Contents

1. Recommendation systems in general

1.1. Introduction

We have probably encountered the following phenomena many times. Youtube automatically plays the clips related to the clip we are watching, or automatically suggest clips that we might like. When we buy an item on Amazon, the system will automatically suggest frequently used products bought together, or it knows what you might like based on your purchase history. Google displays ads for products related to the keyword you just searched on Google. Facebook suggests making friends. Netflix automatically suggests movies for users. And there are many other examples where the system has the ability to automatically suggest to users the products they may like. By advertising to that right audience, the effectiveness of marketing will also increase.

The algorithms behind these applications are machine learning algorithms collectively known as recommendation systems.

In many different industries, including e-commerce, entertainment, social media, and education, recommendation systems are becoming more and more crucial. The main objective of a recommendation system is to offer users personalized suggestions that are pertinent to their needs and preferences. These tools assist users in sifting through a sea of data and coming to wise conclusions.

Customer satisfaction, engagement, and revenue have all increased as a result of recommendation systems, which have revolutionized how users interact with online platforms. For instance, e-commerce businesses use recommendation systems to make product recommendations to customers based on their past purchases and browsing habits, increasing sales and patronage. Similar to this, entertainment platforms make movies, TV shows, or music suggestions based on user preferences and viewing habits, increasing user engagement and retention. Recommendation systems are used in education to offer students personalized learning paths and resources based on their performance and learning objectives.

The development of a machine learning-based movie recommendation system is the main goal of this thesis. The goal is to create a system that uses users' prior movie ratings and preferences to deliver accurate and pertinent movie recommendations to them. To generate recommendations, the system will employ collaborative filtering and content-based filtering strategies. It will also assess how well various machine learning algorithms perform. The findings of this study will shed light on the efficacy of various recommendation methods and aid in the creation of personalized and useful recommendation systems.

1.2. History

Recommendation systems date back to the early 1990s, when commercial websites first started to appear. The Tapestry system created by the University of Minnesota's GroupLens research team was one of the first and most significant systems. According to the users' preferences and ratings, this system recommended Usenet news articles to them. The "Customers who bought this also bought" feature, which was based on collaborative filtering and was introduced by Amazon in the middle of the 1990s, was one of the first successful applications of a recommendation system in e-commerce. Amazon was able to boost sales and customer engagement thanks to this feature.

Recommendation systems are a wide field of machine learning and are less old than classification or regression since the internet has only really exploded in the last 15-20 years. There are two main entities in a recommendation system: user and item. User is the user; item is the product, such as movies, songs, books, clips, or other users in the friend recommendation problem. The main purpose of recommender systems is to predict a user's interest in a certain product, thereby having an appropriate recommendation strategy.

1.3. The long tail phenomenon in commerce

Let's compare the fundamental differences between physical stores and online retailers in terms of the products they choose to promote. Here, we momentarily overlook the tangible touch component of the goods found in physical stores. Let's concentrate on how to market the appropriate products to the appropriate consumers.

The Pareto Principle (also known as the 20/80 rule) states that most effects are caused by a small number of causes. The majority of words found in dictionaries make up a very small portion of all words. A small group of people control the majority of the wealth. Even in the world of business, the most popular goods make up a small portion of all available goods.

The physical stores typically have two sections: a display area and a warehouse. The obvious maxim for maximizing sales is to keep the least popular items in stock and to prominently display the most popular ones. This strategy has a clear drawback in that the products shown are well-liked but may not be ideal for a particular client. Even if a store has the item a customer is looking for, it might not be able to sell it to them because they can't see it displayed; as a result, customers may not access the goods even after they are put on display. Additionally, due to space constraints, the store is unable to display all of the products, and each category only offers a small selection. Here, a small number of the most well-liked products (20%) account for the majority of revenue (80%). It's possible that a small percentage of the products in your store account for the majority of sales, while the vast majority of the products in the back make up only a small portion of the total. The long tail of less well-liked products makes up this phenomenon, which is also referred to as the long tail phenomenon.

The aforementioned drawback can be completely avoided with electronic stores. Every product can be displayed because electronic stores have showrooms that seem to go on forever.

Additionally, online ordering is versatile and practical with virtually no conversion costs, making it easier to deliver the right products to customers. As a result, revenue can rise.

1.4. Two main techniques for recommendation system

Recommendations systems are often divided into two main techniques:

1. Content-based system: recommendations based on the properties of the product. For instance, if a user frequently watches films about criminal police, suggest to them a film from database that has criminal elements, like Se7en (1995). In order to use this strategy, products must be grouped, or their features must be identified. To identify the group or feature of each product, however, is not always possible because some products lack a clear grouping.
2. Collaborative filtering: the system recommends products based on the similarity between users and/or products. It is conceivable that a user in this group will receive a product recommendation based on other users who exhibit similar behaviors. As an illustration, three users A, B and C all enjoy Queen's music. Additionally, the system is aware that users B and C also enjoy Justin Bieber's music, but there is no information whether user A like Justin Bieber's music or not. Based on the information of similar users B and C, the system can predict that user A

1.5. Utility matrix

A key idea in recommendation systems, particularly in collaborative filtering, is the utility matrix. It is a matrix that records how a recommendation system's users interact with the items or simply rate them. The cells of the matrix contain the ratings or interactions between the users and the items, while the rows and columns of the matrix represent the users and the items, respectively.

As mentioned before, user and item are the two main components of a recommendation system. Each user will have a degree of preference for each item. For each user-item pair, this interest is given a value if it is known in advance. Information about a user's interest in an item can be collected through a rating system (review and rating), or it may be based on user's clicks on the item's information on the website, or it can be based the time and frequency of the user's views of the item's information, or any combination of these methods can be used to gather information about a user's interest in a particular item. In this thesis, this information is all based on the rating system.

1.6. An example of utility matrix

	A	B	C	D	E	F
Bohemian Rhapsody	5	5	0	0	1	?
I Love Rock 'N Roll	5	?	?	0	?	?
Hotel California	?	4	1	?	?	1
Twinkle, Twinkle Little Star	1	1	4	4	4	?
Happy and You Know It	1	0	5	?	?	?

Figure 3.6: An example of utility matrix with song recommendation system. Songs (items) are rated by users on a scale from 0 to 5 stars. The "?" marks indicate that the data does not yet exist in the database. The recommendation system needs to manually fill in these values.

With a rating system, a user's interest in an item is measured by the value the user has rated for that item, such as the number of stars out of a total of five stars. The set of all ratings, including the unknown values that need to be predicted, forms a matrix called the utility matrix. Consider the example shown above. In this example, there are six users A, B, C, D, E, F and five songs. The boxes with numbers represent that a user has rated a song with a rating from 0 (dislike) to 5 (like it very much). The boxes marked with a '?' correspond to cells with no data. The job of a recommendation system is to predict the value in these gray boxes, thereby giving suggestions to the user. Therefore, the recommendation system problem is sometimes referred to as a matrix completion problem.

In this simple example, it is easy to see that there are two different genres of music: the first three are rock music and the second two are children's music. From this data, we can also guess that A, B like the Bolero genre, while C, D, E, F like Children's music. From there, a good system should suggest I Love Rock 'N Roll to B; Hotel California for A; Happy and You Know It for D, E, F. Assuming there are only these two types of music, when there is a new song, we just need to classify it into any genre, thereby giving suggestions to each user.

Usually, there are many users and items in the system, and each user usually only rates a very small number of items, even if there are users who do not rate any items. Therefore, the number of “?” cells of the utility matrix is usually very large, and the number of filled cells is a very small number.

The more cells are filled in, the better the accuracy of the system will be. Therefore, systems always encourage users to express their interest in items by rating those items. The evaluation of items, therefore, not only helps other users know the quality of that item, but also helps the system know the user's preferences, thereby having a reasonable advertising policy.

1.7. How to build utility matrix

The system can hardly make any other suggestions to the user besides the most well-liked items without the utility matrix. The utility matrix’s construction is therefore crucial in recommender systems. But creating this matrix is frequently fraught with challenges. Each user-item pair in the utility matrix is rated using one of two typical methods:

1. Ask users to rate products. Amazon frequently sends reminder emails to users asking them to rate their bought products. This strategy does, however, have some drawbacks because consumers frequently don’t rate products. If so, it might be biased judgments made by people who want to be valuable.
2. The second approach is based on user behavior. A user is said to like something if they purchase it from Amazon, watch it on Youtube (possibly several times), or read an article about it. Facebook uses our favorite content to determine what relevant content to display in your newsfeed. Facebook gains more benefits the more you use it, so it always

sends you information that you probably want to read. Typically, using this method, we can only create a matrix with the components 1 and 0, with 1 denoting that the user likes the item and 0 denoting that there is no information. In this context, 0 simply indicates that the user has not provided the information, not that it is lower than 1. The length of time or the number of times a user views an item can also be used to create matrices with values higher than 1. Additionally, the dislike button can occasionally benefit the system in other ways, in which case the corresponding value can be set to -1.

1.8. Summary

Recommendation systems are a type of artificial intelligence that provide personalized recommendations to users based on their preferences and behavior. They have grown in popularity over the past few years as e-commerce and content consumption platforms have expanded.

As traditional recommendation systems may not be able to capture product diversity in the long run, the long tail phenomenon has a significant impact on recommendation systems in trade. This issue can be resolved, and users can receive more varied and individualized recommendations with the help of some techniques.

There are two main techniques for recommendation system:

1. Content-based system.
2. Collaborative filtering.

The utility matrix is a fundamental concept in recommendation systems that captures user-item interactions and is used to generate personalized recommendations and evaluate the performance of the recommendation system. Our task is to predict values of the unfilled cells in the utility matrix using techniques that are applied to the filled cells.

2. Content-based recommendation system

2.1. Introduction

Content-based recommendation system is a type of recommendation system that makes recommendations to users based on the characteristics of the items and their preferences. The system generates recommendations based on the content of the items, such as text, images, or audio.

A set of features or attributes are used to describe items in content-based recommendation system. For example, a movie's genre, director, actors, and plot keywords could all be considered as its features. Then, based on the similarities between features of the items and the user's preferences, the system can generate recommendations for users.

2.2. Item profile

In content-based systems, we must create a profile for each item based on the content of that item. A feature vector is a mathematical representation of this profile. In straightforward

situations, this vector is taken directly from the item. Consider the details of a song that can be incorporated into the recommendation system, for instance:

1. Singer. For the same song, user A prefers version from singer K, user B prefer singer L.
2. Composer. For the same genre but different composer.
3. Release Year. Some users prefer old music to modern music.
4. Genre.

	A	B	C	D	E	F	Item's feature vectors
Bohemian Rhapsody	5	5	0	0	1	?	$x_1 = [0.91, 0.02]^T$
I Love Rock 'N Roll	5	?	?	0	?	?	$x_2 = [0.99, 0.11]^T$
Hotel California	?	4	1	?	?	1	$x_3 = [0.95, 0.05]^T$
Twinkle, Twinkle Little Star	1	1	4	4	4	?	$x_4 = [0.01, 0.99]^T$
Happy and You Know It	1	0	5	?	?	?	$x_5 = [0.03, 0.98]^T$
User's models	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	← need to optimize

Figure 4.2: Assume that the last column contains the feature vector for each product. Finding a corresponding model θ_i for each user is necessary in order to get the best model possible.

There are numerous other aspects of a song that can be mentioned. All other elements, with the exception of the Hard-to-define Category, can be precisely defined.

For each song in the example in Figure 3.6, we create a two-dimensional feature vector, with the first dimension representing the song's Rock level and the second representing its Children's level. Let each song's feature vectors be x_1, x_2, x_3, x_4, x_5. Assume the feature vectors (in columnar form) for each song are given in Figure 17.2. Here, we consider these vectors to be defined in some way.

Similarly, the behavior of each user can also be modeled as a set of parameters θ_i. The training data to build each model θ_u are pairs (item profile, rating) corresponding to the items that the user has rated. Filling in the missing values in the utility matrix is to predict the level of interest when applying the θ_u model to them. This output can be written as a function $f(\theta_u, x_i)$. The choice of the form of $f(\theta_u, x_i)$ depends on the problem. In this chapter, we will be interested in the simplest form, the linear form.

2.3. Loss function

Assume that the number of users is N, the number of items is M. Profile matrix $X = [x_1, x_2, \dots, x_M] \in R^{d \times M}$, and the utility matrix is $\boldsymbol{Y} \in R^{M \times N}$. The component in the $m-th$ row, $n-th$ column of $\boldsymbol{Y}$ is the interest (here is the number of stars rated) of the $n-th$ user on the $m-th$ item that the system has collected. The $\boldsymbol{Y}$ matrix is missing a lot of components corresponding to the values that the system needs to predict. In addition, let $\boldsymbol{R}$ be the rated or not matrix representing whether a user has rated an item or not. Specifically, r_{mn} equals 1 if the $m-th$ item has been evaluated by the $n-th$ user, 0 otherwise.

Linear model

Suppose that we can find a model for each user, illustrated by a column vector of coefficients $w_n \in R^d$ and bias b_n such that a user's interest in an item can be calculated using a function linear:

$$y_{mn} = w_n^T x_m + b_n$$

Considering any $n-th$ user, if we consider the training set as the set of filled components of y_n (the $n-th$ column of the $\boldsymbol{Y}$ matrix), we can construct a loss function similar to ridge regression (linear). regression with l_2 regularization) as follows:

$$L_n(w_n, b_n) = \frac{1}{2s_n} \sum_{m:r_{mn}=1} (w_n^T x_m + b_n - y_{mn})^2 + \frac{\lambda}{2s_n} \|w_n\|_2^2$$

where, the second component is regularization and λ is a positive parameter; s_n is the number of items that the $n-th$ user has rated, which is the sum of the elements on the $n-th$ column of the matrix $\boldsymbol{R}$, i.e., $s_n = \sum_{m=1}^{M} r_{mn}$. Note that regularization is not usually applied to bias b_n.

Since the loss function expression depends only on the items that have been evaluated, we can reduce it by making $\hat{y}_n \in R^{s_n}$ a sub-vector of y_n, constructed by extracting components with different signs. "?" in the $n-th$ column of Y. At the same time, let $\hat{X}_n \in R^{d \times s_n}$ be a submatrix of the feature matrix X, created by extracting the columns corresponding to the items evaluated by the user $n-th$. (See the example below for better understanding). Then, the loss function expression of the model for the $n-th$ user is abbreviated as:

$$L_n(w_n, b_n) = \frac{1}{2s_n} \left\| \hat{X}_n^T w_n + b_n e_n - \hat{y}_n \right\|_2^2 + \frac{\lambda}{2s_n} \|w_n\|_2^2$$

where e_n is the column vector with all components being 1. This is exactly the loss function of ridge regression. The pair of solutions w_n, b_n can be found through gradient descent algorithms. In this chapter, we will directly use the Ridge class in sklearn.linear_model. It is worth noting here that w_n is only determined if the $n-th$ user has rated at least one product.

2.4. Example of loss function for user E

Back to the example in the Figure 4.2, features matrix for items (each column corresponding to an item):

$$X = \begin{bmatrix} 0.99 & 0.91 & 0.95 & 0.01 & 0.03 \\ 0.02 & 0.11 & 0.05 & 0.99 & 0.98 \end{bmatrix}$$

Consider the case of user E with $n = 5$, $y_5 = [1, ?, ?, 4, ?]^T \Rightarrow r_5 = [1,0,0,1,0]^T$. Since E rates only the first and the fourth item so $s_5 = 2$. Additionally:

$$\hat{X}_5 = \begin{bmatrix} 0.99 & 0.01 \\ 0.02 & 0.99 \end{bmatrix}, \hat{y}_5 = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, e_5 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Then, the loss function for the coefficient corresponding to user E is:

$$L_5(w_5, b_5) = \frac{1}{4}\left\| \begin{bmatrix} 0.99 & 0.02 \\ 0.01 & 0.99 \end{bmatrix} w_5 + b_5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 4 \end{bmatrix} \right\|_2^2 + \frac{\lambda}{4}\|w_5\|_2^2$$

The analysis presented above will be used to apply to data set in the next chapter, which could be considered as a real-world issue.

2.5. Development in Python

2.5.1. Building Item Profile

An important job in a content-based recommendation system is to build a profile for each item, that is, a feature vector for each item. First, we need to load all the information about the items into the item's variable:

```
#Reading items file:
i_cols = ['movie id', 'movie title' ,'release date','video release date', 'IMDb
URL', 'unknown', 'Action', 'Adventure',
 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama',
'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller',
'War', 'Western']

items = pd.read_csv('ml-100k/u.item', sep='|', names=i_cols, encoding='latin-1')

n_items = items.shape[0]
print('Number of items: ', n_items)
```

Result:

```
Number of items: 1682
```

Since we're building our profile by movie genre, we'll only be interested in the 19 binary values at the end of each row:

```
# Build Item Profile with Movie Genres, they are the last 19 columns
X0 = np.asmatrix(items)
X_train_counts = X0[:,-19:]
```

Next, we display the first few rows of the rate_train matrix:

```
print(rate_train[:4, :])
```

Result:

```
[[ 1 1 5 874965758]
 [ 1 2 3 876893171]
 [ 1 3 4 878542960]
 [ 1 4 3 876893119]]
```

The first row is understood as the first user to rate the first movie 5 stars. The last column is a number indicating the evaluation time, we will ignore this parameter.

Next, we will construct the feature vector for each item based on the TF-IDF feature and movie genre matrix in the sklearn library.

```
from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer(smooth_idf=True, norm ='l2')
X = transformer.fit_transform(X_train_counts.tolist()).toarray()
```

After this step, each row of X corresponds to the feature vector of a movie.

2.5.2. Find model for each user

For each user, we need to find out what movies that user has rated, and the value of those ratings.

```
def get_items_rated_by_users(rate_matrix,user_id):
    """
    in each line of rate_matrix, we have infor: user_id, item_id, rating
(scores), time_stamp
    we care about the first three values
    """
    y=rate_matrix[:,0] # all users
    # item indices rated by user_id
    # we need to +1 to user_id since in the rate_matrix, id starts from 1
    # while index in python starts from 0
    ids = np.where(y == user_id +1)[0]
    item_ids = rate_matrix[ids, 1] - 1 # index starts from 0
    scores = rate_matrix[ids, 2]
    return (item_ids, scores)
```

Now, we can find the Ridge Regression coefficients for each user:

```
from sklearn.linear_model import Ridge
from sklearn import linear_model
d=X.shape[1] #data dimension
W = np.zeros((d, n_users))
b = np.zeros((1, n_users))
for n in range(n_users):
    ids, scores = get_items_rated_by_users(rate_train,n)
    if(len(ids)==0):
        W[:,n]=0
        b[0,n]=0
        continue
    clf=Ridge(alpha=0.01, fit_intercept=True)
    Xhat=X[ids,:]
    clf.fit(Xhat,scores)
    W[:,n]=clf.coef_
    b[0,n]=clf.intercept_
```

2.5.3. Predict user's ratings

After calculating the W and b coefficients, the rating that each user rated each movie is predicted:

```
#predicted scores
Yhat=X.dot(W)+b
```

Here is an example with a user whose id is 30.

```
n = 30
np.set_printoptions(precision=2) # 2 digits after .
ids, scores = get_items_rated_by_users(rate_test, n)
print('Rated movies ids : ', ids )
print('True ratings : ', scores)
print('Predicted ratings: ', Yhat[ids, n])
```

Result:

```
Rated movies ids : [134 301 320 483 492 497 503 681 704 885]

True ratings : [4 4 4 5 5 4 5 2 5 2]

Predicted ratings: [3.42 3.56 4.7 3.82 3.66 3.53 4.19 3.67 4.03 3.64]
```

2.6. Summary and discussion

Over other recommendation systems like collaborative filtering, content-based recommendation systems have a number of advantages. For instance, content-based recommendation systems are more privacy-protective since they don't need information about other users. Additionally, they can provide recommendations for new or less popular items, as long as they have the same features as the items the user has interacted with previously.

However, this approach has two basic disadvantages:

1. When building a model for a single user, content-based systems do not take advantage of information from other users. This information is often very useful because the purchasing behavior of users is often grouped into a few simple groups. If it knows the buying behavior of some users in the group, the system should be able to infer the behavior of the rest of the users.
2. It's not always possible to build a profile for each item.

It is possible to think of creating a model for each user as a regression or classification problem, with the training data being the pairs of the user's ratings (item profile and rating). Item profiles describe the item rather than relying on the user; they can also be recognized by requiring the user to tag them.

In conclusion, content-based recommendation system is a common method for creating individualized suggestions based on the content of items. It offers various benefits over other recommendation systems, but they also have limits that must be considered while designing and evaluating the system.

3. Neighborhood-based collaborative filtering

3.1. Introduction

In the last chapter, I investigated a basic recommendation system based on feature vectors of each individual item. The content-based recommendation system has the advantage that the model built for each user is independent of other users and is based on the profiles of the items. This offers the benefit of reducing memory and calculation time. This technique has two major drawbacks. First, while developing a model for a single user, content-based recommendation system fails to leverage information from other users. This information is frequently quite valuable since user purchase behavior is frequently categorized into a few easy groupings. If the system is aware of the purchasing habits of certain users in the group, it should be able to deduce the purchasing habits of the other users. Second, creating a profile for each item is not always possible.

These disadvantages can be addressed by a technique called collaborative filtering.

The ideal behind neighborhood-based collaborative filtering is to predict a user's interest in an item based on the behavior of other users who are similar to this user. The degree of interest these individuals have in other objects known to the system can be used to assess their proximity. For example, A and B both enjoy the film Se7en (1995) and have given it five stars rating. We already know that A like John Wick (2014), so it's probable that B will enjoy this film as well.

The two most important questions in a neighborhood-based collaborative filtering system are:

1. How to determine the similarity between two users?
2. Once similar users have been identified, how can one predict a user's interest in an item?

User-user collaborative filtering is the process of determining each user's interest in an item based on how similar users are to that thing. Item-item collaborative filtering is another way that is considered to be more productive. Instead of identifying similarities between people, the system detects similarities between products in this technique. From there, the system suggests items that are close to the items that the user has a high level of interest in.

In the next few chapters, I will use user-user collaborative filtering to explain the technique, item-item collaborative filtering will be explained shorty.

3.2. Similarity determination function

The most significant task in user-user collaborative filtering is to determine the degree of similarity between two users. Assuming that the only data we have is the utility matrix Y, the similarity needs to be determined based on the columns corresponding to the two users in this matrix.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	3	?	?	0	?	?	?
i_2	?	4	1	?	?	1	2
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5

Figure 5-1: Example of utility matrix based on ratings users give to items.

Suppose there are users from u_0 to u_6 and items from i_0 to i_4 where the numbers in each square represent the number of stars each user has rated the item with a higher value representing a higher level of interest. The question marks are the values the system needs to look for. Set the similarity of two users u_i , u_j to $sim(u_i, u_0)$. The first observation that can be noticed is that u_0, u_1 like i_0, i_1, i_2 and don't like i_3, i_4 very much. The opposite happens for the rest of the users. So, a good similarity function should ensure:

$$sim(u_0, u_1) > sim(u_0, u_i), \forall i > 1$$

To determine the interest of u_0 to i_2, we should base u_1's behavior on this item. Fortunately, u_1 already likes i_2 so the system needs to recommend i_2 to u_0.

The question is, how should the similarity function be constructed? To measure similarity between two users, a common practice is to construct a feature vector for each user and then apply a function capable of measuring the similarity between those two vectors. Note that this

feature vector construction is different from the item profile construction as in content-based recommendation systems. These vectors are built directly on the utility matrix and do not use additional external information such as item profiles. For each user, the only information we know is the ratings that user has made, i.e., the column corresponding to that user in the utility matrix. However, the difficulty is that these columns often have a lot of missing values because each user usually only evaluates a very small number of items. One workaround is to help the original system rough estimate these values so that filling doesn't affect the similarity between the two vectors much. This estimation is for similarity calculation only, not the result of the system to be estimated.

So, each '?' should be replaced by what value to limit the estimation bias? The first option that can be thought of is to replace the '?' with the value 0. This is not good because the value 0 corresponds to the lowest level of interest; Just because a user hasn't rated an item doesn't mean they don't care about the item at all. A safer value is 2.5 because it is the average of 0, the lowest, and 5, the highest. However, this value has limitations for easy or fastidious users. Easy users can rate three stars for items they don't like, conversely, difficult users can rate three stars for items they like. Mass replacement of missing elements by 2.5 in this case has not been effective. A more likely value for this is to estimate the missing elements as the average value that a user evaluates. This helps avoid a user being too fastidious or easygoing. And these estimates depend on each user. Now we observe an example that describes user-user collaborative filtering:

Figure 5-2. Example about user-user collaborative filtering.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	?	?
i_1	3	?	?	0	?	?	?
i_2	?	4	1	?	?	1	2
i_3	2	2	3	4	4	?	4
i_4	2	0	4	?	?	?	5
	↓	↓	↓	↓	↓	↓	↓
$\bar{u}_j$	3.25	2.75	2.50	1.33	2.50	1.50	3.33

Figure 5-2: Original utility matrix Y and mean user ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0	0
i_1	0.75	0	0	-1.33	0	0.5	0
i_2	0	1.25	-1.5	0	0	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0	0.67
i_4	-1.25	-2.75	1.5	0	0	0	1.67

Figure 5-3: Normalized utility matrix $\bar{Y}$

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
u_0	1	0.83	-0.58	-0.79	-0.82	0.2	-0.38
u_1	0.83	1	-0.87	-0.40	-0.55	-0.23	-0.71
u_2	-0.58	-0.87	1	0.27	0.32	0.47	0.96
u_3	-0.79	-0.40	0.27	1	0.87	-0.29	0.18
u_4	-0.82	-0.55	0.32	0.87	1	0	0.16
u_5	0.2	-0.23	0.47	-0.29	0	1	0.56
u_6	-0.38	-0.71	0.96	0.18	0.16	0.56	1

Figure 5-4: User similarity matrix S

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	1.75	2.25	-0.5	-1.33	-1.5	0.18	-0.63
i_1	0.75	0.48	-0.17	-1.33	-1.33	0.5	0.05

Figure 5-5: $\hat{Y}$

i_2	0.91	1.25	-1.5	-1.84	-1.78	-0.5	-2.33
i_3	-1.25	-0.75	0.5	2.67	1.5	0.59	0.67
i_4	-1.25	-2.75	1.5	1.57	1.56	1.59	1.67

Figure 5-6: Example: Predict normalized rating of u_1 on i_1 with $k = 2$

Users who rated i_1: $\{u_0, u_3, u_5\}$

Corresponding similarities: $\{0.83, -0.40, -0.23\}$

→ most similar users: $N(u_0, u_i) = \{u_0, u_5\}$ with normalized ratings $\{0.75, 0.5\}$

→ $\hat{y}_{i_1,u_1} = \frac{0.83*0.75+(-0.23)*0.5}{0.83+|-0.23|} \approx 0.48$

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	5	5	2	0	1	1.68	2.70
i_1	4	3.23	2.33	0	1.67	2	3.38
i_2	4.15	4	1	-0.5	0.71	1	1
i_3	2	2	3	4	4	2.1	4
i_4	2	0	4	2.9	4.06	3.10	5

Figure 5-7: Full Y

The last row in Figure 5-2 is the average of each user's ratings. High values correspond to easy-going users and vice versa. Then, if we continue to subtract this average value from each rating and replace the unknown values with 0, we will get a normalized utility matrix as shown in Figure 5-3. Doing this has a few advantages:

- Subtracting the average from each column causes each column to have both positive and negative values. Items corresponding to positive values can be considered as items of greater interest to the user than those corresponding to negative values. Items with a value of 0 mainly correspond to the unknown interest of that user.

- Technically, the number of dimensions of the utility matrix is very large with millions of users and items, if all these values are stored in a matrix, there is a high chance that there will not be enough memory. Observe that since the number of known evaluations is usually a very small number compared to the size of the utility matrix, it would be better to store this matrix as a sparse matrix, i.e., store only other values. no and their location. Therefore, it is better to replace the '?' with the value '0', which is not yet determined whether the user likes the item or not. This not only optimizes memory, but also calculates similarity matrices later on more efficiently. Here, the element in the $i-th$ row, $j-th$ column of the similarity matrix is the similarity of the $i-th$ and $j-th$ users.

After the data has been normalized, the commonly used similarity function is cosine similarity:

$$cosine\_similarity(u_1, u_2) = \cos(u_1, u_2) = \frac{u_1^T u_2}{\|u_1\|_2 . \|u_2\|_2}$$

Where $u_{1,2}$ are vectors corresponding to user 1 and user 2 as above. There is a function in Python that serves to efficiently calculate this value, which we will see in the programming section.

The degree of similarity of two vectors is a real number in the interval [-1, 1]. A value of 1 represents two vectors that are completely similar. The cosine function of an angle equals 1 means that the angle between the two vectors is 0, that is, the two vectors have the same direction and the same direction. A cos value of -1 indicates that these two vectors are completely opposite, i.e., have the same direction but different flavor. This means that if the behavior of two users is completely opposite, the degree of similarity between the two vectors is the lowest.

An example of the cosine similarity of users (normalized) in Figure 5-3 is shown in Figure 5-4. Similarity matrix S is a symmetric matrix because cos is an even function, and if user A is the same as user B, the converse is also true. The blue cells on the diagonal are all cos of the angle between a vector and itself, i.e., $cos(0) = 1$. When calculating in the following steps, we do not need to care about this 1 value. Continuing to look at the row vectors corresponding to u_0, u_1, u_2 we will see a few interesting things:

- u_0 is closer to u_1 and u_5 (the similarity is positive) than the rest of the users. The high similarity between u_0 and u_1 is understandable since both tend to care more about i_0, i_1, i_2 than the rest. The fact that u_0 is close to u_5 may at first seem absurd because u_5 undervalues the items that u_0 appreciates (Figure 5-2); however, looking at the normalized utility matrix in Figure 5-3, we see that this makes sense since the only item that both of these users have provided is i_1 with the corresponding values being positive.
- u_1 is close to u_0 and far from the rest of the users.
- u_2 is close to u_3, u_4, u_5, u_6 and far from the rest of the users.

From this similarity matrix, we can group users into two groups (u_0, u_1) and $(u_2, u_3, u_4, u_5, u_6)$. Since this matrix S is small, we can easily observe this; When the number of users is larger, visual identification is not feasible.

It is important to note here that when the number of users is large, the matrix S is also very large and it is likely that there is not enough memory to store it, even if only more than half of the elements of the matrix is stored. this symmetry. For those cases, for each new user, we only need to calculate and save the result of a row of the similarity matrix, corresponding to the similarity between that user and the other users.

3.3. Fill in the missing values in the utility matrix

Predicting a user's predicted rating of an item based on these closest users is very similar to what we see in K-nearest neighbors (KNN) with a distance function of cosine similarity.

Similar to KNN, NBCF also uses information of k neighbors to make predictions. Of course, to predict a user's interest in an item, we are only interested in the neighboring users who have rated the item. The value to be filled in is usually defined as the weighted average of the normalized ratings. There is a point to note, in KNN, the weights are determined based on the distance between two points, and these distances are non-negative numbers. In NBCF, weights are determined based on similarity between two users, these weights can be less than 0. The common formula used to predict the number of stars that $user\ u$ rated $item\ i$ is:

$$\hat{y}_{i,u} = \frac{\sum_{u_j \in N(u,i)} \bar{y}_{i,u_j} sim(u, u_j)}{\sum_{u_j \in N(u,i)} |sim(u, u_j)|}$$

where $N(u, i)$ is the set of $k\ users$ that are most similar, i.e., have the highest similarity of u who have evaluated i. d) shows filling in the missing values in the normalized utility matrix. The red background cells represent positive values, i.e., items that may be of interest to that user. Here, the threshold is taken to be 0, this threshold can be completely changed depending on whether we want to suggest more or less items.

An example of calculating the normalized rating of u_1 for i_1 is shown in Figure 5-6 with nearest neighbors $k = 2$. The steps are as follows.

1. Identify the users who rated i_1, they are u_0, u_3, u_5.
2. The similarity of u_1 with these users is $\{0.83, -0.40, -0.23\}$, respectively. The two ($k = 2$) maximum values are 0.83 and -0.23 for u_0 and u_5 respectively.
3. Determining the (normalized) evaluations of u_0 and u_5 for i_1, we get two values of 0.75 and 0.5, respectively.
4. Predicted result:

$$\hat{y}_{i_1,u_1} = \frac{0.83 \times 0.75 + (-0.23) \times 0.5}{0.83 + |-0.23|} \approx 0.48$$

Conversion of the normalized rating values to a scale of 5 can be done by adding the columns of the $\hat{Y}$ matrix with the average rating value of each user as calculated in a). How the system decides which item to recommend for each user can be determined in a variety of ways. The system can sort unrated items by predicted rating descending, or it can just select items with a positive normalized predicted rating–corresponding to which this user is more likely to like.

3.4. Item-item collaborative filtering

User-user CF has some limitations as follows:

- When the number of users is much larger than the number of items (which is often the case), the size of the similarity matrix is very large (each dimension of this matrix has the same number of key elements as the number of users). Storing a large matrix is often not feasible.
- The utility matrix Y is usually very sparse, i.e., only a small percentage of the elements are known. With a very large number of users compared to the number of items, many columns of this matrix have very few or even no non-zero elements because users are often lazy to evaluate items. Therefore, once that user changes the previous ratings or evaluates more items, the average of the ratings as well as the normalized vector corresponding to this user changes a lot. Accordingly, the similarity matrix calculation, which takes a lot of memory and time, also needs to be done again.

There is another approach, instead of finding similarity between users, we can find similarity between items. Since then, if a user likes an item, the system should suggest similar items to that user. This has several advantages:

- When the number of items is smaller than the number of users, the similarity matrix has a smaller size, which makes the storage and computation in later steps more efficient.
- Also assume that the number of items is less than the number of users. Since the total number of reviews is constant, the average number of items rated by a user will be less than the average number of users who have rated an item. In other words, if the utility matrix has fewer rows than columns, the average number of known elements in each row will be more than the average number of known elements in each column. Accordingly, information about each item is more than information about each user, and the calculation of similarity between rows is also more reliable. Furthermore, the average value of each row also changes less with a few more reviews. As such, updating the similarity matrix can be done less frequently.

This second approach is called item-item collaborative filtering (item-item CF). When the number of items is less than the number of users, this method is preferred.

The procedure for predicting missing ratings is the same as in user-user CF, except that now we need to calculate the similarity between rows.

Example of Item-Item Collaborative Filtering:

	u_0	u_1	u_2	u_3	u_4	u_5	u_6	→	
i_0	5	5	2	0	1	?	?	→	2.6
i_1	4	?	?	0	?	2	?	→	2
i_2	?	4	1	?	?	1	1	→	1.75
i_3	2	2	3	4	4	?	4	→	3.17
i_4	2	0	4	?	?	?	5	→	2.75

Figure 5-8: Original utility matrix Y and mean item ratings.

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-2.6	-1.6	0	0
i_1	2	0	0	-2	0	0	0
i_2	0	2.25	-0.75	0	0	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0	0.83
i_4	-0.75	-2.75	1.25	0	0	0	2.25

Figure 5-9: Normalized utility matrix $\bar{Y}$

	i_0	i_1	i_2	i_3	i_4
i_0	1	0.77	0.49	-0.89	-0.52
i_1	0.77	1	0	-0.64	-0.14
i_2	0.49	0	1	-0.55	-0.88
i_3	-0.89	-0.64	-0.55	1	0.68

Figure 5-10: Item similarity matrix S

i_4	-0.52	-0.14	-0.88	0.68	1

	u_0	u_1	u_2	u_3	u_4	u_5	u_6
i_0	2.4	2.4	-0.6	-2.6	-1.6	-0.29	-1.52
i_1	2	2.4	-0.6	-2	-1.25	0	-2.25
i_2	2.4	2.25	-0.75	-2.6	-1.2	-0.75	-0.75
i_3	-1.17	-1.17	-0.17	0.83	0.83	0.34	0.83
i_4	-0.75	-2.75	1.25	1.03	1.16	0.65	2.25

Figure 5-11: Full Y

This result is slightly different from the result found by user-user CF in the last two columns corresponding to u_5, u_6. It seems that this result is more reasonable because from the utility matrix, we see that there are two groups of users who like two different groups of items. The first group is u_0 and u_1; The second group is the remaining users.

3.5. Development in Python

The collaborative filtering algorithm in this chapter is relatively simple and contains no optimization problems. Below is the code that demonstrates the $uuCF$ class for user-user collaborative filtering. There are two main methods of this class, fit – calculates a matrix similarity, and predict – predicts the number of stars a user will rate an item.

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy import sparse
```

```
class CF(object):
    def __init__(self, Y_data, k, sim_func = cosine_similarity, uuCF = 1):
        self.uuCF       = uuCF # user-user (1) or item-item (0) CF
        # Y_data is a 2d array of shape (n_users, 3), each row of Y_data has form
[user_id, item_id, rating]
        self.Y_data     = Y_data if uuCF else Y_data[:, [1, 0, 2]]
        self.k          = k # number of neighborhoods
        self.sim_func   = sim_func # sim function, default cosine similarity
        self.Ybar_data  = None # normalized data
        self.n_users    = int(np.max(self.Y_data[:, 0])) + 1 # number of users
        self.n_items    = int(np.max(self.Y_data[:, 1])) + 1 # number of items

    def normalize_Y(self):
        users = self.Y_data[:, 0] # all users - first col of the Y_data
        self.Ybar_data = self.Y_data.copy()
        self.mu = np.zeros((self.n_users,))
        for n in range(self.n_users):
            # row indices of ratings made by user n
            ids = np.where(users == n)[0].astype(np.int32)
            ratings = self.Y_data[ids, 2]  # ratings made by user n
            self.mu[n] = np.mean(ratings) if ids.size > 0 else 0 # avoid zero
division
            self.Ybar_data[ids, 2] = (ratings - self.mu[n]).flatten()

        ## form the rating matrix as a sparse matrix
        data = np.asarray(self.Ybar_data[:, 2]).flatten()
        row = np.asarray(self.Ybar_data[:, 1]).flatten()
        col = np.asarray(self.Ybar_data[:, 0]).flatten()
        shape = np.asarray((int(self.n_items),int(self.n_users))).flatten()
        self.Ybar = sparse.coo_matrix((data,(row, col)), shape=shape).tocsr()

    def similarity(self):
        self.S = self.sim_func(self.Ybar.T, self.Ybar.T)

    def fit(self):
        self.normalize_Y()
        self.similarity()

    # predict the rating of user u for item i
    def __pred(self, u, i):
        # find item
        ids = np.where(self.Y_data[:, 1] == i)[0].astype(np.int32)
        # all users who rated i
        users_rated_i = (self.Y_data[ids, 0]).astype(np.int32)
        # similarity between u and users_rated_i
```

```
        # similarity between u and users_rated_i
        sim = self.S[u, users_rated_i].flatten()
        # find k most similarity users
        a = np.argsort(sim)[-self.k:]
        nearest_s = sim[a] # and the corresponding similarity values
        # the corresponding ratings
        r = self.Ybar[i, users_rated_i[a]]
        return (r*nearest_s).sum()/(np.abs(nearest_s).sum() + 1e-8) + self.mu[u]

    def pred(self, u, i):
        if self.uuCF: return self.__pred(u, i)
        return self.__pred(i, u)

    def evaluate_RMSE(self, rate_test):
        n_tests = rate_test.shape[0]
        SE = 0
        for n in range(n_tests):
            pred = self.pred(rate_test[n, 0], rate_test[n, 1])
            SE += (pred - rate_test[n, 2])**2

        RMSE = np.sqrt(SE/n_tests)
        return RMSE
```

3.6. Summary and discussion

Collaborative filtering is an item recommendation method with the main idea based on the behavior of other similar users on the same item. This inference is made based on a similarity matrix that measures the similarity between users.

To compute the similarity matrix, we first need to normalize the data. The common method is mean offset, which subtracts ratings from the average value a user gives for items.

The commonly used similarity function is cosine similarity.

A similar approach is that instead of looking for users that are close to a user (user-user CF), we look for items that are close to a given item (item-item CF). In fact, item-item CF usually gives better results.

4. Matrix factorization collaborative filtering

4.1. Introduction

In the last chapter, I researched a collaborative filtering (CF) method based on the behavior of neighboring users or items. In this chapter, we will get acquainted with another approach for collaborative filtering based on the problem of matrix factorization or decomposition matrix. This method is called matrix factorization collaborative filtering (MFCF).

In content-based recommendation systems, each item is described by a vector x called the item profile. In that method, we need to find a coefficient vector w corresponding to each user such that the known rating that user gives the item is approximately equal to:

$$y \approx w^T x = x^T w$$

With this approach, the utility matrix Y, assuming it is completely filled, will approximate to:

$$Y \approx \begin{bmatrix} x_1^T w_1 & x_1^T w_2 & \dots & x_1^T w_N \\ x_2^T w_1 & x_2^T w_2 & \dots & x_2^T w_N \\ \dots & \dots & \dots & \dots \\ x_M^T w_1 & x_M^T w_2 & \dots & x_M^T w_N \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ \dots \\ x_M^T \end{bmatrix} [w_1\ w_2\ \dots\ w_N] = X^T W$$

M: number of items.

N: number of users.

In content-based collaborative filtering, x is built based on only the item's descriptive information (TF-IDF), and this construction is independent of the matching coefficient for each user. The construction of the item profile plays a very important role and has a direct influence on the performance of the model. In addition, building each model individually for each user leads to poor results because it does not exploit the relationship between users.

Now, suppose that we do not need to pre-construct the x profile items, but that the feature vector for each of these items can be trained concurrently with each user's model (here, a coefficient vector). This means, the variables in the optimization problem are both X and W; where, X is the matrix of the entire profile item, each column corresponds to an item, W is the matrix of the entire user model, each column corresponds to a user.

With this approach, we are trying to approximate the utility matrix $Y \in R^{MxN}$ is equal to the product of two matrices $X \in R^{MxK}$ and $W \in R^{KxN}$. Usually, K is chosen to be a much smaller number than M, N. Then, both the X and W matrices have a rank not exceeding K.

$$\mathbf{Y} \approx \hat{\mathbf{Y}} = \mathbf{X}^T\mathbf{W}$$

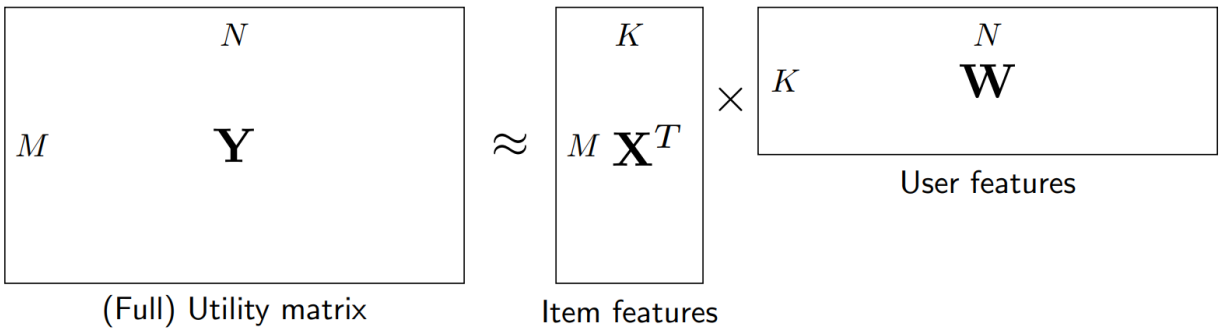

Figure 6-1: Matrix factorization. Utility matrix $Y \in R^{MxN}$ decomposed into the product of two matrix $X \in R^{MxK}$ and $W \in R^{KxN}$.

Now, suppose that we do not need to pre-construct the x profile items, but that the feature vector for each of these items can be trained concurrently with each user's model (here, a coefficient vector). This means, the variables in the optimization problem are both X and W; where, X is the matrix of the entire profile item, each column corresponds to an item, W is the matrix of the entire user model, each column corresponds to a user.

With this approach, we are trying to approximate the utility matrix $Y \in R^{MxN}$ by the product of two matrices $X \in R^{MxK}$ and $W \in R^{KxN}$. Usually, K is chosen to be a much smaller number than M, N. Then, both the X and W matrices have a rank not exceeding K. Therefore, this method is also called low-rank matrix factorization (Figure 6-1).

The main idea behind matrix factorization for recommendation systems is that there exist latent features that describe the relationship between items and users. For example, in the movie recommendation system, the hidden features can be criminal, political, action, comedy, etc.; may also be some combination of these genres; Or it could be anything we don't really need to name. Each item will be hidden to some extent corresponding to the coefficients in its vector x, the higher the coefficient corresponds to having that property. Similarly, each user will also tend to prefer certain hidden properties described by the coefficients in its vector w. The high coefficient corresponds to the user liking the movies with that hidden feature. The value of the expression $x^T w$ will be high if the corresponding components of x and w are both high (and positive). This means that the item has hidden properties that the user likes, so we should suggest this item to that user.

Why is matrix factorization classified as collaborative filtering? The answer comes from optimizing the loss function that we will discuss in the next parts of this chapter. Basically, to find the solution of the optimization problem, we have to find X and W respectively when the remaining components are fixed. Thus, each column of X will depend on all columns of W. In contrast, each column of W depends on all columns of X. Thus, there are interlaced constraints

between the columns. components of the two matrices above. That is, we need to use the information of all to deduce them all. Therefore, this method is also classified as collaborative filtering.

In practical problems, the number of items M and the number of users N are often very large. Finding simple models to predict ratings needs to be done as quickly as possible. Neighborhood-based collaborative filtering does not require too much training, but in the prediction process, we need to find the similarity of the user in question with all the remaining users and then infer the result. In contrast, with matrix factorization, the training can be a bit complicated because we have to iteratively optimize one matrix while fixing the other, but the prediction is simpler since we only need to take the product. scalar of two vectors $x^T w$, each vector of length K is a much smaller number than M, N. Therefore, the prediction process does not require high computational power. This makes it suitable for models with large data sets.

In addition, storing two matrices X and W requires a small amount of memory compared to storing the entire utility and similarity matrix in neighborhood-based collaborative filtering. Specifically, we need memory to hold $K(M + N)$ elements instead of M^2 or N^2 of a similarity matrix.

4.2. Approximate known ratings

As mentioned, the rating of $user\ n$ on $item\ m$ can be approximated by $y_{mn} = x_m^T w_n$. We can also add biases to this approximation and optimize those biases. Specifically:

$$y_{mn} \approx x_m^T w_n + b_m + d_n$$

b_m, d_n are the coefficients respectively for item m and user n.

Vector $b = [b_1, b_2, \dots b_M]^T$ is the bias vector for items.

Vector $d = [d_1, d_2, \dots d_N]^T$ is the bias vector for items.

As in neighborhood-based collaborative filtering (NBCF), these values can also be thought of as data normalization values with b corresponding to item-item CF and d corresponding to user-user CF. Unlike in NBCF, these values will be optimized to find the values that best approximate the training set. In addition, training both d and b simultaneously helps to combine both user-user CF and item-item CF into an optimization problem. Therefore, we expect that this method will yield better results.

4.3. Loss function

The loss function for the Matrix Factorization Collaborative Filtering can be written as:

$$\mathcal{L}(X, W, b, d) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} (x_m^T w_n + b_m + d_n - y_{mn}) + \frac{\lambda}{2} (\|X\|_F^2 + \|W\|_F^2)$$

$r_{mn} = 1$ if m-th item has been rated by n-th user.

s: number of ratings in dataset.

y_{mn}: rating of $n-th$ user for $m-th$ item.

$\|X\|_F$: Frobenius norm of X.

$\frac{1}{2s}\sum_{n=1}^{N}\ \sum_{m:r_{mn}=1}\ (x_m^T w_n + b_m + d_n - y_{mn})$ is the loss data, is the mean error of the model.

$\frac{\lambda}{2}(\|X\|_F^2 + \|W\|_F^2)$ is l_2 regularization loss, help avoid overfitting.

Simultaneous optimization of X, W, b, d is relatively complicated. Instead, the method used is to optimize one of the two pairs $(X, b), (W, d)$ in turn, while the other pair is fixed. This process is repeated until the loss function converges.

4.4. Optimizing the loss function

When the pair (X, b) is fixed, the pair (W, d) optimization problem can be decomposed into N subproblems:

$$\mathcal{L}_1(w_n, d_n) = \frac{1}{2s}\sum_{m:r_{mn}=1}(x_m^T w_n + b_m + d_n - y_{mn})^2 + \frac{\lambda}{2}(\|w_n\|_2^2)$$

Each problem can be optimized using gradient descent. Our important job is to calculate the derivatives of each of these small loss functions with respect to w_n, d_n. Since the expression in the sign Sigma depends only on the items that have been rated by the user in question (corresponding to $r_{mn} = 1$), we can simply by making $\hat{X}_n$ a submatrix generated by the columns of X corresponds to the items that have been rated by user n, $\hat{b}_n$is the corresponding sub-bias vector, and $\hat{y}_n$ is the corresponding rating. Then:

$$\mathcal{L}_1(w_n, d_n) = \frac{1}{2s}\left\|\hat{X}_n^T w_n + \hat{b}_n + d_n\mathbf{1} - \hat{y}_n\right\|^2 + \frac{\lambda}{2}(\|w_n\|_2^2)$$

With 1 is the vector with all elements equal to 1 and the appropriate size. Its derivative is

$$\nabla_{w_n}\mathcal{L}_1 = \frac{1}{s}\hat{X}_n\left(\hat{X}_n^T w_n + \hat{b}_n + d_n\mathbf{1} - \hat{y}_n\right) + \lambda w_n$$

$$\nabla_{b_n}\mathcal{L}_1 = \frac{1}{s}\mathbf{1}^T\left(\hat{X}_n^T w_n + \hat{b}_n + d_n\mathbf{1} - \hat{y}_n\right)$$

Update equation for w_n, d_n:

$$w_n \leftarrow w_n - \eta\left(\frac{1}{s}\hat{X}_n\left(\hat{X}_n^T w_n + \hat{b}_n + d_n\mathbf{1} - \hat{y}_n\right) + \lambda w_n\right)$$

$$d_n \leftarrow d_n - \eta\left(\frac{1}{s}\mathbf{1}^T\left(\hat{X}_n^T w_n + \hat{b}_n + d_n\mathbf{1} - \hat{y}_n\right)\right)$$

η is the learning rate.

Similarly, each column x_m of X, which is the feature vector for each item, and b_m can be found by optimizing the problem:

$$\mathcal{L}_2(x_m, b_m) = \frac{1}{2s} \sum_{n:r_{mn}=1} (w_n^T x_m + d_n + b_m - y_{mn})^2 + \frac{\lambda}{2}(\|x_m\|_2^2)$$

Let $\widehat{W}_m$ be the matrix created with the columns of W corresponding to the users who have rated item m, $\hat{d}_m$ the corresponding bias sub-vector, and $\hat{y}_m$ the corresponding rating vector. The problem becomes:

$$\mathcal{L}_2(x_m, b_m) = \frac{1}{2s} \left\| \widehat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m \right\|^2 + \frac{\lambda}{2}(\|x_m\|_2^2)$$

Update equation for x_m, b_m:

$$x_m \leftarrow x_m - \eta \left(\frac{1}{s} \widehat{W}_m \left(\widehat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m \right) + \lambda w_m \right)$$

$$b_m \leftarrow d_m - \eta \left(\frac{1}{s} \mathbf{1}^T \left(\widehat{W}_m^T x_m + \hat{d}_m + b_m \mathbf{1} - \hat{y}_m \right) \right)$$

4.5. Development in Python

First, we will write a Matrix Factorization class that performs the optimization of variables with a utility matrix given in the form of Y_data just like with NBCF. First, we declare some necessary libraries and initialize the Matrix Factorization class:

```
import json
import pandas as pd
import numpy as np

class MF(object):
    def __init__(self, Y, K, lam = 0.1, Xinit = None, Winit = None,
    learning_rate = 0.5, max_iter = 10000, print_every = 100):
        self.Y = Y      # represents the utility matrix
        self.K = K      # number of features
        self.lam = lam  # regularization parameter
        self.learning_rate = learning_rate  # for gradient descent
        self.max_iter = max_iter            # maximum number of iterations
        self.print_every = print_every      # print loss after each a few iters
        self.n_users = int(np.max(Y[:, 0])) + 1
        self.users_ids = np.unique(np.asarray(Y[:,0].reshape(Y[:,0].shape[0])))
        self.n_items = int(np.max(Y[:, 1])) + 1
        self.items_ids = np.unique(np.asarray(Y[:,1].reshape(Y[:,1].shape[0])))
        self.n_ratings = Y.shape[0]
        self.X = np.random.randn(self.n_items, K) if Xinit is None else Xinit
        self.W = np.random.randn(K, self.n_users) if Winit is None else Winit
        self.b = np.random.randn(self.n_items)  # item biases
        self.d = np.random.randn(self.n_users)  # user biases
```

Next, we write loss, updateXb, updateWd methods for class MF:

```
# return current loss value
def loss(self):
    L = 0
    for i in range(self.n_ratings):
        # user_id, item_id, rating
        n, m, rating = int(self.Y[i, 0]), int(self.Y[i, 1]), self.Y[i, 2]
        L += 0.5*(self.X[m].dot(self.W[:,n]) + self.b[m] + self.d[n] - rating)**2
    L /= self.n_ratings
    # regularization, don't ever forget this
    return L + 0.5*self.lam*(np.sum(self.X**2) + np.sum(self.W**2))
```

```
def updateWd(self):
    for n in self.users_ids:
        # get all items rated by user n, and the corresponding ratings
        ids = np.where(self.Y[:, 0] == n)[0]
        item_ids, ratings = self.Y[ids, 1].astype(np.int32), self.Y[ids, 2]
        Xn, bn = self.X[item_ids], self.b[item_ids]
        for i in range(30): # 30 iteration for each sub problem
            wn = self.W[:, n]
            error = Xn.dot(wn) + bn + self.d[n] - ratings
            grad_wn = Xn.T.dot(error)/self.n_ratings + self.lam*wn
            grad_dn = np.sum(error)/self.n_ratings
            # gradient descent
            self.W[:, n] -= np.array(self.learning_rate*grad_wn.reshape(-1))[0]
            self.d[n]    -= self.learning_rate*grad_dn
```

```
def updateXb(self):
    for m in self.items_ids:
        ids = np.where(self.Y[:, 1] == m)[0] # row indices of items m
        user_ids, ratings = self.Y[ids, 0].astype(np.int32), self.Y[ids, 2]
        Wm, dm = self.W[:, user_ids], self.d[user_ids]
        Wm = Wm.reshape(Wm.shape[0], Wm.shape[1])
        for i in range(30): # 30 iteration for each sub problem
            xm = self.X[m]
            error = Wm.T.dot(xm).reshape(-1,1) + self.b[m] + dm - ratings
            grad_xm = Wm.dot(error)/self.n_ratings + (self.lam*xm).reshape(-1,1)
            grad_bm = np.sum(error)/self.n_ratings
            # gradient descent
            self.X[m] -= np.array((self.learning_rate*grad_xm).T)[0]
            self.b[m] -= rs.learning_rate*grad_bm
```

The next part is the main optimization process of MF (fit), predicting the new rating (pred) and evaluating the model quality by root-mean-square error (evaluate_RMSE).

```
def fit(self):
    for it in range(self.max_iter):
        self.updateXb()
        self.updateWd()
        if (it+1) % self.print_every == 0:
            rsme_train = self.evaluate_RMSE(self.Y)
            # print("iter = ",it+1 ,", loss = "+self.loss(),", RMSE train =
",rsme_train)
            print(it+1)
            print(self.loss())
            print(rsme_train)
            print(self.evaluate_RMSE(rate_test))
```

```
def pred(self, u, i):
    # predict the rating of user u for item i
    u, i = int(u), int(i)
    try:
        pred = self.X[i, :].dot(self.W[:, u]) + self.b[i] + self.d[u]
    except:
        return 0
    return max(0, min(5, pred))

def evaluate_RMSE(self, rate_test):
    n_tests = rate_test.shape[0]
    SE = 0
    for n in range(n_tests):
        pred = self.pred(rate_test[n, 0], rate_test[n, 1])
        SE += (pred - rate_test[n, 2])**2

    RMSE = np.sqrt(SE/n_tests)
    return RMSE
```

At this point, we have built into the Matrix Factorization class with the necessary methods.

4.6. Summary and discussion

Nonnegative matrix factorization. When the data is not normalized, they all carry non-negative values. Even in the case that the rating range contains negative values, we only need to add a reasonable value to the utility matrix to get ratings that are non-negative. At that time, another matrix factorization method with additional constraints is also widely used and highly effective in the recommendation system, which is nonnegative matrix factorization (NMF), i.e., the analysis of the performance matrix of matrices with elements non-negative.

Through matrix factorization, users and items are linked together by hidden features. The association of each user and item to each hidden feature is measured by the corresponding component in their feature vector, the larger the value representing the greater the relevance of the user or item to that hidden feature. Intuitively, the relevance of a user or item to a hidden feature should be a non-negative number with a value of 0 indicating non-relevance. Furthermore, each user and item is only associated with certain hidden features. Therefore, the feature vectors for user and item should be non-negative vectors and have lots of zero values. These solutions can be obtained by adding non-negative constraints to the components of X and W. This is the origin of the idea and name of nonnegative matrix factorization.

Incremental matrix factorization. As mentioned, the prediction time of a recommendation system using matrix factorization is very fast, but the training time is quite long with large data sets. In fact, the utility matrix changes constantly because there are more users, items as well as new ratings or users want to change their rating, so the model parameters must also be regularly updated. This means that we must continue to perform the training process, which takes a lot of time. This is partially solved by incremental matrix factorization. The word incremental can be understood as a small adjustment to the data.