BanditMF: Multi-Armed Bandit Based Matrix Factorization Recommender System

by

Shenghao XU

1155148147

A report

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

The Chinese University of Hong Kong

May. 2021

Supervisor: Prof. John C.S. Lui

Abstract

Multi-armed bandits (MAB) provide a principled online learning approach to attain the balance between exploration and exploitation. Generally speaking, in a multi-armed bandit problem, to obtain a higher reward, the agent must choose the optimal action in various states based on previous experience (*exploit*) known actions to obtain a higher score; to discover these actions, the necessary discovery is required (*exploration*). Due to the superior performance and low feedback learning without the learning to act in multiple situations, Multi-armed Bandits drawing widespread attention in applications ranging such as recommender systems. Likewise, within the recommender system, collaborative filtering (CF) is arguably the earliest and most influential method in the recommender system. The meaning of collaboration is to filter the information through the relationship between the users and the feedback of the user's rating of the items together to find the target users' preferences. Crucially, new users and an ever-changing pool of recommend items are the challenges that recommender systems need to address. For collaborative filtering, the classical method is training the model offline, then perform the online testing, but this approach can no longer handle the dynamic changes in user preferences which is the so-called *cold start*. So how to effectively recommend items to users in the absence of effective information?

To address the aforementioned problems, a multi-armed bandit based collaborative filtering recommender system has been proposed, named BanditMF. BanditMF is designed to address two challenges in the multi-armed bandits algorithm and collaborative filtering: (1) how to solve the cold start problem for collaborative filtering under the condition of scarcity of valid information, (2) how to solve the sub-optimal problem of bandit algorithms in strong social relations domains caused by independently estimating unknown parameters associated with each user and ignoring correlations between users.

Acknowledgements

I would like to thanks for every person who comes to my life and deeply indebted to my beloved parents for their generous and unconditional support all over the years.

My deepest gratitude to my supervisor, Prof. John C.S. Lui who point me the direction of my research and gives valuable advice. Without his consistent and illuminating instruction, this report could not have reached its present stage.

Contents

Chapter 1

Introduction

In this section, first, a detailed account of the recommender system is given, followed by the general information about multi-armed bandits (MAB) and collaborative filtering (CF) will be introduced. The problem statement will also be covered in this chapter. Also, the aims and contributions of this project will have to be indicated.

1.1   General Introduction

Nowadays, with the rapid development of social networks, e-commerce, and sharing economy, it has become a core link of Internet services to discover users' needs, understand users' behaviors, and screen out the most relevant information and products for users. There is a huge amount of information on the Internet: YouTube users upload more than 400 hours of video every minute; 10 billion clicks on WeChat Moments within 1 day; Instagram Stories have 500 million active users per day [1]. Now the recommendation system has become an essential part of various video websites and e-commerce websites to provide various personalized services for users to interact with the system.

Early recommendation systems relied more on simple models or algorithms guided by intuition. For example, the recommendation system based on information retrieval treats the user information as a *query* and baesd on various types of information to express the item to be recommended as a *document*. Thus, the problem of recommending the most relevant set of items is transformed into the problem of finding the most relevant documents in information retrieval. As the field of recommendation systems has evolved,

---

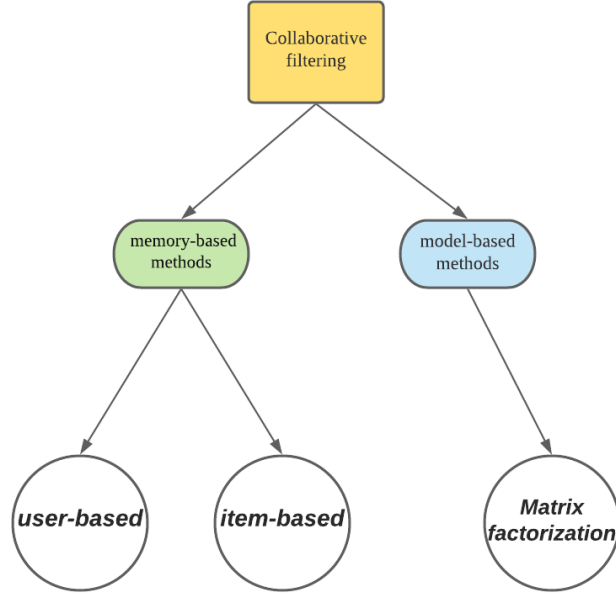[1] Data sources: Instagram Revenue and Usage Statistics (2021).

Figure 1.1: Relation diagram of collaborative filtering in recommender systems

traditional recommendation systems have been divided into two main categories: content-based filtering (CBF) and collaborative filtering (CF) [17]. The content-based approach describes the nature of users and items by defining explicit attributes (usually defined by experts with professional knowledge) and then recommends items that are *"compatible"* with the user's nature. For traditional collaborative filtering, as shown in Figure 1.1, it is categorized into two categories [1]: memory-based methods and model-based methods. Item-based collaborative filtering and user-based collaborative filtering fall under the category of memory-based methods. For matrix factorization (MF), it is the most popular collaborative filtering technique that belongs to model-based methods [9].The three methods mentioned above will be elaborated and analyzed in the next chapter.

Matrix factorization attracts a ton of attention due to its good scalability and ease of implementation. In [11], the authors comprehensively introduce the matrix factorization algorithm which integrates machine learning technology. Other than that, the authors in [9] proposed a method that combines matrix factorization with convolutional neural networks (CNN).As mentioned earlier, matrix factorization is scalable, on the one hand, it can be seamlessly integrated with mainstream models, on the other hand, it can do feature fusion with multiple information sources. So that in [8], a model that fusion the text comment information, geographic neighborhood information, project category information, and popularity information has been proposed to promote the precision of rating

prediction. However, these recommender systems focus on how to improve the accuracy of algorithms for a single problem (e.g. rating prediction for matrix factorization), thus neglecting the timeliness and systematicness of human-machine interaction which makes it difficult to model the vagaries of user behavior and the rapidly changing external environment completely. These matrix factorization-based recommendation systems only exploit the trained model for the recommendation and no properly exploration. Thus, these recommendation systems tend to produce cookie-cutter results.

In order to achieve a dynamic balance between the system and the user, and to shift away from viewing the recommendation system as a running fragment of one recommendation result at a time. Some researchers, such as the authors in [12] have started to explore how to apply some concepts of reinforcement learning to the scenario of recommendation systems. Reinforcement learning often represent a *"feedback system"* in which the user and the system interact to achieve the optimization of the target. The interaction process between the system and users is no longer a *"one-shot deal"*, which instead by the dynamic balance between the system and users that achieved through a series of results on the dimension of time. A number of algorithms have been proposed based on the context-free bandit and contextual bandit. Further detailed concepts and algorithms of context-free bandit and contextual bandit will be conduct in chapter 2.

However, in the more complex scenario of online mass recommendation systems, above algorithm has not been favored. Although such algorithms have beautiful proofs and mathematical properties in theory, there are often insurmountable barriers to product experience in practice. As the authors affirms in [20], the contextual bandit algorithm requires a tremendous amount of exploration resulting in a slow speed. Most of these algorithms are not familiar with the user in the initial period, so the algorithm tends to explore more items to understand the user. Therefore, during this period, the user may be confronted with a number of items that are not as relevant. In these algorithms, the assumption is that once the exploration period is over, the algorithm will be able to learn the user's preferences better and provide more *"trustworthy"* recommendations. However, this hypothetical scenario does not hold in reality. In the real world, only highly loyal users may have the tolerance to receive less relevant recommendations. From the perspective of new users or users with low loyalty, the frequency of use of the product is not high, and when these users find some irrelevant recommendation results in a few

times of use, it is likely to cause new users to abandon the product permanently.

As alluded to above, collaborative filtering is the process of sifting and filtering information together with the relationship between users and their feedback on the items, to recommend items of interest to the target users. Since coordinated filtering focuses more on judging the similarity between historical records and user's similarity to make recommendations that result in the content universe of collaborative filtering is almost static and loses the ability to recommend new items. Unfortunately, in many recommendation scenarios, where the universe of content experience rapid turnover, and the popularity of content changes over time. Furthermore, the cold start [14] problem is also an insurmountable chasm for collaborative filtering. In former research [5], it was demonstrated that these problems led to poor performance and difficulty in applying traditional collaborative filtering-based recommender systems.

To address the aforementioned problems, in this work, BanditMF has been proposed which is a multi-armed bandit based matrix factorization recommender system. This system combines the matrix factorization (MF) which is model-based collaborative filtering with the multi-armed bandit algorithm. BanditMF contains an offline subsystem focusing on matrix factorization and an online subsystem with a multi-armed bandit algorithm as the core. With the power of the multi-armed bandit algorithm, BanditMF solves the cold-start problem of the collaborative filtering method and gives the ability to recommend new items for users. Meanwhile, the bandit algorithm in the online subsystem accepts parameters from the matrix factorization in the offline subsystem to make recommendations. Since the matrix factorization takes into account the connections and similarities between users, the multi-armed bandit algorithm uses these parameters to reduce the irrelevance of the recommended items when exploring, thus reducing the loss of new users with low loyalty due to irrelevant recommendations from the bandit algorithm during the exploration period. The detailed discussion of BanditMF will introduce in chapter 3.

The remainder of this report is organized as follows. In chapter 2, the representative multi-armed bandit and collaborative filtering methods will be reviewed and formulate the problem. Chapter 3, two traditional systems: collaborative filtering based recommender system and hybrid recommender system will be illustrated. Then based on those traditional systems, BanditMF will be introduced in detail. In chapter 4, the two traditional systems are implemented and experimentally evaluates BanditMF. We propose the future

work and the conclusion of the entire work will be present in chapter 5.

Chapter 2

Formulation

In this chapter, context-free bandit and contextual bandit will be introduced that followed by a brief review of the collaborative filtering method.

2.1 Multi-Armed Bandit

The multi-armed Bandit (MAB) framework provides principled solutions for the dilemma between exploration and exploitation.

As one of the classic problems of reinforcement learning - multi-armed bandit (MAB), it is different from machine learning (e.g. supervised learning), it is based on environmental interaction, trial-and-error to achieve the effect of learning, such a learning mode is more similar to the human learning mode.

	Action don't change state of world	Action change state of the world
Learn model of outcomes	Multi-armed bandit	Reinforcement Learning
Given model of stochastic outcomes	Decision theory	Markov Decision Process

Table 2.1: Four scenarios when making decisions under uncertainty

In Table 2.1[2], four different scenarios have been described which are the scenarios when making decisions under uncertainty. In the setting of multi-armed bandits, the outcomes (reward) are unknown, and the outcomes can be stochastic or adversarial, also the state of the world can not be changed by the actions.

The reinforcement learning process consists of five major elements: *agent*, *environment*, *state*, *action*, and *reward*. At a certain instant, the intelligence is in a certain state,

---

[2]Table from the paper: A Survey on Contextual Multi-armed Bandits.

and after executing an action, the environment receives the action, prompting the agent to move to the next state, while feeding back the reward, the agent's purpose is to maximize the accumulated reward and adjust the action which according to the amount of the reward to get higher reward.

For reinforcement learning can using Markov decision process (MDP) to formalize [18]. Formally, A Markov decision process is a 4 tuple (S, A, P_a, R_a), where:

- S is denote the set of states called the state space, where $S = S_1, S_2, S_3, S_4, \cdots$

- A is denote the set of actions called the action space, where $A = A_1, A_2, A_3, A_4, \cdots$

- P_a is denote the probability, $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability from state s to sate s'.

- R_a represent the reward. where $R_a(s, s') = \mathrm{E}(r_t \mid s_t = s, a_t = a)$.

Multi-armed bandits is a classic reinforcement learning problem (essentially a simplified class of reinforcement learning problem) that have non-associative states (learning from only one situation at a time, losing or winning) and that focus only on evaluable feedback. Suppose there is a bandit with K arms, each arm will have a certain probability to get a reward, so that we have K possible actions (each arm corresponds to an action), and the result of each action is only associated with the current state and is not affected by the results of historical actions which means the reward of each arm is only related to the probability set by the bandit, the previous winning and losing results will not affect this action. So that we can formally define multi-armed bandits problems as Markov decision processes with a single state as following:

- No sate S, but S_0 can denote as hypothetical constant state.

- K arms correspond to K different actions, where $A = a_1, a_2, A_3, , \cdots, a_K$.

- Do not have probability P, but it can be assumed that the state always transitions from S_0 to S_0.

- Have a reward function R dependence on action A instead of state S.

The key idea of the multi-armed bandits is make the optimal decisions to maximize the total reward, through exploration and exploitation.

2.1.1 Context-Free Bandit

Bandit algorithms can be categorized into *context-free* bandit and *contextual* bandit depending on whether or not contextual features are taken into account. For the traditional *K-armed* bandit due to:

1. The arm set $\mathcal{A}_t = \{a_1, a_2, \ldots, a_k\}$ with K actions are fixed for all $t \in T$, where T denote the total rounds.

2. Both user u_t and context $\mathbf{x}_{t,a}$ are unchanged for all $t \in T$.

Whenever the arm is pulled, as arm set $\mathcal{A}_t$ and context $\mathbf{x}_{t,a}$ are constant, therefore they do not affect the algorithm. So, for this kind of bandit algorithm, we refer to them as the *context-free* bandit [12].

For the above context-free bandit, we formulated as follows:

Problem Formulation: The *context-free* bandit

Given: T rounds, K actions, 1 agent.

In each round, for $t \in [T]$, agent choose the policy :

1. Policy choose action a_t.

2. For the chosen action, Policy observes the rewards r_t, with $r_t \in [0, 1]$.

For context-free bandit, μ denoting the mean reward vector, where $\mu \in [0, 1]^K$. The mean reward of action a is expressed as $\mu(a) = \mathbb{E}(\mathcal{D}_a)$. The μ^* is used to define the best mean reward, where $\mu^* := max_{a_i \in \{a_i\}_{i=1}^K} \mu(a)$. We use the notation $\Delta(a)$ to represent the *gap* of action a. The *gap* describes the difference between the reward received for action a and the reward received for the optimal action, that $\Delta(a) := \mu^* - \mu(a)$ and a^* is used to denote the optimal action, which action with $\mu(a) = \mu^*$. Formally, *regret* define as follows:

$$R(T) \triangleq \sum_{t=1}^{T} (\mu^* - \mathbb{E}[\mu_{(a_t)}]) = T \cdot \mu^* - \mathbb{E}\left[\sum_{t=1}^{T} \mu_{(a_t)}\right] \tag{2.1}$$

The $R(T)$ is called as *regret* at round T, where a_t is denote the action chosen at round t.

2.1.2 Contextual Bandit

As opposed to the *context-free* bandit, the *contextual* bandit is based on contextual information of the user and items (i.e. feature vector $\mathbf{x}_{t,a}$) to inferring the conditionally expected reward of an action.

For the above contextual bandit, we formulated as follows:

Problem Formulation: The *contextual* bandit

Given: T rounds, users u_t, arm set $\mathcal{A}_t$, feature vector (context) $\mathbf{x}_{t,a}$.

In each round, for $t \in [T]$, do :

1. For all $a \in \mathcal{A}_t$, algorithm observers the current user u_t and arm set $\mathcal{A}_t$ with correspond feature vector $\mathbf{x}_{t,a}$.

2. Algorithm chosen action a_t, and revives the rewards r_{t,a_t}.

3. With new observation $\mathbf{x}_{t,a}$, a_t, and r_{t,a_t}, the algorithm improves its arm selection strategy π.

As the variant of the multi-armed bandit, the contextual multi-armed bandits (CMAB) are different from the stochastic bandit and adversarial bandits. In the setting of the stochastic setting, the reward of an arm is sampled from the unknown distribution or so call reward distribution. In an adversarial environment, the reward for one arm is chosen by the adversarial, rather than drawn from any distribution. Conversely, under the setting of the contextual bandits, what we are interested in is the situation that we need to observe side information at each time slot, where side in formation is the so call contexts $\mathbf{x}_{t,a}$. For the contextual bandit, there is a set of actions $\mathcal{A}_t$, and each action maps a context to an arm, where at each iteration, before the arm selection, the agent will observe the d-dimensional context, where $\mathbf{x}_{t,a} \in \mathbb{R}^d$. Based on this context, along with the rewards of arms played in the past, to select the arms to be chosen in the current iteration. The goal of the algorithm is maximize the reward over the finite times $t \in [T]$. $\sum_{t=1}^{T} r_{t,a_t}$ denote the total rewards of the algorithm and define the optimal expected cumulative reward as $\mathbb{E}\left[\sum_{t=1}^{T} r_{t,a_t^*}\right]$, where a_t^* denote the optimal arm at time t. The *regret* of the contextual

bandit algorithm is defined as:

$$R(T) \triangleq \mathbb{E}\left[\sum_{t=1}^{T} r_{t,a_t^*}\right] - \mathbb{E}\left[\sum_{t=1}^{T} r_{t,a_t}\right] \tag{2.2}$$

To address the general situation, different algorithms are proposed, which include LIN-UCB proposed at [12], Contextual Thompson Sampling (CTS) be described in [2], and Neural Bandit in [3], in these algorithms, a linear dependency between the expected payoff of an action and its context is generally hypothesized.

2.2 Collaborative Filtering

In this section, two categories of methods involved in collaborative filtering will be discussed: memory-based methods and model-based methods.

2.2.1 Memory-Based Methods

Collaborative filtering takes into account the fact that there is a relationship between the items and the user's preference. In memory-based approaches, as the name suggests, a memory-based recommender system typically needs to leverage the entire user-item dataset and make recommendations accordingly. Once we have the user rating data for

	i_1	i_2	i_3	i_4	i_5
S_1	1.4	?	1.5	0.4	?
S_2	?	1.3	?	1.6	0.9
S_3	0.8	?	0.3	?	0.9
S_4	0.7	0.2	0.2	0.9	?

Figure 2.1: User−item rating matrix

the items, we can convert it into the form of the matrix as shown in the Figure 2.1. The horizontal coordinates are the users and the vertical coordinates are the items. The cor-

responding number is the user's rating of the item. The question mark within the matrix indicates that the user has not rated the item, and our goal is to use collaborative filtering to predict unknown ratings. Memory-based collaborative filtering to make recommendations by similar users or similar items, then the question arises how to define similar users or similar items? The answer is similarity. After we have the user-item rating matrix, as shown above, the row vectors in the matrix represent the preferences of each user, and the column vectors represent the attributes of each item that can be interpreted as there exists an attribute of the item that attracts users to provide corresponding rating. So that, by computes the similarity between row vectors, we will get the user similarity, and compute the similarity of vertical vectors, item similarity will get. *Cosine similarity*, *Pearson correlation coefficient*, and *Euclidean distance* are the common methods to compute the similarity.

This chapter has demonstrated the concept of memory-based collaborative filtering. It is now necessary to explain the two basic approaches of memory-based collaborative filtering: user-based method and item-based method.

2.2.1.1 User-Based Collaborative Filtering

For user-based collaborative filtering, in each recommendation, we have a target user who will be recommended. The user-based collaborative filtering algorithm first identifies users that are similar to that particular target user, i.e., users that share the target user's rating pattern. User-based collaborative filtering predicated on similarities including history, preferences, and choices made by users when purchasing, viewing, or certain other behaviors. Based on the ability conferred by similarity, we can estimate the ratings that a target user might give based on similar users' ratings of items that the target user did not give a rating on. Based on the predicted rating, the user-based collaborative filtering algorithm can recommend items with higher ratings to the target users.

User-based collaborative filtering can be illustrated briefly by Figure 2.2. Recall that the recommendation of the user-based method is based on similar users who share the same preference with the target user. As the example shown in Figure 2.2. Both *user1* and *user3* show a preference for *item3* and *item4*, and for this reason, we consider *user1* and *user3* to be similar users. Then, the user-based collaborative filtering algorithm will recommend *item1* to *user3* that *item1* is the item that wins the positive rating by *user3*'s
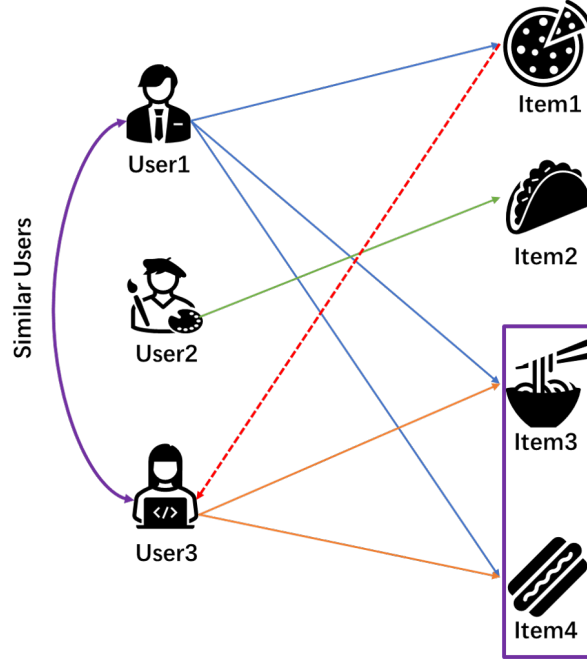
Figure 2.2: User-based collaborative filtering

similar user-*user1*.

2.2.1.2 Item-Based Collaborative Filtering

Item-based collaborative filtering and user-based collaborative filtering are essentially the same. The only difference is that item-based collaborative filtering is based on the similarity between items instead of the similarity between users in user-based collaborative filtering. The idea of item-based collaborative filtering is to find items similar to those you have purchased or browsed, then recommend those items to you, i.e., "Birds of a feather flock together" [16].

We use Figure 2.3 to interpret the process of item-based collaborative filtering. The first step is to identify the similar item through the behavior of users instead of the contents of the items (if we use the content of the item as the similarity criterion, then the system will morph into a content-based recommendation system). In the example in Figure 2.3, *item2* and *item3* are regarded as similar items due to this two items were received a positive rating from both *user1* and *user2*. By the figure, we can see that *user3* shows the preference on *item3*, so the item-based collaborative filtering algorithm will choose *item2* which is the similar item of *item3* to recommend to the *user3*.
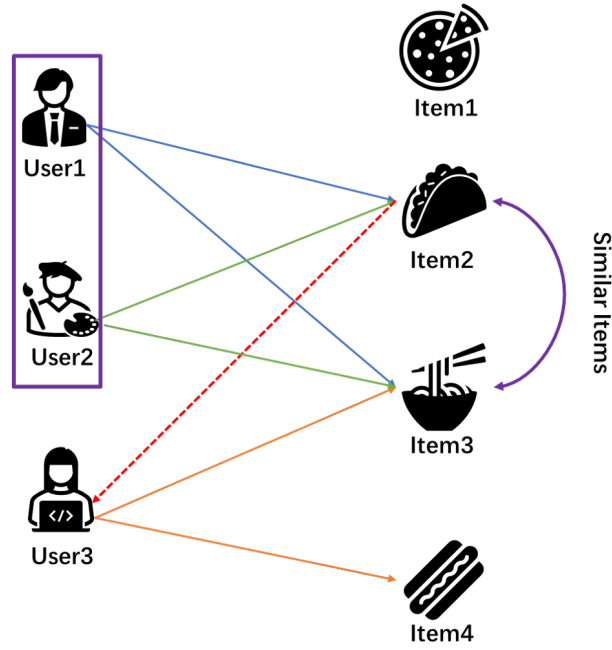
Figure 2.3: Item-based collaborative filtering

2.2.2 Model-Based Methods

It is worth noting that in the real world, the number of users is generally huge (e.g. Netflix, millions of users), compared to the number and type of items that are relatively fixed. Under this scenario, calculate the similarity between items offline by using item-based collaborative filtering is preferred, in order to optimize the efficiency of recommendation. On the contrary, in the case of a small number of users, User-based collaborative filtering is preferred. It can be seen from this that there is no good or bad recommendation system, and the important thing is to choose the right recommendation method in the right scene.

As can be seen from the above, memory-based methods are intuitive and highly interpretable models. Users can easily figure out why the memory-based system recommends these items to them (due to their similar users prefer this item or the item is similar to the one you have purchased or viewed). But on the other hand, the effect of memory-based methods to recommend popular products is more obvious. That is to say if a lot of people like *item1*, under memory-based methods, the probability of recommending it to you will be higher, although item *item2* also matches your interest, due to the rating for *item2* is sparse, so the similarity of item *item2* is greatly reduced when calculating the similarity.

So, is there a method that can handle sparse data with better generalization ability? Turning now to the methods to address the above question which so call model-based

methods.

2.2.2.1 Latent Factor Models

Model-based collaborative filtering has been studied for decades, and many methods have been proposed. Such as in [10], a multifaceted collaborative filtering model has been proposed or the probabilistic matrix factorization be described in [15]. In general, model-based collaborative filtering can produce a more accurate recommendation [10]. In this section, matrix factorization (MF) which is the most popular latent factor model will be illustrated.

The purpose of matrix factorization is to discover latent factor models of users and items in a shared latent space where the user-item relationship (e.g., user rating of the item) is calculated from the inner product of the user matrix and the item matrix.For a user-item rating matrix R consisting of m users and n items, where $R \in \mathbb{R}^{m \times n}$. Through matrix factorization, the $m \times n$ user-item matrix R be factorized into user latent matrix p ($m \times k$ matrix) and item latent matrix q ($k \times n$ matrix), where k is the dimension of the *latent factor*. In matrix factorization, the latent model of user u and item i are denote by p_u and q_i respectively, where $p_u \in \mathbb{R}^k$, $q_i \in \mathbb{R}^k$. The notation $r_{u,i}$ is used to represent the rating of user u on item i. The predicted rating of user u on item i can be described as :

$$\hat{r}_{u,i} = q_i^T p_u \tag{2.3}$$

A general approach to training the latent model is to minimize the gap between predicted rating $\hat{r}_{u,i}$ and real rating $r_{u,i}$ that can be express as $r_{u,i} - q_i^T p_u$, by adopting *root mean square error* (RMSE) as the loss function, we have:

$$\text{Loss} = \min_{q*,p*} \sum_{(u,i) \in K} \left(r_{u,i} - q_i^T p_u \right)^2 \tag{2.4}$$

where, K denote the set of (u,i) pairs. By adopting l_2 regularization to void over-fitting problem, we have:

$$\text{Loss} = \min_{q*,p*} \sum_{(u,i) \in K} \left(r_{u,i} - q_i^T p_u \right)^2 + \lambda \left(\|q_i\| + \|p_u\| \right)^2 \tag{2.5}$$

Chapter 3

Methodologies

In this chapter, we will discuss the techniques mentioned above in the manner that from the shallow to the deep. We will first discuss the methods individually, then we will diverge from a single method to a recommendation system that contains multiple methods, and finally, we will provide details of the proposed model, multi-armed bandit-based matrix factorization recommender system (BanditMF).

3.1   Matrix Factorization

As discussed above, to reduce the gap between predicted rating $\hat{r}_{u,i}$ and real rating $r_{u,i}$, *root mean square error* (RMSE) is used as the loss function and l_2 norm is adopt as the regularization term. Then when we already have the loss function, the next step is to minimize the loss function. Here we use the stochastic gradient descent (SGD) method to minimize the loss function. Next, the derivation of stochastic gradient descent will provide. By Taylor expansions, we have the following equation:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot f'(\theta_0) \tag{3.1}$$

where $(\theta - \theta_0)$ is a vector, it represents the distance of change, we can denote it by η. η is a scalar, and the unit vector of $(\theta - \theta_0)$ is expressed by v:

$$(\theta - \theta_0) = \eta \cdot v \tag{3.2}$$

By formula 3.2, the formula 3.1 can change as:

$$f(\theta) \approx f(\theta_0) + \eta v \cdot f'(\theta_0) \tag{3.3}$$

To minimize the loss function, we need minimize the $f(\theta)$. In another word, We want to make the value of the $f(\theta)$ smaller when each time we update the θ. So that $f(\theta) - f(\theta_0) < 0$, by this we have:

$$f(\theta) - f(\theta_0) \approx \eta v \cdot f'(\theta_0) < 0 \tag{3.4}$$

Where, η is scalar and generally positive, which can be ignored. Therefore:

$$v \cdot f'(\theta_0) < 0 \tag{3.5}$$

From vector multiplication that when the angle between two vectors is 180 degrees, the product of vector multiplication is less than 0, and minimal. So, when the direction of v and $f'(\theta_0)$ is opposite (i.e. included angle is 180 degrees), $v \cdot f'(\theta_0) < 0$ and have minimal value. By this we can get:

$$v = -\frac{f'(\theta)}{\|f'(\theta)\|} \tag{3.6}$$

where v is a unit vector, so we divide it by the norm of the $f'(\theta)$, substitute the formula 3.6 into formula 3.2:

$$\theta = \theta_0 - \eta \frac{f'(\theta)}{\|f'(\theta)\|} \tag{3.7}$$

Due to the norm of vector is a scalar, we obtain:

$$\theta = \theta_0 - \eta f'(\theta) \tag{3.8}$$

where θ_0 is is the current parameter, η denote the stride or so call learn rate, $f'(\theta)$ is the direction of the gradient. However, our loss function is the multivariate function, so we need calculate the partial derivatives of it. So the formula of θ become:

$$\theta = \theta_0 - \eta \frac{\partial}{\partial \theta_0} J(\theta) \tag{3.9}$$

For our loss function:

$$\text{Loss} : J(p, q) = \min_{q*,p*} \sum_{(u,i) \in K} \left(r_{u,i} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\| + \|p_u\|\right)^2 \tag{3.10}$$

Where q_i and p_u are the variables which need to optimize, λ is the regularization factor. Our goal is to optimize the q_i and p_u and find the minimum value of the loss function. Perform partial derivative on both q_i and p_u:

$$\frac{\partial E}{\partial q_i} = -2 \left(r_{u,i} - q_i^T p_u\right) p_u + 2\lambda q_i \tag{3.11}$$

$$\frac{\partial E}{\partial p_u} = -2 \left(r_{u,i} - q_i^T p_u\right) q_i + 2\lambda p_u \tag{3.12}$$

Substituted two partial derivatives into the gradient descent formula 3.9:

$$q_i = q_i + 2\eta \left(\left(r_{u,i} - q_i^T p_u\right) p_u - \lambda q_i\right) \tag{3.13}$$

$$p_u = p_u + 2\eta \left(\left(r_{u,i} - q_i^T p_u\right) q_i - \lambda p_u\right) \tag{3.14}$$

By define $e_{u,i} = r_{u,i} - q_i^T p_u$. we have:

$$q_i \leftarrow q_i + \eta \left(e_{u,i} p_u - \lambda q_i\right) \tag{3.15}$$

$$p_u \leftarrow p_u + \eta \left(e_{u,i} q_i - \lambda p_u\right) \tag{3.16}$$

To sum up, the above process is the derivation process of stochastic gradient descent.By stochastic gradient descent algorithm, we train two latent vector matrices: user matrix p and item matrix q. Through the inner product of user matrix p and item matrix q, we can have the predicted user-item matrix. If the user matrix and item matrix fitted by the loss function perfectly, then our predicted rating will be infinitely close to the real rating.

In this project, we consider another scenario that is some harsh users give a low rating but some users are more tolerant than giving a relevant high score for bad items. By this intuition, we trying to add bias into matrix factorization.

$$\hat{r}_{u,i} = \underbrace{q_i^T p_u}_{basic} + \underbrace{b_{ui}}_{bias} \tag{3.17}$$

So that our predicted rating become:

$$\hat{r}_{u,i} = b_{u,i} + q_i^T * p_u \tag{3.18}$$

And the loss function is:

$$\text{Loss} = \min_{q^*,p^*} \sum_{(u,i) \in K} \left(r_{u,i} - b_{u,i} - q_i^T p_u \right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2 \right) \tag{3.19}$$

where $b_{u,i} = \mu + b_u + b_i$. we use μ denote the average rating for the entire rating matrix, b_u denote the user's bias, and b_i is represent the item bias. In the next chapter we will simulate this two approaches with real data and evaluate the performance between two approaches .

3.2 Memory-Based Collaborative Filtering Recommender System

As was pointed out in the formulation of this report, memory-based collaborative filtering makes recommendations based on the similarity between the users or the items. The detailed step of the memory-based collaborative filtering can be stated as follows:

1. Identify a target user who will be recommended.

2. Figure out the items that the target user has given ratings for.

3. Based on the target user's rating pattern, find similar users.

4. Get the rating records of similar users for items.

5. Calculate the similarity between the target user and similar user by formula (e.g. *Euclidean Distance, Pearson Correlation, Cosine Similarity*).

6. Recommend the items with the highest score to the target user.

In the following, an example will be demonstrated to elaborate on the above steps.

As the user-item rating matrix shown in Figure 3.1, assume *user4* is our target user and rated three out of five items. The goal of the system is to figure out which one of the

Figure 3.1: Example of User−item rating matrix

two unrated items (i.e. *item1* and *item5*) should recommend to the target user that can probably get a high rating. After we defined our target user and figure out some users, the next step that comes up is to calculate the similarity between the target user and similar users. The common methods of similarity calculation include *Euclidean Distance*, *Pearson Correlation*, *Cosine Similarity*. To calculate the similarity between two users, we base on the rating of *item2*, *item3*, and *item4* which are the items that all users have been rated. Here we make a hypothesis that the similarity between the target user and other

Figure 3.2: Similarity between target user and other users

users are 0.4, 0.9, and 0.6 respectively, as shown in Figure 3.2. The next step is that base on the value of similarity, we can calculate the predicted rating for the target user. What can be seen in Figure 3.3 is the so call weighted rating matrix because it provides more weight to those users who have higher similarity to the target user. For the final step, we
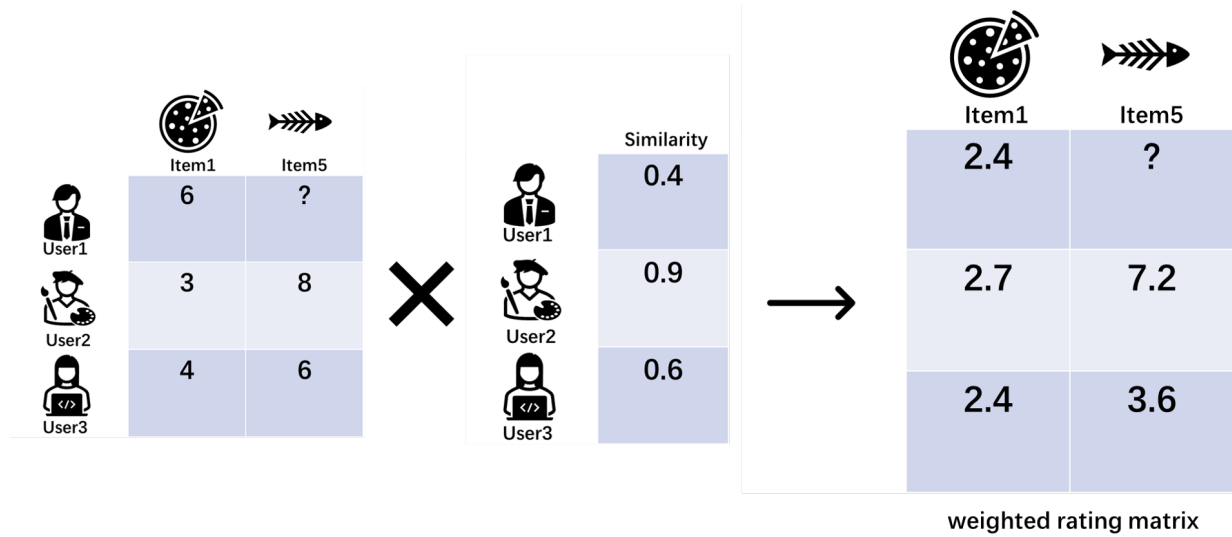
Figure 3.3: Weighted rating matrix

summing up all weighted rates to create the recommender matrix. Due to the sparsity of the weighted rating matrix (e.g. only *user2* and *user3* provided the rating for *item5* , we normalize the value of weighted rating by dividing the sum of weighted ratings by the sum of the similarity for users. According to the result of user-based collaborative filtering
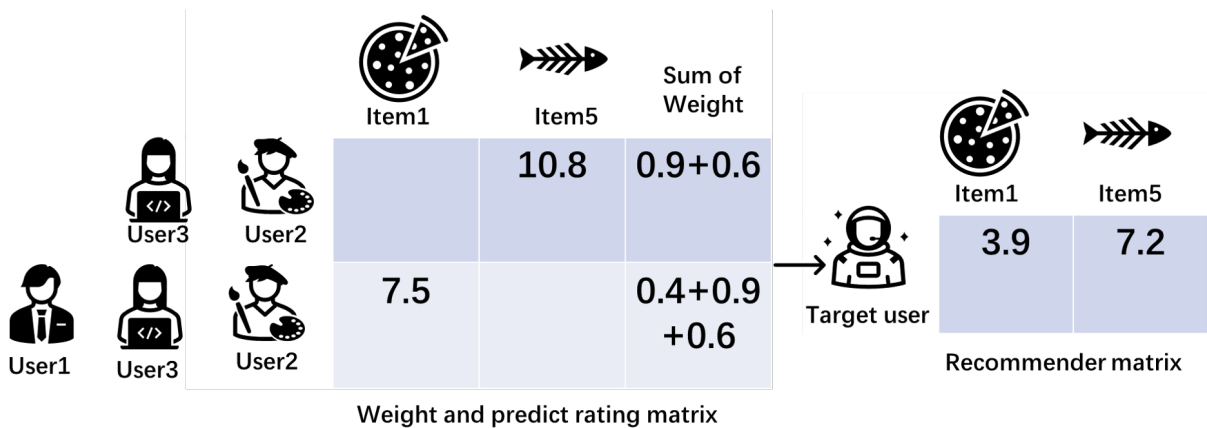
Figure 3.4: Recommender matrix

shown in Figure 3.4, the value in the recommender matrix described the potential rating for the target user that will be arising for the items based on the similarity between it and other users. The target user potentially will provide higher rating to *item5* than *item1*.

25

So that through the result, the user-based collaborative filtering recommender system will choose *item5* to recommend to the target user.

3.3   Hybrid Recommender System: Combination of Content-Based Method and Matrix Factorization

In the above paper, matrix factorization has been discussed. For the content-based method, the concept is straightforward which is recommended items that similar to what the target user liked before. Now, the intuition is whether we can combine the two approaches into a hybrid recommender system. Recall that in matrix factorization, by stochastic gradient descent algorithm, we train two latent vector matrices: user matrix p and item matrix q. Through the inner product of user matrix p and item matrix q, we can have the predicted user-item matrix. In the predicted user-item matrix, row vectors represent the preferences of each user, and column vectors represent the attributes of each item which means the user similarity can be obtained by calculating the similarity between row vectors, and similarly the item similarity can be obtained by calculating the similarity of column vectors.

For the proposed hybrid system, the content-based approach is performed base on the matrix factorization (i.e. content-based approach calculate the similarity through the user-item rating matrix derive by the result of matrix factorization). Consider that the detailed methodology of matrix factorization is already stated in section 3.1, so there we will illustrate the methodology after obtaining the user-item rating matrix which derives from the result of matrix factorization.

Assume, the predicted user-item rating matrix is available, to recommend items to the target user, we have the following steps:

1. List all the item indexes rated by the target user.

2. Sort the items that have not been yet rated by the target user (i.e. the item indexes not list in the *step1*) through our predicted rating score.

3. Get the top rank item of the list created by *step2*.

Then we use the *cosine similarity* for similarity measurement to measure how similar the

items are in attractiveness to the target user, where:

$$\text{similar}(i,j) = \cos(i,j) = \frac{i \cdot j}{\|i\| \cdot \|j\|} \tag{3.20}$$

The formula uses the angle between two vectors to calculate the cosine value. The angle between the vector i and j is the thing this formula cares about, instead of the magnitude. So that the value of similarity is within the interval [-1,1], and 1 meaning "exactly the same". After calculating the cosine similarity, sort the items by the value of similarity and then we can get the items that are most similar to the target user's favorite items. The implementation of this hybrid recommender system with real-world data will be present in the next chapter.

For this hybrid recommendation system, it is foreseeable that if it is combined memory-based collaborative filtering with a content-based approach, it can solve the problems faced by memory-based collaborative filtering, such as sparsity and loss of information. Similarly, as can be seen from our method description, hybrid systems increased computational steps and complexity.

3.4 BanditMF: Multi-Armed Bandit Based Matrix Factorization Recommender System

The authors in [4] proposed a method that based on implicit and explicit user feedback, agents use machine learning algorithms to inscribe an individual user model and use the user model to make recommendations. Inspired by this method, the method called BanditMF, a matrix factorization recommendation system based on multi-arm bandit, is proposed to address the limitations of collaborative filtering and bandit algorithms, as mentioned earlier. The system combines model-based collaborative filtering matrix factorization (MF) with a multi-armed bandit algorithm. BanditMF incorporates an offline subsystem centered on matrix factorization and an online subsystem focused on a multi-armed bandit algorithm.

	i_1	i_2	i_5	i_4	i_5
u_1	1.4	?	1.1	0.7	?
u_2	?	0.3	?	0.7	0.5
u_3	0.4	0.3	?	?	0.3
u_4	1.4	?	1.2	?	0.8

Table 3.1: User-item rating matrix

	i_1	i_2	i_5	i_4	i_5
u_1	1.4	0.8	1.1	0.7	0.9
u_2	1.0	0.3	1.0	0.7	0.5
u_3	0.4	0.3	0.3	0.1	0.3
u_4	1.4	0.7	1.2	0.8	0.8

Table 3.2: Predicted rating matrix

As the user-item rating matrix is shown in Table 3.1, by applying the matrix factorization, we can have the predicted user-item rating matrix as shown in Table 3.2. For the sake of explanation, we consider each row in Table 3.2 as a predictive model, in another world, consider the row vectors which represent the preferences of each user as a model. When a new user enters the system, a predictive model is selected and used as a basis to make recommendations for the new user. However, in the real world, the number of users is generally in the millions, so the above approach is rendered impractical due to the $m \times n$ scaling rating matrix, where the m is millions of orders in size.

Whether we can cluster the rows in the matrix to reduce the size of the rating matrix(i.e. reduce the number of predictive models)? Inspired by collaborative filtering, it is possible to cluster the rows of the rating matrix: each row of the matrix represents a row vector, and each row vector represents the preferences of each user so that each cluster is composed of users with similar rating preferences. Another important reason for matrix factorization is the very widespread assumption that the user-item rating matrix is a low-rank matrix, So the clustering can be performed on the rows of the user-item rating matrix. We treat BanditMF as an algorithm consisting of two stages: (1) prediction,(2) recommend.

For the offline subsystem, we define the prediction model as follow steps: matrix factorization based rating prediction, clustering, and unified rating

Matrix factorization based rating prediction. As we illustrated in section 3.1, the non-cold user formed the user-item rating matrix R and based on the matrix factorization, we can get the predicted user-item rating matrix $\hat{R}$. The notation $\hat{r}_{u,i}$ is used to represent the rating of user u on item i. Recall that in this project our predicted rating is:

$$\hat{\boldsymbol{r}}_{u,i} = b_{u,i} + q_i^T * p_u \tag{3.21}$$
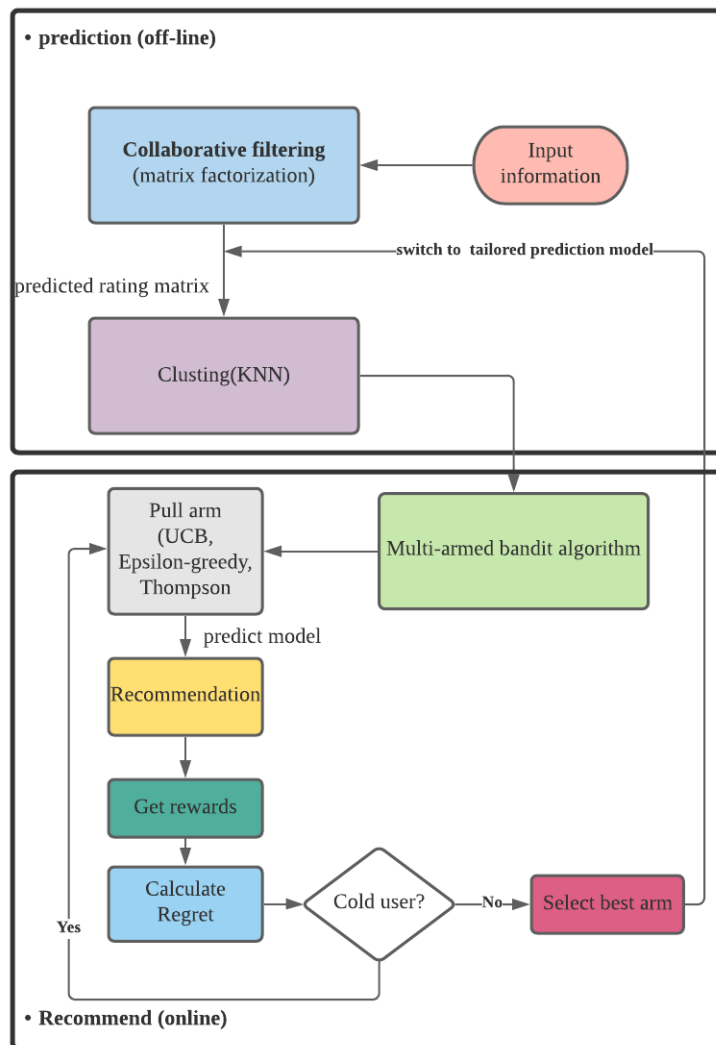
Figure 3.5: Flow chart of the BanditMF

And the loss function is:

$$\text{Loss} = \min_{q^*, p^*} \sum_{(u,i) \in K} \left(r_{u,i} - b_{u,i} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2\right) \tag{3.22}$$

where $b_{u,i} = \mu + b_u + b_i$. we use μ denote the average rating for the entire rating matrix, b_u denote the user's bias, and b_i is represent the item bias.

Clustering. According to the user's preference vectors of the predicted user-item rating matrix $\hat{R}$, we can perform the clustering on the row of the $\hat{R}$. In this system, *K-means* algorithm is adopted which measured by *Euclidean distance*. After performing the clustering on user's preference vectors, we will have a set of clusters , where $\mathcal{C} = \{c_1, c_2 \cdots c_n\}$.

Unified rating. For the clusters in $\mathcal{C}$, we perform the average rating on it to get the unified user preference vectors, where $\hat{\alpha}$ denote the unified user preference vectors. For example, as shown in Table 3.3, by hypothetical that u_1 and u_2 be clustered into c_1, so the unified user preference vectors $\hat{\alpha} = \{1.2, 0.55, 1.05, 0.7, 0.7\}$, which is the average of the u_1, u_2.

	i_1	i_2	i_5	i_4	i_5
u_1	1.4	0.8	1.1	0.7	0.9
u_1	1.0	0.3	1.0	0.7	0.5
$\hat{\alpha}_1$	1.2	0.55	1.05	0.7	0.7

Table 3.3: Unified user preference vectors

The online recommendation module is mainly composed of multi-armed bandit and this online module run as follows:

1. Select a predictive model $\mathcal{S}_n$ using the MAB algorithm π

2. Select and recommend the item i^* with the top rating score in $\hat{\alpha}$ which do not recommended to user u.

3. Receive the user u's rating for item i^*.

4. calculate the reward and regret.

5. Determine whether a user is a cold user by a threshold τ.

6. If yes, update the algorithm π. Otherwise, turn to the offline module.

The online recommendation module will provide recommendations to new users until the new user is no longer cold (i.e. some feedback on the recommended items or preferences has been provided to the system). Nevertheless, if a system obliges the user to make excessive ratings, the user will lose interest in the system [6]. For the above reasons, we cannot run the bandit algorithm too many times to get the optimal recommendation and online systems need to get user preferences in a limited number of recommendations(i.e. within threshold τ).

Now let's walk through the entire BanditFM workflow from scratch which is the flowchart shown in Figure 3.5. The input information is first fed into the offline prediction module, and through matrix factorization, we get the predicted user-item rating matrix. Then each row of the matrix, namely the user preference vector, is clustered to reduce the size of the prediction matrix. If we denote the original size predicted user-item rating matrix as $m \times n$, after clustering, we have a $m' \times n$ matrix, where $m' \ll m$. Then passing the clustered predictive model to the online recommendation module. Through the multi-armed bandit algorithm, it selects the predictive model from the clusters. Get user preferences through interaction with users (i.e. algorithm recommends items to users and users provide feedback). The online module determines whether a user is a cold user through a threshold τ, and if the system has collected certain user rating preferences, then the user is no longer a cold user. At this point, we switch to the offline prediction module and append the user preferences obtained from the online module to the user-item rating matrix, then we can accurately make recommendations to new users through the matrix factorization technique.

Reward and Regret. In BanditMF, we define the *reward* as the equation shown in the formula 3.23.

$$\mu_{\mathcal{S}_n}(t) = \frac{r_{u,i}}{r^*} \tag{3.23}$$

where r^* denote the maximum rating in the dataset. $r_{u,i}$ denote the rating of user u for recommended item i according to the algorithm selected predictive model $\mathcal{S}_n$. The reward interval is $\mu_{\mathcal{S}_n}(t) \in [0,1]$.This intuition of this reward can be interpreted as the higher the feedback received for the recommendation the higher the reward will receive. By the

reward, then we define the *regret* as:

$$R(T) \triangleq \sum_{t=1}^{T} \left(\mathbb{E}\left[\mu^*\right] - \mathbb{E}\left[\mu_{\mathcal{S}_n}\left(t\right)\right] \right) \tag{3.24}$$

where $\mathbb{E}\left[\mu^*\right]$ denote the maximum expected cumulative rewards.

In the next chapter, we will verify the BanditMF using various criteria and compare the performance of different multi-armed bandit algorithms.

Chapter 4

Experiments and Implementation

In this chapter, the methods mentioned in the previous chapters will be implemented and validated with different real-world datasets.

4.1 Contextual Bandit: LinUCB

We introduced the context-free algorithm in detail last semester, so we will not continue to elaborate on it in this report. Instead, we will focus on the contextual bandit, implementing and verifying one of the very famous algorithms – LinUCB which was proposed by the authors in [12]. LinUCB is an extension of the UCB algorithm, the algorithm chooses an arm based on observation of the current user and set of arms together with the context feature. Also, in the LinUCB algorithm there exists a factor α, which is used to control the trade-off between exploration and exploitation.

Dataset. For this experiment, the personalization dataset[3] has been used. The Figure 4.2 demonstrates the format of the dataset. This dataset consists of a matrix of size 10000×102. The data in the first column indicate the action has been chosen. The second column is value indicate the obtained reward, where y_t denote the real reward that was obtained at time t and $y_t \in \mathbb{B}$. From column 3 to column 103 are the context which is represented as the feature vector $\mathbf{x_t} \in \mathbb{R}^{100}$.

Compute click-through rate (CTR). The click-through rate (CTR) proposed in [13] is used to evaluate the performance of the algorithm. The click-through rate replay

---

[3]Dataset from: http://www.cs.columbia.edu/ jebara/6998/dataset.txt

Algorithm 1 LinUCB with disjoint linear models.

0: Inputs: $\alpha \in \mathbb{R}_+$
1: **for** $t = 1, 2, 3, \ldots, T$ **do**
2:     Observe features of all arms $a \in \mathcal{A}_t$: $\mathbf{x}_{t,a} \in \mathbb{R}^d$
3:     **for all** $a \in \mathcal{A}_t$ **do**
4:         **if** a is new **then**
5:             $\mathbf{A}_a \leftarrow \mathbf{I}_d$ (d-dimensional identity matrix)
6:             $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ (d-dimensional zero vector)
7:         **end if**
8:         $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\mathbf{b}_a$
9:         $p_{t,a} \leftarrow \hat{\boldsymbol{\theta}}_a^\top \mathbf{x}_{t,a} + \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}}$
10:     **end for**
11:     Choose arm $a_t = \arg\max_{a\in\mathcal{A}_t} p_{t,a}$ with ties broken arbitrarily, and observe a real-valued payoff r_t
12:     $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
13:     $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
14: **end for**

Figure 4.1: LinUCB with disjoint linear models [12]

Figure 4.2: Format of the personalization dataset

at time T is:

$$C(T) = \frac{\sum_{t=1}^{T} y_t \times \mathbf{1}\left[\pi_{t-1}\left(\mathbf{x}_t\right) = a_t\right]}{\sum_{t=1}^{T} \mathbf{1}\left[\pi_{t-1}\left(\mathbf{x}_t\right) = a_t\right]} \tag{4.1}$$

where, π_{t-1} denote the algorithm trained on data up to time $t - 1$. At time t, if the context $\mathbf{x}_t$ been observed by algorithm π_{t-1}, then we use the notation $\pi_{t-1}(\mathbf{x}_t)$ denote the chosen action. a_t and y_t denote the real action that was taken in the dataset at time t and the real reward that was obtained at time t respectively.

Experiment. In this experiment, the size of the context is 100, where $\mathbf{x_t} \in \mathbb{R}^{100}$, the number of arm $K = 10$, total iteration $T = 10000$. For the α, as the only input in the algorithm, optimizing this parameter may result in higher total payoffs in practice. So, We will experiment with the blow values of α:

1. For Alpha $= 1$

2. For Alpha $= 0.25$

3. For Alpha $= 1/\sqrt{t}$

4. For Alpha $= 0.001$

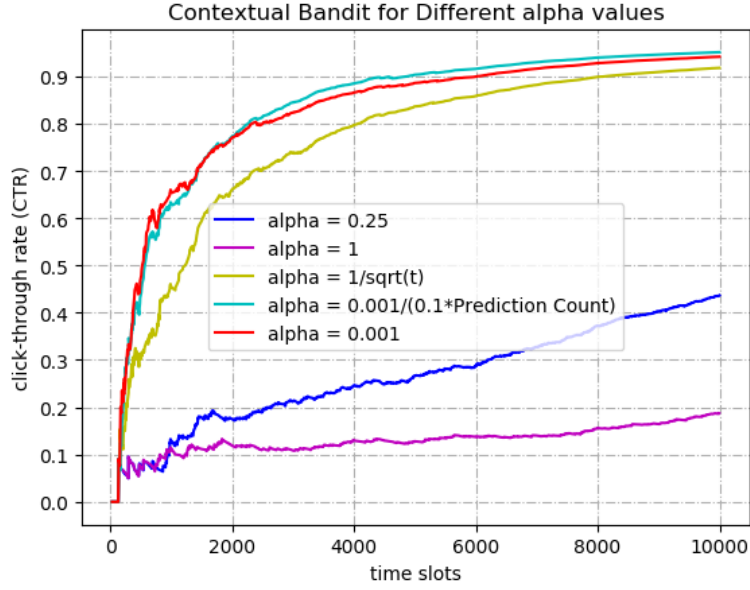5. For Alpha $= 0.001/(0.1 * PredictionCount)$

Figure 4.3: CTR of LinUCB for varying Alphas

The click-through rate (CTR) proposed in A is used to evaluate the performance of the algorithm. As it is evident from Figure 4.3, the best CTR value was achieved with alpha equal to $0.001/(0.1 * PredictionCount)$ when the CTR value is 0.95, where the prediction count is the number of correct predictions.

To understand the trade-off between exploration and exploitation, for each alpha value, first, the mean value of the upper confidence bound for 10 arms will be plotted. We will then compare the number of predictions made by each arm with the number of true predictions it made. .

As it is evident from Figure 4.4, when the α value is 1, the exploration and exploitation here is at a minimum, which can be represented by strips with nearly the same length, that implies that the upper confidence bound of each arm is very approximate. For this cause, we can also observe that the predicted counts are nearly the same for all arms,thus when the α value equals to 1, the algorithm giving the lowest click rate value which is 0.18 as shown in Figure 4.3. For Figure 4.5, When the α value change to 0.25, the improvement is not significant compare to the result when α value equals to 1 . Now, when we change our α to the time-dependent values (i.e square root of the time), as shown in Figure 4.6, the result receives a significant improvement. This is mainly caused by the fact that we adjust our exploration according to the changes of time and limit our predictions in
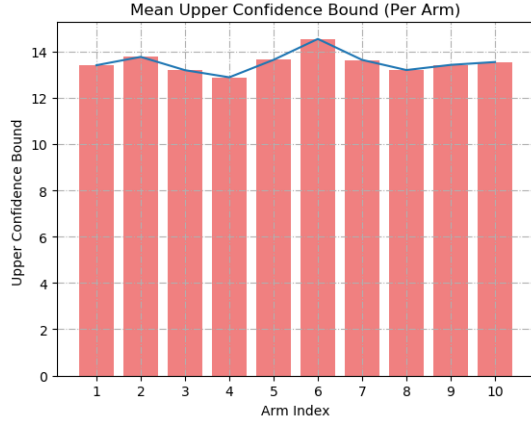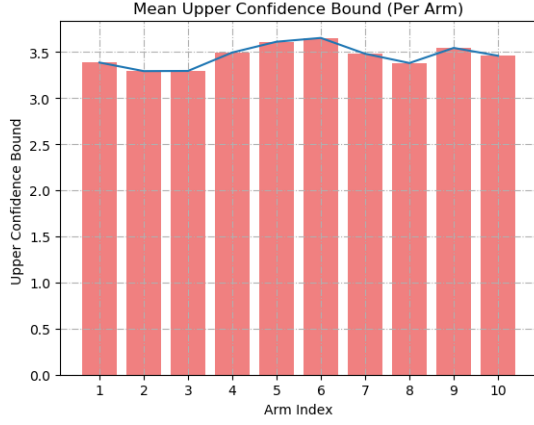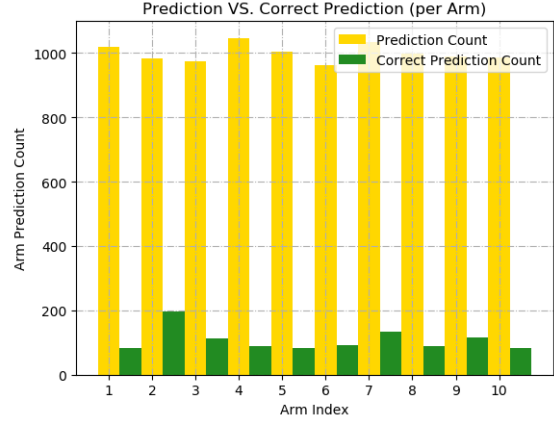
(a) MeanUCB
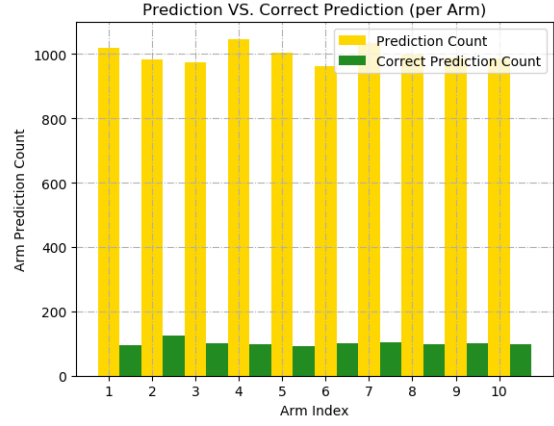
(b) Prediction comparison

Figure 4.4: Mean UCB and Total VS. Correct prediction for Alpha = 1

(a) MeanUCB

(b) Prediction comparison

Figure 4.5: Mean UCB and Total VS. Correct prediction for Alpha = 0.25
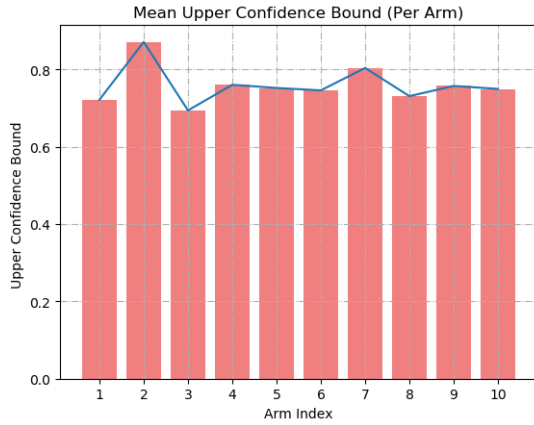
(a) MeanUCB
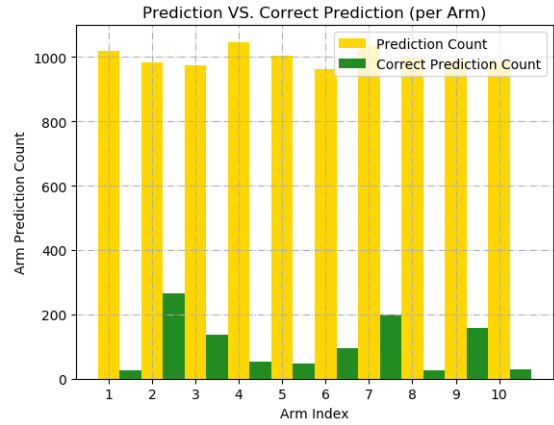
(b) Prediction comparison

Figure 4.6: Mean UCB and Total VS. Correct prediction for Alpha = $1/\sqrt{t}$

order to maximize the use of our trained algorithms over time. Figure 4.7 shows that

(a) MeanUCB

(b) correct prediction comparison
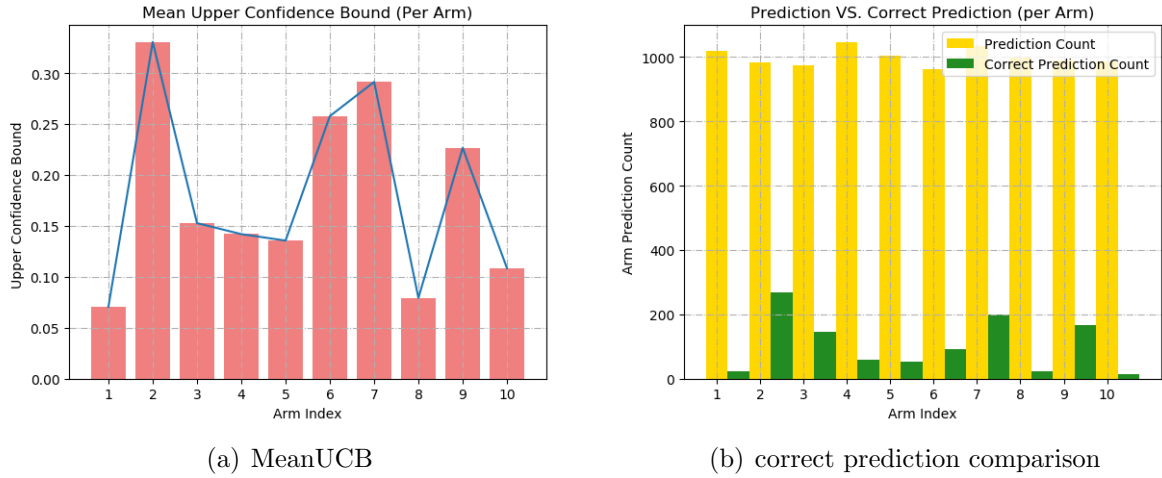
Figure 4.7: Mean UCB and Total VS. Correct prediction for Alpha = 0.001

when $\alpha = 0.001$, a higher CTR can get. Because of the extremely small value of α, the algorithm will perform exploiting most of the iteration and limiting the exploration. In

(a) MeanUCB

(b) Prediction comparison
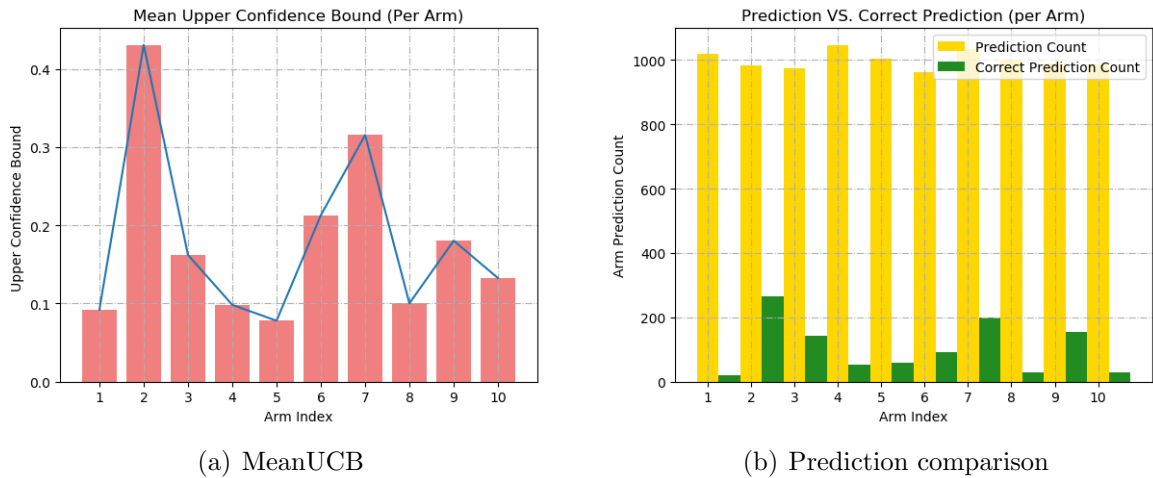
Figure 4.8: Mean UCB and Total VS. Correct prediction for Alpha = $0.001/(\frac{correct-predictions}{10})$

the experiment, we further perform an α value with correct prediction dependent which is the Figure 4.8. The intuition for doing so is that the alpha value related to the correct prediction increases the exploitation of the arm that performs better (i.e. the larger the correct prediction the smaller the α value). On the other hand, the number of explorations for the arm with inaccurate predictions is increased.

From the above figures, we can see that the arm with the largest upper confidence

```
array([[4. , 4. , 4.5, 0. , 4. , 3. , 0. , 0. , 0. , 4.5, 4. , 0. , 0. ,
        4. , 5. , 5. , 5. , 0. , 0. , 0. , 0. , 5. , 4. , 3.5, 0. , 0. ,
        0. , 3. , 0. , 2.5, 4. , 0. , 0. , 3. , 3.5, 3.5, 3. , 2.5, 0. ,
        3. , 3.5, 4.5, 4. , 4. , 0. , 0. , 3.5, 2. , 3. , 3. , 5. , 2.5,
        4. , 0. , 4. , 0. , 0. , 4. , 4.5, 5. , 4. , 5. , 0. , 2.5, 0. ,
        5. , 3.5, 3. , 3.5, 3.5, 3. , 0. , 0. , 3. , 5. , 3.5, 2. , 4.5,
        4. , 3.5, 0. , 0. , 0. , 3.5, 4. , 2.5, 3. , 0. , 0. , 0. , 1. ,
        4. , 0. , 0. , 0. , 3. , 4. , 4. , 3.5, 3. ],
       [4.5, 3.5, 3.5, 3.5, 4. , 4. , 0. , 4. , 4. , 5. , 5. , 3.5, 0. ,
        4. , 3.5, 4. , 5. , 0. , 0. , 3.5, 4. , 5. , 3.5, 4. , 1.5, 3. ,
        4.5, 0. , 0. , 2.5, 4. , 1.5, 0. , 3. , 2.5, 3. , 3. , 1.5, 0. ,
        0. , 0. , 0. , 3. , 3.5, 4. , 4. , 4. , 3. , 3.5, 3. , 4. , 2. ,
        4. , 0. , 0. , 0. , 4.5, 3.5, 3. , 3. , 4. , 0. , 0. , 2.5, 0. ,
        3.5, 1.5, 2.5, 2.5, 0. , 4. , 4. , 3. , 0. , 5. , 0. , 4.5, 5. ,
        3.5, 1.5, 0. , 2.5, 4. , 3.5, 3.5, 1. , 1. , 4. , 0. , 0. , 0. ,
        4. , 3. , 0. , 3.5, 5. , 4. , 3. , 2. , 4. ],
       [0. , 0. , 3.5, 0. , 0. , 5. , 5. , 4. , 4.5, 4. , 3. , 0. , 0. ,
        0. , 5. , 5. , 4. , 0. , 4. , 0. , 4.5, 5. , 3.5, 3.5, 0. , 4.5,
```

Figure 4.9: Partial of the user-item rating matrix R

bound is also the arm with the highest number of predictions. This reflects the nature of the LinUCB algorithm that selects the arm with the highest upper confidence bound as the prediction.

4.2   Matrix Factorization Implementation

To implement the matrix factorization mentioned in section 3.1, a smaller data set was chosen in order for us to carry out a comparison between the two approaches: base method and bias method (i.e. the method of matrix factorization that considers both user bias and item bias).

Dataset. The partial of the dataset is shown in Figure 4.9. The user-item rating matrix R is a 25×100 matrix, which represent 25 users and 100 items. Furthermore, we can see from the figure that R is a sparse matrix (i.e. matrix with many zeros)

Training. Recall our loss function:

$$\text{Loss} : J(p, q) = \min_{q*, p*} \sum_{(u,i) \in K} \left(r_{u,i} - q_i^T p_u \right)^2 + \lambda \left(\|q_i\| + \|p_u\| \right)^2 \tag{4.2}$$

Based on the methods proposed in [11], We use stochastic gradient descent (SGD) algorithm to train two latent vector matrices: user latent matrix p and item latent matrix q.By define $e_{u,i} = r_{u,i} - q_i^T p_u$. We update the two matrices by following formulas:

$$q_i \leftarrow q_i + \eta \left(e_{u,i} p_u - \lambda q_i \right) \tag{4.3}$$

$$p_u \leftarrow p_u + \eta \left(e_{u,i} q_i - \lambda p_u \right) \tag{4.4}$$

In the experiment, we set the number of latent features $k = 2$, then we initiate the user matrix and item matrix with size 25×2 (i.e number of users $\times$ number of latent features). Further, the learn rate$\eta = 0.001$, regularization factor$\lambda = 0.1$ and iteration 1000 times. After iteration, we can get our user latent matrix p and item latent matrix q, as shown

```
 1  -0.64 1.78          1  -0.32 2.10
 2  -0.25 1.78          2  -0.21 2.08
 3  -1.04 1.57          3  -0.88 2.04
 4  -0.89 1.30          4  -1.14 1.69
 5  -0.48 2.27          5  -0.03 1.91
 6  -0.75 1.59          6  -0.58 1.90
 7  -0.64 2.02          7  -1.86 1.46
 8  -0.24 1.94          8  -1.18 1.45
 9  -0.50 1.79          9  -1.31 1.88
10  -2.00 0.95         10  -0.90 1.99
11  -0.77 2.23         11  -0.37 2.01
12  -0.73 1.84         12  -0.86 2.04
13  -0.63 1.77         13  -0.55 1.87
14   0.85 2.17         14  -0.63 1.63
15  -0.51 1.79         15  -0.56 1.94
16  -0.62 1.82         16  -0.07 2.13
17  -0.44 1.78         17  -0.45 2.01
18  -0.76 2.03         18  -1.00 1.51
19  -0.80 2.02         19  -0.92 1.57
20  -0.58 1.90         20  -0.01 1.85
21  -0.11 2.08         21  -0.27 1.98
22  -0.71 1.84         22  -0.97 2.06
23  -0.62 1.99         23  -0.56 2.01
24   0.03 1.81         24  -0.45 1.92
25  -0.93 1.88         25  -1.23 1.37
```

(a) User latent matrix   (b) Item latent matrix (first 25 out of 100 items)

Figure 4.10: User and item matrix

in figure 4.10. Then, by performing the inner product between these two latent matrices, we get the predicted user-item rating matrix $\hat{R}$, shown at Figure 4.11. Expressed by the formula: $\hat{R} =$ user latent matrix $\times$ item latent matrix$^{\mathbf{T}}$.

```
1  3.93 3.82 4.17 3.73 3.42 3.74 3.77 3.33 4.17 4.10 3.80 4.18 3.67 3.30 3.80 3.84 3.86 3.32 3.38 3.29 3.69 4.28 3.93 3.69 3.21
   3.83 3.46 4.17 3.00 3.41 3.92 4.02 4.43 3.70 3.43 3.88 3.73 2.97 4.02 3.78 3.17 4.08 3.20 3.04 3.86 3.37 3.80 3.68 3.42 3.28
   4.15 3.40 3.38 3.94 3.89 3.89 2.98 3.23 2.99 3.91 3.77 3.52 3.22 3.11 3.96 3.52 3.40 3.19 2.98 3.74 3.87 3.41 3.37 4.07 3.81
   3.17 3.05 3.91 3.70 3.39 3.27 2.96 3.32 3.52 3.61 2.57 2.83 3.13 2.79 3.80 2.62 3.78 2.64 2.99 3.42 3.54 3.32 3.80 3.63 2.93
2  3.82 3.75 3.84 4.30 3.41 3.52 3.06 2.88 3.67 3.76 3.67 3.85 3.47 3.07 3.60 3.82 3.70 2.94 3.03 3.29 3.60 3.91 3.72 3.53 2.74
   3.28 3.48 3.93 2.77 3.20 3.60 3.62 4.29 3.29 3.21 3.71 3.71 2.75 3.88 3.65 2.85 3.77 3.28 3.09 3.88 3.45 3.33 3.34 3.21 3.27
   3.78 3.32 2.92 3.49 3.67 3.61 3.05 3.34 3.23 3.53 3.41 3.02 3.25 2.99 3.28 3.51 3.38 2.86 2.70 3.73 3.55 3.38 3.34 3.80 3.29
   2.85 3.15 3.93 3.47 3.18 3.13 2.73 2.98 3.43 3.69 2.48 2.46 2.78 2.53 3.51 2.65 3.62 2.69 2.86 3.38 3.67 3.06 3.65 3.42 2.76
3  3.64 3.49 4.12 3.86 3.04 3.59 4.23 3.52 4.32 4.07 3.55 4.12 3.52 3.23 3.64 3.44 3.64 3.42 3.43 2.92 3.40 4.26 3.75 3.49 3.43
   4.10 3.04 4.00 2.94 3.28 3.88 4.07 4.12 3.80 3.32 3.66 6.34 2.92 3.75 3.52 3.20 4.02 2.74 2.64 3.41 2.89 3.97 3.70 3.30 2.93
   4.15 3.12 3.58 4.06 3.74 3.81 2.57 2.73 2.36 3.96 3.81 3.75 2.82 2.91 4.37 3.15 3.05 3.25 2.99 3.34 3.84 3.06 3.02 3.95 4.03
   3.23 2.59 3.45 3.56 3.27 3.09 2.92 3.38 3.24 3.12 2.40 2.97 3.21 2.81 3.74 2.29 3.56 2.29 2.81 3.08 3.00 3.29 3.57 3.50 2.83
```

Figure 4.11: Predicated rating matrix $\hat{R}$ (Partial,first 3 out of 100 user's estimate rating)

Adding bias.Now, we will experiment the bias method which considers both user bias and item bias. As mentioned before, when we consider the bias then the loss function

becomes:

$$\text{Loss} = \min_{q^*,p^*} \sum_{(u,i)\in K} \left(r_{u,i} - b_{u,i} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2\right) \qquad (4.5)$$

The formula used to iterate over the user latent matrix and the item latent matrix becomes:

$$q_i \leftarrow q_i + \eta \left(\left(r_{ui} - q_i^T p_u - b_{u,i}\right) p_u - \lambda q_i\right) \qquad (4.6)$$

$$p_u \leftarrow p_u + \eta \left(\left(r_{ui} - q_i^T p_u - b_{u,i}\right) q_i - \lambda p_u\right) \qquad (4.7)$$

The rest of the steps are consistent with the base matrix factorization method.After the training, we get the user and item latent matrix base on the bias method. By visualization method, Figure 4.12 plot the latent vectors of users and items in the two-dimensional coordinate. Through multiply the red dot coordinate by the blue cross's coordinate in the figure, you can get the predicted rating score of user u on item i.
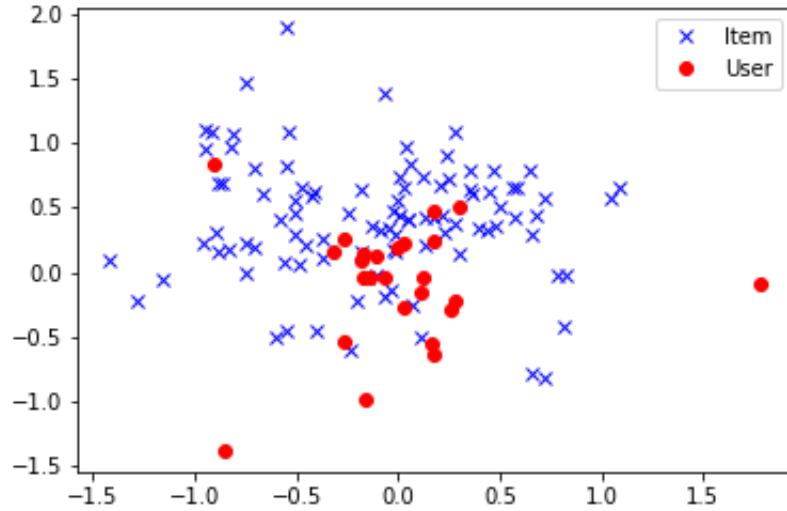
Figure 4.12: visualization of user and item latent vector

Evaluation. To compare the performance between the base method and bias method, Mean Square Error (MSE) is used to evaluate these two algorithms. In order not to lose generality, we conducted 10 experiments and took the average of the results. What is striking in Table 4.1 is the dramatic decline of MSE, when we consider user bias and item bias, the value of MSE is reduced by 20% which proved that consider both user and item bias when perform the matrix factorization is effective.

40

	MSE	Improvement rate
Base method $\hat{r}_{u,i} = q_i^T p_u$	0.6530	
Bias method $\hat{\boldsymbol{r}}_{u,i} = b_{u,i} + q_i^T p_u$	0.5178	20.715%

Table 4.1: Evaluation between base method and bias method

4.3 Collaborative Filtering Recommender System: User-Based

In this chapter, we will demonstrate the implementation of a user-based recommender system.

Dataset. In this system, we adopt the dataset called MovieLen [7]. MovieLen contains the 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

Implementation. By *pandas* library, we read movies and ratings into the data frame, the format of both movies and rating s data frames are shown in Figure 4.13.

(a) movies data-frame

(b) ratings data-frame

Figure 4.13: Format of data frame

According to the process for creating the memory-based collaborative filtering, the next step is to identify a target user who will be recommended.

Identify a target user. We define a target user and their rating in the way shown in the frame below. As in the image shown in Figure 4.14, the target user data frame involved the movie title with its ID and target user's rating.

Target user input

---

targetUserInput = [

'title':'Grumpier Old Men', 'rating':4,

'title':'Get Shorty', 'rating':2.5,

'title':'Pulp Fiction', 'rating':3,

'title':"Heat", 'rating':3.5,

'title':'Toy Story', 'rating':5

]

targetUserMovies = pd.DataFrame(targetUserInput)

targetUserMovies

	movieId	title	rating
0	1	Toy Story	5.0
1	3	Grumpier Old Men	4.0
2	6	Heat	3.5
3	21	Get Shorty	2.5
4	296	Pulp Fiction	3.0

Figure 4.14: Target user data frame

Find similar user. With the movie ID in the target user input, we can now figure out the users who have seen and rated the movie which same as the target user input. Then according to the user ID, we group up the rows of the similar user set. We sort these groups, with the rule that the group with more number of the movies watched or rated the same as the target user will be given a higher priority. With the information contained in Figure 4.15, the group with user ID 32, 91, and 414 are the groups that have the top 3 highest number of movies watched or rated the same as the target user.

```
[(32,       userId  movieId  rating
 4929        32        1      3.0
 4930        32        3      3.0
 4931        32        6      3.0
 4933        32       21      4.0
 4961        32      296      5.0), (91,       userId  movieId  rating
 14121       91        1      4.0
 14123       91        3      3.0
 14124       91        6      5.0
 14128       91       21      4.0
 14173       91      296      4.5), (414,       userId  movieId  rating
 62294      414        1      4.0
 62296      414        3      4.0
 62298      414        6      3.0
 62307      414       21      4.0
 62425      414      296      5.0)]
```

Figure 4.15: Top three similar groups

Calculate the similarity. In this step, we are going to compare all groups with the

```
dict_items([(32, -0.6684239227769595), (91, -0.28039985489311137), (414, -0.18380365552345185), (470, 0.8137334712067352), (480, -0.11997845
478343619), (599, -0.5714176025623253), (1, 0.6831300510639733), (6, 0.9467292624062575), (45, -0.3086066999241838), (57, 0.925820099772551
4), (64, -0.5606119105813882), (68, -0.0773308893397006), (102, 0.4472135954999579), (117, -0.8944271909999159), (140, -0.566946709513840
9), (182, 0.2857142857142857), (191, -0.3086066999241838), (217, 0.659231724180059), (219, -0.3086066999241838), (239, -0.2672612419124244),
(266, -0.8583950752789522), (288, 0.03925343359894298), (294, -0.7181848464596079), (307, 0.4181129057288785), (314, 0.3086066999241838), (3
68, -0.13483997249264842), (372, -0.16116459280507606), (373, 0.1611645928050606), (380, 0.6172133998483676), (385, -0.3779644730092272),
(448, 0.5261522196019802), (474, 0.0), (555, 0.22549380840084865), (588, 0.0), (590, -0.22677868380553634), (603, -0.8017837257372732), (60
8, -0.6534196381862065), (5, -0.32732683535398915), (18, -0.970725343941623), (28, 0.3273268353539889), (33, -0.7559289460184544), (40, 0.6
546536707079749), (42, -0.18898223650461116), (43, 0.866025403784439), (44, 0), (54, 0), (58, -0.6546536707079769), (66, -0.3273268353539891
5), (84, 0), (109, 0.0), (112, 0.2894736842105256), (121, -0.2773500981126172), (135, 0.6546536707079783), (156, -0.9819805060619622), (160,
-0.052414241836095214), (166, 0.5765566601970545), (179, 0), (181, 0.8660254037844378), (199, 0), (202, -0.2773500981126172), (220, 0.277350
0981126147), (226, -0.8660254037844402), (229, 0.6546536707079749), (240, 0.981980506061966), (269, 0.18898223650461046), (270, 0.9449111825
230682), (273, 0.3273268353539874), (274, -0.69337524528154), (275, 0), (282, 0.9707253433941623), (302, -0.6546536707079769), (304, 0.6546
536707079749), (308, -0.755928946018457), (330, 0.0), (336, -0.69337524528154), (337, -0.7559289460184538), (339, 0.6933752452815368), (353,
0.2773500981126157), (386, -0.5), (391, -0.9607689228305233), (410, 0.6546536707079769), (411, 0.6546536707079749), (434, -0.6933752452815
4), (438, 0.56362148019068), (444, 0.0), (453, 0.6546536707079774), (462, -0.9011271137791658), (469, -0.24019223070763082), (477, -0.327326
8353539889), (486, 0.8660254037844355), (492, 0.7559289460184538), (501, 0.755928946018457), (524, 0.8029550685469663), (559, 0.693375245281
5362), (561, -0.6933752452815377), (573, 0.2773500981126068), (580, -0.9607689228305233), (587, 0.6546536707079749), (592, 0.0), (597, -0.39
73597071195133)])
```

Figure 4.16: Similarity value of all similar users to the target user

	similarityIndex	userId
63	0.981981	240
69	0.970725	282
7	0.946729	6
65	0.944911	270
9	0.925820	57

Figure 4.17: Top 5 similar users

target user to figure out the most similar one by calculating the similarity by formula. As mentioned in the methodology chapter there exist many formulae to calculate the similarity, but in this system, *Pearson Correlation Coefficient* is adopted, and it can be represent as :

$$\text{sim}(i, j) = \frac{\sum_{p \in P} \left(R_{i,p} - \bar{R}_i \right) \left(R_{j,p} - \bar{R}_j \right)}{\sqrt{\sum_{p \in P} \left(R_{i,p} - \bar{R}_i \right)^2} \sqrt{\sum_{p \in P} \left(R_{j,p} - \bar{R}_j \right)^2}} \tag{4.8}$$

where $R_{i,p}$ denote the rating of the user i on movie p and $\bar{R}_i$ represents the average rating of all movies by the user i. Similarly, $R_{j,p}$ denote the rating of the target user j on movie p and $\bar{R}_j$ represents the average rating of all movies by the target user j. Also, P represent the set of all movies.

After calculating the similarity, we can list out the similarity of all similar users to the target user, shown in Figure 4.16. As Figure 4.17 shows that, by sort the similar users by similarity, the user with ID 240 is the most similar user with the target user, which the similarity is 0.981981.

By the method of user-based collaborative filtering, we multiply the similar user's

	similarityIndex	userId	movieId	rating	weightedR
0	0.981981	240	1	5.0	4.90
1	0.981981	240	2	5.0	4.90
2	0.981981	240	3	4.0	3.92
3	0.981981	240	10	3.0	2.94
4	0.981981	240	16	3.0	2.94

(a) Weighted rating

movieId	sum_similarityIndex	sum_weightedRating
1	19.733491	90.369833
2	9.421011	33.762455
3	9.907461	38.399688
4	1.601383	4.149495
5	7.635674	26.493279

(b) Sum of the weighted rating

movieId	weighted average recommendation score	movieId
1	4.579516	1
2	3.583740	2
3	3.875835	3
4	2.591195	4
5	3.469671	5

(c) Weighted average score

Figure 4.18: Calculate the score for recommendation (5 out of 9000 movies)

rating by the similarity to get the weighted rating, as shown in Figure 4.18 (a), the intuition here is that the higher the similarity with the target user, the higher the weight of the target user's prediction score. Figure 4.18 (b) demonstrates the result of the sum of weighted rating and the sum of similarity. Finally, in Figure 4.18 (c), by dividing the sum of weighted ratings by the sum of similarity, we can get the predicted weighted average recommendation score of what the target users probably will rate the movie.

Recommendation. For the recommendation, based on the above steps, first, sort the movies by average recommendation score which is shown in Figure 4.19 (a). Then, by the movie ID in the sort list, we can get the top 10 movies with titles recommended by the user-based collaborative filtering system to the target user, shown in Figure 4.19 (b).

movieId	weighted average recommendation score	movieId
8542	5.0	8542
34338	5.0	34338
106100	5.0	106100
74754	5.0	74754
27773	5.0	27773
1734	5.0	1734
1224	5.0	1224
1227	5.0	1227
33834	5.0	33834
549	5.0	549

(a) Sorted by average recommendation score

	movieId	title	year
481	549	Thirty-Two Short Films About Glenn Gould	1993
925	1224	Henry V	1989
928	1227	Once Upon a Time in America	1984
1300	1734	My Life in Pink (Ma vie en rose)	1997
5230	8542	Day at the Races, A	1937
5695	27773	Old Boy	2003
5922	33834	Land of the Dead	2005
5951	34338	Aristocrats, The	2005
7277	74754	Room, The	2003
8290	106100	Dallas Buyers Club	2013

(b) recommend movies

Figure 4.19: Top 10 movies recommend by user-based system

4.4 Hybrid Recommender System

In this section, we will demonstrate the implementation of a hybrid system with combines matrix factorization and the content-based approach.

Dataset. To verify the generality of the system, in addition to the MovieLens dataset we have used in the user-based system, we will also use a dataset called Jester[4] which contains over 1.7 million continuous ratings (-10.00 to +10.00) of 150 jokes from 59,132 users.

Recommendation. Based on the method and implementation of matrix factorization described above, we can get our user latent matrix p and item latent matrix q. By performing the inner product between the above two latent matrices, we get the predicted user-item rating matrix $\hat{R}$. For the example shown in Figure 4.20, if we want to get the predicted rating of user 1 on item 1, we can through the inner product between two matrices to get the predicted rating, the process is shown in (a) and the predicted rating here is 3.999. Compare with the grand true rating shown in (b), we can see that the approximation of the predicted rating to the true rating is 99.975%.

```
User latent vector:

[ 0.83067924  1.0331751  -1.0091356  -1.3799053  -0.7302516  -1.2855773
 -0.9584481   1.4543629   0.39436764 -1.063871   -1.3623861  -1.4639814
 -0.7324671   1.2642244   1.0967345  -1.3307375 ]
<NDArray 16 @cpu(0)>

item latent vector:

[ 0.27533662  0.15328383 -0.33647132 -0.04745744 -0.44057822  0.13066939
 -0.4638861   0.27264315 -0.17999323  0.3974794  -0.39498428 -0.4291723
 -0.29537013  0.40978968  0.43552142 -0.24620494]
<NDArray 16 @cpu(0)>

predicted rating:

[3.9992692]
<NDArray 1 @cpu(0)>
```

1	userId	movieId	rating
2	1	1	4
3	1	3	4
4	1	6	4
5	1	47	5
6	1	50	5
7	1	70	3
8	1	101	5

(a) Predicted rating   (b) Grand true rating

Figure 4.20: comparison between predicted rating and true rating

As the methods that we talk about in section 3.3, the next step is to sort the movies that have not been yet rated by the target user through our predicted rating score and recommend the top rank movie to the target user. Through Figure 4.21, we can get the information that movie with ID 910 is the movie with highest predicted rating.

Content-based recommendation. By the movie ID, we know that this movie is called *"Some Like It Hot (1959)"*. All that is left for the hybrid recommendation system to do is to suggest movies similar to the movie *"Some Like It Hot (1959)"* to the target user.

Like what we described before, in order to measure how similar the movies are in

---

[4]Dataset available at: http://eigentaste.berkeley.edu/dataset

```
[(910, 6.8839207),
 (1198, 6.415406),
 (3450, 6.31452),
 (2149, 6.307396),
 (72128, 6.1239767),
 (27800, 5.8808103),
 (246, 5.8303404),
 (152076, 5.8138604),
 (8967, 5.797981),
 (60068, 5.724992),
```

Figure 4.21: Top 10 movies with high predicted rating

attractiveness to the target user, we need to calculate the similarity by some formula. To test the ability of different similarity formulas, different from the *Pearson Correlation Coefficient* used in the previous user-based system, in this system we use *Cosine similarity* to calculate the similarity, where the following formula can used to calculate the similarity between movie i and movie j:

$$\text{similar}(i, j) = \cos(i, j) = \frac{i \cdot j}{\|i\| \cdot \|j\|} \tag{4.9}$$

Figure 4.22, (a) shows the top 10 movies which similar to the movie *"Some Like It Hot (1959)"*. In (b), if we want to recommend two more movies to the target user, as the recommendation result of this hybrid system, movie *" The Fate of the Furious (2017)"* which ID is 170875 and movie *" Withnail & I (1987)"* which ID is 1202 will be recommended to the target user who likes the movie *"Some Like It Hot (1959)"*.

```
[(170875, 1.0),
 (1202, 0.98270786),
 (73322, 0.97912574),
 (116796, 0.97876674),
 (2421, 0.97823906),
 (103209, 0.97795004),
 (134251, 0.9769946),
 (108539, 0.9768616),
 (52603, 0.9756624),
 (26346, 0.97549564),
```

```
movies making same people impressive compared to: Some Like It Hot (1959)
___
The Fate of the Furious (2017)
___

___
Withnail & I (1987)
___
```

(a) Top 10 items with high similarity   (b) Recommendation result

Figure 4.22: Content-based recommendation

When we repeated the above steps on another dataset Jester on jokes, our results are shown in Figure 4.23. By matrix factorization, we get the most recommended jokes to

For the joke:

================

Recently a teacher, a garbage collector, and a lawyer wound up together at the Pearly Gates. St. Peter informed them that in order to get in to Heaven, they would each have to answer one question.

St. Peter addressed the teacher and asked, "What was the name of the ship that crashed into the iceberg? They just made a movie about it."

The teacher answered quickly, "That would be the Titanic." St. Peter let him through the gate.

St. Peter turned to the garbage man and, figuring Heaven didn't really need all the odors that this guy would bring with him, decided to make the question a little harder: "How many people died on the ship?"

Fortunately for him, the trash man had just seen the movie. "1,228," he answered.

"That's right! You may enter."

St. Peter turned to the lawyer: "Name them."

================

——

It was the day of the big sale. Rumors of the sale (and some advertising in the local paper) were the main reason for the long line that formed by 8:30, the store's opening time, in front of the store.

A small man pushed his way to the front of the line, only to be pushed back, amid loud and colorful curses. On the man's second attempt, he was punched square in the jaw, and knocked around a bit, and then thrown to the end of the line again. As he got up the second time, he said to the person at the end of the line...

"That does it! If they hit me one more time, I won't open the store!"

——

——

Reaching the end of a job interview, the human resources person asked a young engineer fresh out of Stanford,

"And what starting salary were you looking for?"

The engineer said, "In the neighborhood of $125,000 a year, depending on the benefits package."

The interviewer said, "Well, what would you say to a package of 5-weeks vacation, 14 paid holidays, full medical and dental, company matching retirement fund to 50% of salary, and a company car leased every 2 years—say, a red Corvette?"

The engineer sat up straight and said, "Wow! Are you kidding?"

And the interviewer replied, "Yeah, but you started it."

——

——

A guy walked past a mental hospital and heard a moaning voice: "13...13...13...13..."

The man looked over to the hospital and saw a hole in the wall. He looked through the hole and got poked in the eye. The moaning voice then groaned: "14...14...14...14..."

——

Figure 4.23: Joke recommendation

the target user (bounded by double lines), and based on content-based recommendations, three jokes similar to the matrix factorization recommended jokes are derived (bounded by signal line).

4.5 BanditMF

In this section, we will implement the BanditMF proposed in Section 3.4 and compare the experimental results obtained using different multi-armed bandit algorithms according to different criteria. In this experiment, the *ε-greedy*, *UCB*, and *Thompson sampling* algorithm will be used.We will implement BanditMF's process one by one as follows.

Dataset. MovieLens will be used in this experiment which contains the same data as we described in the last two experiments.

Matrix factorization. For BanditMF we use the bias method to implement matrix factorization, where $\hat{r}_{u,i} = b_{u,i} + q_i^T p_u$ and $b_{u,i}$ is the combination of user bias, item bias, and average rating.

For the result shown in Figure 4.24, we generate a predicted user-item rating matrix with a size of 610×9724

```
            0          1          2          3          4          5          6       \
0       4.896083   4.386592   4.530958   3.455741   3.605139   4.962445   4.121052
1       4.308184   3.491218   3.546596   2.882115   2.640107   4.200679   3.214883
2       3.145855   2.885803   2.790828   2.377366   2.235639   3.643841   2.665657
3       4.300299   3.474928   3.454586   3.035396   2.639504   4.256399   3.206916
4       3.547616   3.002737   2.935020   2.551711   2.261716   3.782230   2.759691
5       4.346290   3.709563   3.783751   2.985024   2.907290   4.371814   3.443478
6       3.822477   3.442792   3.428836   2.805846   2.737766   4.144951   3.204845
7       4.417197   3.509267   3.653003   2.743757   2.610019   4.151285   3.216251
8       3.926547   3.156179   3.225137   2.498111   2.315602   3.845305   2.881469
9       3.586708   3.208347   3.147003   2.674816   2.515089   3.954489   2.976065
10      4.535122   4.052258   4.022734   3.491926   3.320427   4.785921   3.809009
11      5.215317   4.467566   4.371107   4.163076   3.673389   5.307257   4.213872
12      4.137957   3.569241   3.549886   3.021927   2.809554   4.307548   3.318859
13      4.114391   3.436710   3.451015   2.860396   2.636689   4.161557   3.174836
14      3.863664   3.441856   3.453999   2.764989   2.718097   4.126605   3.197928
15      4.021058   3.328825   3.362246   2.716560   2.519907   4.038236   3.063696
16      4.317994   3.754635   3.813282   3.033999   2.977811   4.419307   3.495451
17      4.395970   3.582433   3.660615   2.921892   2.726863   4.270060   3.303653
```

Figure 4.24: Partial of user-item predicted matrix

Clustering. We perform the clustering on the row of the predicted user-item rating matrix $\hat{R}$, which means that we cluster the users who share similar rating preferences. In this experiment, we adopt a machine learning algorithms Library called *scikit-learn*. Based on the library, *K-means* algorithm is adopted which is measured by *Euclidean distance*. We set the parameter *n_clusters*=3, which means that we will cluster all users into three clusters. Also, we set the value of parameter *n_init* to 20 which means that the *K-means* algorithm will be run with 20 different centroid seeds and output the best result in 20 consecutive runs as the final result. Figure 4.25 shows the above iterations,As the number of iterations increases (i.e. x-axis), the sum of the Euclidean distances decreases (i.e. y-axis).

Finally, by clustering, we got a 3×9724 predicted user-item rating matrix, shown in Figure 4.26, which means that we convert the original 610×9724 predicted user-item rating matrix to the 3×9724 clustered user-item rating matrix.

Online recommendation module.In the online recommendation, we used three different multi-armed bandit algorithms for the comparison, including *ϵ-greedy*, *UCB*, and *Thompson sampling*. The bandit algorithm takes the clustered predicted user-item rating matrix as data, and then selects and recommends items to the target users. After receiving the user's rating, the reward and regret are calculated. Then judge whether the user is still a cold user. If so, continue to update the Bandit algorithm with the received
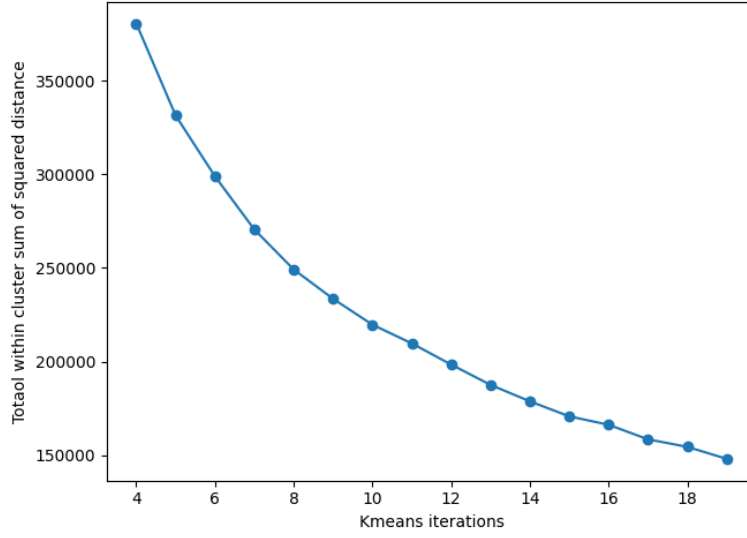
Figure 4.25: Iterations of K-means

cluster	0	1	2	3	4	5	6	7	8	...
0	1	4.545291	3.938018	3.925851	3.390804	3.165193	4.675968	3.684125	3.705106	3.534816	...
1	2	3.404064	2.742872	2.767940	2.136159	1.945148	3.454965	2.480872	2.506613	2.349315	...
2	3	4.027009	3.395659	3.390010	2.844051	2.614142	4.131490	3.139334	3.180036	2.997031	...

Figure 4.26: Predicted user-item rating matrix after clustering

data; if not, transfer the user to the offline system and append the user-item rating matrix.

Evaluation criteria. In addition to the _rewards_ and _regrets_ which are the common criteria in the multi-armed bandit algorithm, we have introduced two criteria: _discounted cumulative gain_ (DCG) and _normalized discounted cumulative gain_ (NDCG) [19], their formulas are shown below:

$$DCG(u) = r_{u,1} + \sum_{t=2}^{T} \frac{r_{u,t}}{\log_2 t} \tag{4.10}$$

$$NDCG = \frac{1}{N} \sum_{u} \frac{DCG(u)}{DCG^*(u)} \tag{4.11}$$

We can see from the formula that these two criteria will discount the reward at a later stage, which means that the accuracy (i.e., user satisfaction) of early recommendations is more important than that of late recommendations, which is why these two criteria are applied here. Since we focus on solving the user's cold start problem, we need the algorithm to make accurate recommendations in a shorter period. However, when the user is no longer a cold user, accurate recommendations lose value, because the user may

give up using the server before you can make accurate recommendations.

Result. We use T to denote the total rounds, and N denotes the number of new users. Further, R_t represent the cumulative regret at t rounds.

	R_1	R_2	R_3	R_4	R_5
Thompson sampling	0.00738798	0.01144652	0.04573369	0.08002087	0.11430805
UCB	0.30009534	0.61558008	0.93106481	1.23116015	1.53125549
ϵ-greedy	0.03761663	0.07190381	0.10619099	0.14047816	0.17476534

Table 4.2: Cumulative regret when $T = 5$, $N = 2$

	Normalized discounted cumulative gain (NDCG)
Thompson sampling	0.974304261691225
UCB	0.6949893798084221
ϵ-greedy	0.9646494155015297

Table 4.3: NDCG for three algorithms when $T = 5$, $N = 2$

From Table 4.2 we can see that Thompson sampling achieves the smallest regret, followed by ϵ-greedy, while the UCB algorithm has the largest regret. Similarly,as the shown in Table 4.3, Thompson sampling performs the highest NDCG which means it took less time to get the new user's preferences and provide a more accurate recommendation. The gap of NDCG between Thompson sampling and ϵ-greedy is small, but for UCB the NDCG value is low which means UCB cannot quickly get user preferences and make accurate recommendations.

	Cumulative regret	NDCG
Thompson sampling	0.60372807	0.9588463913680102
UCB	3.62539861	0.7600041518097639
ϵ-greedy	0.55885873	0.9627427515312132

Table 4.4: Cumulative regret and NDCG when $T = 15$, $N = 2$

Table 4.4 shows the cumulative regret and NDCG when $T = 15$. After 15 rounds of recommendations, we see that the UCB algorithm also has the highest regret, but the difference is that ϵ-greedy achieves a lower regret than Thompson sampling. Using the NDCG values, we conclude that ϵ-greedy and Thompson sampling can make accurate recommendations faster compared to the UCB algorithm.

As Table 4.5 shows that, When we increase T to 50 rounds, Thompson sampling reverts to the algorithm that obtains the best results, both in terms of cumulative regret and

	Cumulative regret	NDCG
Thompson sampling	1.36048677	0.9734318020634418
UCB	15.49012907	0.690944770845324
ϵ-greedy	1.58943474	0.9683140256825828

Table 4.5: Cumulative regret and NDCG when $T = 50$, $N = 2$

	Cumulative regret	NDCG
Thompson sampling	0.27170972	0.9429132401340926
UCB	1.36397039	0.7392537941258233
ϵ-greedy	0.24749948	0.9505001043510048

Table 4.6: Cumulative regret and NDCG when $T = 5$, $N = 5$

NDCG, ϵ-greedy achieves results comparable to Thompson sampling, and the gap between UCB and the two algorithms also exists. In Table 4.6, when we increase the number of new users N to 5, $T = 5$, compared to the previous results in Table 4.2, although the UCB algorithm still performs the worst, the difference between it and Thompson sampling and ϵ-greedy is slightly reduced. In Table 4.7, when we further set the number of new users N to 50, the above conclusion still holds.In Table 4.8, ϵ-greedy achieves a better performance than Thompson sampling when the number of users is relatively large. The regret of the UCB algorithm is 6.4 times higher than Thompson sampling.

	Cumulative regret	NDCG
Thompson sampling	3.11502502	0.9378967672555214
UCB	15.26485644	0.695220217000829
ϵ-greedy	2.91051406	0.9397737295609012

Table 4.7: Cumulative regret and NDCG when $T = 50$, $N = 5$

In summary, both Thompson sampling and ϵ-greedy can achieve lower regret and higher NDCG. In most cases, Thompson sampling performs better, but the difference between the two is very small. In other words, users' preferences can be drawn and accurate recommendations can be made in a short time. As for the UCB algorithm, it performs the worst among the three algorithms and cannot quickly get the preferences of new users. The possible reason is that the UCB algorithm needs to select all the arms once, a move that is fatal for the cold start problem. Because users will lose interest in the system if the system forces them to rate too much[6], it is important to get user preferences and make accurate recommendations in the shortest possible time for the cold start problem. However, if the number of new users is very large, e.g., 10,000 new users,

	Cumulative regret	NDCG
Thompson sampling	4.77815387	0.9520246834804285
UCB	30.58810434	0.6937173872467632
ϵ-greedy	4.48261952	0.954308414431733

Table 4.8: Cumulative regret and NDCG when $T = 100$, $N = 50$

then the UCB algorithm is equivalent to conducting 10,000 random selection strategies, which results in UCB not being able to make accurate recommendations for new users quickly and suffering from huge regret.

Chapter 5

Conclusion

In this report, we firstly reviewed and formulated the representative multi-armed bandit and collaborative filtering methods. Subsequently, we introduce the matrix factorization methods, and a matrix factorization considering bias is proposed. Based on it, we illustrate the memory-based collaborative filtering recommender system and a hybrid recommender system that combines the content-based method with matrix factorization techniques. Importantly, in this work, BanditMF has been proposed which is a multi-armed bandit-based matrix factorization recommender system. This system combines the matrix factorization (MF) which is model-based collaborative filtering with the multi-armed bandit algorithm. BanditMF uses a multi-armed bandit algorithm to solve the cold start problem in collaborative filtering. Also with the help of matrix factorization results, the bandit algorithm can quickly derive user preferences. When we get the user preferences of the new users, we can transfer the new users to the offline module for the recommendation, that is, add the user preference vector obtained from the online module to the user-item rating matrix. The advantage of this is that the collaborative filtering technology in the offline module fills a gap in the multi-armed bandit algorithm where similarity and user relationships are not taken into account. Also for BanditMF, there are disadvantages, as we explain in the report, the contextual bandit algorithm can be based on contextual information to achieve a better balance between exploration and exploitation. However, in our BanditMF, we apply the context-free bandit, i.e., bandit algorithm that does not consider contextual feature information, which may result in the recommended items are not optimal. So for future work is to extend the contextual bandit to the BanditMF.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), pp. 734–749. DOI: `10.1109/TKDE.2005.99`.

[2] Shipra Agrawal and Navin Goyal. "Thompson Sampling for Contextual Bandits with Linear Payoffs". In: *Proceedings of the 30th International Conference on Machine Learning.* Ed. by Sanjoy Dasgupta and David McAllester. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 127–135.

[3] Robin Allesiardo, Raphael Feraud, and Djallel Bouneffouf. *A Neural Networks Committee for the Contextual Bandit Problem.* 2014. arXiv: `1409.8191 [cs.NE]`.

[4] Daniel Billsus and Michael J Pazzani. "User modeling for adaptive news access". In: *User modeling and user-adapted interaction* 10.2 (2000), pp. 147–180.

[5] Wei Chu and Seung-Taek Park. "Personalized Recommendation on Dynamic Content Using Predictive Bilinear Models". In: *Proceedings of the 18th International Conference on World Wide Web.* WWW '09. Madrid, Spain: Association for Computing Machinery, 2009, pp. 691–700. ISBN: 9781605584874. DOI: `10.1145/1526709.1526802`. URL: `https://doi.org/10.1145/1526709.1526802`.

[6] Mehdi Elahi, Francesco Ricci, and Neil Rubens. "Active Learning Strategies for Rating Elicitation in Collaborative Filtering: A System-Wide Perspective". In: *ACM Trans. Intell. Syst. Technol.* 5.1 (Jan. 2014). ISSN: 2157-6904. DOI: `10.1145/2542182.2542195`. URL: `https://doi.org/10.1145/2542182.2542195`.

[7] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015). ISSN: 2160-6455. DOI: `10.1145/2827872`. URL: `https://doi.org/10.1145/2827872`.

[8] Longke Hu, Aixin Sun, and Yong Liu. "Your Neighbors Affect Your Ratings: On Geographical Neighborhood Influence to Rating Prediction". In: *Proceedings of the 37th International ACM SIGIR Conference on Research and amp; Development in Information Retrieval*. SIGIR '14. Gold Coast, Queensland, Australia: Association for Computing Machinery, 2014, pp. 345–354. ISBN: 9781450322577. DOI: 10.1145/ 2600428.2609593. URL: https://doi.org/10.1145/2600428.2609593.

[9] Donghyun Kim et al. "Convolutional Matrix Factorization for Document Context-Aware Recommendation". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 233–240. ISBN: 9781450340359. DOI: 10.1145/2959100. 2959165. URL: https://doi.org/10.1145/2959100.2959165.

[10] Yehuda Koren. "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model". In: KDD '08. Las Vegas, Nevada, USA: Association for Computing Machinery, 2008, pp. 426–434. ISBN: 9781605581934. DOI: 10.1145/ 1401890.1401944. URL: https://doi.org/10.1145/1401890.1401944.

[11] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/ MC.2009.263.

[12] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. DOI: 10.1145/1772690.1772758. URL: https://doi.org/10.1145/1772690.1772758.

[13] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 661–670. ISBN: 9781605587998. DOI: 10.1145/1772690.1772758. URL: https://doi.org/10.1145/1772690.1772758.

[14] Seung-Taek Park et al. "Naive Filterbots for Robust Cold-Start Recommendations". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. Philadelphia, PA, USA: Association for Com-

puting Machinery, 2006, pp. 699–705. ISBN: 1595933395. DOI: `10.1145/1150402.1150490`. URL: `https://doi.org/10.1145/1150402.1150490`.

[15] Ruslan Salakhutdinov and Andriy Mnih. "Probabilistic Matrix Factorization". In: *Advances in Neural Information Processing Systems.* Vol. 20. 2008.

[16] Badrul Sarwar et al. "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th International Conference on World Wide Web.* WWW '01. Hong Kong, Hong Kong: Association for Computing Machinery, 2001, pp. 285–295. ISBN: 1581133480. DOI: `10.1145/371920.372071`. URL: `https://doi.org/10.1145/371920.372071`.

[17] J. Ben Schafer, Joseph Konstan, and John Riedl. "Recommender Systems in E-Commerce". In: *Proceedings of the 1st ACM Conference on Electronic Commerce.* EC '99. Denver, Colorado, USA: Association for Computing Machinery, 1999, pp. 158–166. ISBN: 1581131763. DOI: `10.1145/336992.337035`. URL: `https://doi.org/10.1145/336992.337035`.

[18] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[19] Yining Wang et al. "A Theoretical Analysis of NDCG Type Ranking Measures". In: *CoRR* abs/1304.6480 (2013). arXiv: `1304.6480`. URL: `http://arxiv.org/abs/1304.6480`.

[20] Xiaoying Zhang et al. "Conversational Contextual Bandit: Algorithm and Application". In: *Proceedings of The Web Conference 2020.* WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 662–672. ISBN: 9781450370233. DOI: `10.1145/3366423.3380148`. URL: `https://doi.org/10.1145/3366423.3380148`.