig. 2: CDAE Evaluation Performance

CHAPTER 12

Implemented Models

Recommender Type	Name
Learn-to-rank	CDAE (Collaborative Denoising Auto-Encoder)
Learn-to-rank	DMF (Deep Matrix Factorization)

12.1 Implemented Baselines (non deep learning based)

Recommender Type	Name
Predictive	User/Item KNN

CHAPTER 13

Benchmarks

TODO

CHAPTER 14

License

Check LICENCE.md.

CHAPTER 15

Contributors

This work was conducted under the supervision of Prof. Francisco M. Couto, and during the initial development phase the project was financially supported by a FCT research scholarship UID/CEC/00408/2019, under the research institution LASIGE, from the Faculty of Sciences, University of Lisbon.

Public contribution is welcomed, and if you wish to contribute just open a PR or contect me fabioiuri@live.com.

CHAPTER 16

Development Status

Project in alpha stage.

Planned work:

- Wrap up missing documentation
- Implement more models
- Refine and clean unit tests

If you have any bugs to report or update suggestions, you can use DRecPy's github issues page or email me directly to fabioiuri@live.com.

Python Module Index

d

Index

E

F

G

H

I

K

L

M

N

P

R

S

U

V