Published in Hacktive Devs

Ola Follow
Mar 10, 2019 · 2 min read · Listen

Save

Recommender System made easy with Scikit-Surprise

https://www.offerzen.com/blog/how-to-build-a-content-based-recommender-system-for-your-product

> *A **recommender system** is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general.*

- Collaborative filtering

- Content-based filtering
- Hybrid recommender system

In this tutorial, I'll be focusing on Collaborative filtering. In Collaborative filtering, the model learns from the user's past behavior, user's decision, preference to predict items the user might have an interest in.

Scikit-Surprise is an easy-to-use Python scikit for recommender systems, another example of python scikit is Scikit-learn which has lots of awesome estimators. To install surprise, type this on your CMD/Terminal

```
pip install scikit-surprise
```

Preprocessing

The first thing is to preprocess our data. We have to check the shape, description, a number of unique value, columns and analyze to get more insights from our data.

```
data = pd.read_csv('ratings.csv')
```

In [9]:
```
data.head()
```

Out[9]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

In [12]:
```
data.rating.value_counts()
```

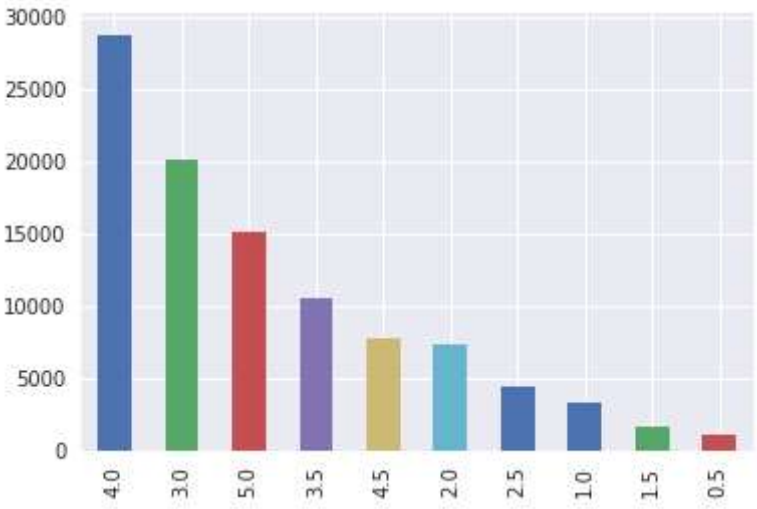
Out[12]:
```
4.0    28750
3.0    20064
5.0    15095
3.5    10538
4.5     7723
2.0     7271
2.5     4449
```

```
1.0     3326
```

load.ipynb hosted with ❤ by GitHub view raw

few rows. We have four columns *userId, moveId, rating and timestamp,* and checked the value counts of *rating.* From here we can see rating of 4.0 has highest value counts. This means more people rated the movie 4.0 has shown in the plot below.

In [15]:
```
data.rating.value_counts().plot(kind='bar')
plt.show()
```



In [ ]:
```
```

plot.ipynb hosted with ❤ by GitHub view raw

Now I have to check the number of null value in my data.

```
In [16]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
userId       100004 non-null int64
movieId      100004 non-null int64
rating       100004 non-null float64
timestamp    100004 non-null int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB

In [17]: data.isnull().sum()

Out[17]: userId       0
         movieId      0
         rating       0
         timestamp    0
         dtype: int64
```

null.ipynb hosted with ❤ by GitHub view raw

To load a dataset from a pandas dataframe, you will need the **`load_from_df()`** method. You will also need a **`Reader`** object, but only the `rating_scale` the parameter must be specified the default rating_scaale is (2,5). The dataframe must have three columns, corresponding to the user (raw) ids, the item (raw) ids, and the ratings in this order.

In [24]:
```
data = data[['userId', 'movieId', 'rating', 'timestamp']]
data = data.iloc[:, :-1]
```

In [25]:
```
data.head()
```

Out[25]:

	userId	movieId	rating
0	1	31	2.5
1	1	1029	3.0
2	1	1061	3.0
3	1	1129	2.0
4	1	1172	4.0

In [27]:
```
from surprise import Reader, Dataset
reader = Reader()
data = Dataset.load_from_df(data[['userId', 'movieId', 'rating']], reader)
```

loadsurprise.ipynb hosted with ❤️ by GitHub view raw

The next step is splitting our dataset in train and test set in a ratio of 75%:25%

In [31]:
```
from surprise.model_selection import train_test_split

trainset, testset = train_test_split(data, test_size=0.25)
```

In [ ]:
```

```

split.ipynb hosted with ❤ by GitHub view raw

```
In [33]:
from surprise import SVD, accuracy
algo = SVD()
algo.fit(trainset)

Out[33]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x7faef9e1e588>

In [34]:
predictions = algo.test(testset)

In [ ]:
```

predict.ipynb hosted with ❤ by GitHub view raw

Evaluation

Singular vector decomposition (SVD) shown here employs the use of gradient descent to minimize the squared error between predicted rating and actual rating, eventually getting the best model.

```
In [38]: from surprise import accuracy
         accuracy.rmse(predictions)

         RMSE: 0.8976
Out[38]: 0.8976472082192053
```


evaluation.ipynb hosted with ❤ by GitHub view raw

You can perform Cross-validation and heavy hyperparameters tuning with surprise to get more accurate predictions.

I love feedback please let me know what you think, hit the clap button and share this post with friends and colleagues. You can get access to the full code here
Thanks for reading!

Resources :

Recommender system - Wikipedia

The majority of existing approaches to recommender systems focus on recommending the most relevant content to users...

en.wikipedia.org

Data source: https://grouplens.org/datasets/movielens/100k/

Link to surprise documentation: https://surprise.readthedocs.io/en/stable/index.html

Data Science Recommender Systems Machine Learning Scikit