

 [learn-co-curriculum](#) / [dsc-implementing-recommender-systems-lab](#) Public [View license](#) 3 stars  167 forks Star Watch ▾[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) solution ▾

...

This branch is [3 commits ahead](#), [4 commits behind](#) master.

sumedh10 update readme ...

on Dec 12, 2019  6[View code](#) README.md

Implementing Recommender Systems - Lab

Introduction

In this lab, you'll practice creating a recommender system model using `surprise`. You'll also get the chance to create a more complete recommender system pipeline to obtain the top recommendations for a specific user.

Objectives

In this lab you will:

- Use surprise's built-in reader class to process data to work with recommender algorithms
- Obtain a prediction for a specific user for a particular item
- Introduce a new user with rating to a rating matrix and make recommendations for them

- Create a function that will return the top n recommendations for a user

For this lab, we will be using the famous 1M movie dataset. It contains a collection of user ratings for many different movies. In the last lesson, you were exposed to working with `surprise` datasets. In this lab, you will also go through the process of reading in a dataset into the `surprise` dataset format. To begin with, load the dataset into a Pandas `DataFrame`. Determine which columns are necessary for your recommendation system and drop any extraneous ones.

```
import pandas as pd
df = pd.read_csv('./ml-latest-small/ratings.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
userId      100836 non-null int64
movieId     100836 non-null int64
rating      100836 non-null float64
timestamp   100836 non-null int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

```
# Drop unnecessary columns
new_df = df.drop(columns='timestamp')
```

It's now time to transform the dataset into something compatible with `surprise`. In order to do this, you're going to need `Reader` and `Dataset` classes. There's a method in `Dataset` specifically for loading dataframes.

```
from surprise import Reader, Dataset
reader = Reader()
data = Dataset.load_from_df(new_df, reader)
```

Let's look at how many users and items we have in our dataset. If using neighborhood-based methods, this will help us determine whether or not we should perform user-user or item-item similarity

```
dataset = data.build_full_trainset()
print('Number of users: ', dataset.n_users, '\n')
```

```
print('Number of items: ', dataset.n_items)
```

Number of users: 610

Number of items: 9724

Determine the best model

Now, compare the different models and see which ones perform best. For consistency sake, use RMSE to evaluate models. Remember to cross-validate! Can you get a model with a higher average RMSE on test data than 0.869?

```
# importing relevant libraries
from surprise.model_selection import cross_validate
from surprise.prediction_algorithms import SVD
from surprise.prediction_algorithms import KNNWithMeans, KNNBasic, KNNBaseline
from surprise.model_selection import GridSearchCV
import numpy as np

## Perform a gridsearch with SVD
# ⚠ This cell may take several minutes to run
params = {'n_factors': [20, 50, 100],
          'reg_all': [0.02, 0.05, 0.1]}
g_s_svd = GridSearchCV(SVD, param_grid=params, n_jobs=-1)
g_s_svd.fit(data)

print(g_s_svd.best_score)
print(g_s_svd.best_params)

{'rmse': 0.8689250510051669, 'mae': 0.6679404366294037}
{'rmse': {'n_factors': 50, 'reg_all': 0.05}, 'mae': {'n_factors': 100, 'reg_all': 0.05}}
```

```
# cross validating with KNNBasic
knn_basic = KNNBasic(sim_options={'name': 'pearson', 'user_based': True})
cv_knn_basic = cross_validate(knn_basic, data, n_jobs=-1)
```

```

for i in cv_knn_basic.items():
    print(i)
print('-----')
print(np.mean(cv_knn_basic['test_rmse']))

('test_rmse', array([0.97646619, 0.97270627, 0.97874535, 0.97029184,
0.96776748]))
('test_mae', array([0.75444119, 0.75251222, 0.7531242 , 0.74938542, 0.75152129]))
('fit_time', (0.46678805351257324, 0.54010009765625, 0.7059998512268066,
0.5852491855621338, 1.0139541625976562))
('test_time', (2.308177947998047, 2.4834508895874023, 2.6563329696655273,
2.652374029159546, 1.2219891548156738))
-----
0.9731954260849399

# cross validating with KNNBaseline
knn_baseline = KNNBaseline(sim_options={'name':'pearson', 'user_based':True})
cv_knn_baseline = cross_validate(knn_baseline,data)

Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.

for i in cv_knn_baseline.items():
    print(i)

np.mean(cv_knn_baseline['test_rmse'])

```

```
( 'test_rmse', array([0.87268017, 0.88765352, 0.87311917, 0.88706914,
0.87043399]))
( 'test_mae', array([0.66796685, 0.676203 , 0.66790869, 0.67904038, 0.66459155]))
( 'fit_time', (0.6972200870513916, 0.7296440601348877, 0.5842609405517578,
0.609612226486206, 0.61130690574646))
( 'test_time', (1.5466029644012451, 1.567044973373413, 1.6441452503204346,
1.5709199905395508, 1.6216418743133545))
```

```
0.8781911983703239
```

Based off these outputs, it seems like the best performing model is the SVD model with `n_factors = 50` and a regularization rate of 0.05. Use that model or if you found one that performs better, feel free to use that to make some predictions.

Making Recommendations

It's important that the output for the recommendation is interpretable to people. Rather than returning the `movie_id` values, it would be far more valuable to return the actual title of the movie. As a first step, let's read in the movies to a dataframe and take a peek at what information we have about them.

```
df_movies = pd.read_csv('./ml-latest-small/movies.csv')
```

```
df_movies.head()
```

```
<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

```
</style>
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

Making simple predictions

Just as a reminder, let's look at how you make a prediction for an individual user and item. First, we'll fit the SVD model we had from before.

```
svd = SVD(n_factors= 50, reg_all=0.05)
svd.fit(dataset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x11952ab38>
```

```
svd.predict(2, 4)
```

```
Prediction(uid=2, iid=4, r_ui=None, est=3.0129484092252135, details=
{'was_impossible': False})
```

This prediction value is a tuple and each of the values within it can be accessed by way of indexing. Now let's put our knowledge of recommendation systems to do something interesting: making predictions for a new user!

Obtaining User Ratings

It's great that we have working models and everything, but wouldn't it be nice to get to recommendations specifically tailored to your preferences? That's what we'll be doing now. The first step is to create a function that allows us to pick randomly selected movies. The function should present users with a movie and ask them to rate it. If they have not seen the movie, they should be able to skip rating it.

The function `movie_rater()` should take as parameters:

- `movie_df` : DataFrame - a dataframe containing the movie ids, name of movie, and genres
- `num` : int - number of ratings
- `genre` : string - a specific genre from which to draw movies

The function returns:

- `rating_list` : list - a collection of dictionaries in the format of `{'userId': int , 'movieId': int , 'rating': float}`

This function is optional, but fun :)

```
def movie_rater(movie_df,num, genre=None):
    userID = 1000
    rating_list = []
    while num > 0:
        if genre:
            movie = movie_df[movie_df['genres'].str.contains(genre)].sample(1)
        else:
            movie = movie_df.sample(1)
        print(movie)
        rating = input('How do you rate this movie on a scale of 1-5, press n if you
            if rating == 'n':
                continue
            else:
                rating_one_movie = {'userId':userID,'movieId':movie['movieId'].values[0]}
                rating_list.append(rating_one_movie)
                num -= 1
    return rating_list
```

```
user_rating = movie_rater(df_movies, 4, 'Comedy')
```

```

      movieId      title      genres
6579    55245  Good Luck Chuck (2007)  Comedy|Romance
How do you rate this movie on a scale of 1-5, press n if you have not seen :
5

      movieId      title      genres
1873    2491  Simply Irresistible (1999)  Comedy|Romance
How do you rate this movie on a scale of 1-5, press n if you have not seen :
4

      movieId      title      genres
3459    4718  American Pie 2 (2001)  Comedy
How do you rate this movie on a scale of 1-5, press n if you have not seen :
4

      movieId      title      genres
4160    5990  Pinocchio (2002)  Children|Comedy|Fantasy
How do you rate this movie on a scale of 1-5, press n if you have not seen :
3

```

If you're struggling to come up with the above function, you can use this list of user ratings to complete the next segment

`user_rating`

```

[{'userId': 1000, 'movieId': 55245, 'rating': '5'},
 {'userId': 1000, 'movieId': 2491, 'rating': '4'},
 {'userId': 1000, 'movieId': 4718, 'rating': '4'},
 {'userId': 1000, 'movieId': 5990, 'rating': '3'}]

```

Making Predictions With the New Ratings

Now that you have new ratings, you can use them to make predictions for this new user. The proper way this should work is:

- add the new ratings to the original ratings DataFrame, read into a `surprise` dataset
- train a model using the new combined DataFrame
- make predictions for the user
- order those predictions from highest rated to lowest rated
- return the top n recommendations with the text of the actual movie (rather than just the index number)


```
## add the new ratings to the original ratings DataFrame
new_ratings_df = new_df.append(user_rating,ignore_index=True)
new_data = Dataset.load_from_df(new_ratings_df,reader)
```

```
# train a model using the new combined DataFrame
svd_ = SVD(n_factors= 50, reg_all=0.05)
svd_.fit(new_data.build_full_trainset())
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x11daeb898>
```

```
# make predictions for the user
# you'll probably want to create a list of tuples in the format (movie_id, predicted
list_of_movies = []
for m_id in new_df['movieId'].unique():
    list_of_movies.append( (m_id,svd_.predict(1000,m_id)[3]))
```

```
# order the predictions from highest to lowest rated
ranked_movies = sorted(list_of_movies, key=lambda x:x[1], reverse=True)
```


For the final component of this challenge, it could be useful to create a function `recommended_movies()` that takes in the parameters:

- `user_ratings` : list - list of tuples formulated as (user_id, movie_id) (should be in order of best to worst for this individual)
- `movie_title_df` : DataFrame
- `n` : int - number of recommended movies

The function should use a `for` loop to print out each recommended n movies in order from best to worst

```
# return the top n recommendations using the
def recommended_movies(user_ratings,movie_title_df,n):
    for idx, rec in enumerate(user_ratings):
        title = movie_title_df.loc[movie_title_df['movieId'] == int(rec[0])]['ti
        print('Recommendation # ', idx+1, ': ', title, '\n')
        n -= 1
    if n == 0:
        break
```

```
recommended_movies(ranked_movies,df_movies,5)
```



```
Recommendation # 1 : 277    Shawshank Redemption, The (1994)
Name: title, dtype: object

Recommendation # 2 : 680    Philadelphia Story, The (1940)
Name: title, dtype: object

Recommendation # 3 : 686    Rear Window (1954)
Name: title, dtype: object

Recommendation # 4 : 602    Dr. Strangelove or: How I Learned to Stop Worr...
Name: title, dtype: object

Recommendation # 5 : 926    Amadeus (1984)
Name: title, dtype: object
```

Level Up (Optional)

- Try and chain all of the steps together into one function that asks users for ratings for a certain number of movies, then all of the above steps are performed to return the top n recommendations
- Make a recommender system that only returns items that come from a specified genre

Summary

In this lab, you got the chance to implement a collaborative filtering model as well as retrieve recommendations from that model. You also got the opportunity to add your own recommendations to the system to get new recommendations for yourself! Next, you will learn how to use Spark to make recommender systems.

Releases

No releases published

Packages

No packages published

Contributors 3



fpolchow Forest Polchow



sumedh10 Sumedh Panchadhar



Imcm18

Languages

● **Jupyter Notebook** 100.0%