DeepLearning RS Evaluation

This repository was developed by Maurizio Ferrari Dacrema, postdoctoral researcher at Politecnico di Milano. See our website for more information on our research group. This reporsitory contains the source code of the following articles:

- **"Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches", RecSys 2019** BibTex. Full text available on ACM DL (open), ResearchGate or ArXiv, source code of our experiments and full results are available here. The slides and poster are also available.
- **"A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research". ACM TOIS 2021** The paper is available on ACM DL (open), ArXiv.
- **"Critically Examining the Claimed Value of Convolutions over User-Item Embedding Maps for Recommender Systems", CIKM 2020** ACM DL (open), ArXiv, BibTex see related documentation HERE.
- **"Methodological Issues in Recommender Systems Research (Extended Abstract)", IJCAI 2020** BibTex, PDF.

A small example on how to use the baseline models is in *run_example_usage.py*.

We are still actively pursuing this research direction in evaluation and reproducibility, we are open to collaboration with other reseachers. Follow our project on ResearchGate!

Please cite our articles if you use this repository or our implementations of baseline algorithms, remember also to cite the original authors if you use our porting of the DL algorithms. The BibTex code is linked above, next to the article.

Update 06/11/2021: A recent paper observed some issues in the description and implementation of HR and NDCG. The issue is present under random holdout, hence it only affects very few NDCG results. The implementation has been updated to fix the issue, the additional material contains the updated metric description and results for the few affected cases. The analysis and conclusions of the study are not affected.

Full results and hyperparameters

The full results and corresponding hyperparameters for all DL algorithms are accessible HERE. For information on the requirements and how to install this repository, see the following Installation section.

Code organization

This repository is organized in several subfolders.

Deep Learning Algorithms

The Deep Learning algorithms are all contained in the *Conferences* folder and further divided in the conferences they were published in. For each DL algorithm the repository contains two subfolders:

- A folder named "_github" which contains the full original repository or "_original" which contains the source code provided by the authors upon request, with the minor fixes needed for the code to run.
- A folder named "_our_interface" which contains the python wrappers needed to allow its testing in our framework. The main class for that algorithm has the "Wrapper" suffix in its name. This folder also contains the functions needed to read and split the data in the appropriate way.

Note that in some cases the original repository contained also the data split used by the original authors, those are included as well.

Baseline algorithms

Folders like "KNN", "GraphBased", "MatrixFactorization", "SLIM_BPR", "SLIM_ElasticNet" and "EASE_R" contain all the baseline algorithms we used in our experiments. The complete list is as follows, details on all algorithms and references can be found HERE:

- Random: recommends a list of random items,
- TopPop: recommends the most popular items,

- UserKNN: User-based collaborative KNN,
- ItemKNN: Item-based collaborative KNN,
- UserKNN CBF: User-based content-based KNN,
- ItemKNN CBF: Item-based content-based KNN,
- UserKNN CFCBF: User-based hybrid content-based collaborative KNN,
- ItemKNN CFCBF: Item-based hybrid content-based collaborative KNN,
- P3alpha: collaborative graph-based algorithm,
- RP3beta: collaborative graph-based algorithm with reranking,
- PureSVD: SVD decomposition of the user-item matrix,
- NMF: Non-negative matrix factorization of the user-item matrix,
- IALS: Implicit alternating least squares,
- MatrixFactorization BPR (BPRMF): machine learning based matrix factorization optimizing ranking with BPR,
- MatrixFactorization FunkSVD: machine learning based matrix factorization optimizing prediction accuracy with MSE,
- EASE_R: collaborative shallow autoencoder,
- SLIM BPR: Item-based machine learning algorithm optimizing ranking with BPR,
- SLIM ElasticNet: Item-based machine learning algorithm optimizing prediction accuracy with MSE.

The following similarities are available for all KNN models: cosine, adjusted cosine, pearson correlation, dice, jaccard, asymmetric cosine, tversky, euclidean

Evaluation

The folder *Base.Evaluation* contains the two evaluator objects (*EvaluatorHoldout*, *EvaluatorNegativeSample*) which compute all the metrics we report.

Data

The data to be used for each experiment is gathered from specific *DataReader* objects within each DL algoritm's folder. Those will load the original data split, if available. If not, automatically download the dataset and perform the split with the appropriate methodology. If the dataset cannot be downloaded automatically, a console message will display the link at which the dataset can be manually downloaded and instructions on where the user should save the compressed file. The data of ConvNCF cannot be automatically handled and should be manually downloaded HERE and decompressed in folder "Conferences/IJCAI/ConvNCF_github/Data".

The folder *Data_manager* contains a number of *DataReader* objects each associated to a specific dataset, which are used to read datasets for which we did not have the original split.

Whenever a new dataset is downloaded and parsed, the preprocessed data is saved in a new folder called *Data_manager_split_datasets*, which contains a subfolder for each dataset. The data split used for the experimental evaluation is saved within the result folder for the relevant algorithm, in a subfolder *data* .

Hyperparameter optimization

Folder *ParameterTuning* contains all the code required to tune the hyperparameters of the baselines. The script *run_parameter_search* contains the fixed hyperparameters search space used in all our experiments. The object *SearchBayesianSkopt* does the hyperparameter optimization for a given recommender instance and hyperparameter space, saving the explored configuration and corresponding recommendation quality.

If you want to execute ConvNCF make sure you extract the data files in folder Conferences/ConvNCF/ConvNCF_github/Data and ConvolutionRS/ConvNCF/ConvNCF_github/Data

Run the experiments

See see the following Installation section for information on how to install this repository. After the installation is complete you can run the experiments.

Comparison with baselines algorithms

All experiments related to a DL algorithm reported in our paper can be executed by running the corresponding script, which is preceeded by *run_*, the conference name and the year of publication. The scripts have the following boolean optional parameters (all default values are False except for the print-results flag):

- '-b' or '--baseline_tune': Run baseline hyperparameter search
- '-a' or '--DL_article_default': Train the deep learning algorithm with the original hyperparameters
- '-p' or '--print_results': Generate the latex tables for this experiment

For example, if you want to run all the experiments for SpectralCF, you should run this command:

```
python run_RecSys_18_SpectralCF.py -b True -a True -p True
```

The script will:

- Load and split the data.
- Run the bayesian hyperparameter optimization on all baselines, saving the best values found.

- Run the fit and test of the DL algorithm
- Create the latex code of the result tables, as well as plot the data splits, when required.
- The results can be accessed in the *result_experiments* folder.

Ablation experiment for CNN algorithms on embeddings

All experiments related to a Convolution algorithm reported in our paper can be executed by running the corresponding script, which is preceeded by *run_*, the conference name and the year of publication. For example, if you want to run the experiments for ConvNCF, you should run this command:

```
python run_IJCAI_18_ConvNCF_CNN_embeddings.py
```

The training of baselines is enabled by default, if you want to disable it use:

```
python run_IJCAI_18_ConvNCF_CNN_embeddings.py --run_baselines False
```

The script will:

- Load and split the data.
- (if required) Run the bayesian hyperparameter optimization on all baselines, saving the best values found.
- Run the pretraining of the embeddings, if needed.
- Run Study 1, applying 20 permutations of the pretrained embeddings and for each evaluating the pretraining model, fitting the Convolution algorithm on the full map and evaluating it for each component: full map, diagonal (element wise), off-diagonal (embeddings correlation).
- Run Study 2, reading the previously generated permutations and fitting the model on the specified portion of the interaction map: full map, diagonal (element wise), off-diagonal (embeddings correlation).
- Create the latex code of the result tables.
- The results can be accessed in the *result_experiments* folder.

Installation

Note that this repository requires Python 3.6

First we suggest you create an environment for this project using virtualenv (or another tool like conda)

First checkout this repository, then enter in the repository folder and run this commands to create and activate a new environment:

If you are using virtualenv:

```
virtualenv -p python3 DLevaluation
source DLevaluation/bin/activate
```

If you are using conda:

```
conda create -n DLevaluation python=3.6 anaconda
conda activate DLevaluation
```

Then if you want to run the experiments on CPU you should install all the requirements and dependencies using the following command. If you wish a GPU installation please install the dependencies as described in subsection Installation on GPU.

```
pip install -r requirements.txt
```

At this point, having installed all dependencies for either CPU or GPU usage, you have to compile all Cython algorithms.

In order to compile you must first have installed: *gcc* and *python3 dev*. Under Linux those can be installed with the following commands:

```
sudo apt install gcc
sudo apt-get install python3-dev
```

If you are using Windows as operating system, the installation procedure is a bit more complex. You may refer to THIS guide.

Now you can compile all Cython algorithms by running the following command. The script will compile within the current active environment. The code has been developed for Linux and Windows platforms. During the compilation you may see some warnings.

```
python run_compile_all_cython.py
```

Installation on GPU

In order to run the experiments on GPU you should install requirements and dependencies in the following way. This commands only work if you are using conda to manage your virtual environment. It is possible to install them using other tools or pip but it may prove to be a much more complex task.

```
conda install tensorflow-gpu
conda install -c anaconda keras-gpu
conda install -c hcc dm-sonnet-gpu

pip install -r requirements_gpu.txt
```

Matlab engine

In addition to the repository dependencies, KDD CollaborativeDL also requires the Matlab engine, due to the fact that the algorithm is developed in Matlab. To install the engine you can use a script provided directly with your Matlab distribution, as described in the Matlab Documentation. The algorithm requires also a GSL distribution, whose installation folder can be provided as a parameter in the fit function of our Python wrapper. Please refer to the original CollaborativeDL README for all installation details.