**Recommender system code walkthrough**

The dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from [MovieLens](<http://movielens.org>), a movie recommendation service. It contains 100004 ratings and 1296 tag applications across 9125 movies. These data were created by 671

users between January 09, 1995 and October 16, 2016. This dataset was generated on October 17, 2016.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included.

Each user is represented by an id, and no other information is provided.

The data are contained in the files `links.csv`, `movies.csv`, `ratings.csv` and `tags.csv`.

This is a development dataset. As such, it may change over time.

This and other GroupLens data sets are publicly available for download at <http://grouplens.org/datasets/>.

User Ids

--------

MovieLens users were selected at random for inclusion. Their ids have been anonymized. User ids are consistent between `ratings.csv` and `tags.csv` (i.e., the same id refers to the same user across the two files).

Movie Ids

---------

Only movies with at least one rating or tag are included in the dataset. These movie ids are consistent with those used on the MovieLens web site (e.g., id `1` corresponds to the URL <https://movielens.org/movies/1>).

Movie ids are consistent between `ratings.csv`, tags.csv`, `movies.csv`, and `links.csv` (i.e., the same id refers to the same movie across these four data files).

Ratings Data File Structure (ratings.csv)

-----------------------------------------

All ratings are contained in the file `ratings.csv`. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:

userId,movieId,rating,timestamp

The lines within this file are ordered first by userId, then, within user, by movieId.

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Tags Data File Structure (tags.csv)

-----------------------------------

All tags are contained in the file `tags.csv`. Each line of this file after the header row represents one tag applied to one movie by one user, and has the following format:

userId,movieId,tag,timestamp

The lines within this file are ordered first by userId, then, within user, by movieId.

Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user.

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Movies Data File Structure (movies.csv)

---------------------------------------

Movie information is contained in the file `movies.csv`. Each line of this file after the header row represents one movie, and has the following format:

movieId,title,genres

Movie titles are entered manually or imported from <https://www.themoviedb.org/>, and include the year of release in parentheses. Errors and inconsistencies may exist in these titles.

Genres are a pipe-separated list, and are selected from the following:

\* Action

\* Adventure

\* Animation

\* Children's

\* Comedy

\* Crime

\* Documentary

\* Drama

\* Fantasy

\* Film-Noir

\* Horror

\* Musical

\* Mystery

\* Romance

\* Sci-Fi

\* Thriller

\* War

\* Western

\* (no genres listed)

Links Data File Structure (links.csv)

---------------------------------------

Identifiers that can be used to link to other sources of movie data are contained in the file `links.csv`. Each line of this file after the header row represents one movie, and has the following format:

movieId,imdbId,tmdbId

movieId is an identifier for movies used by <https://movielens.org>.

E.g., the movie Toy Story has the link <https://movielens.org/movies/1>.

imdbId is an identifier for movies used by <http://www.imdb.com>.

E.g., the movie Toy Story has the link <http://www.imdb.com/title/tt0114709/>.

tmdbId is an identifier for movies used by <https://www.themoviedb.org>.

E.g., the movie Toy Story has the link <https://www.themoviedb.org/movie/862>.

Use of the resources listed above is subject to the terms of each provider.

For all of the code we are using surprise open source library for building and analysing recommender system.

<http://surpriselib.com/>

c:\>pip install surprise

**MovieLens.py**

**loadMovieLensLatestSmall()**

This function loads the data from ratings.csv file and returns a ratings dataset and creates a dictionary of movieid to moviname and vice versa.

Reader class from surprise library used create a format to read data from ratings.csv file as a reader object.

Then Dataset class is used to create ratingsDataset from ratings.csv file using reader object.

Finally ratingsDataset is returned.

**getUserRatings(user)**

This function takes in user id as a parameter to provide list of user ratings.

This function reads the ratings.csv file and creates list of tuples of movieid and rating for the

User id passed as parameter and returns this list.

**getPopularityRanks()**

This function reads in the ratings.csv file and returns the dictionary rankings which contain popular movies at top.

This is done by sorting ratings dictionary in descending order based on the movie count and rank highest count (most popular) movies in ascending order from 1 to n.

**getGenres()**

This function reads in the movies.csv file, split the string containing different genres and stores into a genreList and also creates a genre id list which contains all genre ids.

And then convert integer-encoded genre lists to bitfields that we can treat as vectors.

And returns the genres dictionary containing pairs of movie ids and bitfield vectors.

**getYears()**

This function reads in the movies.csv file and extracts the release year from the title column

using regular expression and returns the dictionary containing pairs of movie ids and their release years.

**getMiseEnScene()**

This function reads LLVisualFeatures13K\_Log.csv file extracts list of information like avgShotLength, meanColorVariance, stddevColorVariance, meanMotion, stddevMotion, meanLightingKey, numShots from file for each movie id and creates a mise data dictionary

Containing pair of movie id and the above created list as value and returns this mise dictionary.

**getMovieName(movieID)**

This function returns Movie Name using movieID\_to\_name dictionary for the input movie id.

**getMovieID()**

This function returns Movie ID using name\_to\_movieID dictionary for the input movieName.

**Content based collaborative filtering**

Here content based algorithm is implemented using custom ContentKNNAlgorithm with the help of surpriselib library by deriving AlgoBase class.

Here by default k neighbours are 40 which is something we experimentin practice i.e. k=40.

Next we implemented fit() function which gets called by surpriselib when we train the algorithm.

Here we are building up 2D array that serves as a lookup of the content based similarity score between any two movies. This takes some run time when it process every 100 movies.

We translate item ids and users ids to their inner ids that’s the job of predict() function.

Then we compute genre based and release year based similarity scores for every possible pair and multiply them together to generate combined content based similarity score.

The computeGenreSimilarity() function is based on cosine similarity metric between each movie treating each movie’s genre has coordinates in 18 dimensional space that represents all possible genres. computeYearSimilarity() function is using exponential decay function to give more way to movies released at around the same time.

computeMiseEnSceneSimilarity() function is used to get mise en scene similarity can be ignored while explaining.

Estimate() is the function where the K-Nearest Neighbours algorithm is happening.

We select the k-nearest movies a user has rated to the one we are trying to make prediction for based on the genres and release years. Then we compute a weighted average based on the similarity scores and user ratings.

The output contains **outperforming random recommendations**:

The extract of the output showing the scores for both random and ContentKNN:

**Algorithm RMSE MAE**

**ContentKNN 0.9375 0.7263**

**Random 1.4385 1.1478**

**Legend:**

**RMSE: Root Mean Squared Error. Lower values mean better accuracy.**

**MAE: Mean Absolute Error. Lower values mean better accuracy.**

--------------------------------------------------------------------------------------------------------------------------

Here the code level workflow is as follows:

Loading movie ratings...

Computing movie popularity ranks so we can measure novelty later...

Estimating biases using als...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Evaluating ContentKNN ...

Evaluating accuracy...

Computing content-based similarity matrix...

0 of 8211

100 of 8211

200 of 8211

…………………..

…………………..

…………………..

…………………..

8000 of 8211

8100 of 8211

8200 of 8211

...done.

Analysis complete.

Evaluating Random ...

Evaluating accuracy...

Analysis complete.

Algorithm RMSE MAE

ContentKNN 0.9375 0.7263

Random 1.4385 1.1478

Legend:

RMSE: Root Mean Squared Error. Lower values mean better accuracy.

MAE: Mean Absolute Error. Lower values mean better accuracy.

Using recommender ContentKNN

Building recommendation model...

Computing content-based similarity matrix...

0 of 9066

100 of 9066

200 of 9066

300 of 9066

…………………..

…………………..

…………………..

8800 of 9066

8900 of 9066

9000 of 9066

...done.

Computing recommendations...

We recommend:

Presidio, The (1988) 3.841314676872932

Femme Nikita, La (Nikita) (1990) 3.839613347087336

Wyatt Earp (1994) 3.8125061475551796

Shooter, The (1997) 3.8125061475551796

Bad Girls (1994) 3.8125061475551796

The Hateful Eight (2015) 3.812506147555179

True Grit (2010) 3.812506147555179

Open Range (2003) 3.812506147555179

Big Easy, The (1987) 3.7835412549266985

Point Break (1991) 3.764158410102279

Using recommender Random

Building recommendation model...

Computing recommendations...

We recommend:

Sleepers (1996) 5

Beavis and Butt-Head Do America (1996) 5

Fear and Loathing in Las Vegas (1998) 5

Happiness (1998) 5

Summer of Sam (1999) 5

Bowling for Columbine (2002) 5

Babe (1995) 5

Birdcage, The (1996) 5

Carlito's Way (1993) 5

Wizard of Oz, The (1939) 5

===========================================================================

**Neighbourhood based collaborative filtering**

**1. User based collaborative filtering**

This is strictly for generating top N recommendations and no point to predict user ratings, so we have not included into evaluation framework in the earlier project.

Also we are not measuring accuracy here we just doing top N recommenations.

Step1. Loads up data from movilens. And builds up complete training set so we are not going to build any test set as we are not going to measure accuracy here.

Next we builds up the similarity metrics between every possible user pair. To do this quickly and easily we are using surpriselib’s KNNBasic algorithm which build up the similarity metrics.

The sim\_options parameter here specifies we are use cosine similarity metrics, we can also specify MSD or Pearson here. We are also specifying that we want user based similarities so we get metrics mapping user to user similarity scores. Then we use surpriselib to actually build that similarity scores. Then we extract all the similar users to a test user i.e. user 85.

Then we use heapq.nlargest() function to quickly sort all of the users by their similarity to user 85 and pluck out the top k results to get a neighbourhood of similar users.

Next we build up recommendation candidates. Finally we sort the recommendations candidates by their final scores and through the top N of those and print the movies.

O/p:

Computing the cosine similarity matrix...

Done computing similarity matrix.

Computing the cosine similarity matrix...

Done computing similarity matrix.

Inception (2010) 3.3

Star Wars: Episode V - The Empire Strikes Back (1980) 2.4

Bourne Identity, The (1988) 2.0

Crouching Tiger, Hidden Dragon (Wo hu cang long) (2000) 2.0

Dark Knight, The (2008) 2.0

Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966) 1.9

Departed, The (2006) 1.9

Dark Knight Rises, The (2012) 1.9

Back to the Future (1985) 1.9

Gravity (2013) 1.8

Fight Club (1999) 1.8

**2. Item based collaborative filtering**

Here the general approach is similar to user based collaborative filtering so most of the code is similar. Here we just focus on relationship between items than users.

Here we are selecting sim\_options in KNNBasic with user\_based = False, this tells surprise library to generate item to item similarity metrics using cosine similarity metrics.

Rest of the code is same where we get similarity movie recommendation based on item to item similarity scores.

O/p:

Computing the cosine similarity matrix...

Done computing similarity matrix.

Computing the cosine similarity matrix...

Done computing similarity matrix.

James Dean Story, The (1957) 10.0

Get Real (1998) 9.987241120712646

Kiss of Death (1995) 9.966881877751941

Set It Off (1996) 9.963732215657119

How Green Was My Valley (1941) 9.943984081065269

Amos & Andrew (1993) 9.93973694500253

My Crazy Life (Mi vida loca) (1993) 9.938290487546041

Grace of My Heart (1996) 9.926255896645218

Fanny and Alexander (Fanny och Alexander) (1982) 9.925699671455906

Wild Reeds (Les roseaux sauvages) (1994) 9.916226404418774

Edge of Seventeen (1998) 9.913028764691676

**3. Neighbourhood based (KNN based) collaborative filtering**

Here we are importing KNNBasic package as it implements both user based and item KNN recommendations. To get user based KNN recommendation we set sim\_options user\_based = True and to get item based KNN recommendation we set sim\_options user\_based = False.

Rest surprise library will take care of.

Here in the output user based and item based accuracy came out to be almost similary.

User based accuracy is slightly better than the accuracy of item based recommendations.

But this slight difference we don’t need to worry about.

O/p:

Loading movie ratings...

Computing movie popularity ranks so we can measure novelty later...

Estimating biases using als...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Evaluating User KNN ...

Evaluating accuracy...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Analysis complete.

Evaluating Item KNN ...

Evaluating accuracy...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Analysis complete.

Evaluating Random ...

Evaluating accuracy...

Analysis complete.

Algorithm RMSE MAE

User KNN 0.9961 0.7711

Item KNN 0.9995 0.7798

Random 1.4385 1.1478

Legend:

RMSE: Root Mean Squared Error. Lower values mean better accuracy.

MAE: Mean Absolute Error. Lower values mean better accuracy.

Using recommender User KNN

Building recommendation model...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Computing recommendations...

We recommend:

One Magic Christmas (1985) 5

Step Into Liquid (2002) 5

Art of War, The (2000) 5

Taste of Cherry (Ta'm e guilass) (1997) 5

King Is Alive, The (2000) 5

Innocence (2000) 5

MaelstrÃ¶m (2000) 5

Faust (1926) 5

Seconds (1966) 5

Amazing Grace (2006) 5

Using recommender Item KNN

Building recommendation model...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Computing recommendations...

We recommend:

Life in a Day (2011) 5

Under Suspicion (2000) 5

Asterix and the Gauls (AstÃ©rix le Gaulois) (1967) 5

Find Me Guilty (2006) 5

Elementary Particles, The (Elementarteilchen) (2006) 5

Asterix and the Vikings (AstÃ©rix et les Vikings) (2006) 5

From the Sky Down (2011) 5

Vive L'Amour (Ai qing wan sui) (1994) 5

Vagabond (Sans toit ni loi) (1985) 5

Ariel (1988) 5

Using recommender Random

Building recommendation model...

Computing recommendations...

We recommend:

Sleepers (1996) 5

Beavis and Butt-Head Do America (1996) 5

Fear and Loathing in Las Vegas (1998) 5

Happiness (1998) 5

Summer of Sam (1999) 5

Bowling for Columbine (2002) 5

Babe (1995) 5

Birdcage, The (1996) 5

Carlito's Way (1993) 5

Wizard of Oz, The (1939) 5

===========================================================================

**Matrix Factorization Methods**

**SVD recommendation based of matrix factorization techniques using PCA (Principal Component Analysis)**

Surprise contains a couple of SVD implementations i.e. the regular SVD and slight variant of it called SVDpp (SVD++) for providing recommendations.

The difference in SVD++ is to use a loss function while running stochastic gradient descent.

In SVD++ this loss function takes into account the idea that nearly rating an item at all is some sort of implicit interest in the item no matter what the rating was. The implementation is taken cared by the surprise library.

The O/p contains below accuracy where SVD++ have better accuracy than SVD.

Then we work on improving SVD by tuning the hyper-parameters for SVD.

Here the code is almost similar except GridSearchCV object which takes SVD and param\_grid as an argument with measures RMSE and MAE and CV=3.

We settled on 20 epochs and learning rate lr\_all of 0.005 and 50 factors.

The tuned result gives RMSE of 0.9002 over unturned result with RMSE of 0.9033.

The difference is small but it result into very different top N movie recommendations result.

We can consider both these results are good.

O/p:

Loading movie ratings...

Computing movie popularity ranks so we can measure novelty later...

Estimating biases using als...

Computing the cosine similarity matrix...

Done computing similarity matrix.

Evaluating SVD ...

Evaluating accuracy...

Analysis complete.

Evaluating SVD++ ...

Evaluating accuracy...

Analysis complete.

Evaluating Random ...

Evaluating accuracy...

Analysis complete.

Algorithm RMSE MAE

SVD 0.9039 0.6984

SVD++ 0.8943 0.6887

Random 1.4359 1.1493

Legend:

RMSE: Root Mean Squared Error. Lower values mean better accuracy.

MAE: Mean Absolute Error. Lower values mean better accuracy.

Using recommender SVD

Building recommendation model...

Computing recommendations...

We recommend:

Gladiator (1992) 4.520884890007874

Philadelphia Story, The (1940) 4.420701711947352

Stand by Me (1986) 4.3959589752178365

Moon (2009) 4.372613693384055

Happiness (1998) 4.369493252705134

American Graffiti (1973) 4.353470600109924

And Your Mother Too (Y tu mamÃ¡ tambiÃ©n) (2001) 4.349192492630499

Wallace & Gromit: A Close Shave (1995) 4.3154412154304085

Band of Brothers (2001) 4.315414828016616

Seven Samurai (Shichinin no samurai) (1954) 4.311102920673881

Using recommender SVD++

Building recommendation model...