mkpyros Documentation

Release latest

Jan 17, 2018

Contents

The PYROS python module offers some tools to build and evaluate recommender systems for implicit feedback.

PYROS is available in the PyPi repository and it can be installed with

```
pip install mkpyros
```

and then it can be imported in python with

```
import pyros
```

CHAPTER 1

Loading a dataset

First of all you have to load the dataset. This module provides useful methods for reading Comma Separated Values (CSV) files.

```
from pyros.data import CSVReader
import pyros.data.dataset as ds
reader = CSVReader("path\to\the\csv\file", " ")
data = ds.UDataset(Mapping(), Mapping())
reader.read(dataset, True) #True means that the ratings are binary
```

the code above reads the content of the given CSV file (space separated) and saves it in the dataset variable. In the example the dataset is user-centered, that is ratings are stored as set of items rated by a user.

CHAPTER 2

Creating a recommender

Once the dataset is ready the recommender can be instanciated. Firstly, let us import the engine module

```
from pyros import engine as exp
```

Currently the module offers, beyond the common baselines (e.g., popularity-based), the following recommendation algorithms:

- Matrix-based implementation of the algorithm described in "Efficient Top-N Recommendation for Very Large Scale Binary Rated Datasets" by F. Aiolli, which is based on the asymmetric cosine similarity

```
rec = exp.I2I_Asym_Cos(data, alpha, q)
```

where 'alpha' is the asimmetric weight and 'q' the locality parameter.

- "Convex AUC Optimization for Top-N Recommendation with Implicit Feedback" by F. Aiolli

```
rec = exp.CF_OMD(data, lambda_p, lambda_n, sparse)
```

where 'lambda_p', 'lambda_n' are respectively the regularization terms for the positives and negatives distribution, while 'sparse' is a boolean parameter that says whether to use a sparse matrix implementation or not.

- Implementation of the algorithm (which is a simplification of CF-OMD) described in "Kernel based collaborative filtering for very large scale top-N item recommendation" by M.Polato and F. Aiolli

```
exp.ECF_OMD(data, lambda_p, sparse)
```

where the parameters has the same meaning as in CF_OMD but in this one 'lambda_n' is not required (it is assumed to be +inf).

- Implementation of the algorithm (which is a "kernelification" of ECF-OMD) described in "Kernel based collaborative filtering for very large scale top-N item recommendation" by M.Polato and F. Aiolli, and in "Exploiting sparsity to build efficient kernel based collaborative filtering for top-N item recommendation" by M.Polato and F. Aiolli.

```
import pyros.utils as ut
K = ut.kernels.normalize(ut.kernels.linear(data.to_cvxopt_matrix()))
rec = exp.CF_KOMD(data, K, lambda_p, sparse)
```

in this case a kernel 'K' is required as parameter. The code shows an example of linear kernel built using the support methods provided by the 'utils' module. The 'utils' module includes also the 'kernels' submodule which contains some useful methods related to kernels and also some kernel functions implementation as the one described in

"Disjunctive Boolean Kernels for Collaborative Filtering in Top-N Recommendation" by M.Polato and F. Aiolli.

- Implementation of the algorithm SLIM described in "Sparse linear methods with side information for top-n recommendations" by Xia Ning and George Karypis.

```
rec = exp.SLIM(data, beta, lbda)
```

where 'beta' and 'lbda' are the regularization of the frobenius norm and the Taxicab norm, respectively, as described in the paper.

- Implementation of the algorithm WRMF described in "Collaborative Filtering for Implicit Feedback Datasets" by Yifan Hu, Yehuda Koren and Chris Volinsky, and it is also presented in "One-Class Collaborative Filtering" by Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz and Qiang Yang.

```
rec = exp.WRMF(data, latent_factors, alpha, lbda, num_iters)
```

where 'latent_factors' are the number of latent features, 'alpha' is the weight value for the ratings, 'lbda' the regularization parameter and 'num_iters' the maximum number of iterations of the algorithm.

- Implementation of the algorithm BPRMF described in "BPR: Bayesian personalized ranking from implicit feedback" by Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme.

```
rec = exp.BPRMF(data, factors, learn_rate, num_iters, reg_u, reg_i, reg_bias)
```

where 'factors' are the number of latent features, 'learn_rate' is the learning rate, 'num_iters' the maximum number of iterations of the algorithm 'reg_i', 'reg_u' and 'reg_bias' are the regularization parameters for uesrs, items and the bias respectively.

CHAPTER 3

Training a recommender

After the instanciation of the recommender it has to be trained:

```
rec.train(users)
```

where ‘users’ is the list of users for which the items ranking will be calculated.

CHAPTER 4

Evaluating a trained recommender

Finally, the evaluation step is:

```
import pyros.core.evaluation as ev
result = ev.evaluate(rec, data_test)
```

where 'data_test' is the test dataset which contains the ratings to predict (unknown at training time!!). The evaluation is done using AUC, mAP and NDCG.

For more details please refer to the papers and to the code @ GITHUB.

CHAPTER 5

Version

0.9.32

CHAPTER 6

Tech

PYROS requires the following python modules:

- Scipy
- Numpy
- CVXOPT