SLIM 2.0 — Sparse LInear Models for Top-N Recommendations

User/Reference Manual

Karypis Lab

Department of Computer Science & Engineering

University of Minnesota

September 2019

Chapter 1

SLIM Documentation

Sparse LInear Method (SLIM) [1] is an item-based top-N recommendation approach that combines the advantages of neighborhood- and model-based collaborative filtering methods. It achieves state-of-the-art recommendation performance and has low computational requirements.

This package provides a C-based optimized multi-threaded implementation of SLIM that consists of a set of command-line programs and a user-callable library for estimating and applying SLIM models as well as an easy to use Python interface.

1.1 Downloading SLIM

SLIM uses Git submodules to manage external dependencies. Hence, please specify the `--recursive` option while cloning the repo as follow:

```
git clone --recursive https://github.com/KarypisLab/SLIM.git
```

1.2 Building standalone SLIM binary and shared library

To build SLIM you can follow the instructions below:

1.2.1 Dependencies

General dependencies for building slim are: gcc, cmake, build-essential. In Ubuntu systems these can be obtained from the apt package manager (e.g., apt-get install cmake, etc)

```
sudo apt-get install build-essential
sudo apt-get install cmake
```

1.2.2 Building and installing SLIM

In order to build SLIM, first build GKlib by running:

```
cd lib/GKlib
make config openmp=set
make
cd ../../
```

After building GKlib, you can build and install SLIM by running:

```
make config shared=1 cc=gcc cxx=gcc prefix=~/.local
make install
```

1.2.2.1 Building SLIM with Intel's MKL support (optional)

In order to use SLIM's ADMM solver, you will need to install `Intel's MKL library`.

For Ubuntu machines on which you have `sudo` privileges, we provided the `depmkl.sh` script that automates the process of obtaining and installing MKL, which can be used as follows:

```
bash depmkl.sh
source ~/.bashrc
```

For machines on which you do not have `sudo` privileges, you should download the MKL tarball from `Intel's website` and then install it locally using the `install.sh` script they provide. After installing it you should add `your-path-to-intel/intel/mkl/bin/mklvars.sh intel64`in your bashrc and run `source ~/.bashrc`.

You can build and install SLIM with MKL support by running:

```
make config shared=1 cc=gcc cxx=gcc with_mkl=1 prefix=~/.local
make install
```

Note that SLIM's ADMM solver usually outperforms the default optimizer included in SLIM when the number of items in the dataset is relatively small compared to the number of users and the number of non-zeros in the dataset is large.

1.3 Python Package installation

The Python package is located at `python-package/`. The installation of python-package requires Python `distutils` module and is often part of the core Python package or can be installed using a package manager, e.g., in Debian use

```
sudo apt-get install python-setuptools
```

After building the SLIM library, follow one of the following steps to install the python-package:

1. Install python-package system-wide (this requires sudo privileges):

   ```
   cd python-package
   sudo python setup.py install
   ```

2. Install python-package only for the current users (without sudo privileges):

   ```
   cd python-package
   python setup.py install --user
   ```

1.4 Getting Started

Here are some examples to quickly try out SLIM on the sample datasets that are provided with SLIM.

1.4.1 Python

```
import pandas as pd
from SLIM import SLIM, SLIMatrix

#read training data stored as triplets <user> <item> <rating>
traindata = pd.read_csv('../test/AutomotiveTrain.ijv', delimiter = ' ', header=None)
trainmat = SLIMatrix(traindata)

#set up parameters to learn model, e.g., use Coordinate Descent with L1 and L2
#regularization
params = {'algo':'cd', 'nthreads':2, 'l1r':1., 'l2r':1.}

#learn the model using training data and desired parameters
model = SLIM()
model.train(params, trnmat)

#read test data having candidate items for users
testdata = pd.read_csv('../test/AutomotiveTest.ijv', delimiter = ' ', header=None)
#NOTE: model object is passed as an argument while generating test matrix
testmat = SLIMatrix(testdata, model)

#generate top-10 recommendations
prediction_res = model.predict(testmat, nrcmds=10, outfile = 'output.txt')

#dump the model to files on disk
model.save_model(modelfname='model.csr', # filename to save the model as a csr matrix
                 mapfname='map.csr' # filename to save the item map
                )

#load the model from from disk
model_new = SLIM()
model_new.load_model(modelfname='model.csr', # filename of the model
                 mapfname='map.csr' # filename of the item map
                )
```

The users can also refer to the python notebook `UserGuide.ipynb` located at `./python-package/UserGuide.ipynb` for more examples on using the python api.

1.4.2 Standalone

SLIM can also be used standalone by running the built binaries located under `./build` directory. The `slim_learn` and `slim_predict` program can be used to learn the model and predict using an existing model, respectively. The usage of `slim_learn` is generally as follow:

```
slim_learn [options] train-file [output-model-file]
```

In above command, `train-file` refers to the rating matrix stored in a file on disk in sparse format (by default it expects CSR format) and `output-model-file` is the name of the file that will be used to save the learned model. The available options will be described later in detail. Following shows an example of using the `slim_learn` program:

```
./build/Linux-x86_64/src/programs/slim_learn test/ml100k-train.csr model_output.slim
```

Similarly, the recommendations can be generated by using the program `slim_predict` and the usage of `slim_predict` is as follow:

```
slim_predict [options] model-file train-file [test-file]
```

In above command, `model-file` refers to the SLIM model saved in a file, `train-file` refers to the training data or past historical data used to generate the model and `test-file` is the test data containing hidden ratings for the users.

1.5 SLIM parameters

The following optional parameters can be provided to the `slim_learn` program.

- ifmt [default=csr]
  - Specifies the format of the input file. Available options are: csr - CSR format [default]. csrnv - CSR format without ratings. cluto - Format used by CLUTO. ijv - One (row#, col#, val) per line.
- binarize [default=0]
  - Specifies that the ratings should be binarized.
- l1r [default=1.0]
  - Specifies the L1 regularization parameter.
- ipmdlfile [default=""]
  - Specifies the file used to initilize the model.
- l2r [default=1.0]
  - Specifies the L2 regularization parameter.
- nnbrs [default=0]
  - Selects fSLIM model and specifies the number of item nearest neighbors to be used. Specifying few neighbors will speed-up the model learning but it may lower the accuracy. The parameter *simtype* sets the measurement of similarity. This package supports three similarity measurements, Jaccard similarity ("jac"), Cosine similarity ("cos"), and inner product ("dotp"). The default value for *simtype* is "cos". A fSLIM model can be used in the same way with a SLIM model. Note that, a fSLIM model can only be trained using coordinate descent.
- simtype [default=cos]
  - Specifies the similarity function for determining the neighbors. Available options are:
    - cos - cosine similarity [default].
    - jac - extended Jacquard similarit.
    - dotp - dot-product similarity.
- algo [default=cd]
  - Specifies the optimization algorithms for learning the model. Available options are:
    - admm - ADMM.

        * cd - Coordinate Descent [default].

- optTol [default=1e-7]
  - Specifies the threshold used during optimization for termination.
- niters [default=10000]
  - Specifies the maximum number of allowed optimization iterations.
- nthreads
  - Specifies the number of threads to be used for estimation. Default value is maximum number of threads available on the machine.
- dbglvl
  - Specifies the debug level. The default value turns on info and timing.
- help
  - Prints the parameter details.

1.6 Citing

If you use any part of this library in your research, please cite it using the following BibTex entry:

```
@online{slim,
  title = {{SLIM}: Sparse LInear Model library},
  author = {Ning, Xia and Nikolakopoulos, Athanasios N. and Shui, Zeren and Sharma, Mohit and Karypis, George},
  url = {https://github.com/KarypisLab/SLIM},
  publisher = {GitHub},
  journal = {GitHub Repository},
  year = {2019},
}
```

1.7 Credits & Contact Information

SLIM was written by George Karypis with contributions by Xia Ning, Athanasios N. Nikolakopoulos, Zeren Shui and Mohit Sharma.

If you encounter any problems or have any suggestions, please contact George Karypis at karypis@cs.umn.edu.

1.8 Copyright & License Notice

Chapter 2

Module Documentation

2.1 Functions for using the SLIM library

2.1.1 Detailed Description

This is the set of methods that can used from SLIM library.

Functions

- slim_t * SLIM_Learn (int32_t nrows, ssize_t *rowptr, int32_t *rowind, float *rowval, int32_t *ioptions, double *doptions, slim_t *imodel, int32_t *r_status)

  Entry point for the model estimation routine. It uses the sparse matrix in CSR format.
- int32_t SLIM_GetTopN (slim_t *model, int32_t nratings, int32_t *itemids, float *ratings, int32_t *ioptions, int32_t nrcmds, int32_t *rids, float *rscores)

  get Top-N recommendations given a historical rating profile
- int32_t SLIM_iSetDefaults (int32_t *options)

  Sets the default value (-1) for passed options.
- int32_t SLIM_dSetDefaults (double *options)

  Sets the default value (-1) for passed options.
- int32_t SLIM_WriteModel (slim_t *model, char *filename)

  Writes the model to a supplied file.
- slim_t * SLIM_ReadModel (char *filename)

  Reads the model from the passed file in CSR format. For example:
- void SLIM_FreeModel (slim_t **r_model)

  Frees the memory allocated for the SLIM model matrix. For example:
- int32_t * SLIM_DetermineHeadAndTail (int32_t nrows, int32_t ncols, ssize_t *rowptr, int32_t *rowind)

  Returns an array marking as 0 the columns that belong to the head and as 1 the columns that belong to the tail. The split is based on an 50-50 split (head: the most frequent items that correspond to the 50% of the ratings).

2.1.2 Detailed Description

This is the set of methods that can used from SLIM library.

2.1.3 Function Documentation

2.1.3.1 SLIM_Learn()

```
slim_t* SLIM_Learn (
            int32_t nrows,
            ssize_t * rowptr,
            int32_t * rowind,
            float * rowval,
            int32_t * ioptions,
            double * doptions,
            slim_t * imodel,
            int32_t * r_status )
```

Entry point for the model estimation routine. It uses the sparse matrix in CSR format.

Parameters

nrows	number of rows in the matrix
rowptr	[rowptr[i], rowptr[i+1]) points to the indices in rowind and rowval
rowind	contains the column indices of the non-zero elements in matrix.
rowval	contains the values of the non-zero elements in matrix.
ioptions	integer or boolean options to the estimation routine.
doptions	double options to the estimation routine.
imodel	if not null then it is pointer to model that will be used for initialization.
r_status	set to 1 on success

Returns

sparse representation of model in CSR format.

2.1.3.2 SLIM_GetTopN()

```
int32_t SLIM_GetTopN (
            slim_t * model,
            int32_t nratings,
```

```
            int32_t * itemids,
            float * ratings,
            int32_t * ioptions,
            int32_t nrcmds,
            int32_t * rids,
            float * rscores )
```

get Top-N recommendations given a historical rating profile

Parameters

model	the SLIM model matrix
nratings	number of ratings in the historical profile
itemids	ids of rated items in the historical profile
ratings	ratings of items in the historical profile
ioptions	integer options passed to the routine
nrcmds	N in Top-N, i.e., size of recommendation list
rids	ids of items in the recommendation list
rscores	predicted ratings items in the recommendation list

Returns

size of recommendation list on success else a value < 0

2.1.3.3 SLIM_iSetDefaults()

```
int32_t SLIM_iSetDefaults (
            int32_t * options )
```

Sets the default value (-1) for passed options.

Parameters

options	the integer array having option values

Returns

1 on success

2.1.3.4 SLIM_dSetDefaults()

```
int32_t SLIM_dSetDefaults (
            double * options )
```

Sets the default value (-1) for passed options.

Parameters

options	the double array having option values

Returns

1 on success

2.1.3.5 SLIM_WriteModel()

```
int32_t SLIM_WriteModel (
            slim_t * model,
            char * filename )
```

Writes the model to a supplied file.

Parameters

model	the SLIM model matrix
filename	the name of the file to write model to in CSR format

Returns

1 on success

2.1.3.6 SLIM_ReadModel()

```
slim_t* SLIM_ReadModel (
            char * filename )
```

Reads the model from the passed file in CSR format. For example:

```
slim_t *model = SLIM_ReadModel(input_model_filename);
```

Parameters

filename	the name of the file having model in CSR format

Returns

return the SLIM model sparse matrix in CSR format

2.1.3.7 SLIM_FreeModel()

```
void SLIM_FreeModel (
             slim_t ** r_model )
```

Frees the memory allocated for the SLIM model matrix. For example:

```
slim_t *model = SLIM_ReadModel(input_model_filename);
SLIM_FreeModel(&model);
```

Parameters

r_model	the SLIM model sparse matrix

Bibliography

[1] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011. 1

Index