rankfm Documentation

Release 0.2.5

Eric Lundquist

Jul 19, 2020

Contents

RankFM is a python implementation of the general Factorization Machines model class described in Rendle 2010 adapted for collaborative filtering recommendation/ranking problems with implicit feedback user-item interaction data. It uses Bayesian Personalized Ranking (BPR) and a variant of Weighted Approximate-Rank Pairwise (WARP) loss to learn model weights via Stochastic Gradient Descent (SGD). It can (optionally) incorporate individual training sample weights and/or user/item auxiliary features to augment the main interaction data for model training.

The core training/prediction/recommendation methods are written in Cython. This makes it possible to scale to millions of users, items, and interactions. Designed for ease-of-use, RankFM accepts both *pd.DataFrame* and *np.ndarray* inputs. You do not have to convert your data to *scipy.sparse* matrices or re-map user/item identifiers to matrix indexes prior to use - RankFM internally maps all user/item identifiers to zero-based integer indexes, but always converts its outputs back to the original user/item identifiers from your data, which can be arbitrary (non-zero-based, non-consecutive) integers or even strings.

In addition to the familiar *fit()*, *predict()*, *recommend()* methods, RankFM includes additional utilities *similiar_users()* and *similar_items()* to find the most similar users/items to a given user/item based on latent factor space embeddings. A number of popular recommendation/ranking evaluation metric functions have been included in the separate *evaluation* module to streamline model tuning and validation.

CHAPTER 1

Dependencies

- Python 3.6+
- numpy >= 1.15
- pandas >= 0.24

CHAPTER 2

Installation

2.1 Prerequisites

To install RankFM's C extensions you will need the GNU Compiler Collection (GCC). Check to see whether you already have it installed:

```
gcc --version
```

If you don't have it already you can easily install it using Homebrew on OSX or your default linux package manager:

```
# OSX
brew install gcc

# linux
sudo yum install gcc

# ensure [gcc] has been installed correctly and is on the system PATH
gcc --version
```

2.2 Package Installation

You can install the latest published version from PyPI using *pip*:

```
pip install rankfm
```

Or alternatively install the current development build directly from GitHub:

```
pip install git+https://github.com/etlundquist/rankfm.git#egg=rankfm
```

CHAPTER 3

Contents

3.1 Welcome to RankFM's Documentation!

RankFM is a python implementation of the general Factorization Machines model class described in Rendle 2010 adapted for collaborative filtering recommendation/ranking problems with implicit feedback user-item interaction data. It uses Bayesian Personalized Ranking (BPR) and a variant of Weighted Approximate-Rank Pairwise (WARP) loss to learn model weights via Stochastic Gradient Descent (SGD). It can (optionally) incorporate individual training sample weights and/or user/item auxiliary features to augment the main interaction data for model training.

The core training/prediction/recommendation methods are written in Cython. This makes it possible to scale to millions of users, items, and interactions. Designed for ease-of-use, RankFM accepts both *pd.DataFrame* and *np.ndarray* inputs. You do not have to convert your data to *scipy.sparse* matrices or re-map user/item identifiers to matrix indexes prior to use - RankFM internally maps all user/item identifiers to zero-based integer indexes, but always converts its outputs back to the original user/item identifiers from your data, which can be arbitrary (non-zero-based, non-consecutive) integers or even strings.

In addition to the familiar *fit()*, *predict()*, *recommend()* methods, RankFM includes additional utilities *similiar_users()* and *similar_items()* to find the most similar users/items to a given user/item based on latent factor space embeddings. A number of popular recommendation/ranking evaluation metric functions have been included in the separate *evaluation* module to streamline model tuning and validation.

3.1.1 Dependencies

- Python 3.6+
- numpy >= 1.15
- pandas >= 0.24

3.1.2 Installation

Prerequisites

To install RankFM's C extensions you will need the GNU Compiler Collection (GCC). Check to see whether you already have it installed:

```
gcc --version
```

If you don't have it already you can easily install it using Homebrew on OSX or your default linux package manager:

```
# OSX
brew install gcc

# linux
sudo yum install gcc

# ensure [gcc] has been installed correctly and is on the system PATH
gcc --version
```

Package Installation

You can install the latest published version from PyPI using *pip*:

```
pip install rankfm
```

Or alternatively install the current development build directly from GitHub:

```
pip install git+https://github.com/etlundquist/rankfm.git#egg=rankfm
```

3.2 Quickstart

Let's work through a simple example of fitting a model, generating recommendations, evaluating performance, and assessing some item-item similarities. The data we'll be using here may already be somewhat familiar: you know it, you love it, it's the MovieLens 1M!

Let's first look at the required shape of the interaction data:

user_id	item_id
3	233
5	377
8	610

It has just two columns: a *user_id* and an *item_id* (you can name these fields whatever you want or use a numpy array instead). Notice that there is no *rating* column - this library is for **implicit feedback** data (e.g. watches, page views, purchases, clicks) as opposed to **explicit feedback** data (e.g. 1-5 ratings, thumbs up/down). Implicit feedback is far more common in real-world recommendation contexts and doesn't suffer from the missing-not-at-random problem of pure explicit feedback approaches.

Now let's import the library, initialize our model, and fit on the training data:

```
from rankfm.rankfm import RankFM
model = RankFM(factors=20, loss='warp', max_samples=20, learning_rate=0.1, learning_
↪schedule='invscaling')
model.fit(interactions_train, epochs=20, verbose=True)
```

If you set *verbose=True* the model will print the current epoch number as well as the epoch's log-likelihood during training. This can be useful to gauge both computational speed and training gains by epoch. If the log likelihood is not increasing then try upping the *learning_rate* or lowering the (*alpha*, *beta*) regularization strength terms. If the log likelihood is starting to bounce up and down try lowering the *learning_rate* or using *learning_schedule='invscaling'* to decrease the learning rate over time. If you run into overflow errors then decrease the feature and/or sample-weight magnitudes and try upping *beta*, especially if you have a small number of dense user-features and/or item-features. Selecting *BPR* loss will lead to faster training times, but *WARP* loss typically yields superior model performance.

Now let's generate some user-item model scores from the validation data:

```
valid_scores = model.predict(interactions_valid, cold_start='nan')
```

this will produce an array of real-valued model scores generated using the Factorization Machines model equation. You can interpret it as a measure of the predicted utility of item (i) for user (u). The *cold_start='nan'* option can be used to set scores to *np.nan* for user/item pairs not found in the training data, or *cold_start='drop'* can be specified to drop those pairs so the results contain no missing values.

Now let's generate our topN recommended movies for each user:

```
valid_recs = model.recommend(valid_users, n_items=10, filter_previous=True, cold_
↪start='drop')
```

The input should be a *pd.Series*, *np.ndarray* or *list* of *user_id* values. You can use *filter_previous=True* to prevent generating recommendations that include any items observed by the user in the training data, which could be useful depending on your application context. The result will be a *pd.DataFrame* where *user_id* values will be the index and the rows will be each user's top recommended items in descending order (best item is in column 0):

user_id	0	1	2	3	4	5	6	7	8	9
3	2396	1265	357	34	2858	3175	1	2028	17	356
5	608	1617	1610	3418	590	474	858	377	924	1036
8	589	1036	2571	2028	2000	1220	1197	110	780	1954

Now let's see how the model is performing wrt the included validation metrics evaluated on the hold-out data:

```
from rankfm.evaluation import hit_rate, reciprocal_rank, discounted_cumulative_gain,␣
↪precision, recall

valid_hit_rate = hit_rate(model, interactions_valid, k=10)
valid_reciprocal_rank = reciprocal_rank(model, interactions_valid, k=10)
valid_dcg = discounted_cumulative_gain(model, interactions_valid, k=10)
valid_precision = precision(model, interactions_valid, k=10)
valid_recall = recall(model, interactions_valid, k=10)
```

```
hit_rate: 0.796
reciprocal_rank: 0.339
dcg: 0.734
precision: 0.159
recall: 0.077
```

That's a Bingo!

Now let's find the most similar other movies for a few movies based on their embedding representations in latent factor space:

```
# Terminator 2: Judgment Day (1991)
model.similar_items(589, n_items=10)
```

```
2571                       Matrix, The (1999)
1527                Fifth Element, The (1997)
2916                      Total Recall (1990)
3527                          Predator (1987)
780             Independence Day (ID4) (1996)
1909    X-Files: Fight the Future, The (1998)
733                         Rock, The (1996)
1376     Star Trek IV: The Voyage Home (1986)
480                      Jurassic Park (1993)
1200                            Aliens (1986)
```

I hope you like explosions. . .

```
# Being John Malkovich (1999)
model.similar_items(2997, n_items=10)
```

```
2599           Election (1999)
3174    Man on the Moon (1999)
2858    American Beauty (1999)
3317       Wonder Boys (2000)
223             Clerks (1994)
3897     Almost Famous (2000)
2395          Rushmore (1998)
2502      Office Space (1999)
2908     Boys Don't Cry (1999)
3481     High Fidelity (2000)
```

Let's get weird. . .

3.3 RankFM

class rankfm.rankfm.**RankFM**(*factors=10*, *loss='bpr'*, *max_samples=10*, *alpha=0.01*, *beta=0.1*, *sigma=0.1*, *learning_rate=0.1*, *learning_schedule='constant'*, *learning_exponent=0.25*)

Factorization Machines for Ranking Problems with Implicit Feedback Data

__init__(*factors=10*, *loss='bpr'*, *max_samples=10*, *alpha=0.01*, *beta=0.1*, *sigma=0.1*, *learning_rate=0.1*, *learning_schedule='constant'*, *learning_exponent=0.25*)

store hyperparameters and initialize internal model state

Parameters

- **factors** – latent factor rank
- **loss** – optimization/loss function to use for training: ['bpr', 'warp']
- **max_samples** – maximum number of negative samples to draw for WARP loss
- **alpha** – L2 regularization penalty on [user, item] model weights
- **beta** – L2 regularization penalty on [user-feature, item-feature] model weights
- **sigma** – standard deviation to use for random initialization of factor weights
- **learning_rate** – initial learning rate for gradient step updates
- **learning_schedule** – schedule for adjusting learning rates by training epoch: ['constant', 'invscaling']

- **learning_exponent** – exponent applied to epoch number to adjust learning rate: scaling = 1 / pow(epoch + 1, learning_exponent)

Returns None

fit (*interactions*, *user_features=None*, *item_features=None*, *sample_weight=None*, *epochs=1*, *verbose=False*)
clear previous model state and learn new model weights using the input data

Parameters

- **interactions** – dataframe of observed user/item interactions: [user_id, item_id]
- **user_features** – dataframe of user metadata features: [user_id, uf_1, … , uf_n]
- **item_features** – dataframe of item metadata features: [item_id, if_1, … , if_n]
- **sample_weight** – vector of importance weights for each observed interaction
- **epochs** – number of training epochs (full passes through observed interactions)
- **verbose** – whether to print epoch number and log-likelihood during training

Returns self

fit_partial (*interactions*, *user_features=None*, *item_features=None*, *sample_weight=None*, *epochs=1*, *verbose=False*)
learn or update model weights using the input data and resuming from the current model state

Parameters

- **interactions** – dataframe of observed user/item interactions: [user_id, item_id]
- **user_features** – dataframe of user metadata features: [user_id, uf_1, … , uf_n]
- **item_features** – dataframe of item metadata features: [item_id, if_1, … , if_n]
- **sample_weight** – vector of importance weights for each observed interaction
- **epochs** – number of training epochs (full passes through observed interactions)
- **verbose** – whether to print epoch number and log-likelihood during training

Returns self

predict (*pairs*, *cold_start='nan'*)
calculate the predicted pointwise utilities for all (user, item) pairs

Parameters

- **pairs** – dataframe of [user, item] pairs to score
- **cold_start** – whether to generate missing values ('nan') or drop ('drop') user/item pairs not found in training data

Returns np.array of real-valued model scores

recommend (*users*, *n_items=10*, *filter_previous=False*, *cold_start='nan'*)
calculate the topN items for each user

Parameters

- **users** – iterable of user identifiers for which to generate recommendations
- **n_items** – number of recommended items to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

- **cold_start** – whether to generate missing values ('nan') or drop ('drop') users not found in training data

Returns pandas dataframe where the index values are user identifiers and the columns are recommended items

similar_items (*item_id*, *n_items=10*)
find the most similar items wrt latent factor space representation

Parameters

- **item_id** – item to search
- **n_items** – number of similar items to return

Returns np.array of topN most similar items wrt latent factor representations

similar_users (*user_id*, *n_users=10*)
find the most similar users wrt latent factor space representation

Parameters

- **user_id** – user to search
- **n_users** – number of similar users to return

Returns np.array of topN most similar users wrt latent factor representations

3.4 Model Evaluation

rankfm model tuning and evaluation functions

rankfm.evaluation.**discounted_cumulative_gain** (*model*, *test_interactions*, *k=10*, *filter_previous=False*)
evaluate discounted cumulative gain wrt out-of-sample observed interactions

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns mean discounted cumulative gain wrt the test users

rankfm.evaluation.**diversity** (*model*, *test_interactions*, *k=10*, *filter_previous=False*)
evaluate the diversity of the model recommendations

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns dataframe of cnt/pct of users recommended for each item

`rankfm.evaluation.hit_rate(model, test_interactions, k=10, filter_previous=False)`
evaluate hit-rate (any match) wrt out-of-sample observed interactions

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns the hit rate or proportion of test users with any matching items

`rankfm.evaluation.precision(model, test_interactions, k=10, filter_previous=False)`
evaluate precision wrt out-of-sample observed interactions

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns mean precision wrt the test users

`rankfm.evaluation.recall(model, test_interactions, k=10, filter_previous=False)`
evaluate recall wrt out-of-sample observed interactions

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns mean recall wrt the test users

`rankfm.evaluation.reciprocal_rank(model, test_interactions, k=10, filter_previous=False)`
evaluate reciprocal rank wrt out-of-sample observed interactions

Parameters

- **model** – trained RankFM model instance
- **test_interactions** – pandas dataframe of out-of-sample observed user/item interactions
- **k** – number of recommendations to generate for each user
- **filter_previous** – remove observed training items from generated recommendations

Returns mean reciprocal rank wrt the test users

Python Module Index

r

Index

Symbols

D

F

H

P

R

S