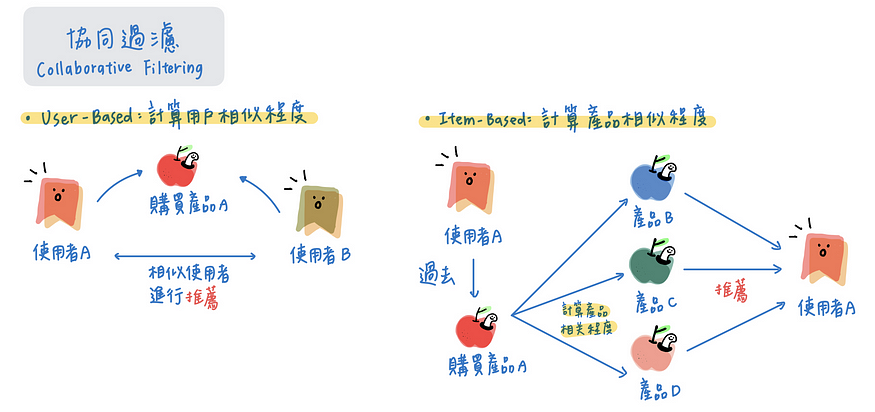
### 1_eBUyt7ZgnsBXohT-ab5uNw



### User Base recommender system

### Option 1: calculate SVD by manual

user\_ratings\_mean = np.mean(final\_df\_matrix.values, axis=1)#calculate the mean rating of each user

ratings\_demeaned = final\_df\_matrix.values - user\_ratings\_mean.reshape(-1, 1)

ratings\_demeaned

在进行奇异值分解（Singular Value Decomposition, SVD）时，使用的是"去均值化"的评分数据（ratings\_demeaned），而不是原始的评分数据（final\_df\_matrix）。这是因为我们要在用户的评分数据上捕捉出差异，而不是绝对量。

简单来说，SVD 是一种能够将复杂的矩阵分解为一系列简单矩阵的技术，它是许多推荐系统算法背后的关键一步。当我们对用户的评分数据进行 SVD 分解时，我们的目标是发现那些能够解释用户评分变动的底层因素。例如，这些因素可能包括电影的类型、导演、演员等。然后，我们可以用这些底层因素来预测用户对他们尚未评分的电影的评分。

问题的关键在于，我们并不关心用户的绝对评分，而关心的是他们的相对评分。也就是说，如果有一个用户对所有的电影都给出了很高的评分，那么他对一个电影的5分评价和另一个电影的4分评价，其实是在说后者相对于前者稍逊一筹。如果我们没有去除用户的平均评分，那么这种差异性就可能被忽略。

因此，我们在进行 SVD 分解之前，会先从每个用户的评分中减去他们的平均评分，也就是进行"去均值化"。这样，我们就可以更准确地捕捉到用户的相对偏好，从而为他们提供更好的推荐。

from scipy.sparse.linalg import svds

U, sigma, Vt = svds(ratings\_demeaned, k=50)  # Number of singular values and vectors to compute設成一個小於5953和 3312 的數字

sigma = np.diag(sigma)

all\_user\_predicted\_ratings = np.dot(np.dot(U, sigma), Vt) + user\_ratings\_mean.reshape(-1, 1)

preds = pd.DataFrame(all\_user\_predicted\_ratings, columns = final\_df\_matrix.columns)

这是因为在奇异值分解（SVD）的过程中，我们将用户的评分数据进行了"去均值化"处理，即从每个用户的评分中减去了他们的平均评分。这一步操作使得数据中心化在零点，使得我们可以在相对差异上进行建模，而不是在绝对评分上。

然而，当我们要做出预测时，我们需要将这个过程反转，将均值添加回去。这是因为，我们的目标是预测用户可能给出的实际评分，这些评分是在具体的评分尺度（例如，1分到5分）上的，而不是相对于他们的平均评分的相对值。

因此，`user\_ratings\_mean.reshape(-1, 1)`将每个用户的平均评分变为列向量，并将其添加到预测的矩阵中，将预测值重新转换到原来的评分尺度上。这样，我们就可以得到一个预测矩阵，其中的每一个值都表示了一个预测的具体评分，可以直接用于推荐。

def recommend\_movies(predictions, userID, movies, reviews, num\_recommendations):

    # Get and sort the user's predictions

    user\_row\_number = userID - 1 # User ID starts at 1, not 0

    sorted\_user\_predictions = preds.iloc[user\_row\_number].sort\_values(ascending=False)

    # Get the specified user's data and merge in the movie information.

    user\_data = reviews[reviews.userId == str(userID)]

    user\_full = (user\_data.merge(movies, how = 'left', on = 'movieId').sort\_values(['rating'], ascending=False))#specified user fulldata

    print('User {0} has already rated {1} movies.'.format(userID, user\_full.shape[0]))

    print('Recommending highest {0} predicted ratings movies not already rated.'.format(num\_recommendations))

    # Recommend the highest predicted rating movies that the user hasn't seen yet.

hasnot\_seen\_yet = movies[~movies['movieId'].isin(user\_full['movieId'])]#user\_full是specified user 看过的所有movie，把它排除

sorted\_user\_predictions = pd.DataFrame(sorted\_user\_predictions).reset\_index()#把movieID拉出来，不要作为index，reset

recommendations = hasnot\_seen\_yet.merge(sorted\_user\_predictions, how = 'left', on = 'movieId')#没有看过的movies中，predict specified user 会给多少分该movie

    recommendations = recommendations.rename(columns = {user\_row\_number: 'Predictions'})

    recommendations = recommendations.sort\_values('Predictions', ascending = False).iloc[:num\_recommendations, :]

    return recommendations

### Option 2 (faster) : use Surprise library

from surprise import Reader, Dataset, SVD, SVDpp

from surprise import accuracy

reader = Reader(rating\_scale=(1, 5))

dataset = Dataset.load\_from\_df(final\_df[['userId', 'movieId', 'rating']], reader=reader)

svd = SVD(n\_factors=50)

svd\_plusplus = SVDpp(n\_factors=50)

**Reader**: This is a class in the **surprise** library that is used to parse a file containing ratings.

**rating\_scale=(1, 5)** is specifying the range of the ratings in your dataset. This helps the algorithm interpret your ratings correctly. For example, a rating of 5 on a scale of 1 to 5 means something very different than a rating of 5 on a scale of 1 to 10.

**Dataset.load\_from\_df**: This function loads a pandas DataFrame into a **surprise** dataset. It's used here to convert the **final\_df** DataFrame, which contains user IDs, movie IDs, and ratings, into a form that the **surprise** library can work with.

trainset = dataset.build\_full\_trainset()

print(dataset.build\_full\_trainset())

svd.fit(trainset)  # old version use svd.train

When using GridSearchCV, you don't need to manually create a training set because GridSearchCV internally handles splitting of the data into training and validation sets for performing cross-validation.

However, when directly fitting a model (like SVD or SVD++), you need to provide a trainset object. The **build\_full\_trainset()** method provides an easy way to create a trainset object that includes all of your data.

trainset.to\_inner\_uid('1')

0

you can convert between internal IDs and raw IDs using the **to\_inner\_uid**, **to\_inner\_iid**, **to\_raw\_uid**, and **to\_raw\_iid** methods.the **to\_inner\_uid()** function does not change any data in the dataset.

id\_2\_names = dict()

for idx, names in zip(movies['movieId'], movies['movie\_names']):

    id\_2\_names[idx] = names

Build a dict for movie,idx is key,movie name is value

def Build\_Anti\_Testset4User(user\_id):

    fill = trainset.global\_mean

    anti\_testset = list()

    u = trainset.to\_inner\_uid(user\_id)

    # ur == users ratings

    user\_items = set([item\_inner\_id for (item\_inner\_id, rating) in trainset.ur[u]])#把specified user对每一样item的rating以及index 取出来，只要idx 不要items，用set过滤掉所有重复的，出来的就会是所有items的idx 只针对specified user！！！！

    anti\_testset += [(trainset.to\_raw\_uid(u), trainset.to\_raw\_iid(i), fill) for

                            i in trainset.all\_items() if i not in user\_items]

#for i in trainset.all\_items()是创立一个iterator 来iterate trainset.all\_items()是所有items的index without repetition,如果item不在specified user的 item中，也就是specified user没使用过的item，取出idx，用tuple装住原来的specified user id和原来的item id，以及global mean，即('1', '1187', 3.52986)，然后再append进去anti\_testset这个list里面。

现在这个list中就有specified user的user id，未看过的item id 以及global mean rating

    return anti\_testset

用customized anti testset generation跟直接用testset=**build\_anti\_testset()，**customized anti testset generation主要是生成specified user没有看过的包含predicted raring的test data，而testset=**build\_anti\_testset()则只是生成trainset里面没有的data。**

### First, let's try SVD for make Top-N recommendation

def TopNRecs\_SVD(user\_id, num\_recommender=10):

    testSet = Build\_Anti\_Testset4User(user\_id)#use this function to find out those items the specified user not interact with

predict = svd.test(testSet)  # we can change to SVD++ later

#The **svd.test()** method is used to predict ratings for the items in the anti-testset based on the trained SVD model. It returns a list of **Prediction** objects.

Prediction(uid='1', iid='292', r\_ui=3.52986, est=3.707634532995649, details={'was\_impossible': False}),

    recommendation = list()

    for userID, movieID, actualRating, estimatedRating, \_ in predict:

        intMovieID = int(movieID)

        recommendation.append((intMovieID, estimatedRating))

#用for loop把这些specified user未看过的movie 以及预测的rating append进去推荐列表

    recommendation.sort(key=lambda x: x[1], reverse=True)#以rating来排序

    movie\_names = []

    movie\_ratings = []

    for name, ratings in recommendation[:20]: #for loop取出头20个，一个一个在id\_2\_names的dictionary中选取出value，也就是movie name，然后append进去，rating也是

      movie\_names.append(id\_2\_names[str(name)])

      movie\_ratings.append(ratings)

    movie\_dataframe =  pd.DataFrame({'movie\_names': movie\_names, 'rating': movie\_ratings}).merge(movies[['movie\_names']],on='movie\_names', how='left')

#创一个df，data是用movie name和rating。 然后把movie的这个df 按照movie\_name left join到第一个df中。 #意义是把看过的并且打了分的，与没看过的但是预测他打分的放在一起

    return movie\_dataframe.head(num\_recommender)

### Second option:

* Sort by release year
* System will recommend the latest movies
* But the best predicted rating will go down from the top

movie\_dataframe.sort(key = lambda x:x)

## Model Evaluation

# Than predict ratings for all pairs (u, i) that are NOT in the training set.

testset = trainset.build\_anti\_testset()

predictions\_svd = svd.test(testset)

Randomly select the data from original dataset but have not been used in train dataset

Use test set data to test the accuracy

## Optional

you can use function below to give a recommendation to all users

from collections import defaultdict

def GetTopN(predictions, n=10, minimumRating=4.0):

        topN = defaultdict(list)

        for userID, movieID, actualRating, estimatedRating, \_ in predictions:

            if (estimatedRating >= minimumRating):

                topN[int(userID)].append((int(movieID), estimatedRating))

        for userID, ratings in topN.items():

            ratings.sort(key=lambda x: x[1], reverse=True)

            topN[int(userID)] = ratings[:n]

        return topN