这里我对ml-1m重新计算

- 使用在dl中用到的数据集
- 计算rmse / recall
- 计算在不同年龄分割 / 不同性别分割的情况
    - 不分割：全测试集
    - 男性
    - 女性
    - 7种年龄

In [1]:
```
import pandas as pd
import numpy as np
from glob import glob
from time import time

from surprise import Reader
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise import NormalPredictor
from surprise import KNNBasic
from surprise import KNNWithMeans
from surprise import KNNWithZScore
from surprise import KNNBaseline
from surprise import SVD
from surprise import BaselineOnly
from surprise import SVDpp
from surprise import NMF
from surprise import SlopeOne
from surprise import CoClustering
from surprise.accuracy import rmse, mae
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise.model_selection import GridSearchCV

import math
import copy
import pickle
from pathlib import Path
from itertools import zip_longest
from collections import defaultdict
```

1. 加载dl时用到的数据集

In [4]:
```
df = pd.read_csv("ml-1m_dl.csv")
print(df.shape)
df.head()
```

```
(1000209, 9)
```

Out[4]:

	movie_id	movie_title	user_id	age	sex	occupation	rating	sex_index	age_index
0	1	Toy Story (1995)	1	1	F	10	5	0	0
1	48	Pocahontas (1995)	1	1	F	10	5	0	0
2	150	Apollo 13 (1995)	1	1	F	10	5	0	0
3	260	Star Wars: Episode IV - A New Hope (1977)	1	1	F	10	4	0	0
4	527	Schindler's List (1993)	1	1	F	10	5	0	0

In [24]:
```
set(df['age'])
```

Out[24]:
```
{1, 18, 25, 35, 45, 50, 56}
```

In [5]:
```
# load dataset
datasets = pickle.load(open('ml-1m_dl.pkl','rb'))
datasets['val'][1]
```

Out[5]:
```
630120    4.0
229398    5.0
758377    3.0
159240    5.0
254252    4.0
         ...
875199    4.0
743921    4.0
527163    4.0
623363    3.0
120098    3.0
Name: rating, Length: 200042, dtype: float32
```

In [16]:
```
# df_train
df_train = datasets['train'][0].copy()
df_train['rating'] = datasets['train'][1].astype(np.int64)
print(len(df_train))
df_train.head(3)
```

```
800167
```

Out[16]:

	user_id	movie_id	sex_index	age_index	rating
529184	5530	1488	0	1	1
341591	3600	609	1	3	4
470922	4889	1291	1	1	5

In [15]:
```
df_test = datasets['val'][0].copy()
df_test['rating'] = datasets['val'][1].astype(np.int64)
df_test.head(3)
```

Out[15]:

	user_id	movie_id	sex_index	age_index	rating
630120	5837	2353	1	2	4
229398	2242	3114	1	1	5
758377	103	1801	1	4	3

In [20]:
```
def build_test_dataset(reader, df_test):
    data_test = Dataset.load_from_df(df_test[['user_id', 'movie_id', 'rating']], reader)
    data_test = data_test.build_full_trainset().build_testset()
    return data_test
```

In [23]:
```
# 构建support数据集 --> 基于年龄和性别分割测试集
reader = Reader(rating_scale=(1, 5))
data_train = Dataset.load_from_df(df_train[['user_id', 'movie_id', 'rating']], reader)
data_train = data_train.build_full_trainset()
data_test = build_test_dataset(reader, df_test)

df_test_m = df_test[df_test['sex_index'] == 1]
print("Male: {}".format(len(df_test_m)))
data_test_m = build_test_dataset(reader, df_test_m)

df_test_f = df_test[df_test['sex_index'] == 0]
print("Female: {}".format(len(df_test_f)))
data_test_f = build_test_dataset(reader, df_test_f)

split_data_test_list = [data_test_m, data_test_f]

for age_index in range(7):
    df_test_age = df_test[df_test['age_index'] == age_index]
    print("AgeIndex {}: {}".format(age_index, len(df_test_age)))
    data_test_age_curr = build_test_dataset(reader, df_test_age)
    split_data_test_list.append(data_test_age_curr)
```

```
Male: 150891
Female: 49151
AgeIndex 0: 5368
AgeIndex 1: 36797
AgeIndex 2: 79300
AgeIndex 3: 39883
AgeIndex 4: 16537
AgeIndex 5: 14470
AgeIndex 6: 7687
```

In [28]:
```
split_data_test_names = ['Male', 'Female', 'Age1', 'Age18', 'Age25', 'Age35', 'Age45', 'Age50', 'Age56']
```

In [27]:
```
# 不使用SVDpp --> 太慢了
algorithms = {'SVD':SVD(), 'SlopeOne':SlopeOne(), 'NMF':NMF(), 'NormalPredictor':NormalPredictor(),
              'KNNBaseline':KNNBaseline(), 'KNNBasic':KNNBasic(), 'KNNWithMeans':KNNWithMeans(),
              'KNNWithZScore':KNNWithZScore(), 'BaselineOnly':BaselineOnly(), 'CoClustering':CoClustering()}
```

2. Helper function: rmse / recall, etc.

In [37]:
```
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    '''Return precision and recall at k metrics for each user.'''
    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))
    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)
        # Number of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])
        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                              for (est, true_r) in user_ratings[:k])
        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1
        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

        precisions_mean = sum(prec for prec in precisions.values()) / len(precisions)
        recalls_mean = sum(rec for rec in recalls.values()) / len(recalls)
    return precisions_mean, recalls_mean
```

In [51]:
```
def train_single_algorithm(algorithm_name, data_train, data_test, split_data_test_list, save_model=False):
    algorithms = {'SVD':SVD(), 'SVDpp':SVDpp(), 'SlopeOne':SlopeOne(), 'NMF':NMF(), 'NormalPredictor':NormalP
redictor(),
              'KNNBaseline':KNNBaseline(), 'KNNBasic':KNNBasic(), 'KNNWithMeans':KNNWithMeans(),
              'KNNWithZScore':KNNWithZScore(), 'BaselineOnly':BaselineOnly(), 'CoClustering':CoClustering()}
    split_data_test_names = ['Male', 'Female', 'Age1', 'Age18', 'Age25', 'Age35', 'Age45', 'Age50', 'Age56']
    assert(algorithm_name in algorithms), "{} does not exist!".format(algorithm_name)
    assert(len(split_data_test_list) == 9), "9 split test sets!"
    algo = algorithms[algorithm_name]
    start_time = time()
    print("Start training: {}".format(algorithm_name))
    algo.fit(data_train)

    # test
    print("Start testing on full test set")
    predictions = algo.test(data_test)
    result = {}
    result['rmse_full'] = accuracy.rmse(predictions, verbose=True)
    result['recall_full'] = precision_recall_at_k(predictions, k=10, threshold=3.5)[1]
    # result['mae'] = accuracy.mae(predictions, verbose=True)


    # test on split datasets
    print("Start testing on split test sets")
    for i in range(9):
        split_data_test_name = split_data_test_names[i]
        split_data_test = split_data_test_list[i]
        print(split_data_test_name)
        predictions = algo.test(split_data_test)
        result['rmse_{}'.format(split_data_test_name)] = accuracy.rmse(predictions, verbose=True)
        result['recall_{}'.format(split_data_test_name)] = precision_recall_at_k(predictions, k=10, threshold
=3.5)[1]

    if save_model:
        print("Save model")
        result['model'] = algo
    else:
        print("Do not save model")

    # print_result = "{:<20}|{:.2f} mins|rmse: {:.4f}|rmse_m: {:.4f}|rmse_f: {:.4f}|mae: {:.4f}|mae_m: {:.4f}
|mae_f: {:.4f}"
    # print_result = print_result.format(algorithm_name, (time() - start_time) / 60.,
    #                                    result['rmse'], result['rmse_m'], result['rmse_f'],
    #                                    result['mae'],result['mae_m'],result['mae_f'])
    # print(print_result)
    print("Algorithm {} finished with {:.2f} mins".format(algorithm_name, (time() - start_time) / 60.))
    return result
```

In [56]:
```
def show_single_result(algo_name, result):
    print("Algo: {}".format(algo_name))
    for key in sorted(result.keys(), reverse=True):
        print("{:<13}: {:.4f}".format(key, result[key]))
```

Now let's start all results

In [50]:
```
all_results = {}
for algorithm_name in algorithms.keys():
    result = train_single_algorithm(algorithm_name, data_train, data_test, split_data_test_list, save_model=F
alse)
    all_results[algorithm_name] = result
    print("===== ===== ===== =====")
```

```
Start training: SVD
Start testing on full test set
RMSE: 0.8735
Start testing on split test sets
Male
RMSE: 0.8624
Female
RMSE: 0.9068
Age1
RMSE: 0.9561
Age18
RMSE: 0.9140
Age25
RMSE: 0.8690
Age35
RMSE: 0.8556
Age45
RMSE: 0.8483
Age50
RMSE: 0.8551
Age56
RMSE: 0.8413
Do not save model
Algorithm SVD finished with 0.91 mins
===== ===== ===== =====
Start training: SlopeOne
Start testing on full test set
RMSE: 0.9055
Start testing on split test sets
Male
RMSE: 0.8948
Female
RMSE: 0.9375
Age1
RMSE: 1.0172
Age18
RMSE: 0.9489
Age25
RMSE: 0.9007
Age35
RMSE: 0.8848
Age45
RMSE: 0.8780
Age50
RMSE: 0.8805
Age56
RMSE: 0.8690
Do not save model
Algorithm SlopeOne finished with 2.42 mins
===== ===== ===== =====
Start training: NMF
Start testing on full test set
RMSE: 0.9151
Start testing on split test sets
Male
RMSE: 0.9056
Female
RMSE: 0.9436
Age1
RMSE: 1.0256
Age18
RMSE: 0.9591
Age25
RMSE: 0.9113
Age35
RMSE: 0.8932
Age45
RMSE: 0.8864
Age50
RMSE: 0.8887
Age56
RMSE: 0.8773
Do not save model
Algorithm NMF finished with 0.99 mins
===== ===== ===== =====
Start training: NormalPredictor
Start testing on full test set
RMSE: 1.5034
Start testing on split test sets
Male
RMSE: 1.5102
Female
RMSE: 1.5036
Age1
RMSE: 1.5558
Age18
RMSE: 1.5427
Age25
```

```
RMSE: 1.5128
Age35
RMSE: 1.4793
Age45
RMSE: 1.4725
Age50
RMSE: 1.4749
Age56
RMSE: 1.4730
Do not save model
Algorithm NormalPredictor finished with 0.15 mins
===== ===== ===== =====
Start training: KNNBaseline
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Start testing on full test set
RMSE: 0.8936
Start testing on split test sets
Male
RMSE: 0.8843
Female
RMSE: 0.9214
Age1
RMSE: 0.9977
Age18
RMSE: 0.9382
Age25
RMSE: 0.8897
Age35
RMSE: 0.8729
Age45
RMSE: 0.8641
Age50
RMSE: 0.8669
Age56
RMSE: 0.8543
Do not save model
Algorithm KNNBaseline finished with 5.78 mins
===== ===== ===== =====
Start training: KNNBasic
Computing the msd similarity matrix...
Done computing similarity matrix.
Start testing on full test set
RMSE: 0.9219
Start testing on split test sets
Male
RMSE: 0.9135
Female
RMSE: 0.9471
Age1
RMSE: 1.0161
Age18
RMSE: 0.9696
Age25
RMSE: 0.9179
Age35
RMSE: 0.8975
Age45
RMSE: 0.8929
Age50
RMSE: 0.8976
Age56
RMSE: 0.8907
Do not save model
Algorithm KNNBasic finished with 5.34 mins
===== ===== ===== =====
Start training: KNNWithMeans
Computing the msd similarity matrix...
Done computing similarity matrix.
Start testing on full test set
RMSE: 0.9277
Start testing on split test sets
Male
RMSE: 0.9209
Female
RMSE: 0.9484
Age1
RMSE: 1.0476
Age18
RMSE: 0.9755
Age25
RMSE: 0.9281
Age35
RMSE: 0.9021
Age45
RMSE: 0.8892
Age50
RMSE: 0.8921
```

```
Age56
RMSE: 0.8773
Do not save model
Algorithm KNNWithMeans finished with 5.74 mins
===== ===== ===== =====
Start training: KNNWithZScore
Computing the msd similarity matrix...
Done computing similarity matrix.
Start testing on full test set
RMSE: 0.9292
Start testing on split test sets
Male
RMSE: 0.9223
Female
RMSE: 0.9500
Age1
RMSE: 1.0560
Age18
RMSE: 0.9803
Age25
RMSE: 0.9297
Age35
RMSE: 0.9021
Age45
RMSE: 0.8890
Age50
RMSE: 0.8896
Age56
RMSE: 0.8726
Do not save model
Algorithm KNNWithZScore finished with 5.83 mins
===== ===== ===== =====
Start training: BaselineOnly
Estimating biases using als...
Start testing on full test set
RMSE: 0.9075
Start testing on split test sets
Male
RMSE: 0.8984
Female
RMSE: 0.9348
Age1
RMSE: 1.0172
Age18
RMSE: 0.9521
Age25
RMSE: 0.9049
Age35
RMSE: 0.8850
Age45
RMSE: 0.8766
Age50
RMSE: 0.8801
Age56
RMSE: 0.8644
Do not save model
Algorithm BaselineOnly finished with 0.22 mins
===== ===== ===== =====
Start training: CoClustering
Start testing on full test set
RMSE: 0.9135
Start testing on split test sets
Male
RMSE: 0.9037
Female
RMSE: 0.9428
Age1
RMSE: 1.0217
Age18
RMSE: 0.9581
Age25
RMSE: 0.9068
Age35
RMSE: 0.8956
Age45
RMSE: 0.8855
Age50
RMSE: 0.8893
Age56
RMSE: 0.8788
Do not save model
Algorithm CoClustering finished with 0.45 mins
===== ===== ===== =====
```

In [57]:
```
# show results
for algo_name in all_results.keys():
    show_single_result(algo_name, all_results[algo_name])
    print("=====")
```

```
Algo: SVD
rmse_full    : 0.8735
rmse_Male    : 0.8624
rmse_Female  : 0.9068
rmse_Age56   : 0.8413
rmse_Age50   : 0.8551
rmse_Age45   : 0.8483
rmse_Age35   : 0.8556
rmse_Age25   : 0.8690
rmse_Age18   : 0.9140
rmse_Age1    : 0.9561
recall_full  : 0.5534
recall_Male  : 0.5438
recall_Female: 0.5775
recall_Age56 : 0.6545
recall_Age50 : 0.5756
recall_Age45 : 0.5998
recall_Age35 : 0.5486
recall_Age25 : 0.5283
recall_Age18 : 0.5363
recall_Age1  : 0.5622
=====
Algo: SlopeOne
rmse_full    : 0.9055
rmse_Male    : 0.8948
rmse_Female  : 0.9375
rmse_Age56   : 0.8690
rmse_Age50   : 0.8805
rmse_Age45   : 0.8780
rmse_Age35   : 0.8848
rmse_Age25   : 0.9007
rmse_Age18   : 0.9489
rmse_Age1    : 1.0172
recall_full  : 0.5399
recall_Male  : 0.5313
recall_Female: 0.5617
recall_Age56 : 0.6339
recall_Age50 : 0.5666
recall_Age45 : 0.5931
recall_Age35 : 0.5387
recall_Age25 : 0.5121
recall_Age18 : 0.5224
recall_Age1  : 0.5441
=====
Algo: NMF
rmse_full    : 0.9151
rmse_Male    : 0.9056
rmse_Female  : 0.9436
rmse_Age56   : 0.8773
rmse_Age50   : 0.8887
rmse_Age45   : 0.8864
rmse_Age35   : 0.8932
rmse_Age25   : 0.9113
rmse_Age18   : 0.9591
rmse_Age1    : 1.0256
recall_full  : 0.5281
recall_Male  : 0.5178
recall_Female: 0.5540
recall_Age56 : 0.6201
recall_Age50 : 0.5541
recall_Age45 : 0.5685
recall_Age35 : 0.5350
recall_Age25 : 0.5024
recall_Age18 : 0.5071
recall_Age1  : 0.5212
=====
Algo: NormalPredictor
rmse_full    : 1.5034
rmse_Male    : 1.5102
rmse_Female  : 1.5036
rmse_Age56   : 1.4730
rmse_Age50   : 1.4749
rmse_Age45   : 1.4725
rmse_Age35   : 1.4793
rmse_Age25   : 1.5128
rmse_Age18   : 1.5427
rmse_Age1    : 1.5558
recall_full  : 0.3963
recall_Male  : 0.3835
recall_Female: 0.4036
recall_Age56 : 0.4609
recall_Age50 : 0.4161
recall_Age45 : 0.4038
recall_Age35 : 0.3891
recall_Age25 : 0.3656
recall_Age18 : 0.3828
recall_Age1  : 0.4323
=====
```

```
Algo: KNNBaseline
rmse_full    : 0.8936
rmse_Male    : 0.8843
rmse_Female  : 0.9214
rmse_Age56   : 0.8543
rmse_Age50   : 0.8669
rmse_Age45   : 0.8641
rmse_Age35   : 0.8729
rmse_Age25   : 0.8897
rmse_Age18   : 0.9382
rmse_Age1    : 0.9977
recall_full  : 0.5537
recall_Male  : 0.5419
recall_Female: 0.5838
recall_Age56 : 0.6556
recall_Age50 : 0.5887
recall_Age45 : 0.6028
recall_Age35 : 0.5526
recall_Age25 : 0.5264
recall_Age18 : 0.5338
recall_Age1  : 0.5425
=====
Algo: KNNBasic
rmse_full    : 0.9219
rmse_Male    : 0.9135
rmse_Female  : 0.9471
rmse_Age56   : 0.8907
rmse_Age50   : 0.8976
rmse_Age45   : 0.8929
rmse_Age35   : 0.8975
rmse_Age25   : 0.9179
rmse_Age18   : 0.9696
rmse_Age1    : 1.0161
recall_full  : 0.5774
recall_Male  : 0.5664
recall_Female: 0.6052
recall_Age56 : 0.6907
recall_Age50 : 0.6056
recall_Age45 : 0.6138
recall_Age35 : 0.5785
recall_Age25 : 0.5537
recall_Age18 : 0.5532
recall_Age1  : 0.5679
=====
Algo: KNNWithMeans
rmse_full    : 0.9277
rmse_Male    : 0.9209
rmse_Female  : 0.9484
rmse_Age56   : 0.8773
rmse_Age50   : 0.8921
rmse_Age45   : 0.8892
rmse_Age35   : 0.9021
rmse_Age25   : 0.9281
rmse_Age18   : 0.9755
rmse_Age1    : 1.0476
recall_full  : 0.5171
recall_Male  : 0.5019
recall_Female: 0.5556
recall_Age56 : 0.6283
recall_Age50 : 0.5565
recall_Age45 : 0.5797
recall_Age35 : 0.5267
recall_Age25 : 0.4880
recall_Age18 : 0.4841
recall_Age1  : 0.4691
=====
Algo: KNNWithZScore
rmse_full    : 0.9292
rmse_Male    : 0.9223
rmse_Female  : 0.9500
rmse_Age56   : 0.8726
rmse_Age50   : 0.8896
rmse_Age45   : 0.8890
rmse_Age35   : 0.9021
rmse_Age25   : 0.9297
rmse_Age18   : 0.9803
rmse_Age1    : 1.0560
recall_full  : 0.5215
recall_Male  : 0.5069
recall_Female: 0.5585
recall_Age56 : 0.6310
recall_Age50 : 0.5580
recall_Age45 : 0.5811
recall_Age35 : 0.5293
recall_Age25 : 0.4940
recall_Age18 : 0.4904
recall_Age1  : 0.4759
=====
Algo: BaselineOnly
```

```
rmse_full    : 0.9075
rmse_Male    : 0.8984
rmse_Female  : 0.9348
rmse_Age56   : 0.8644
rmse_Age50   : 0.8801
rmse_Age45   : 0.8766
rmse_Age35   : 0.8850
rmse_Age25   : 0.9049
rmse_Age18   : 0.9521
rmse_Age1    : 1.0172
recall_full  : 0.5525
recall_Male  : 0.5423
recall_Female: 0.5785
recall_Age56 : 0.6587
recall_Age50 : 0.5883
recall_Age45 : 0.6008
recall_Age35 : 0.5552
recall_Age25 : 0.5251
recall_Age18 : 0.5261
recall_Age1  : 0.5474
=====
Algo: CoClustering
rmse_full    : 0.9135
rmse_Male    : 0.9037
rmse_Female  : 0.9428
rmse_Age56   : 0.8788
rmse_Age50   : 0.8893
rmse_Age45   : 0.8855
rmse_Age35   : 0.8956
rmse_Age25   : 0.9068
rmse_Age18   : 0.9581
rmse_Age1    : 1.0217
recall_full  : 0.5470
recall_Male  : 0.5345
recall_Female: 0.5788
recall_Age56 : 0.6638
recall_Age50 : 0.5888
recall_Age45 : 0.5958
recall_Age35 : 0.5503
recall_Age25 : 0.5197
recall_Age18 : 0.5171
recall_Age1  : 0.5203
=====
```