LensKit Documentation

Release 0.2.0

Michael D. Ekstrand

Jan 10, 2019

Contents:

LensKit is a set of Python tools for experimenting with and studying recommender systems. It provides support for training, running, and evaluating recommender algorithms in a flexible fashion suitable for research and education.

LensKit for Python (also known as LKPY) is the successor to the Java-based LensKit project.

CHAPTER 1

Installation

To install the current release with Anaconda (recommended):

```
conda install -c lenskit lenskit
```

Or you can use `pip`:

```
pip install lenskit
```

To use the latest development version, install directly from GitHub:

```
pip install git+https://github.com/lenskit/lkpy
```

Then see Getting Started.

CHAPTER 2

Resources

- Mailing list, etc.
- Source and issues on GitHub

2.1 Getting Started

We're working on documentation!

For now, this example computes nDCG for an item-based k-NN collaborative filter:

```
import pandas as pd
from lenskit import batch, topn
from lenskit import crossfold as xf
from lenskit.algorithms import knn

ratings = pd.read_csv('ml-100k/u.data', sep='\t',
        names=['user', 'item', 'rating', 'timestamp'])

algo = knn.ItemItem(30)

def eval(train, test):
    model = algo.train(train)
    users = test.user.unique()
    recs = batch.recommend(algo, model, users, 100,
            topn.UnratedCandidates(train))
    # combine with test ratings for relevance data
    res = pd.merge(recs, test, how='left',
                on=('user', 'item'))
    # fill in missing 0s
    res.loc[res.rating.isna(), 'rating'] = 0
    return res

# compute evaluation
```

(continues on next page)

(continued from previous page)

```
splits = xf.partition_users(ratings, 5,
        xf.SampleFrac(0.2)
recs = pd.concat((eval(train, test)
                  for (train, test) in splits))

# compile results
ndcg = recs.groupby('user').rating.apply(topn.ndcg)
```

2.2 Crossfold preparation

The LKPY *crossfold* module provides support for preparing data sets for cross-validation. Crossfold methods are implemented as functions that operate on data frames and return generators of *(train, test)* pairs (*lenskit.crossfold.TTPair* objects). The train and test objects in each pair are also data frames, suitable for evaluation or writing out to a file.

Crossfold methods make minimal assumptions about their input data frames, so the frames can be ratings, purchases, or whatever. They do assume that each row represents a single data point for the purpose of splitting and sampling.

Experiment code should generally use these functions to prepare train-test files for training and evaluating algorithms. For example, the following will perform a user-based 5-fold cross-validation as was the default in the old LensKit:

```
import pandas as pd
import lenskit.crossfold as xf
ratings = pd.read_csv('ml-20m/ratings.csv')
ratings = ratings.rename(columns={'userId': 'user', 'movieId': 'item'})
for i, tp in enumerate(xf.partition_users(ratings, 5, xf.SampleN(5))):
    tp.train.to_csv('ml-20m.exp/train-%d.csv' % (i,))
    tp.train.to_parquet('ml-20m.exp/train-%d.parquet % (i,))
    tp.test.to_csv('ml-20m.exp/test-%d.csv' % (i,))
    tp.test.to_parquet('ml-20m.exp/test-%d.parquet % (i,))
```

2.2.1 Row-based splitting

The simplest preparation methods sample or partition the rows in the input frame. A 5-fold *partition_rows()* split will result in 5 splits, each of which extracts 20% of the rows for testing and leaves 80% for training.

lenskit.crossfold.**partition_rows**(*data*, *partitions*)

Partition a frame of ratings or other datainto train-test partitions. This function does not care what kind of data is in *data*, so long as it is a Pandas DataFrame (or equivalent).

Parameters

- **data** (pandas.DataFrame or equivalent) – a data frame containing ratings or other data you wish to partition.
- **partitions** (*integer*) – the number of partitions to produce

Return type iterator

Returns an iterator of train-test pairs

lenskit.crossfold.**sample_rows**(*data*, *partitions*, *size*, *disjoint=True*)

Sample train-test a frame of ratings into train-test partitions. This function does not care what kind of data is in *data*, so long as it is a Pandas DataFrame (or equivalent).

Parameters

- **data** (pandas.DataFrame or equivalent) – a data frame containing ratings or other data you wish to partition.
- **partitions** (*integer*) – the number of partitions to produce

Return type iterator

Returns an iterator of train-test pairs

2.2.2 User-based splitting

It's often desirable to use users, instead of raw rows, as the basis for splitting data. This allows you to control the experimental conditions on a user-by-user basis, e.g. by making sure each user is tested with the same number of ratings. These methods require that the input data frame have a *user* column with the user names or identifiers.

The algorithm used by each is as follows:

1. Sample or partition the set of user IDs into *n* sets of test users.
2. For each set of test users, select a set of that user's rows to be test rows.
3. **Create a training set for each test set consisting of the non-selected rows from each** of that set's test users, along with all rows from each non-test user.

lenskit.crossfold.**partition_users**(*data*, *partitions: int*, *method: lenskit.crossfold.PartitionMethod*)

Partition a frame of ratings or other data into train-test partitions user-by-user. This function does not care what kind of data is in *data*, so long as it is a Pandas DataFrame (or equivalent) and has a *user* column.

Parameters

- **data** (pandas.DataFrame or equivalent) – a data frame containing ratings or other data you wish to partition.
- **partitions** (*integer*) – the number of partitions to produce
- **method** – The method for selecting test rows for each user.

Return type iterator

Returns an iterator of train-test pairs

lenskit.crossfold.**sample_users**(*data*, *partitions: int*, *size: int*, *method: lenskit.crossfold.PartitionMethod*, *disjoint=True*)

Create train-test partitions by sampling users. This function does not care what kind of data is in *data*, so long as it is a Pandas DataFrame (or equivalent) and has a *user* column.

Parameters

- **data** (pandas.DataFrame or equivalent) – a data frame containing ratings or other data you wish to partition.
- **partitions** – the number of partitions to produce
- **size** – the sample size
- **method** – The method for selecting test rows for each user.
- **disjoint** – whether user samples should be disjoint

Return type iterator

Returns an iterator of train-test pairs

Selecting user test rows

These functions each take a *method* to decide how select each user's test rows. The method is a function that takes a data frame (containing just the user's rows) and returns the test rows. This function is expected to preserve the index of the input data frame (which happens by default with common means of implementing samples).

We provide several partition method factories:

lenskit.crossfold.**SampleN**(*n*)
Randomly select a fixed number of test rows per user/item.

Parameters **n** – The number of test items to select.

lenskit.crossfold.**SampleFrac**(*frac*)
Randomly select a fraction of test rows per user/item.

Parameters **frac** – the fraction of items to select for testing.

lenskit.crossfold.**LastN**(*n*, *col='timestamp'*)
Select a fixed number of test rows per user/item, based on ordering by a column.

Parameters

- **n** – The number of test items to select.
- **col** – The column to sort by.

lenskit.crossfold.**LastFrac**(*frac*, *col='timestamp'*)
Select a fraction of test rows per user/item.

Parameters

- **frac** – the fraction of items to select for testing.
- **col** – The column to sort by.

2.2.3 Utility Classes

class lenskit.crossfold.**PartitionMethod**
Partition methods select test rows for a user or item. Partition methods are callable; when called with a data frame, they return the test rows.

__call__(*udf*)
Subset a data frame.

Parameters **udf** – The input data frame of rows for a user or item.

Returns The data frame of test rows, a subset of *udf*.

class lenskit.crossfold.**TTPair**
Train-test pair (named tuple).

test
Test data for this pair.

train
Train data for this pair.

2.3 Batch-Running Recommendations

The *lenskit.batch* module contains support for *batch-running* recommender and predictor algorithms. This is often used as part of a recommender evaluation experiment.

2.3.1 Rating Prediction

lenskit.batch.**predict**(*algo*, *pairs*, *model=None*)

Generate predictions for user-item pairs. The provided algorithm should be a `algorithms.Predictor` or a function of two arguments: the user ID and a list of item IDs. It should return a dictionary or a `pandas.Series` mapping item IDs to predictions.

Parameters

- **or** (*predictor(callable)*) – py:class:algorithms.Predictor): a rating predictor function or algorithm.
- **pairs** (*pandas.DataFrame*) – a data frame of (`user`, `item`) pairs to predict for. If this frame also contains a `rating` column, it will be included in the result.
- **model** (*any*) – a model for the algorithm.

Returns a frame with columns `user`, `item`, and `prediction` containing the prediction results. If `pairs` contains a *rating* column, this result will also contain a *rating* column.

Return type pandas.DataFrame

2.4 Algorithms

LKPY provides general algorithmic concepts, along with implementations of several algorithms.

2.4.1 Algorithm Interfaces

LKPY's batch routines and utility support for managing algorithms expect algorithms to implement consistent interfaces. This page describes those interfaces.

The interfaces are realized as abstract base classes with the Python `abc` module. Implementations must be registered with their interfaces, either by subclassing the interface or by calling `abc.ABCMeta.register()`.

Rating Prediction

class lenskit.algorithms.**Predictor**

Predicts user ratings of items. Predictions are really estimates of the user's like or dislike, and the `Predictor` interface makes no guarantees about their scale or granularity.

predict(*model*, *user*, *items*, *ratings=None*)

Compute predictions for a user and items.

Parameters

- **model** – the trained model to use. Either `None` or the ratings matrix if the algorithm has no concept of training.
- **user** – the user ID

- **items** (*array-like*) – the items to predict
- **ratings** (*pandas.Series*) – the user's ratings (indexed by item id); if provided, they may be used to override or augment the model's notion of a user's preferences.

Returns scores for the items, indexed by item id.

Return type pandas.Series

Model Training

Most algorithms have some concept of a trained model. The `Trainable` interface captures the ability of a model to be trained and saved to disk.

class lenskit.algorithms.**Trainable**

Models that can be trained and have their models saved.

train(*ratings*)

Train the model on rating/consumption data. Training methods that require additional data may accept it as additional parameters or via class members.

Parameters **ratings** (*pandas.DataFrame*) – rating data, as a matrix with columns 'user', 'item', and 'rating'. The user and item identifiers may be of any type.

Returns the trained model (of an implementation-defined type).

save_model(*model*, *file*)

Save a trained model to a file. The default implementation pickles the model.

Algorithms are allowed to use any format for saving their models, including directories.

Parameters

- **model** – the trained model.
- **file** (*str*) – the file in which to save the model.

load_model(*file*)

Save a trained model to a file.

Parameters **file** (*str*) – the path to file from which to load the model.

Returns the re-loaded model (of an implementation-defined type).

2.4.2 Basic and Utility Algorithms

The *lenskit.algorithms.basic* module contains baseline and utility algorithms for nonpersonalized recommendation and testing.

Personalized Mean Rating Prediction

Fallback Predictor

The `Fallback` rating predictor is a simple hybrid that takes a list of composite algorithms, and uses the first one to return a result to predict the rating for each item.

A common case is to fill in with `Bias` when a primary predictor cannot score an item.

Memorized Predictor

The `Memorized` recommender is primarily useful for test cases. It memorizes a set of rating predictions and returns them.

2.4.3 k-NN Collaborative Filtering

LKPY provides user- and item-based classical k-NN collaborative Filtering implementations. These lightly-configurable implementations are intended to capture the behavior of the Java-based LensKit implementations to provide a good upgrade path and enable basic experiments out of the box.

User-based k-NN

2.4.4 Classic Matrix Factorization

LKPY provides classical matrix factorization implementations.

FunkSVD

FunkSVD is an SVD-like matrix factorization that uses stochastic gradient descent, configured much like coordinate descent, to train the user-feature and item-feature matrices.

2.5 Utility Functions

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

Symbols

L

P

S

T