Implicit Documentation

Release 0.4.8

Ben Frederickson

Oct 02, 2021

Contents:

Fast Python Collaborative Filtering for Implicit Datasets.

This project provides fast Python implementations of several different popular recommendation algorithms for implicit feedback datasets:

- Alternating Least Squares as described in the papers Collaborative Filtering for Implicit Feedback Datasets and in Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering.
- Bayesian Personalized Ranking
- Logistic Matrix Factorization
- Item-Item Nearest Neighbour models, using Cosine, TFIDF or BM25 as a distance metric

All models have multi-threaded training routines, using Cython and OpenMP to fit the models in parallel among all available CPU cores. In addition, the ALS and BPR models both have custom CUDA kernels - enabling fitting on compatible GPU's. This library also supports using approximate nearest neighbours libraries such as Annoy, NMSLIB and Faiss for speeding up making recommendations.

CHAPTER 1

Implicit

Fast Python Collaborative Filtering for Implicit Datasets

This project provides fast Python implementations of several different popular recommendation algorithms for implicit feedback datasets:

- Alternating Least Squares as described in the papers Collaborative Filtering for Implicit Feedback Datasets and in Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering.
- Bayesian Personalized Ranking
- Item-Item Nearest Neighbour models, using Cosine, TFIDF or BM25 as a distance metric

All models have multi-threaded training routines, using Cython and OpenMP to fit the models in parallel among all available CPU cores. In addition, the ALS and BPR models both have custom CUDA kernels - enabling fitting on compatible GPU's. This library also supports using approximate nearest neighbours libraries such as Annoy, NMSLIB and Faiss for speeding up making recommendations.

1.1 Installation

To install:

```
pip install implicit
```

1.2 Basic Usage

```
import implicit

# initialize a model
model = implicit.als.AlternatingLeastSquares(factors=50)

# train the model on a sparse matrix of item/user/confidence weights
```

(continues on next page)

(continued from previous page)

```
model.fit(item_user_data)

# recommend items for a user
user_items = item_user_data.T.tocsr()
recommendations = model.recommend(userid, user_items)

# find related items
related = model.similar_items(itemid)
```

1.3 Articles about Implicit

These blog posts describe the algorithms that power this library:

- Finding Similar Music with Matrix Factorization
- Faster Implicit Matrix Factorization
- Implicit Matrix Factorization on the GPU
- Approximate Nearest Neighbours for Recommender Systems
- Distance Metrics for Fun and Profit

There are also several other blog posts about using Implicit to build recommendation systems:

- Recommending GitHub Repositories with Google BigQuery and the implicit library
- Intro to Implicit Matrix Factorization: Classic ALS with Sketchfab Models
- A Gentle Introduction to Recommender Systems with Implicit Feedback

1.4 Requirements

This library requires SciPy version 0.16 or later. Running on OSX requires an OpenMP compiler, which can be installed with homebrew: `brew install gcc`.

CHAPTER 2

RecommenderBase

class implicit.recommender_base.**RecommenderBase**

Defines the interface that all recommendations models here expose

fit()

Trains the model on a sparse matrix of item/user/weight

Parameters **item_user** (*csr_matrix*) – A matrix of shape (number_of_items, number_of_users). The nonzero entries in this matrix are the items that are liked by each user. The values are how confident you are that the item is liked by the user.

rank_items()

Rank given items for a user and returns sorted item list.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us (optionally) recalculate user factors (see *recalculate_user* parameter) as required
- **selected_items** (*List of itemids*) –
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples. it only contains items that appears in input parameter selected_items

Return type list

recommend()

Recommends items for a user

Calculates the N best recommendations for a user, and returns a list of itemids, score.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for

- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.
- **N** (*int, optional*) – The number of results to return
- **filter_already_liked_items** (*bool, optional*) – When true, don't return items present in the training set that were rated by the specificed user.
- **filter_items** (*sequence of ints, optional*) – List of extra item ids to filter out from the output
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples

Return type list

similar_items()

Calculates a list of similar items

Parameters

- **itemid** (*int*) – The row id of the item to retrieve similar items for
- **N** (*int, optional*) – The number of similar items to return
- **react_users** (*csr_matrix, optional*) – A sparse matrix of shape (number_items, number_users). This lets us look up the reacted users and their weights for the item.
- **recalculate_item** (*bool, optional*) – When true, don't rely on stored item state and instead recalculate from the passed in react_users

Returns List of (itemid, score) tuples

Return type list

similar_users()

Calculates a list of similar users

Parameters

- **userid** (*int*) – The row id of the user to retrieve similar users for
- **N** (*int, optional*) – The number of similar users to return

Returns List of (userid, score) tuples

Return type list

CHAPTER 3

AlternatingLeastSquares

class implicit.als.**AlternatingLeastSquares**

Alternating Least Squares

A Recommendation Model based off the algorithms described in the paper 'Collaborative Filtering for Implicit Feedback Datasets' with performance optimizations described in 'Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering.'

This factory function switches between the cpu and gpu implementations found in implicit.cpu.als.AlternatingLeastSquares and implicit.gpu.als.AlternatingLeastSquares depending on the use_gpu flag.

Parameters

- **factors** (*int, optional*) – The number of latent factors to compute
- **regularization** (*float, optional*) – The regularization factor to use
- **dtype** (*data-type, optional*) – Specifies whether to generate 64 bit or 32 bit floating point factors
- **use_native** (*bool, optional*) – Use native extensions to speed up model fitting
- **use_cg** (*bool, optional*) – Use a faster Conjugate Gradient solver to calculate factors
- **use_gpu** (*bool, optional*) – Fit on the GPU if available, default is to run on GPU only if available
- **iterations** (*int, optional*) – The number of ALS iterations to use when fitting data
- **calculate_training_loss** (*bool, optional*) – Whether to log out the training loss at each iteration
- **num_threads** (*int, optional*) – The number of threads to use for fitting the model. This only applies for the native extensions. Specifying 0 means to default to the number of cores on the machine.

- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

CHAPTER 4

BayesianPersonalizedRanking

class implicit.bpr.**BayesianPersonalizedRanking**

Bayesian Personalized Ranking

A recommender model that learns a matrix factorization embedding based off minimizing the pairwise ranking loss described in the paper BPR: Bayesian Personalized Ranking from Implicit Feedback.

This factory function returns either the cpu implementation from implicit.cpu.bpr or the gpu implementation from implicit.gpu.bpr depending on the value of the use_gpu flag.

Parameters

- **factors** (*int, optional*) – The number of latent factors to compute
- **learning_rate** (*float, optional*) – The learning rate to apply for SGD updates during training
- **regularization** (*float, optional*) – The regularization factor to use
- **dtype** (*data-type, optional*) – Specifies whether to generate 64 bit or 32 bit floating point factors
- **use_gpu** (*bool, optional*) – Fit on the GPU if available
- **iterations** (*int, optional*) – The number of training epochs to use when fitting the data
- **verify_negative_samples** (*bool, optional*) – When sampling negative items, check if the randomly picked negative item has actually been liked by the user. This check increases the time needed to train but usually leads to better predictions.
- **num_threads** (*int, optional*) – The number of threads to use for fitting the model. This only applies for the native extensions. Specifying 0 means to default to the number of cores on the machine.
- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

CHAPTER 5

LogisticMatrixFactorization

class implicit.lmf.**LogisticMatrixFactorization**

Logistic Matrix Factorization

A collaborative filtering recommender model that learns probabilistic distribution whether user like it or not. Algorithm of the model is described in *Logistic Matrix Factorization for Implicit Feedback Data <https://web.stanford.edu/~rezab/nips2014workshop/submits/logmat.pdf>*

Parameters

- **factors** (*int, optional*) – The number of latent factors to compute
- **learning_rate** (*float, optional*) – The learning rate to apply for updates during training
- **regularization** (*float, optional*) – The regularization factor to use
- **dtype** (*data-type, optional*) – Specifies whether to generate 64 bit or 32 bit floating point factors
- **iterations** (*int, optional*) – The number of training epochs to use when fitting the data
- **neg_prop** (*int, optional*) – The proportion of negative samples. i.e.) "neg_prop = 30" means if user have seen 5 items, then 5 * 30 = 150 negative samples are used for training.
- **use_gpu** (*bool, optional*) – Fit on the GPU if available
- **num_threads** (*int, optional*) – The number of threads to use for fitting the model. This only applies for the native extensions. Specifying 0 means to default to the number of cores on the machine.
- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

item_factors

Array of latent factors for each item in the training set

Type ndarray

user_factors
Array of latent factors for each user in the training set

Type ndarray

fit()
Factorizes the item_users matrix

Parameters

- **item_users** (*coo_matrix*) – Matrix of confidences for the liked items. This matrix should be a coo_matrix where the rows of the matrix are the item, and the columns are the users that liked that item. BPR ignores the weight value of the matrix right now - it treats non zero entries as a binary signal that the user liked the item.
- **show_progress** (*bool, optional*) – Whether to show a progress bar

rank_items()
Rank given items for a user and returns sorted item list.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us (optionally) recalculate user factors (see *recalculate_user* parameter) as required
- **selected_items** (*List of itemids*) –
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples. it only contains items that appears in input parameter selected_items

Return type list

recommend()
Recommends items for a user

Calculates the N best recommendations for a user, and returns a list of itemids, score.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.
- **N** (*int, optional*) – The number of results to return
- **filter_already_liked_items** (*bool, optional*) – When true, don't return items present in the training set that were rated by the specificed user.
- **filter_items** (*sequence of ints, optional*) – List of extra item ids to filter out from the output
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples

Return type list

recommend_all()
Recommends items for all users

Calculates the N best recommendations for all users, and returns numpy ndarray of shape (number_users, N) with item's ids in reversed probability order

Parameters

- **self** (implicit.als.AlternatingLeastSquares) – The fitted recommendation model
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.
- **N** (*int, optional*) – The number of results to return
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items
- **filter_already_liked_items** (*bool, optional*) – This is used to filter out items that have already been liked from the user_items
- **filter_items** (*list, optional*) – List of item id's to exclude from recommendations for all users
- **num_threads** (*int, optional*) – The number of threads to use for sorting scores in parallel by users. Default is number of cores on machine
- **show_progress** (*bool, optional*) – Whether to show a progress bar
- **batch_size** (*int, optional*) – To optimise memory usage while matrix multiplication, users are separated into groups and scored iteratively. By default batch_size == num_threads * 100
- **users_items_offset** (*int, optional*) – Allow to pass a slice of user_items matrix to split calculations

Returns Array of (number_users, N) with item's ids in descending probability order

Return type numpy ndarray

similar_items()
Calculates a list of similar items

Parameters

- **itemid** (*int*) – The row id of the item to retrieve similar items for
- **N** (*int, optional*) – The number of similar items to return
- **react_users** (*csr_matrix, optional*) – A sparse matrix of shape (number_items, number_users). This lets us look up the reacted users and their weights for the item.
- **recalculate_item** (*bool, optional*) – When true, don't rely on stored item state and instead recalculate from the passed in react_users

Returns List of (itemid, score) tuples

Return type list

similar_users()
Calculates a list of similar users

Parameters

- **userid** (*int*) – The row id of the user to retrieve similar users for
- **N** (*int, optional*) – The number of similar users to return

Returns List of (userid, score) tuples

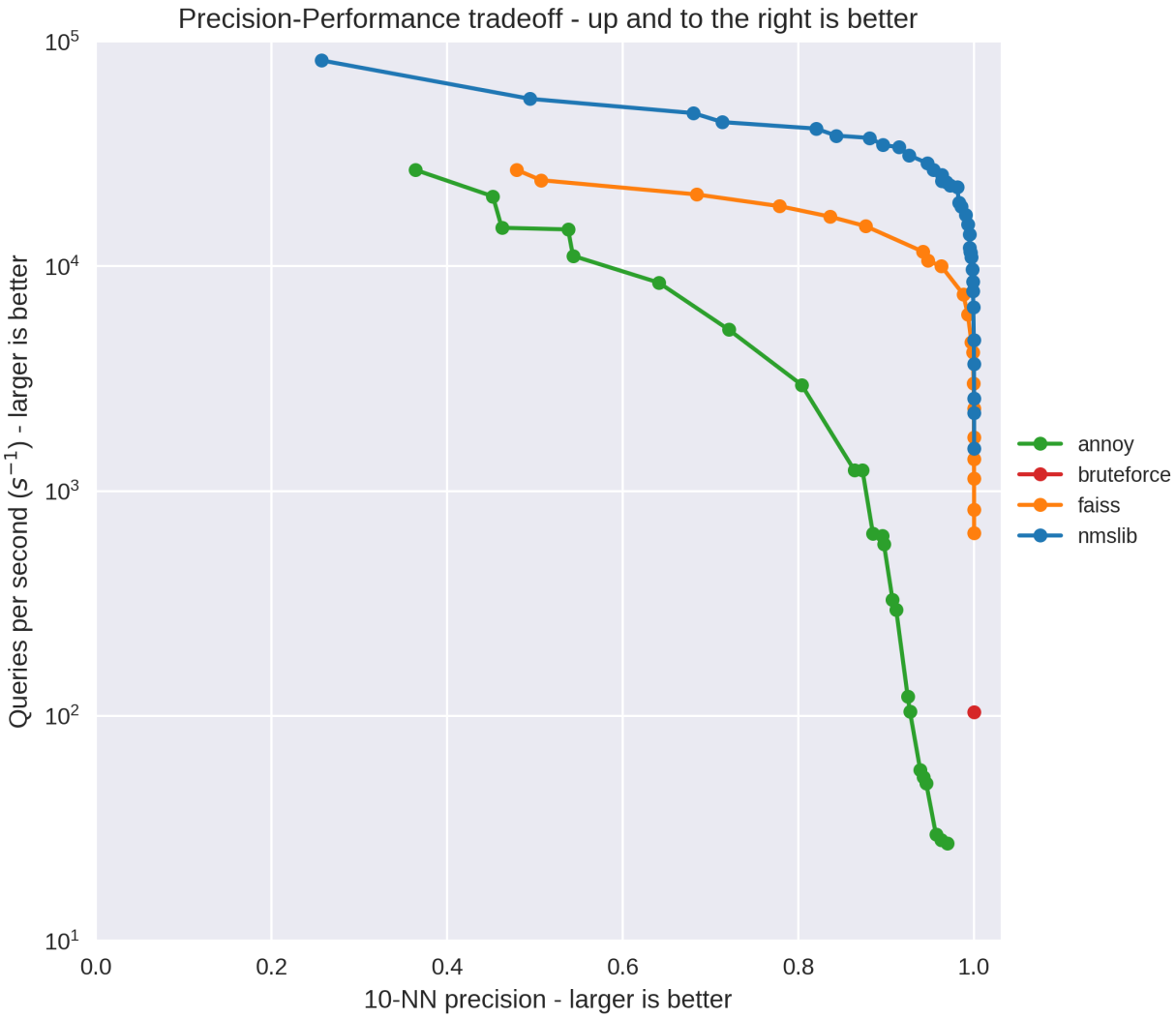
Return type list

CHAPTER 6

Approximate Alternating Least Squares

This library supports using a couple of different approximate nearest neighbours libraries to speed up the recommend and similar_items methods of the AlternatingLeastSquares model.

The potential speedup of using these methods can be quite significant, at the risk of potentially missing relevant results:



See this post comparing the different ANN libraries for more details.

6.1 NMSLibAlternatingLeastSquares

class implicit.approximate_als.**NMSLibAlternatingLeastSquares**(*approximate_similar_items=True*, *approximate_recommend=True*, *method='hnsw'*, *index_params=None*, *query_params=None*, *random_state=None*, **args*, ***kwargs*)

Bases: implicit.cpu.als.AlternatingLeastSquares

Speeds up the base *AlternatingLeastSquares* model by using NMSLib to create approximate nearest neighbours indices of the latent factors.

Parameters

- **method** (*str, optional*) – The NMSLib method to use
- **index_params** (*dict, optional*) – Optional params to send to the createIndex call in NMSLib
- **query_params** (*dict, optional*) – Optional query time params for the NMSLib 'setQueryTimeParams' call
- **approximate_similar_items** (*bool, optional*) – whether or not to build an NMSLIB index for computing similar_items
- **approximate_recommend** (*bool, optional*) – whether or not to build an NMSLIB index for the recommend call
- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

similar_items_index
NMSLib index for looking up similar items in the cosine space formed by the latent item_factors

Type nmslib.FloatIndex

recommend_index
NMSLib index for looking up similar items in the inner product space formed by the latent item_factors

Type nmslib.FloatIndex

fit(*Ciu*, *show_progress=True*)
Factorizes the item_users matrix.

After calling this method, the members 'user_factors' and 'item_factors' will be initialized with a latent factor model of the input data.

The item_users matrix does double duty here. It defines which items are liked by which users (P_iu in the original paper), as well as how much confidence we have that the user liked the item (C_iu).

The negative items are implicitly defined: This code assumes that positive items in the item_users matrix means that the user liked the item. The negatives are left unset in this sparse matrix: the library will assume that means Piu = 0 and Ciu = 1 for all these items. Negative items can also be passed with a higher confidence value by passing a negative value, indicating that the user disliked the item.

Parameters

- **item_users** (*csr_matrix*) – Matrix of confidences for the liked items. This matrix should be a csr_matrix where the rows of the matrix are the item, the columns are the users that liked that item, and the value is the confidence that the user liked the item.
- **show_progress** (*bool, optional*) – Whether to show a progress bar during fitting

recommend(*userid*, *user_items*, *N=10*, *filter_already_liked_items=True*, *filter_items=None*, *recalculate_user=False*)
Recommends items for a user

Calculates the N best recommendations for a user, and returns a list of itemids, score.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.

- **N** (*int, optional*) – The number of results to return
- **filter_already_liked_items** (*bool, optional*) – When true, don't return items present in the training set that were rated by the specificed user.
- **filter_items** (*sequence of ints, optional*) – List of extra item ids to filter out from the output
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples

Return type list

similar_items(*itemid*, *N=10*)

Calculates a list of similar items

Parameters

- **itemid** (*int*) – The row id of the item to retrieve similar items for
- **N** (*int, optional*) – The number of similar items to return
- **react_users** (*csr_matrix, optional*) – A sparse matrix of shape (number_items, number_users). This lets us look up the reacted users and their weights for the item.
- **recalculate_item** (*bool, optional*) – When true, don't rely on stored item state and instead recalculate from the passed in react_users

Returns List of (itemid, score) tuples

Return type list

6.2 AnnoyAlternatingLeastSquares

class implicit.approximate_als.**AnnoyAlternatingLeastSquares**(*approximate_similar_items=True*, *approximate_recommend=True*, *n_trees=50*, *search_k=-1*, *random_state=None*, **args*, ***kwargs*)

Bases: implicit.cpu.als.AlternatingLeastSquares

A version of the *AlternatingLeastSquares* model that uses an Annoy index to calculate similar items and recommend items.

Parameters

- **n_trees** (*int, optional*) – The number of trees to use when building the Annoy index. More trees gives higher precision when querying.
- **search_k** (*int, optional*) – Provides a way to search more trees at runtime, giving the ability to have more accurate results at the cost of taking more time.
- **approximate_similar_items** (*bool, optional*) – whether or not to build an Annoy index for computing similar_items
- **approximate_recommend** (*bool, optional*) – whether or not to build an Annoy index for the recommend call

- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

similar_items_index
Annoy index for looking up similar items in the cosine space formed by the latent item_factors

Type annoy.AnnoyIndex

recommend_index
Annoy index for looking up similar items in the inner product space formed by the latent item_factors

Type annoy.AnnoyIndex

fit (*Ciu*, *show_progress=True*)
Factorizes the item_users matrix.

After calling this method, the members 'user_factors' and 'item_factors' will be initialized with a latent factor model of the input data.

The item_users matrix does double duty here. It defines which items are liked by which users (P_iu in the original paper), as well as how much confidence we have that the user liked the item (C_iu).

The negative items are implicitly defined: This code assumes that positive items in the item_users matrix means that the user liked the item. The negatives are left unset in this sparse matrix: the library will assume that means Piu = 0 and Ciu = 1 for all these items. Negative items can also be passed with a higher confidence value by passing a negative value, indicating that the user disliked the item.

Parameters

- **item_users** (*csr_matrix*) – Matrix of confidences for the liked items. This matrix should be a csr_matrix where the rows of the matrix are the item, the columns are the users that liked that item, and the value is the confidence that the user liked the item.
- **show_progress** (*bool, optional*) – Whether to show a progress bar during fitting

recommend (*userid*, *user_items*, *N=10*, *filter_already_liked_items=True*, *filter_items=None*, *recalculate_user=False*)
Recommends items for a user

Calculates the N best recommendations for a user, and returns a list of itemids, score.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.
- **N** (*int, optional*) – The number of results to return
- **filter_already_liked_items** (*bool, optional*) – When true, don't return items present in the training set that were rated by the specificed user.
- **filter_items** (*sequence of ints, optional*) – List of extra item ids to filter out from the output
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples

Return type list

similar_items(*itemid*, *N=10*)
Calculates a list of similar items

Parameters

- **itemid** (*int*) – The row id of the item to retrieve similar items for
- **N** (*int, optional*) – The number of similar items to return
- **react_users** (*csr_matrix, optional*) – A sparse matrix of shape (number_items, number_users). This lets us look up the reacted users and their weights for the item.
- **recalculate_item** (*bool, optional*) – When true, don't rely on stored item state and instead recalculate from the passed in react_users

Returns List of (itemid, score) tuples

Return type list

6.3 FaissAlternatingLeastSquares

class implicit.approximate_als.**FaissAlternatingLeastSquares**(*approximate_similar_items=True*, *approximate_recommend=True*, *nlist=400*, *nprobe=20*, *use_gpu=False*, *random_state=None*, **args*, ***kwargs*)

Bases: implicit.cpu.als.AlternatingLeastSquares

Speeds up the base *AlternatingLeastSquares* model by using Faiss to create approximate nearest neighbours indices of the latent factors.

Parameters

- **nlist** (*int, optional*) – The number of cells to use when building the Faiss index.
- **nprobe** (*int, optional*) – The number of cells to visit to perform a search.
- **use_gpu** (*bool, optional*) – Whether or not to enable run Faiss on the GPU. Requires faiss to have been built with GPU support.
- **approximate_similar_items** (*bool, optional*) – whether or not to build an Faiss index for computing similar_items
- **approximate_recommend** (*bool, optional*) – whether or not to build an Faiss index for the recommend call
- **random_state** (*int, RandomState or None, optional*) – The random state for seeding the initial item and user factors. Default is None.

similar_items_index
Faiss index for looking up similar items in the cosine space formed by the latent item_factors

Type faiss.IndexIVFFlat

recommend_index
Faiss index for looking up similar items in the inner product space formed by the latent item_factors

Type faiss.IndexIVFFlat

fit(*Ciu*, *show_progress=True*)
Factorizes the item_users matrix.

After calling this method, the members 'user_factors' and 'item_factors' will be initialized with a latent factor model of the input data.

The item_users matrix does double duty here. It defines which items are liked by which users (P_iu in the original paper), as well as how much confidence we have that the user liked the item (C_iu).

The negative items are implicitly defined: This code assumes that positive items in the item_users matrix means that the user liked the item. The negatives are left unset in this sparse matrix: the library will assume that means Piu = 0 and Ciu = 1 for all these items. Negative items can also be passed with a higher confidence value by passing a negative value, indicating that the user disliked the item.

Parameters

- **item_users** (*csr_matrix*) – Matrix of confidences for the liked items. This matrix should be a csr_matrix where the rows of the matrix are the item, the columns are the users that liked that item, and the value is the confidence that the user liked the item.
- **show_progress** (*bool, optional*) – Whether to show a progress bar during fitting

recommend(*userid*, *user_items*, *N=10*, *filter_already_liked_items=True*, *filter_items=None*, *recalculate_user=False*)
Recommends items for a user

Calculates the N best recommendations for a user, and returns a list of itemids, score.

Parameters

- **userid** (*int*) – The userid to calculate recommendations for
- **user_items** (*csr_matrix*) – A sparse matrix of shape (number_users, number_items). This lets us look up the liked items and their weights for the user. This is used to filter out items that have already been liked from the output, and to also potentially calculate the best items for this user.
- **N** (*int, optional*) – The number of results to return
- **filter_already_liked_items** (*bool, optional*) – When true, don't return items present in the training set that were rated by the specificed user.
- **filter_items** (*sequence of ints, optional*) – List of extra item ids to filter out from the output
- **recalculate_user** (*bool, optional*) – When true, don't rely on stored user state and instead recalculate from the passed in user_items

Returns List of (itemid, score) tuples

Return type list

similar_items(*itemid*, *N=10*)
Calculates a list of similar items

Parameters

- **itemid** (*int*) – The row id of the item to retrieve similar items for
- **N** (*int, optional*) – The number of similar items to return
- **react_users** (*csr_matrix, optional*) – A sparse matrix of shape (number_items, number_users). This lets us look up the reacted users and their weights for the item.

- **recalculate_item** (*bool, optional*) – When true, don't rely on stored item state and instead recalculate from the passed in react_users

Returns List of (itemid, score) tuples

Return type list

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

U