Parallel Probabilistic Matrix Factorization

1.0

Generated by Doxygen 1.9.1

Chapter 1

Parallel Probabilistic Matrix Factorization using C++

1.1 About

Probablistic Matrix Factorization is a class of graphical models commonly used for recommender systems. This project provides a parallel implementation of a Gaussian matrix factorization model utilizing stochastic gradient ascent with no locking to obtain unbiased Maximum A Posteriori (MAP) estimates of the latent user preference and attribute vectors.

1.2 Requirements & Prequisite libraries

- Boost >= 1.7.0
- `Eigen3` >= 3.3.9

1.3 Installation

```
git clone https://github.com/ageil/parallel-pmf.git
cd parallel-pmf/
cmake .
make
```

To compile & run the unit tests:

```
cmake .
make test
```

1.3.1 Python wrapper

We provide a simple python wrapper library `pmf` to enable interactive analysis, including model recommendations and plottings in jupyter notebooks. To install it:

```
cd pypmf
./install.sh
```

Please refer to the /Users/ageil/Google Drive/Columbia/W4995 Design using C++/Project/example/pmf_tutorial.md "tutorial notebooks" for details.

1.4 Running options

```
Parameters for Probabilistic Matrix Factorization (PMF):
  -h [ --help ]              Help
  -i [ --input ] arg         Input file name
  -m [ --map ] arg           Item mapping file name
  --task arg                 Task to perform
                              [Options: 'train', 'recommend']
  -o [ --output ] arg        Output directory
                               [default: current_path/results/]
  -k [ --n_components ] arg Number of components (k)
                              [default: 3]
  -n [ --n_epochs ] arg      Num. of learning iterations
                               [default: 200]
  -r [ --ratio ] arg         Ratio for training/test set splitting
                              [default: 0.7]
  --thread arg               Number of threads for parallelization
  --gamma arg                Learning rate for gradient descent
                               [default: 0.01]
  --std_theta arg            Std. of theta's prior normal distribution
                               [default: 1]
  --std_beta arg             Std. of beta's prior normal distribution
                               [default: 1]
  --user                     Recommend items for given user
  --item                     Recommend similar items for a given item
  -s [--run_sequential]      Enable running fit model sequentially
  -l [--loss_interval] arg   Number of epochs between each loss computation. [default: 10]
```

1.5 Quick start

Please refer to the sample running scripts for `training` and `recommendation`.

1.6 References

- Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. *Advances in neural information processing systems*, *20*, 1257-1264
- Niu, F., Recht, B., Ré, C., & Wright, S. J. (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. arXiv preprint arXiv:1106.5730
- GroupLens Research (2021). MovieLens dataset. https://grouplens.org/datasets/movielens/

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 DataManager::DataManager Class Reference

Public Member Functions

- DataManager (const string &input, const double ratio)
- shared_ptr< MatrixXd > getTrain () const
- shared_ptr< MatrixXd > getTest () const
- ItemMap loadItemMap (const string &input)

Private Member Functions

- tuple< TrainingData, TestingData > split (const double ratio)

4.1.1 Constructor & Destructor Documentation

4.1.1.1 DataManager()

```
DataManager::DataManager::DataManager (
             const string & input,
             const double ratio )
```

Initialize DataManager by loading the csv file found in the given input. The data is zero-centered, shuffled, and stored. Additionally, the given ratio will determine how to split the processed data into training data vs. testing data. (e.g. ratio=0.7 will split to 70% train, 30% test)

Parameters

input	A file path to the csv file of data to load.
ratio	The ratio to split the data into training data vs. testing data.

4.1.2 Member Function Documentation

4.1.2.1 getTest()

```
shared_ptr< MatrixXd > DataManager::DataManager::getTest ( ) const
```

Gets the testing data set.

Returns

A shared_ptr of the matrix of the testing data set.

4.1.2.2 getTrain()

```
shared_ptr< MatrixXd > DataManager::DataManager::getTrain ( ) const
```

Gets the training data set.

Returns

A shared_ptr of the matrix of the training data set.

4.1.2.3 loadItemMap()

```
ItemMap DataManager::DataManager::loadItemMap (
            const string & input )
```

Load the mappings between items' ID (integer), titles (string), and genres (string)

Parameters

input	Input file name

Returns

Struct of multiple Maps between ID, titles & genres: ItemMap.id_name - ID->title, ItemMap.name_id - title->ID, ItemMap.id_genre - ID->genre, ItemMap.name_genre - title->genre, Item.genre_ids - genre->Set of IDs of the given genre

4.1.2.4 split()

```
tuple< TrainingData, TestingData > DataManager::DataManager::split (
            const double ratio )  [private]
```

Splits the m_data rows into a train and test set by ratio (e.g. ratio=0.7 will split to 70% train, 30% test)

Parameters

ratio	The ratio to split the data into training data vs. testing data.

Returns

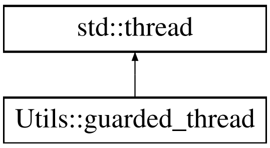
> A tuple of <MatrixXd, MatrixXd> type, in which the first matrix is the training data and the second matrix is the testing data.

The documentation for this class was generated from the following files:

- models/datamanager.h
- models/datamanager.cpp

4.2 Utils::guarded_thread Struct Reference

Inheritance diagram for Utils::guarded_thread:



The documentation for this struct was generated from the following file:

- models/utils.h

4.3 DataManager::ItemMap Struct Reference

The documentation for this struct was generated from the following file:

- models/datamanager.h

4.4 Model::LatentVectorsSnapshot Struct Reference

```
#include <PMF.h>
```

4.4.1 Detailed Description

Stores a 'snapshot' of the given theta and beta inputs by copying the inputs and storeing them in theta and beta member variables.

Parameters

theta	A map connecting each entity ID to its corresponding latent vector.
beta	A map connecting each entity ID to its corresponding latent vector.

The documentation for this struct was generated from the following file:

- models/PMF.h

4.5 Model::Metrics Struct Reference

The documentation for this struct was generated from the following file:

- models/PMF.h

4.6 Model::PMF Class Reference

Public Member Functions

- vector< double > fitSequential (const int epochs, const double gamma)
- vector< double > fitParallel (const int epochs, const double gamma, const int n_threads)
- void load (filesystem::path &indir)
- void save (filesystem::path &outdir)
- VectorXd predict (const MatrixXd &data) const
- vector< string > recommend (const int user_id, const unordered_map< int, string > &item_name, const int N=10) const
- Metrics accuracy (const shared_ptr< MatrixXd > &data, const int N) const
- vector< string > getSimilarItems (int &item_id, unordered_map< int, string > &id_name, int N=10)

Private Member Functions

- void initVectors (normal_distribution<> &dist, const vector< int > &entities, LatentVectors &vmap, const int k)
- double logNormPDF (const VectorXd &x, double loc=0.0, double scale=1.0) const
- double logNormPDF (double x, double loc=0.0, double scale=1.0) const
- MatrixXd subsetByID (const Ref< MatrixXd > &batch, int ID, int column) const
- void computeLoss (const LatentVectors &theta, const LatentVectors &beta)
- void computeLossFromQueue ()
- void loadModel (filesystem::path &indir, LatentVar option)
- void fitUsers (const Ref< MatrixXd > &batch, const double gamma)
- void fitItems (const Ref< MatrixXd > &batch, const double gamma)
- VectorXi recommend (const int user_id, const int N) const

4.6.1 Member Function Documentation

4.6.1.1 accuracy()

```
Metrics Model::PMF::accuracy (
            const shared_ptr< MatrixXd > & data,
            const int N ) const
```

Calculate the accuracy metrics of the top N predicted items for each user with their actual likes

Parameters

data	A 3-column matrix with Col.1 - user IDs, Col.2 - item IDs & Col.3 - user's rating to item
N	Number of the top predicted recommendations (items) compare with

Returns

Struct of {precision, recall} representing how frequency recommendations hit the actual users' likes

4.6.1.2 computeLoss()

```
void Model::PMF::computeLoss (
            const LatentVectors & theta,
            const LatentVectors & beta )  [private]
```

Compute the log-likelihood of the data under the model (assuming only Gaussian distributions).

Parameters

theta	Map of user IDs to user preference vectors
beta	Map of item IDs to item attribute vectors

4.6.1.3 computeLossFromQueue()

```
void Model::PMF::computeLossFromQueue ( )  [private]
```

Compute the log-likelihood of the snapshots of data found in m_loss_queue (assuming only Gaussian distributions). This queue will wait until it gets a signal that there is a new item to process or until it gets a signal to terminate. If it gets the signal to terminate, it will process any remaining items in the queue before exiting.

4.6.1.4 fitItems()

```
void Model::PMF::fitItems (
            const Ref< MatrixXd > & batch,
            const double gamma )  [private]
```

Compute gradient updates of each item in a batch of data, and apply the update to the corresponding beta vectors.

Parameters

batch	Reference to a batch of training data containing columns for user IDs, item IDs, and ratings (in order)
gamma	Learning rate to be used in the gradient ascent procedure

4.6.1.5 fitParallel()

```
vector< double > Model::PMF::fitParallel (
            const int epochs,
            const double gamma,
            const int n_threads )
```

Fit the latent beta and theta vectors to the training dataset in parallel over multiple threads.This performs the loss computation every 10 epochs in parallel on a separate thread.

Parameters

epochs	Number of times the training dataset is passed over in order to compute gradient updates
gamma	Learning rate to be used in the gradient ascent procedure
n_threads	Number of threads the training dataset to distribute the dataset over

Returns

A vector of log-likelihoods of the data under the model for each epoch

4.6.1.6 fitSequential()

```
vector< double > Model::PMF::fitSequential (
            const int epochs,
            const double gamma )
```

Fit the latent beta and theta vectors to the training dataset sequentially. This performs the loss computation every 10 epochs sequentially.

Parameters

epochs	Number of times the training dataset is passed over in order to compute gradient updates
gamma	Learning rate to be used in the gradient ascent procedure

Returns

A vector of log-likelihoods of the data under the model for each epoch

4.6.1.7 fitUsers()

```
void Model::PMF::fitUsers (
            const Ref< MatrixXd > & batch,
            const double gamma )  [private]
```

Compute gradient updates of each user in a batch of data, and apply the update to the corresponding theta vectors.

Parameters

batch	Reference to a batch of training data containing columns for user IDs, item IDs, and ratings (in order)
gamma	Learning rate to be used in the gradient ascent procedure

4.6.1.8 getSimilarItems()

```
vector< string > Model::PMF::getSimilarItems (
            int & item_id,
            unordered_map< int, string > & id_name,
            int N = 10 )
```

Generate a vector of top N most similar items to the input item with Item ID

Parameters

item_id	Item ID of the item to generate item recommendations
id_name	Map of of item ID (int) to their item title (string)
N	Number of item recommendations to generate

Returns

A list of recommended items names sorted from the most to least similar to the input item

4.6.1.9 initVectors()

```
void Model::PMF::initVectors (
            normal_distribution<> & dist,
            const vector< int > & entities,
            LatentVectors & vmap,
            const int k ) [private]
```

Initialize for each entity the corresponding k-length latent vector in vmap by drawing randomly from dist.

Parameters

dist	The distribution from which entry values for the latent vector are randomly drawn
entities	A vector of entity IDs, either user IDs or item IDs
vmap	A map connecting each entity ID to its corresponding latent vector
k	The length of each latent vector

4.6.1.10 load()

```
void Model::PMF::load (
            filesystem::path & indir )
```

Load previously learnt latent theta & beta vectors from file

Parameters

indir	Parent directory to files containing theta & beta vectors

4.6.1.11 loadModel()

```
void Model::PMF::loadModel (
            filesystem::path & indir,
            LatentVar option ) [private]
```

Helper function to load theta & beta vectors from file

Parameters

indir	Parent directory to files containing theta & beta vectors
option	Specify which latent variable to load (LatentVar::theta or LatentVar::beta)

4.6.1.12 logNormPDF() [1/2]

```
double Model::PMF::logNormPDF (
            const VectorXd & x,
            double loc = 0.0,
            double scale = 1.0 ) const [private]
```

Compute the log-likelihood of a vector x under a Gaussian distribution with mean loc and standard deviation scale.

Parameters

x	A vector of doubles to be evaluated
loc	The mean of the Gaussian distribution
scale	The standard deviation of the Gaussian distribution

Returns

The log-probability of observing x

4.6.1.13 logNormPDF() [2/2]

```
double Model::PMF::logNormPDF (
             double x,
             double loc = 0.0,
             double scale = 1.0 ) const  [private]
```

Compute the log-likelihood of a double x under a Gaussian distribution with mean loc and standard deviation scale.

Parameters

x	A point double to be evaluated
loc	The mean of the Gaussian distribution
scale	The standard deviation of the Gaussian distribution

Returns

The log-probability of observing x

4.6.1.14 predict()

```
VectorXd Model::PMF::predict (
             const MatrixXd & data ) const
```

Predict ratings using learnt theta and beta vectors in model.

Parameters

data	A 2-column matrix with the first column denoting user IDs and the second column denoting item IDs

Returns

A vector of predicted ratings for each pair of user and item IDs

4.6.1.15 recommend() [1/2]

```
VectorXi Model::PMF::recommend (
             const int user_id,
             const int N ) const  [private]
```

Generate a vector of top N most recommended items for user with ID user_id.

Parameters

user← _id	User ID of the user to generate item recommendations
N	Number of item recommendations to generate

Returns

A list of recommended item IDs sorted from most to least recommended

4.6.1.16 recommend() [2/2]

```
vector< string > Model::PMF::recommend (
            const int user_id,
            const unordered_map< int, string > & item_name,
            const int N = 10 ) const
```

Generate a vector of top N most recommended items with actual titles for user with ID user_id.

Parameters

user_id	User ID of the user to generate item recommendations
item_name	Hashmap of of item ID (int) to their item title (string)
N	Number of item recommendations to generate

Returns

A list of recommended items names sorted from most to least recommended

4.6.1.17 save()

```
void Model::PMF::save (
            filesystem::path & outdir )
```

Save learnt latent theta & beta vectors to file

Parameters

outdir	Parent directory to files to save theta & beta vectors

4.6.1.18 subsetByID()

```
MatrixXd Model::PMF::subsetByID (
             const Ref< MatrixXd > & batch,
             int ID,
             int column ) const  [private]
```

Extract a subset of a data batch where the value in column is ID.

Parameters

batch	Reference to a batch of data
ID	The ID of a user or item to be extracted
column	Index of either the user or item column in which ID is located

Returns

A matrix of rows where values in column are all ID

The documentation for this class was generated from the following files:

- models/PMF.h
- models/PMF.cpp

Index