filtering

January 8, 2024

[ ]:
```python
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd

from surprise import Dataset, KNNBasic, accuracy, SVD
from surprise.model_selection import train_test_split, ShuffleSplit,␣
 ↪cross_validate

movies = pd.read_csv('./ml-100k/u.item', names=['movie_id', 'movie_title',␣
 ↪'release_date', 'video_release_date', 'imdb_url'], delimiter='|',␣
 ↪engine='python',encoding = "latin-1", usecols=range(5))

data100k = Dataset.load_builtin('ml-100k')
data1m = Dataset.load_builtin('ml-1m')

data_training, data_testing = train_test_split(data100k, random_state=22020,␣
 ↪train_size=0.80)
data_big_training, data_big_testing = train_test_split(data1m,␣
 ↪random_state=22020, train_size=0.80)
```

[ ]:
```python
# Util functions.

from collections import defaultdict


def getTopNRecommendations(predictions, n=10):
    # code from https://surprise.readthedocs.io/en/stable/FAQ.
 ↪html#how-to-get-the-top-n-recommendations-for-each-user
    """Return the top-N recommendation for each user from a set of predictions.

    Args:
        predictions(list of Prediction objects): The list of predictions, as
            returned by the test method of an algorithm.
        n(int): The number of recommendation to output for each user. Default
            is 10.
```

```
    Returns:
    A dict where keys are user (raw) ids and values are lists of tuples:
        [(raw item id, rating estimation), ...] of size n.
    """

    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

def getTopRecommendationsByUserId(predictions, userId, n=10):
    top_n = getTopNRecommendations(predictions, n)
    userRating = top_n.get(userId)

    it = 1
    for iid, rating in userRating:
        movieTitle = movies.loc[movies['movie_id'] == int(iid)]['movie_title']
        print()
        print(str(it) + ". " + movieTitle)
        print("Rating: " + str(round(rating, 2)))
        print()
        it+=1


def customCrossValidate(algorithm, data):
    sSplit = ShuffleSplit(n_splits=5, train_size=0.8, random_state=22020)
    results = cross_validate(algorithm, data, measures=['MSE'], cv=sSplit,␣
 ↪verbose=True)
    return results
```

1 Movielens 100k

1.1 User Based CF

[ ]:
```
# Predict Rating for UserID 20, Movie Id
userId = 22
movieId = 20

userBasedAlgorithm = KNNBasic(sim_options={'name':'pearson', 'user_based':True})
```

```python
def userBasedFiltering(dataTraining, dataTesting):
    algorithm = userBasedAlgorithm
    predictions = algorithm.fit(dataTraining).test(dataTesting)

    if dataTraining.knows_user(userId) & dataTraining.knows_item(movieId):
        algorithm.predict(str(userId), str(movieId), verbose=True)
    else:
        if dataTraining.knows_user(userId) == False:
            unknownId = "userId"
        else:
            unknownId = "movieId"
        print(unknownId + " ist unbekannt. Andere ID wählen.")

    top_n = getTopNRecommendations(predictions, n=10)

    userRecommendations = getTopRecommendationsByUserId(predictions,␣
 ↪str(userId))

userBasedFiltering(data_training, data_testing)

customCrossValidate(userBasedAlgorithm, data100k)
```

```
Computing the pearson similarity matrix…
Done computing similarity matrix.
Mean Squared Error:
MSE: 1.0258

user: 22         item: 20         r_ui = None   est = 3.56   {'actual_k': 40,
'was_impossible': False}

126    1. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 4.55


650    2. Glory (1989)
Name: movie_title, dtype: object
Rating: 4.16


88    3. Blade Runner (1982)
Name: movie_title, dtype: object
Rating: 4.11


227    4. Star Trek: The Wrath of Khan (1982)
Name: movie_title, dtype: object
Rating: 4.07
```

```
208    5. This Is Spinal Tap (1984)
Name: movie_title, dtype: object
Rating: 4.06


209    6. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 3.96


432    7. Heathers (1989)
Name: movie_title, dtype: object
Rating: 3.95


210    8. M*A*S*H (1970)
Name: movie_title, dtype: object
Rating: 3.87


501    9. Bananas (1971)
Name: movie_title, dtype: object
Rating: 3.85


203    10. Back to the Future (1985)
Name: movie_title, dtype: object
Rating: 3.8

Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating MSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     1.0258  1.0170  1.0415  1.0358  1.0366  1.0313  0.0088
Fit time          0.18    0.19    0.19    0.18    0.19    0.19    0.00
Test time         1.02    0.96    1.22    0.95    1.02    1.04    0.10
```

```
[ ]: {'test_mse': array([1.02583297, 1.01700853, 1.0414697 , 1.03578399, 1.0365519
]),
  'fit_time': (0.18326783180236816,
   0.18796014785766602,
   0.18834376335144043,
   0.18455982208251953,
   0.18796277046203613),
  'test_time': (1.0245740413665771,
   0.9630730152130127,
   1.2159900665283203,
   0.9518771171569824,
   1.0196020603179932)}
```

1.2 Item-Based CF

```
[ ]: itemBasedAlgorithm = KNNBasic(sim_options={'name':"cosine", 'user_based':False})
def itemBasedFiltering(dataTraining, dataTesting):

    algorithm = itemBasedAlgorithm
    predictions = algorithm.fit(dataTraining).test(dataTesting)

    algorithm.predict(str(userId), str(movieId), verbose=True)

    getTopRecommendationsByUserId(predictions, str(userId))

itemBasedFiltering(data_training, data_testing)
customCrossValidate(itemBasedAlgorithm, data100k)
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
Mean Squared Error:
MSE: 1.0626

user: 22         item: 20         r_ui = None   est = 3.80   {'actual_k': 40,
'was_impossible': False}

203    1. Back to the Future (1985)
Name: movie_title, dtype: object
Rating: 4.2


126    2. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 4.2


88    3. Blade Runner (1982)
Name: movie_title, dtype: object
```

```
Rating: 4.15


227    4. Star Trek: The Wrath of Khan (1982)
Name: movie_title, dtype: object
Rating: 4.13


152    5. Fish Called Wanda, A (1988)
Name: movie_title, dtype: object
Rating: 4.12


221    6. Star Trek: First Contact (1996)
Name: movie_title, dtype: object
Rating: 4.05


210    7. M*A*S*H (1970)
Name: movie_title, dtype: object
Rating: 3.97


209    8. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 3.92


432    9. Heathers (1989)
Name: movie_title, dtype: object
Rating: 3.89


398    10. Three Musketeers, The (1993)
Name: movie_title, dtype: object
Rating: 3.82

Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating MSE of algorithm KNNBasic on 5 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     1.0626  1.0605  1.0679  1.0512  1.0623  1.0609  0.0054
Fit time          0.22    0.23    0.22    0.22    0.23    0.22    0.00
Test time         1.22    1.34    1.14    1.15    1.14    1.20    0.08
```

[ ]:
```
{'test_mse': array([1.06264478, 1.06052716, 1.06791961, 1.05123638,
1.06232762]),
 'fit_time': (0.2194077968597412,
  0.22818613052368164,
  0.22234225273132324,
  0.22297286987304688,
  0.22675204277038574),
 'test_time': (1.2150342464447021,
  1.343759298324585,
  1.1449267864227295,
  1.1522581577301025,
  1.1442930698394775)}
```

1.3 SVD Based

[ ]:
```python
svdBasedAlgorithm = SVD()
def svdBasedFiltering(dataTraining, dataTesting):
    algo = svdBasedAlgorithm
    predictions = algo.fit(dataTraining).test(dataTesting)

    algo.predict(str(userId), str(movieId), verbose=True)

    print("The top recommendations are: ")
    getTopRecommendationsByUserId(predictions, str(userId))

svdBasedFiltering(data_training, data_testing)
customCrossValidate(svdBasedAlgorithm, data100k)
```

```
Mean Squared Error:
MSE: 0.8794

user: 22         item: 20         r_ui = None   est = 3.53   {'was_impossible':
False}
The top recommendations are:

208    1. This Is Spinal Tap (1984)
Name: movie_title, dtype: object
Rating: 4.47


227    2. Star Trek: The Wrath of Khan (1982)
Name: movie_title, dtype: object
```

```
Rating: 4.47


650    3. Glory (1989)
Name: movie_title, dtype: object
Rating: 4.45


126    4. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 4.41


221    5. Star Trek: First Contact (1996)
Name: movie_title, dtype: object
Rating: 4.29


88    6. Blade Runner (1982)
Name: movie_title, dtype: object
Rating: 4.21


152    7. Fish Called Wanda, A (1988)
Name: movie_title, dtype: object
Rating: 4.17


501    8. Bananas (1971)
Name: movie_title, dtype: object
Rating: 4.08


203    9. Back to the Future (1985)
Name: movie_title, dtype: object
Rating: 4.01


209    10. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 3.89

Evaluating MSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.8784  0.8765  0.8866  0.8819  0.8812  0.8809  0.0035
Fit time          0.47    0.41    0.42    0.41    0.42    0.42    0.02
Test time         0.22    0.04    0.04    0.04    0.04    0.07    0.07
```

[ ]: {'test_mse': array([0.87835723, 0.87646019, 0.88660841, 0.88189351,
0.88124492]),
 'fit_time': (0.46881794929504395,
  0.406527042388916,
  0.41904616355895996,
  0.4075813293457031,
  0.41622400283813477),
 'test_time': (0.21959877014160156,
  0.036596059799194336,
  0.03629279136657715,
  0.03682565689086914,
  0.036402225494384766)}

2 Movielens 1M

2.1 User-Based CF

```
[ ]: userBasedFiltering(data_big_training, data_big_testing)
customCrossValidate(userBasedAlgorithm, data1m)
```

```
Computing the pearson similarity matrix…
Done computing similarity matrix.
Mean Squared Error:
MSE: 0.9223

user: 22         item: 20         r_ui = None   est = 2.09   {'actual_k': 40,
'was_impossible': False}

907    1. Half Baked (1998)
Name: movie_title, dtype: object
Rating: 4.64


Series([], Name: movie_title, dtype: object)
Rating: 4.3


1199    3. Kim (1950)
Name: movie_title, dtype: object
Rating: 4.3


588    4. Wild Bunch, The (1969)
Name: movie_title, dtype: object
Rating: 4.28


1079    5. Celestial Clockwork (1994)
```

```
Name: movie_title, dtype: object
Rating: 4.17


607    6. Spellbound (1945)
Name: movie_title, dtype: object
Rating: 4.14


Series([], Name: movie_title, dtype: object)
Rating: 4.13


Series([], Name: movie_title, dtype: object)
Rating: 4.07


1195    9. Savage Nights (Nuits fauves, Les) (1992)
Name: movie_title, dtype: object
Rating: 4.06


1393    10. Swept from the Sea (1997)
Name: movie_title, dtype: object
Rating: 4.03

Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Computing the pearson similarity matrix…
Done computing similarity matrix.
Evaluating MSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.9223  0.9259  0.9305  0.9248  0.9209  0.9249  0.0033
Fit time          17.12   17.09   16.99   17.05   17.06   17.06   0.04
Test time         51.32   46.22   50.13   48.98   51.39   49.61   1.91
```

[ ]:

```
{'test_mse': array([0.92231648, 0.9258572 , 0.93047129, 0.92476521,
0.92085076]),
 'fit_time': (17.119293212890625,
  17.094645977020264,
```

```
  16.994232892990112,
  17.051733016967773,
  17.059494256973267),
 'test_time': (51.31756067276001,
  46.2184419631958,
  50.12724304199219,
  48.98101019859314,
  51.391844749450684)}
```

2.2 Item-Based CF

[ ]:
```
itemBasedFiltering(data_big_training, data_big_testing)
customCrossValidate(itemBasedAlgorithm, data1m)
```

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
Mean Squared Error:
MSE: 0.9957

user: 22         item: 20         r_ui = None   est = 2.75   {'actual_k': 40,
'was_impossible': False}

Series([], Name: movie_title, dtype: object)
Rating: 3.8


607    2. Spellbound (1945)
Name: movie_title, dtype: object
Rating: 3.78


670    3. Bride of Frankenstein (1935)
Name: movie_title, dtype: object
Rating: 3.7


1393    4. Swept from the Sea (1997)
Name: movie_title, dtype: object
Rating: 3.7


Series([], Name: movie_title, dtype: object)
Rating: 3.7


Series([], Name: movie_title, dtype: object)
Rating: 3.7
```

```
1199    7. Kim (1950)
Name: movie_title, dtype: object
Rating: 3.7


1195    8. Savage Nights (Nuits fauves, Les) (1992)
Name: movie_title, dtype: object
Rating: 3.65


1672    9. Mirage (1995)
Name: movie_title, dtype: object
Rating: 3.65


Series([], Name: movie_title, dtype: object)
Rating: 3.63

Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Evaluating MSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.9957  1.0018  1.0023  0.9959  0.9921  0.9976  0.0039
Fit time          5.65    5.79    5.76    5.63    5.71    5.71    0.06
Test time         21.02   21.85   22.70   19.03   21.46   21.21   1.22
```

```
[ ]: {'test_mse': array([0.9957227 , 1.00178679, 1.002326  , 0.99586078,
     0.99214696]),
      'fit_time': (5.653619289398193,
       5.78642201423645,
       5.762329816818237,
       5.629648208618164,
       5.70727801322937),
      'test_time': (21.018913745880127,
       21.846208095550537,
       22.69937515258789,
       19.02978777885437,
```

```
21.458035945892334)}
```

2.3 SVD Based Filtering

```
[ ]: svdBasedFiltering(data_big_training, data_big_testing)
     customCrossValidate(svdBasedAlgorithm, data1m)
```

```
Mean Squared Error:
MSE: 0.7618

user: 22         item: 20         r_ui = None   est = 1.81   {'was_impossible':
False}
The top recommendations are:

Series([], Name: movie_title, dtype: object)
Rating: 4.5


Series([], Name: movie_title, dtype: object)
Rating: 4.31


1079    3. Celestial Clockwork (1994)
Name: movie_title, dtype: object
Rating: 4.08


1672    4. Mirage (1995)
Name: movie_title, dtype: object
Rating: 4.07


1393    5. Swept from the Sea (1997)
Name: movie_title, dtype: object
Rating: 4.07


Series([], Name: movie_title, dtype: object)
Rating: 3.92


1195    7. Savage Nights (Nuits fauves, Les) (1992)
Name: movie_title, dtype: object
Rating: 3.89


607    8. Spellbound (1945)
Name: movie_title, dtype: object
```

```
Rating: 3.85


907    9. Half Baked (1998)
Name: movie_title, dtype: object
Rating: 3.83


Series([], Name: movie_title, dtype: object)
Rating: 3.77

Evaluating MSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.7624  0.7622  0.7666  0.7604  0.7581  0.7619  0.0028
Fit time          4.91    4.81    4.76    4.05    4.80    4.66    0.31
Test time         0.75    0.76    1.09    0.74    1.06    0.88    0.16
```

[ ]:
```
{'test_mse': array([0.7623805 , 0.76221565, 0.76657712, 0.76035194,
0.75807519]),
  'fit_time': (4.910680055618286,
   4.811629056930542,
   4.756448984146118,
   4.0475499629974365,
   4.797432899475098),
  'test_time': (0.7457480430603027,
   0.7557346820831299,
   1.0948710441589355,
   0.73514723777771,
   1.0551478862762451)}
```

3 Ergebnisse

Als Algorithmen habe ich: * einen Userbased k-Next Neighbors Algorithmus mit Pearson Correlation, * einen Itembased k-Next Neighbors Algorithmus mit Cosine Correlation, * sowie den SVD-Algorithmus.

In Hinblick auf die durchschnittliche Wirksamkeit (Effectiveness) in Bezug auf den **Mean Squared Error** ergibt sich folgendes (gereiht von bester nach schlechtester) - gemessen am großen Datensatz (1m):

1. **Item Based:** 0.9976s
2. **User Based:** 0.9249s
3. **SVD:** 0.7619s

In Hinblick auf die durchschnittliche Effizienz ergibt sich die folgende Reihung (ebenso am größeren Datensatz gemessen, um Rauschen zu vermeiden):

1. SVD: Fit: 4.66s - Test: 0.88s

2. Item Based: Fit: 5.71s - Test: 21.21s
3. User Based: Fit: 17.06s - Test: 49.61s

Somit ergibt sich, dass SVD im Vergleich zu den anderen beiden Algorithmen ungenau ist und weniger effektiv, allerdingst perfort er sehr gut, auch bei großen Datenmengen.

Der Userbased Algorithmus dauert am längsten, erzielt aber auch bessere Ergebnisse.

Der Item Based Algorithmus zeigt sehr geringe Abweichungen bei den erwarteten Ergebnissen von den echten Ergebnissen, und braucht im Fitting nur etwas länger als SVD, allerdings sehr viel länger beim Testen der Ergebnisse.

Die besten Ergebnisse erzielt wohl eine Mischung aus User based und item based Algorithmus. Hier kann man wahrscheinlich die Efficiency sowie Effectiveness optimieren. SVD wird wohl eine gute Methode sein, um halbwegs gute Vorhersagen zu machen, allerdings kann man sich nicht zu sehr auf die Daten verlassen.

> Ich habe zusätzlich eine Funktion aus den Examples der Surprise-Library eingebaut, der zusätzlich die Recommendations ausgibt. Bei den Algorithmen werden unterschiedliche Recommendations gefunden, was allerdings interesstant ist, ist dass beim kleinen (100k) Datensatz in jedem Algorithmus "The Godfather" gefunden wird. Das könnte allerdings damit zu tun haben, dass der Film sehr populär ist, und diese Popularität im Algorithmus nicht berücksichtigt wird.