

---

# Implicit Documentation

*Release 0.4.0*

**Ben Frederickson**

**Apr 29, 2020**



---

## Contents:

---

<b>1</b>	<b>Implicit</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Basic Usage . . . . .	3
1.3	Articles about Implicit . . . . .	4
1.4	Requirements . . . . .	4
<b>2</b>	<b>RecommenderBase</b>	<b>5</b>
<b>3</b>	<b>AlternatingLeastSquares</b>	<b>7</b>
<b>4</b>	<b>BayesianPersonalizedRanking</b>	<b>11</b>
<b>5</b>	<b>LogisticMatrixFactorization</b>	<b>15</b>
<b>6</b>	<b>Approximate Alternating Least Squares</b>	<b>19</b>
6.1	NMSLibAlternatingLeastSquares . . . . .	20
6.2	AnnoyAlternatingLeastSquares . . . . .	22
6.3	FaissAlternatingLeastSquares . . . . .	24
<b>7</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



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.



### 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`.



---

## 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 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.
- **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 specified 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

**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

---

## AlternatingLeastSquares

---

```
class implicit.als.AlternatingLeastSquares (factors=100, regularization=0.01,
                                             dtype=<type 'numpy.float32'>,
                                             use_native=True, use_cg=True,
                                             use_gpu=False, iterations=15, calculate_training_loss=False,
                                             num_threads=0,
                                             random_state=None)
```

### 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.’

#### 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.

**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

**explain** (*userid, user\_items, itemid, user\_weights=None, N=10*)

Provides explanations for why the item is liked by the user.

**Parameters**

- **userid** (*int*) – The userid to explain recommendations for
- **user\_items** (*csr\_matrix*) – Sparse matrix containing the liked items for the user
- **itemid** (*int*) – The itemid to explain recommendations for
- **user\_weights** (*ndarray, optional*) – Precomputed Cholesky decomposition of the weighted user liked items. Useful for speeding up repeated calls to this function, this value is returned
- **N** (*int, optional*) – The number of liked items to show the contribution for

**Returns**

- **total\_score** (*float*) – The total predicted score for this user/item pair
- **top\_contributions** (*list*) – A list of the top N (itemid, score) contributions for this user/item pair
- **user\_weights** (*ndarray*) – A factorized representation of the user. Passing this in to future ‘explain’ calls will lead to noticeable speedups

**fit** (*item\_users, 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<sub>iu</sub> in the original paper), as well as how much confidence we have that the user liked the item (C<sub>iu</sub>).

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 P<sub>iu</sub> = 0 and C<sub>iu</sub> = 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

**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 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.
- **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 specified 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

**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

---

## 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](#).

### 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.

### **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 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.
- **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 specified 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

**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

---

## 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 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.
- **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 specified 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

**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

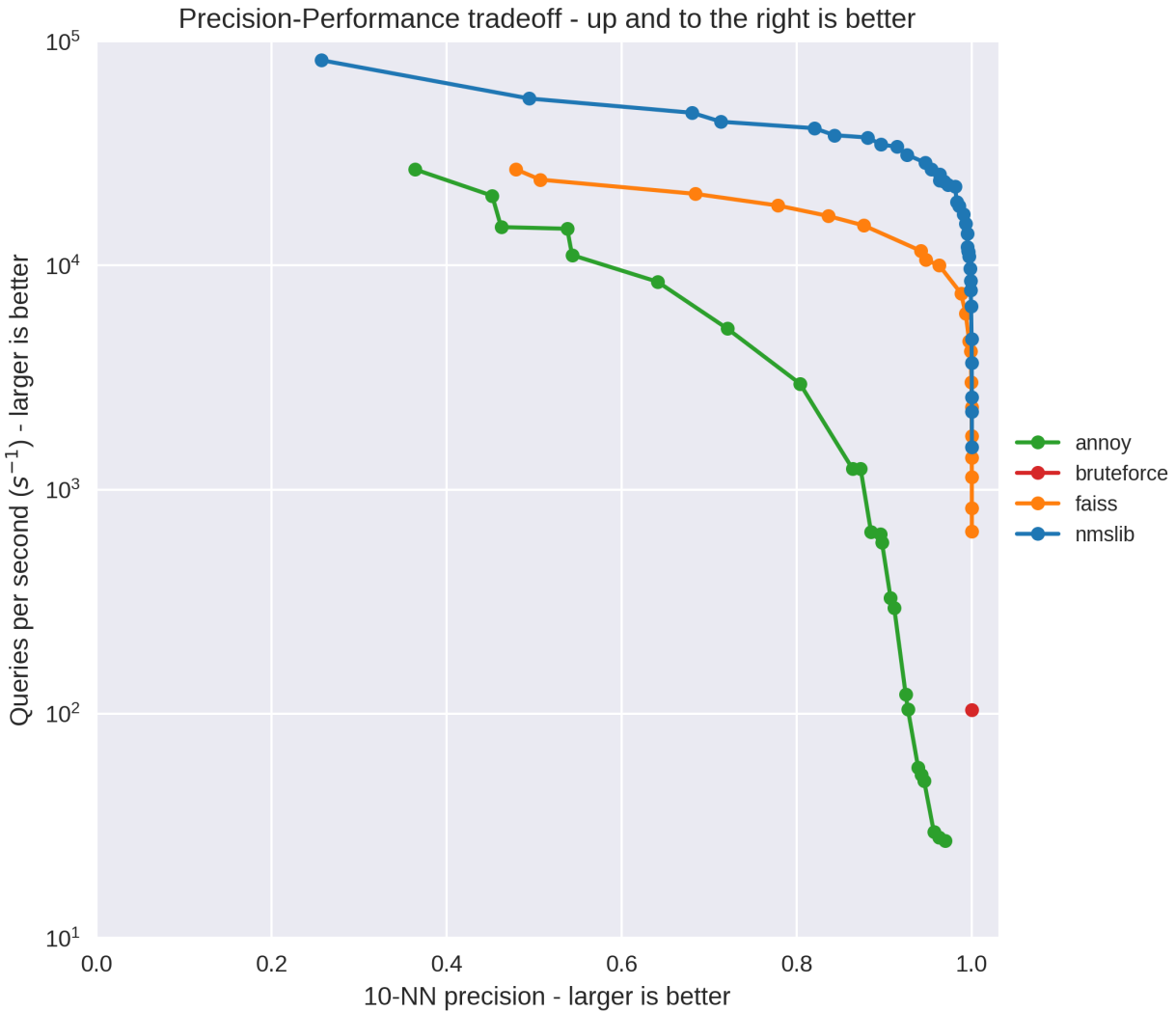
---

### 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.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<sub>iu</sub> in the original paper), as well as how much confidence we have that the user liked the item (C<sub>iu</sub>).

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 P<sub>iu</sub> = 0 and C<sub>iu</sub> = 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 specified 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

**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.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 specified 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

**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.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  $P_{iu} = 0$  and  $C_{iu} = 1$  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 specified 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

**Returns** List of (itemid, score) tuples

**Return type** list



# CHAPTER 7

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`





**A**

AlternatingLeastSquares (class in *implicit.als*), 7

AnnoyAlternatingLeastSquares (class in *implicit.approximate\_als*), 22

**B**

BayesianPersonalizedRanking (class in *implicit.bpr*), 11

**E**

explain() (*implicit.als.AlternatingLeastSquares* method), 8

**F**

FaissAlternatingLeastSquares (class in *implicit.approximate\_als*), 24

fit() (*implicit.als.AlternatingLeastSquares* method), 8

fit() (*implicit.approximate\_als.AnnoyAlternatingLeastSquares* method), 23

fit() (*implicit.approximate\_als.FaissAlternatingLeastSquares* method), 24

fit() (*implicit.approximate\_als.NMSLibAlternatingLeastSquares* method), 21

fit() (*implicit.bpr.BayesianPersonalizedRanking* method), 12

fit() (*implicit.lmf.LogisticMatrixFactorization* method), 16

fit() (*implicit.recommender\_base.RecommenderBase* method), 5

**I**

item\_factors (*implicit.als.AlternatingLeastSquares* attribute), 8

item\_factors (*implicit.bpr.BayesianPersonalizedRanking* attribute), 11

item\_factors (*implicit.lmf.LogisticMatrixFactorization* attribute), 15

**L**

LogisticMatrixFactorization (class in *implicit.lmf*), 15

**N**

NMSLibAlternatingLeastSquares (class in *implicit.approximate\_als*), 20

**R**

rank\_items() (*implicit.als.AlternatingLeastSquares* method), 8

rank\_items() (*implicit.bpr.BayesianPersonalizedRanking* method), 12

rank\_items() (*implicit.lmf.LogisticMatrixFactorization* method), 16

rank\_items() (*implicit.recommender\_base.RecommenderBase* method), 5

recommend() (*implicit.als.AlternatingLeastSquares* method), 9

recommend() (*implicit.approximate\_als.AnnoyAlternatingLeastSquares* method), 23

recommend() (*implicit.approximate\_als.FaissAlternatingLeastSquares* method), 25

recommend() (*implicit.approximate\_als.NMSLibAlternatingLeastSquares* method), 21

recommend() (*implicit.bpr.BayesianPersonalizedRanking* method), 12

recommend() (*implicit.lmf.LogisticMatrixFactorization* method), 16

recommend() (*implicit.recommender\_base.RecommenderBase* method), 5

recommend\_all() (*implicit.als.AlternatingLeastSquares* method), 9

recommend\_all() (*implicit.bpr.BayesianPersonalizedRanking* method), 13

recommend\_all() (*implicit.lmf.LogisticMatrixFactorization* method),

17  
 recommend\_index (implicit.recommender\_base.RecommenderBase similar\_users() (implicit.recommender\_base.RecommenderBase method), 6  
 plicit.approximate\_als.AnnoyAlternatingLeastSquares attribute), 23  
 recommend\_index (implicit.recommender\_base.RecommenderBase similar\_users() (implicit.als.AlternatingLeastSquares attribute), 8  
 plicit.approximate\_als.FaissAlternatingLeastSquares attribute), 24  
 recommend\_index (implicit.recommender\_base.RecommenderBase user\_factors (implicit.bpr.BayesianPersonalizedRanking attribute), 11  
 plicit.approximate\_als.NMSLibAlternatingLeastSquares attribute), 21  
 RecommenderBase (class in implicit.RecommenderBase user\_factors (implicit.lmf.LogisticMatrixFactorization attribute), 16  
 plicit.recommender\_base), 5

## S

similar\_items() (implicit.als.AlternatingLeastSquares method),  
 10  
 similar\_items() (implicit.approximate\_als.AnnoyAlternatingLeastSquares method), 23  
 similar\_items() (implicit.approximate\_als.FaissAlternatingLeastSquares method), 25  
 similar\_items() (implicit.approximate\_als.NMSLibAlternatingLeastSquares method), 22  
 similar\_items() (implicit.bpr.BayesianPersonalizedRanking method), 13  
 similar\_items() (implicit.lmf.LogisticMatrixFactorization method),  
 17  
 similar\_items() (implicit.recommender\_base.RecommenderBase method), 6  
 similar\_items\_index (implicit.approximate\_als.AnnoyAlternatingLeastSquares attribute), 22  
 similar\_items\_index (implicit.approximate\_als.FaissAlternatingLeastSquares attribute), 24  
 similar\_items\_index (implicit.approximate\_als.NMSLibAlternatingLeastSquares attribute), 21  
 similar\_users() (implicit.als.AlternatingLeastSquares method),  
 10  
 similar\_users() (implicit.bpr.BayesianPersonalizedRanking method), 13  
 similar\_users() (implicit.lmf.LogisticMatrixFactorization method),  
 17