filtering

January 13, 2024

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

from joblib import delayed, Parallel
from surprise import Dataset, KNNBasic, SVD
from surprise.model_selection import train_test_split
from surprise.model_selection.validation import fit_and_score, print_summary

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)
```

[ ]:
```
# 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 + " ,Rating: " + str(round(rating,␣
 ↪2)))
        print()
        it+=1

def runAlgo(algorithm, data, measures):
    data_training, data_testing = train_test_split(data, random_state=22020,␣
 ↪train_size=0.80)
    return fit_and_score(algorithm, data_training, data_testing, measures, True)

def customCrossValidate(algorithm, data):
    # manches wurde hier aus der Library-Methode "cross_validate" verwendet.␣
 ↪diese funktion wurde angepasst, da nicht mit Folds gearbeitet werden sollte.
    measures = [m.lower() for m in ['MSE']]

    delayed_list = (
        delayed(runAlgo)(algorithm, data, measures)
        for i in range(5)
    )

    out = Parallel(n_jobs=-1,pre_dispatch='2*n_jobs')(delayed_list)
```

```
    (test_measures_dicts, train_measures_dicts, fit_times, test_times) =␣
↪zip(*out)

    test_measures = defaultdict(dict)
    train_measures = defaultdict(dict)

    for m in measures:
        test_measures[m] = np.asarray([d[m] for d in test_measures_dicts])
        train_measures[m] = np.asarray([d[m] for d in train_measures_dicts])

    print_summary(algorithm, measures, test_measures, train_measures,␣
↪fit_times, test_times, 5)
```

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})

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.
```

```
user: 22         item: 20         r_ui = None   est = 3.39   {'actual_k': 40,
'was_impossible': False}

171    1. Empire Strikes Back, The (1980)
Name: movie_title, dtype: object
Rating: 4.6


126    2. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 4.32


193    3. Sting, The (1973)
Name: movie_title, dtype: object
Rating: 4.16


209    4. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 4.12


194    5. Terminator, The (1984)
Name: movie_title, dtype: object
Rating: 4.11


237    6. Raising Arizona (1987)
Name: movie_title, dtype: object
Rating: 4.05


522    7. Cool Hand Luke (1967)
Name: movie_title, dtype: object
Rating: 3.99


78    8. Fugitive, The (1993)
Name: movie_title, dtype: object
Rating: 3.94


180    9. Return of the Jedi (1983)
Name: movie_title, dtype: object
Rating: 3.94
```

```
432    10. Heathers (1989)
Name: movie_title, dtype: object
Rating: 3.91

Computing the pearson similarity matrix…
Computing the pearson 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.
Done computing 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  1.0258  1.0258  1.0258  1.0258  1.0258  0.0000
MSE (trainset)    0.5706  0.5706  0.5706  0.5706  0.5706  0.5706  0.0000
Fit time          0.24    0.29    0.28    0.24    0.21    0.25    0.03
Test time         1.19    1.17    1.13    1.12    1.13    1.15    0.03
```

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.
user: 22         item: 20         r_ui = None   est = 3.43   {'actual_k': 40,
'was_impossible': False}

237    1. Raising Arizona (1987)
Name: movie_title, dtype: object
Rating: 4.2


180    2. Return of the Jedi (1983)
```

```
Name: movie_title, dtype: object
Rating: 4.15


209    3. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 4.05


171    4. Empire Strikes Back, The (1980)
Name: movie_title, dtype: object
Rating: 4.05


78    5. Fugitive, The (1993)
Name: movie_title, dtype: object
Rating: 4.0


126    6. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 3.95


193    7. Sting, The (1973)
Name: movie_title, dtype: object
Rating: 3.95


230    8. Batman Returns (1992)
Name: movie_title, dtype: object
Rating: 3.94


152    9. Fish Called Wanda, A (1988)
Name: movie_title, dtype: object
Rating: 3.92


432    10. Heathers (1989)
Name: movie_title, dtype: object
Rating: 3.91

Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Done computing similarity matrix.
```

```
Done computing similarity matrix.
Computing the cosine similarity matrix…
Done computing similarity matrix.
Done computing 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  1.0626  1.0626  1.0626  1.0626  1.0626  0.0000
MSE (trainset)    0.8048  0.8048  0.8048  0.8048  0.8048  0.8048  0.0000
Fit time          0.28    0.28    0.28    0.26    0.25    0.27    0.01
Test time         1.29    1.26    1.20    1.21    1.19    1.23    0.04
```

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)
```

```
user: 22         item: 20         r_ui = None   est = 3.23   {'was_impossible':
False}
The top recommendations are:

78    1. Fugitive, The (1993)
Name: movie_title, dtype: object
Rating: 4.36


171    2. Empire Strikes Back, The (1980)
Name: movie_title, dtype: object
Rating: 4.32


126    3. Godfather, The (1972)
Name: movie_title, dtype: object
Rating: 4.29


180    4. Return of the Jedi (1983)
```

```
Name: movie_title, dtype: object
Rating: 4.23


194    5. Terminator, The (1984)
Name: movie_title, dtype: object
Rating: 4.1


152    6. Fish Called Wanda, A (1988)
Name: movie_title, dtype: object
Rating: 4.09


209    7. Indiana Jones and the Last Crusade (1989)
Name: movie_title, dtype: object
Rating: 4.0


429    8. Duck Soup (1933)
Name: movie_title, dtype: object
Rating: 3.89


432    9. Heathers (1989)
Name: movie_title, dtype: object
Rating: 3.83


193    10. Sting, The (1973)
Name: movie_title, dtype: object
Rating: 3.79

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.8787  0.8789  0.8801  0.8788  0.8822  0.8797  0.0013
MSE (trainset)    0.4679  0.4698  0.4676  0.4689  0.4698  0.4688  0.0009
Fit time          0.47    0.46    0.47    0.46    0.46    0.46    0.01
Test time         0.05    0.05    0.05    0.05    0.05    0.05    0.00
```

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.
user: 22         item: 20         r_ui = None   est = 2.07   {'actual_k': 40,
'was_impossible': False}

857    1. Amityville: Dollhouse (1996)
Name: movie_title, dtype: object
Rating: 4.5


918    2. City of Lost Children, The (1995)
Name: movie_title, dtype: object
Rating: 4.44


Series([], Name: movie_title, dtype: object)
Rating: 4.41


1135    4. Ghosts of Mississippi (1996)
Name: movie_title, dtype: object
Rating: 4.32


554    5. White Man's Burden (1995)
Name: movie_title, dtype: object
Rating: 4.19


46    6. Ed Wood (1994)
Name: movie_title, dtype: object
Rating: 4.18


109    7. Operation Dumbo Drop (1995)
Name: movie_title, dtype: object
Rating: 4.15


Series([], Name: movie_title, dtype: object)
Rating: 4.12
```

```
1290    9. Celtic Pride (1996)
Name: movie_title, dtype: object
Rating: 4.06


1357    10. The Deadly Cure (1996)
Name: movie_title, dtype: object
Rating: 4.04

Computing the pearson similarity matrix…
Computing the pearson similarity matrix…
Computing the pearson similarity matrix…
Computing the pearson similarity matrix…
Computing the pearson similarity matrix…
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing 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  0.9223  0.9223  0.9223  0.9223  0.9223  0.0000
MSE (trainset)    0.5154  0.5154  0.5154  0.5154  0.5154  0.5154  0.0000
Fit time          199.26  203.21  202.44  199.90  196.58  200.28  2.37
Test time         49.88   52.00   47.70   47.11   51.29   49.59   1.92
```

2.2 Item-Based CF

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

```
Computing the cosine similarity matrix…
Done computing similarity matrix.
user: 22         item: 20         r_ui = None   est = 2.61   {'actual_k': 40,
'was_impossible': False}

222    1. Sling Blade (1996)
Name: movie_title, dtype: object
Rating: 3.85


Series([], Name: movie_title, dtype: object)
Rating: 3.85
```

```
857    3. Amityville: Dollhouse (1996)
Name: movie_title, dtype: object
Rating: 3.82


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


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


1135    6. Ghosts of Mississippi (1996)
Name: movie_title, dtype: object
Rating: 3.8


Series([], Name: movie_title, dtype: object)
Rating: 3.78


1672    8. Mirage (1995)
Name: movie_title, dtype: object
Rating: 3.78


1357    9. The Deadly Cure (1996)
Name: movie_title, dtype: object
Rating: 3.75


554    10. White Man's Burden (1995)
Name: movie_title, dtype: object
Rating: 3.73

Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Computing the cosine similarity matrix…
Done computing similarity matrix.
Done computing similarity matrix.
Done computing similarity matrix.
Done computing 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  0.9957  0.9957  0.9957  0.9957  0.9957  0.0000
MSE (trainset)   0.8108  0.8108  0.8108  0.8108  0.8108  0.8108  0.0000
Fit time         6.96    7.61    7.57    7.60    7.22    7.39    0.26
Test time        19.78   22.92   19.37   19.15   19.37   20.12   1.42
```

2.3 SVD Based Filtering

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

```
user: 22         item: 20         r_ui = None   est = 1.85   {'was_impossible':
False}
The top recommendations are:

1135    1. Ghosts of Mississippi (1996)
Name: movie_title, dtype: object
Rating: 4.18


46    2. Ed Wood (1994)
Name: movie_title, dtype: object
Rating: 4.1


Series([], Name: movie_title, dtype: object)
Rating: 4.09


1672    4. Mirage (1995)
Name: movie_title, dtype: object
Rating: 4.06


Series([], Name: movie_title, dtype: object)
Rating: 4.05


Series([], Name: movie_title, dtype: object)
Rating: 4.05


222    7. Sling Blade (1996)
Name: movie_title, dtype: object
Rating: 3.99


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

```
Rating: 3.97


1357    9. The Deadly Cure (1996)
Name: movie_title, dtype: object
Rating: 3.96


Series([], Name: movie_title, dtype: object)
Rating: 3.87

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MSE (testset)     0.7617  0.7634  0.7627  0.7621  0.7611  0.7622  0.0008
MSE (trainset)    0.4492  0.4487  0.4507  0.4484  0.4495  0.4493  0.0008
Fit time          4.91    4.30    4.28    4.29    4.56    4.47    0.24
Test time         0.73    0.87    0.76    0.67    0.62    0.73    0.09
```

3 Ergebnisse

> Anzumerken ist: bei den "Folds" in der Ausgabe handelt es sich nicht um Folds, sondern lediglich um die Iterationen. die "Folds"-Ausgabe ergibt sich aus der `print_summary`-Methode, die ich aus der Library verwendet habe.

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:** 0.9957 / 0.8108
2. **User Based:** 0.9223 / 0.5154
3. **SVD:** 0.7622 / 0.4493

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

1. SVD: Fit: 4.47s - Test: 0.73s
2. Item Based: Fit: 7.39s - Test: 20.12s
3. User Based: Fit: 200.28s - Test: 49.59s

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.