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
  -d [ --use_defaults ]                 If enabled, uses './movielens/ratings.c
                                        sv' for the input file and
                                        './movielens/movies.csv' for the map
                                        input file
  -o [ --output ] arg                   Output directory
                                        [default: current_path/results/]


  --task arg (=train)                   Task to perform
                                        [Options: 'train', 'recommend']

  -k [ --n_components ] arg (=5)        Number of components (k)
                                        [default: 3]

  -n [ --n_epochs ] arg (=200)          Num. of learning iterations
                                        [default: 200]

  -r [ --ratio ] arg (=0.7)             Ratio for training/test set splitting
                                        [default: 0.7]

  --thread arg (=4)                     Number of threads for parallelization
                                        This value must be at least 2
                                        [default: 4]

  --gamma arg (=0.01)                   Learning rate for gradient descent
                                        [default: 0.01]

  --std_theta arg (=1)                  Std. of theta's prior normal
                                        distribution
                                        [default: 1]

  --std_beta arg (=1)                   Std. of beta's prior normal
                                        distribution
                                        [default: 1]

  -s [ --run_sequential ]               Enable running model fitting
                                        sequentially

  --user                                Recommend items for given user

  --item                                Recommend similar items for a given
                                        item

  --loss_interval arg (=10)             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 Tutorial

Please kindly find our comprehensive `tutorial`, `manual` and [design document](docs)

1.7 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:

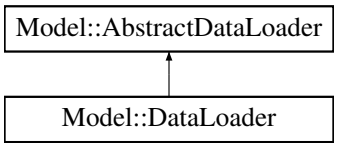
Chapter 4

Class Documentation

4.1 Model::AbstractDataLoader Class Reference

`#include <abstractdataloader.h>`

Inheritance diagram for Model::AbstractDataLoader:



4.1.1 Detailed Description

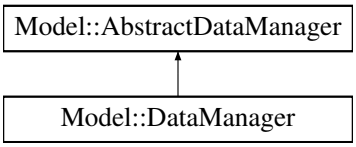
Barebone interface for a `DataLoader`, agnostic of what the source of the data is (e.g. database, csv files on disk etc.)

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

- models/abstractdataloader.h

4.2 Model::AbstractDataManager Class Reference

Inheritance diagram for Model::AbstractDataManager:



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

- models/abstractdatamanager.h

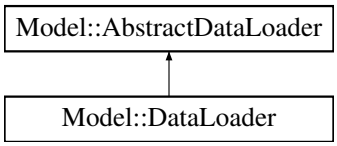
4.3 Model::Utils::Arguments Struct Reference

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

- models/utils.h

4.4 Model::DataLoader Class Reference

Inheritance diagram for Model::DataLoader:



Public Member Functions

- virtual MatrixXd getDataset () const
- virtual tuple< LatentVectors, LatentVectors > getLearntVectors () const
- virtual void saveTrainResults (const LatentVectors &theta, const LatentVectors &beta, const vector< double > &losses) const

4.4.1 Member Function Documentation

4.4.1.1 getDataset()

```
MatrixXd Model::DataLoader::getDataset ( ) const  [virtual]
```

Loads data matrix from a csv file specified in m_dataset_in

Implements Model::AbstractDataLoader.

4.4.1.2 getLearntVectors()

```
tuple< LatentVectors, LatentVectors > Model::DataLoader::getLearntVectors ( ) const  [virtual]
```

Loads previously learnt latent theta & beta vectors from file m_res_dir

Implements Model::AbstractDataLoader.

4.4.1.3 saveTrainResults()

```
void Model::DataLoader::saveTrainResults (
            const LatentVectors & theta,
            const LatentVectors & beta,
            const vector< double > & losses ) const  [virtual]
```

Save learnt latent theta & beta vectors and computed loss to file in m_res_dir

Parameters

theta	Learnt theta vectors to save to file
beta	Learnt beta vectors to save to file
losses	Computed loss vector to save to file

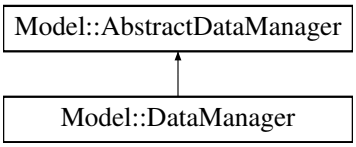
Implements Model::AbstractDataLoader.

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

- models/dataloader.h
- models/dataloader.cpp

4.5 Model::DataManager Class Reference

Inheritance diagram for Model::DataManager:



Public Member Functions

- DataManager (const shared_ptr< DataLoader > &data_loader, const double ratio)
- virtual void loadDataset (const double ratio)
- virtual TrainingData getTrain () const
- virtual TestingData getTest () const
- virtual shared_ptr< vector< int > > getUsers () const
- virtual shared_ptr< vector< int > > getItems () const

Private Member Functions

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

4.5.1 Constructor & Destructor Documentation

4.5.1.1 DataManager()

```
Model::DataManager::DataManager (
            const shared_ptr< DataLoader > & data_loader,
            const double ratio )
```

Initialize DataManager with shared ownership of DataLoader

Parameters

data_loader	shared_ptr to an instance of data_loader
ratio	to split the dataset into train and test

4.5.2 Member Function Documentation

4.5.2.1 getItems()

```
shared_ptr< vector< int > > Model::DataManager::getItems ( ) const  [virtual]
```

Gets all the unique item ids.

Returns

a shared_ptr to the vector of the item ids.

Implements Model::AbstractDataManager.

4.5.2.2 getTest()

```
TestingData Model::DataManager::getTest ( ) const  [virtual]
```

Gets the testing data set.

Returns

TestingData: a shared_ptr of the matrix of the testing data set.

Implements Model::AbstractDataManager.

4.5.2.3 getTrain()

```
TestingData Model::DataManager::getTrain ( ) const  [virtual]
```

Gets the training data set.

Returns

TestingData: a shared_ptr of the matrix of the training data set.

Implements Model::AbstractDataManager.

4.5.2.4 getUsers()

```
shared_ptr< vector< int > > Model::DataManager::getUsers ( ) const  [virtual]
```

Gets all the unique user ids.

Returns

a shared_ptr to the vector of the user ids.

Implements Model::AbstractDataManager.

4.5.2.5 loadDataset()

```
void Model::DataManager::loadDataset (
            const double ratio )  [virtual]
```

Load the user ids, item ids, train data, test data, splitting the dataset by the given 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.

4.5.2.6 split()

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

Splits the 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

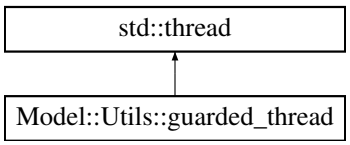
A tuple of <MatrixXd, MatrixXd> type, in which the first matrix is the training data and the second matrix is the testing data.

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

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

4.6 Model::Utils::guarded_thread Struct Reference

Inheritance diagram for Model::Utils::guarded_thread:



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

- models/utils.h

4.7 Model::Utils::ItemMap Struct Reference

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

- models/utils.h

4.8 Model::LatentVectorsSnapshot Struct Reference

`#include <PMF.h>`

4.8.1 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

4.9 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)
- 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
- 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)
- 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 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

4.9.1 Member Function Documentation

4.9.1.1 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

4.9.1.2 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.

4.9.1.3 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

4.9.1.4 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

4.9.1.5 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

4.9.1.6 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

4.9.1.7 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 item_name (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

4.9.1.8 initVectors()

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

4.9.1.9 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

4.9.1.10 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

4.9.1.11 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

4.9.1.12 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

4.9.1.13 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 item_names 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 item_name (string)
N	Number of item recommendations to generate

Returns

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

4.9.1.14 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