Master's Research Project Report

A Collaborative Filtering based Approach for Recommender Systems

KANG Gaoxi

Supervisor Huynh Nam Van

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

02-2020

Abstract

In recent decades, with the continuous development of technology based on information systems and big data, more and more fields have been started the research and applications of personalized recommendation technologies. With the rapid growth of Internet business data, the processing capacity's demand is also in an increasing trend. Accompanied by the fast growth of customer information is that how to use it effectively has become one of the issues that need to be paid attention to.

In this report, we first introduce an overview of recommender systems consisting of its definition, theoretical basis, typical approaches, and implementation. Besides, we present solid background knowledge that is used in these systems such as similarity functions and *k*-Nearest Neighbor, Gradient Descent and Artificial Neural Networks, some standard evaluation metrics for a recommender system. We then applied the previous research to implement three different typical algorithms based on collaborative filtering. They are item-based *k*-Nearest Neighbors (*i*-*k*NN), Alternating Least Square (ALS), and Neural Network-based collaborative filtering recommender systems (NeuNet).

After that, we conduct experiments on a public dataset from Netflix called MovieLens, which contains user information, item information, and the interaction between users and items (e.g., ratings, comments, etc.). The comparison between these algorithms' performance also is considered. Through the experimental results, we found that recommendations from the item-based *k*-Nearest Neighbors approach are less diverse than those from the approach of Alternating Least Square, which can bring "surprise" recommendations to the users. From the statistic point of view, the Root Mean Square Error (RMSE) of the Neural Network based Recommender System is a little better. This result gives us more reference in the practical applications, thus providing a direction on the modern approaches for recommender systems.

Keywords: Recommender System; Collaborative Filtering; Neural Network; *k*-Nearest Neighbors; Alternative Least Square.

Contents

List of Figures

List of Tables

Chapter1
Introduction

1.1 Recommender Systems: an overview

In today's digital era, with the rise of many E-commerce platforms like Youtube, Amazon, Netflix, and many other web services, recommender systems have become more and more important in our lives. Definition: *"A recommender system, or a recommendation system, is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item [wikipedia]*. " Today's recommender systems are inevitable in our daily online journey and consumers are facing various choices of products at any time, from e-commerce with suggestions to articles of potentially interested buyers to online advertising which suggests users the right content to match their preferences.

Technically, the recommender system is an algorithm that aims to recommend relevant items to users. The items may be movies, texts, products or others depending on business contexts.

In this chapter, we will present an overview of the recommender system and their different examples. For each instance, we will introduce how they work, describe their theoretical basis, and discuss their advantages and disadvantages.

1.2 Typical approaches in Recommender Systems

In the first part, we introduce two typical paradigms of a recommender system: content-based approach and collaborative filtering one and how they work. The next part will then describe various methods of collaborative filterings, such as user-based, item-based and matrix factorization. The overall of different paradigms for the recommender system is depicted in Figure 1.1.

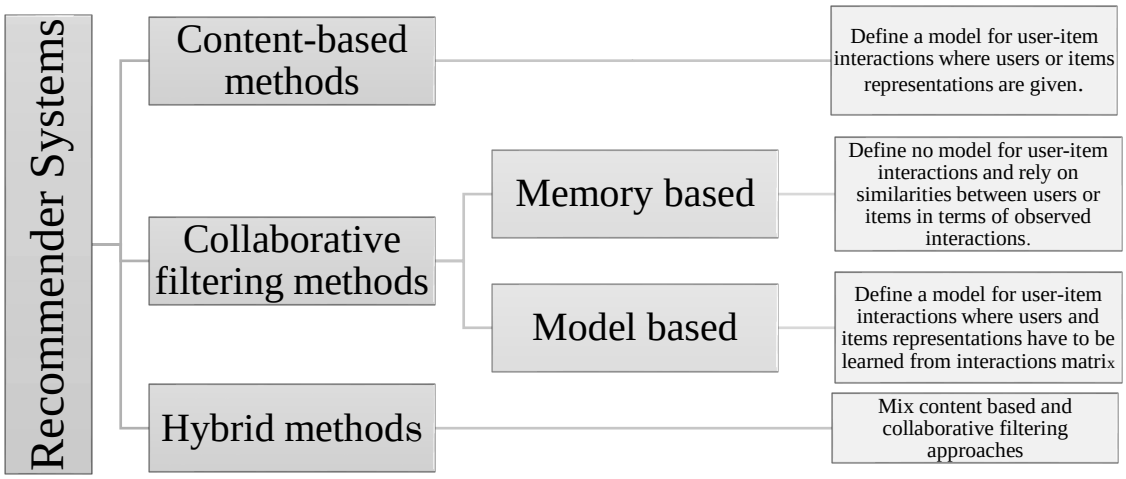

Figure 1.1: Typical paradigms for a recommender system [1]

1.2.1 Content-based methods

Content-based methods use additional information about users or items in making recommendations [2]. If we take the movie recommendation system as an example, this additional information can be, for example, age, gender, any other personal details of the user(like click on a specific item, purchasing item, etc.) as well as category, main actor, duration or other characteristics of the item.

Then, the critical idea of a content-based method is to try to build a model based on available "features" that precisely explain observed user-item interactions. Retake users and movies as an instance; a good recommender system should reflect the fact that young women score higher on some movies, while young men score higher on others, and so on. If we try to get such a model, it's easy to make new predictions for users: we can utilize the user's profile (eg., age, gender, etc.). Basing on this information to determine the relevant movies, then make a recommendation list.

In the content-based approach, the recommendation problem is transformed into a classification problem (predicting whether the user "likes" the product) or a regression problem (predicting the user's evaluation of the product). In both cases, we will set up the

model based on the available user or item features.

If the classification or regression is based on user features, then we say the method is item-centered: modeling, optimization and calculation can be conducted "by item". In this case, we will build and learn a model item by item according to the features of users to try to answer "what is the probability that each user likes this item?" for regression. The model associated with each item will naturally be trained based on the data related to the items, and because many users have interacted with the items, it usually leads to quite powerful models. However, considering that the interaction of the learning model comes from each user, even if these users have similar features, their features may be different. This means that even if this method is more robust, it can be considered less personalized (more biased) than the user-centric methods.

If we use item functionality, the method will be user-centric: modeling, optimization, and calculation can be done "by the user." Then, our goal is to train a model according to the features of the items, which tries to answer the following questions: "what is the probability that the user likes each item?" for regression. Then, we can attach the model to each user trained in the data: therefore, the model obtained is more personalized than the item-centric model, because it only considers the interaction from the users. However, in most cases, users interact with relatively few items, so the model we get is far less robust than the item-centered model.

From a practical view, we should emphasize that in most cases, it is much more difficult to ask new users some information (users don't want to answer too many questions) than new items. That is because the person who added the item is interested in filling in this information in order to recommend its product to the appropriate user. We can also note that according to the complexity of expression relationships, the models we build are more or less complex, ranging from basic models (logical/linear regression for classification/regression) to deep neural network [3]. Finally, it should be mentioned that content-based methods can be neither user-centered nor item-centered: both information of users and items can be used in our model, for example, by stacking two eigenvectors

and passing them through the neural network architecture [4].

1.2.2 Collaborative filtering methods

The collaborative filtering methods of the recommender system are based on past interaction recorded between users and items to generate new recommendations. These interactions are stored in the “user-item interaction matrix.”

Then, the main idea governing the collaborative filtering methods is that these past user-item interactions are sufficient to detect similar users or similar items, and predict based on the proximity of these estimates.

The categories of collaborative filtering algorithms are divided into two subcategories, usually called memory-based methods and model-based methods. The memory-based approach works directly with the recorded interaction values and is basically on the nearest neighbor search. The model-based approach assumes a basic "build" model that interprets user-item interactions and attempts to discover such model in order to make new predictions.

The main advantage of collaborative filtering algorithms is that they do not need information about users or items so that they can be used in many cases. Besides, the more users interact with the project, the more accurate the new suggestions are: for a set of fixed users and items, the new interaction recorded over time will bring new information and make the system more effective.

However, since collaborative filtering only considers history interactions to make suggestions, it suffers some problem such as "Cold Start," "Scalability," "Sparsity":

- Cold Start: when an item inserted into the catalog is minimal, even zero interactions. This causes some difficulty if recommender bases on those interactions to make recommendations

- Scalability: when applying algorithms on a considerable dataset where the number of users and items are big enough. The current, practical datasets are almost extremely massive.

- Sparsity: in many practical E-commerce platforms such as Youtube, Netflix, Amazon, etc. The number of items & users is huge while the interaction between each (user, item) pairs is usually lacked. Because most users did not revise their purchasing products/items. This caused the problem of sparsity.

It is unable to recommend anything to new users or items, and many users or items interactions are rarely effectively handled. This defect can be solved in different ways: such as random strategy [5] which is recommending random items to new users, maximum expectation strategy [6] which is recommending common items to new users or recommend new items to the most active users, exploratory strategy [7] which is recommending various items to new users or a group of new users and so on.

1.2.2.1 Memory-based methods

- **User-based Collaborative Filtering**

To make new suggestions for users, the user-based method tries to identify users with the most similar "interaction" to propose the most popular items among these users. This approach is called "user-centric" because it represents users and evaluates the distance between users based on their interaction with the items.

Supposing we want to make recommendations for a given user. Firstly, each user can be represented by its interaction vector (row in the matrix) with different items. Then, we can calculate a certain "similarity" between interested users and all other users. This similarity measure makes two users who have similar interaction on the same item be regarded as intimate. Once we have calculated the similarity with each user pairs, we can keep *k* nearest neighbor to our users, and then suggest the most popular items.

- **Item-based Collaborative Filtering**

In order to make new suggestions to users, the idea of the item-based method is to find items similar to those that users have already "actively" interacted with. If most users who interact with two items operate similarly, the two items are considered similar. This approach is called "item-centric" because it represents items based on the interaction

between users and them, and evaluates the distance between these items.

Supposing we want to make recommendations for a particular user. Firstly, we consider his/her favorite products and represent them by an interaction vector (column in the matrix) with each user. Then, we can calculate the similarity between the "best item" and all other items. Once the similarity is calculated, we can keep *k* neighbors on the selected "best item", which is new for the users who are interested in, and recommend these items. Table 1.1 summarize the contrast between item-based and user-based in collaborative filtering [8].

Contrast	User-based	Item-based
Search of similar	Users	Items
Variance	High	Low
Bias	Low	High

Table 1.1: Contrast of memory-based methods

1.2.2.2 Model-based methods

The model-based collaboration method only depends on the interaction information of users and items and assumes a potential model can explain these interactions. For example, the matrix decomposition algorithm decomposes the vast and sparse user-item interaction matrix into two smaller, dense matrices: user factor matrix (including user representation) multiplied by item factor matrix (including item representation).

The main assumption behind matrix decomposition is that there is a relatively low dimensional feature space in which we can represent users and items, and we can get the interaction between users and items by the dot product of the corresponding dense vectors in that space.

For example, we have a user-movie rating matrix. To simulate the interaction between users and movies, we can assume that:

(1) Some features can describe movies well.

(2) These features can also describe the user's preferences

However, we don't want to assign these capabilities to our model explicitly. On the contrary, we prefer to let the system discover these useful features and make its own representations to users and items. Since these features are not learned, the features extracted separately are of mathematical significance, but there is no intuitive explanation. However, it is not uncommon that this type of algorithm structure is very close to the intuitive decomposition that human can think of. In fact, the result of this factorization is that in terms of preferences, close to users and features close to items eventually have a close representation in the potential space.

1.2.3 Hybrid recommender system

Recent studies have shown that the hybrid approach of collaborative filtering and content-based filtering may be more effective in some cases [9]. There are several ways to implement hybrid methods, which are content-based prediction and collaboration-based prediction respectively [10], and then combine them to add content-based functions to collaboration based methods (and vice versa) or unify these methods into a method model [11]. Several studies compare the performance of the hybrid method with a pure collaboration-method and content-based approach and show that the hybrid method can provide more accurate suggestions than pure way [12]. These methods can also overcome some common problems in the recommender system, such as cold start and sparsity.

1.3 Report objectives & Structure

In this report, we will first study the typical approaches for a recommender system, then we reimplement these approaches and compare their performance in a specific context. To achieve this objective, we carry out subtasks as below:

- Survey about the general pattern of a recommender system: its definition, theoretical basis, approaches and implementation. This task is represented in Chapter 1.
- Accomplish the background knowledge that needed to constitute a typical recommender system. These backgrounds include Gradient Descent & Artificial Neural Network, similarity measure & *k*-Nearest Neighbor, and valuation metrics for

the recommender system. They are presented in Chapter 2.

- Summarize three typical algorithm principles that provide the theoretical foundation for the whole report from different perspectives that will be present in Chapter 3.
- Implementing the programs, conducting experiments on particular dataset, observing the result and make conclusions will be in Chapter 4 and Chapter 5.

Chapter 2
Background

This chapter will present a background knowledge of recommender systems. We will introduce the similarity functions and *k*-Nearest Neighbor, what is Gradient Descent and Artificial Neural Network in detail, and the evaluation method of the recommender system.

2.1 Similarity functions

Distance is a numerical measure of how far apart an object or point is. In physics or everyday use, distance can refer to physical length or an estimation basing on some predefined criteria. In mathematics, distance function or measure is a generalization version of physical distance's concepts. Measurement is a function that runs according to a specific set of rules and is a way to describe the meaning of "close" or "far" between elements of a space.

2.1.1 Minkowski(Euclidian, Manhattan, Chebyshev)

Minkowski distance is a measure in norm vector space, which can be regarded as the generalization of Euclid distance, Manhattan distance and Chebyshev Distance.

Minkowski distance of order *p* between two points $X = (x_1, x_2, \dots \dots x_n) \in \mathbb{R}^n$ and $Y = (y_1, y_2, \dots \dots y_n) \in \mathbb{R}^n$ is defined as:

$$d(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \quad (2\text{-}1)$$

When p equals different value, the formula can be defined as different distance functions, shown in table 2.1, if we take an example that $= [0,6,0]$ $y = [2,8,9]$, the different distances are also shown in the table.

Manhattan Distance: $p = 1$ **(L₁-norm)**	**Euclidian Distance:** $p = 2$ **(L₂-norm)**	**Chebyshev Distance:** $p \to +\infty$ **(L_∞-norm)**
$d(x,y) = \sum_{i=1}^{n} \lvert x_i - y_i \rvert$	$d(x,y) = \sqrt[2]{\sum_{i=1}^{n} \lvert x_i - y_i \rvert^2}$	$d(x,y) = \lim_{p \to +\infty} \left(\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p \right)^{\frac{1}{p}} = \max_{\forall i} \{ \lvert x_i - y_i \rvert \}$
$d(x,y) = \lvert 0-2 \rvert + \lvert 6-8 \rvert + \lvert 0-9 \rvert = 13$	$d(x,y) = \sqrt[2]{\lvert 0-2 \rvert^2 + \lvert 6-8 \rvert^2 + \lvert 0-9 \rvert^2} = 89$	$d(x,y) = \max\{\lvert 0-2 \rvert + \lvert 6-8 \rvert + \lvert 0-9 \rvert\} = 9$

Table 2.1: Different distance function and example

2.1.2 Cosine Similarity & Pearson Correlation

Cosine similarity is usually useful for text data. It considers the angle between two vectors $u = [u_1, u_2, \dots, u_n]$ and $v = [v_1, v_2, \dots, v_n] \in \mathbb{R}^n$:

$$\text{Cosine}(u,v) = \frac{u \cdot v}{\|u\| * \|v\|} = \frac{\sum_{i=1}^{n} u_i * v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}} \quad (2\text{-}2)$$

The range of similarity is from - 1 to 1, where 0 is orthogonal or decorrelated, and the middle value is similar or not.

Example: Two documents consist of 10 words as below:

	car	***bug***	***gid***	***dir***	***gid***	***ant***	***fox***	***fir***	***cat***	***log***
u	0	6	0	0	2	0	0	0	8	9
v	1	0	0	0	1	0	0	3	0	0

Thus $u \cdot v = 0 * 1 + 6 * 0 + \cdots + 2 * 1 + \cdots + 9 * 0 = 2$

$$\|u\| = \sqrt{0^2 + 6^2 + \cdots + 2^2 + \cdots + 8^2 + 9^2} = 13.60$$

$$\|v\| = \sqrt{1^2 + \cdots + 1^2 + \cdots + 3^2 + \cdots} = 3.32$$

$$\textbf{Cosine}(u,v) = \frac{u \cdot v}{\|u\| * \|v\|} = \frac{2}{13.60 * 3.32} = 0.044$$

These two documents are quite difference

Pearson Correlation is a measure of the linear relationship between the attributes

of the two objects $x = [x_1, x_2, \dots, x_n]$ and $y = [y_1, y_2, \dots, y_n] \in \mathbb{R}^n$:

$$r(x, y) = \frac{S_{xy}}{S_x * S_y} \tag{2-3}$$

where $S_{xy} = covariance\ (x, y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$

$$S_x = Standard\ deviation\ (x) = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$S_y = Standard\ deviation\ (y) = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})^2}$$

$$\bar{x} = mean\ (x) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$\bar{y} = mean\ (y) = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Notice that $r(x, y) \in [-1, +1]$, where $+1$ indicates total positive linear correlation, 0 means no linear correlation, and -1 indicates total negative linear correlation

Example: if hours of study: $x = [8,8,6,5,7,6]$

Test score: $y = [81,80,75,65,91,80]$

Then: $r(x, y) = 0.648$

which is a certain correlation between study hours and test scores.

2.2 *k*-Nearest Neighbors

k-Nearest Neighbor is a method of estimating the distance between an unseen object and a set of observed data in the attribute vector space. It is instance-based or lazy learning in which the classification function is only approximated locally, and all calculations are delayed until the actual classification process. An object can be classified via a majority vote of its *k* neighbors. As shown in Figure 2.1, if $k = 1$, the object is assigned to its nearest neighbor's class.

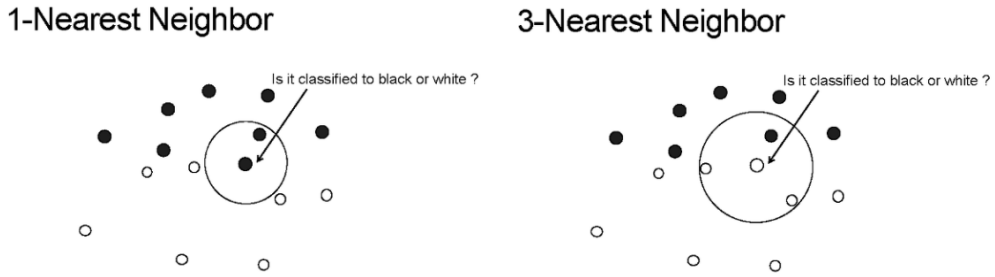

Figure 2.1: Demonstration of *k*-Nearest Neighbors [13]

Table 2.2 illustrates a typical pattern of *k*-Nearest Neighbors' search.

Algorithm: *k*-Nearest Neighbors

//Goal: search the nearest neighbors data point of a target object
//input:
- target object
- A set of labeled objects
- Similarity measure $d(x,y)$
- The value of *k* (usually $k = 1,3,5,7 \ldots$)
//Output: Top *k* data that are nearest to the target object.
1. for each object in the dataset
calculate *d(target object, current object),*
insert the distance into a sorted list.
2. Pick the first *k* entries from the sorted collection.
3. Return the results in step 2.

Table 2.2: Algorithm of *k*-Nearest Neighbors search

2.3 Gradient Descent[1]

Gradient Descent is one of the well-known optimization algorithms in machine learning, especially deep learning. It has a good mathematic background and usually be used in training models. Mathematically, gradient descent's goal is to minimize a convex function to a local optimal by adjusting its parameters which are the gradients of the function

[1] https://builtin.com/data-science/gradient-descent

This method is used to find the value of parameters, which minimize the cost function as much as possible. Gradient descent usually starts with an initial value of the parameters; then it iteratively adjusts the value from the gradient descent using the calculus to minimize the given cost function.

For example that figure 2.2 shows a hill from top to bottom, and the red arrow is the climber's step. In this case, we can think of the gradient as a vector containing the direction to the hill's peak that a climber should take to arrive at the highest point of the hill.

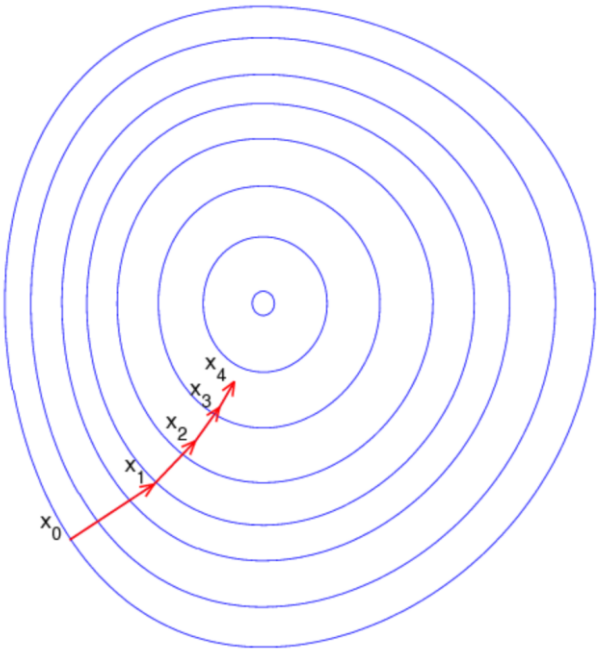

Figure 2.2: Contour plot of a 3D surface [*wikipedia*]

The following equation describes the effect of gradient descent: b is the climber's next position, and a is his current location. A minus sign is the minimum part of the gradient drop. The middle γ is a waiting factor (e.g., known as learning rate) and the gradient ($\Delta f(a)$) is the direction of ascent.

$$b = a - \gamma \Delta f(a) \tag{2-4}$$

Therefore, the formula tells us the next position we need to climb, that is, the direction of the steepest ascent. We want to apply gradient descent to a Machining Learning problem, as shown in Figure 2.3. To minimize the cost function $J(W, b)$ that means we

want to minimize the local cost function $J(W, b)$ by adjusting W & b. That figure 2.3 shows that the first two dimensions represent W & b respectively. The third one represents J which is a convex function.

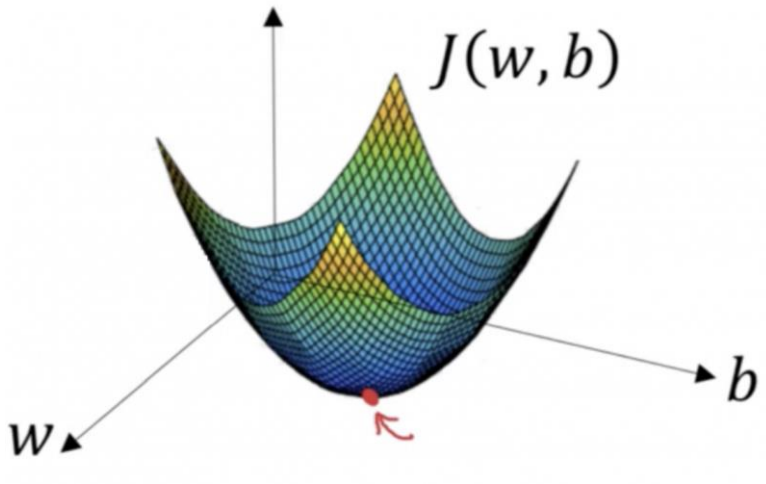

Figure 2.3: Gradient Descent with two parameters [*wikipedia*]

Our goal is to find W & b values (marked with red arrows) corresponding to the local minimum value of the cost function. To arrive at the correct value, W & b usually are initialized at some random value. Then, the gradient descent will start at this position (near the bottom of the hill) and proceeds step by step in the steepest downward direction into a lower position. After a number of iterations, the cost function eventually ends up at hopefully the lowest point, called local minimal.

To guarantee that the gradient decrease to a local minimum, learning rate should be set to an appropriate value not too high neither too low. This is vital because if the step size is too large, you may not be able to reach a local minimum, as it bounces back and forth between convex functions with gradient descent. On the other hand, if the learning rate is set to a too-small value, the function eventually reaches its local minimum but requiring a long time. Figure 2.4 illustrates these situations

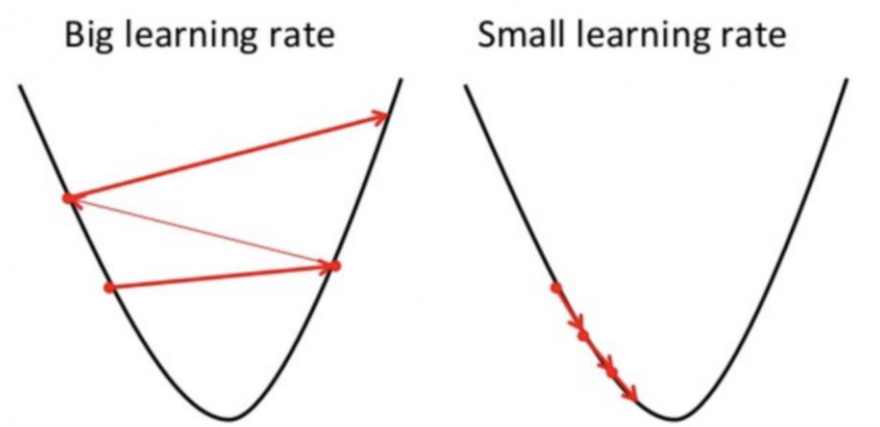

Figure 2.4: The convergence of Gradient Descent

2.4 Artificial Neural Network[2]

Artificial Neural Networks(ANNs) are the computational models that try to imitate the human brain. They have shown huge advancements in a variety of Artificial Intelligence's fields like Image Recognition, Voice Recognition, Robotics.

The term "Neral" derives from the basic functional unit "neuron" of the human nervous system or nerve cells existing in the human brains and other parts of the human body. Artificial neural networks attempt to mimic and simplify these brain behaviors. A neural network is an algorithm that captures the hidden relationship between data points inside the dataset. Technically, it adapts itself to the input so that its resulting output is "best fit" the true ground label

2 https://adventuresinmachinelearning.com/neural-networks-tutorial/#gradient-desc-opt

2.4.1 The structure of a typical ANNs

2.4.1.1 The artificial neuron

Biological neurons are simulated in ANN by activation function. In the classification tasks, this activation function must have the "on" feature - in other words, once the input is higher than a specific value, the output should change state, i.e. from 0 to 1, or from -1 to 1, or from 0 to 0. This mimics the "opening" of biological neurons. A common activation function is usually the sigmoid function:

$$f(z) = \frac{1}{1+\exp(-z)} \tag{2-5}$$

Which looks like this:

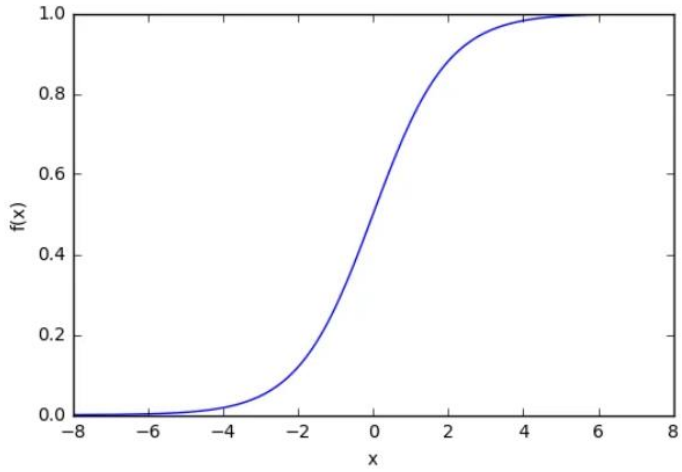

Figure 2.5: Sigmoid function

As can be seen from the above figure, this function is "activated," that is, when the input *x* is higher than a specific value, it moves from 0 to 1. Sigmoid colon function is not a stepping function, but the edge is "soft," and the output will not change immediately. This means that the function has a derivative, which is very important for the training algorithm.

2.4.1.2 Nodes

As mentioned before, biological neurons are connected to layered networks, and the

output of some neurons is the input of other neurons. We can express these networks as the connection layer of nodes. Each node uses multiple weighted inputs, and the activation function is applied to the sum of these inputs to generate the output. Consider the following figure:

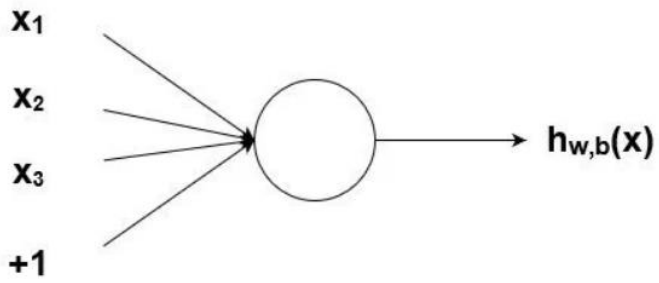

Figure 2.6: Node with input in a Neural Network [14]

The circle in figure 2.6 represents the node. Each node is then fed into the next layer, multiplied by the corresponding weight, sum up, then apply activation function to get the output at that layer unit. That output is shown as $h_{w,b}(x)$ in figure 2.6. Mathematically:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + b \tag{2-6}$$

Where w_i here is the weight (known as parameters), They are variables that are changeable during learning and effects on the node's output along with the input. If the value changes, the active function will change.

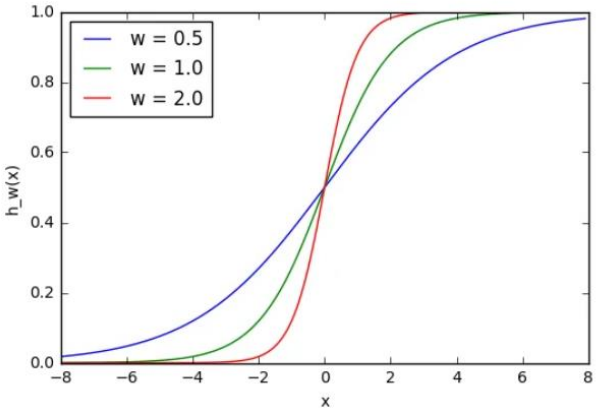

Figure 2.7: Effect of adjusting weights

Here, changing the weight will shift the output slope of the active sigmoid function. If we want to model the different strengths of the relationship between input variables and

output variables, it is beneficial.

2.4.1.3 The bias

b is the weight of the $+1$ offset element - including this bias increases the flexibility of the node. If we only want to change the output when x is higher than 1, that's where the bias occurs.

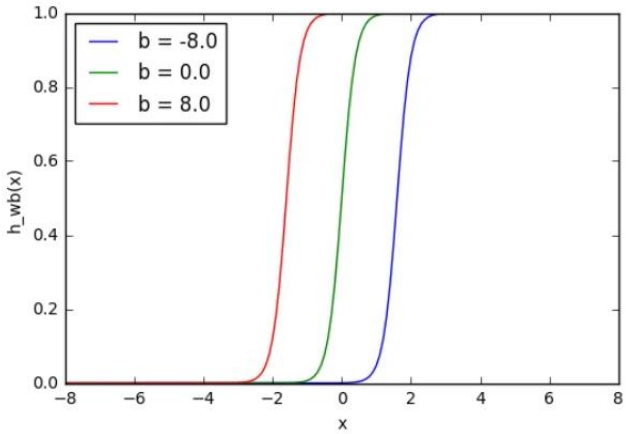

Figure 2.8: Effect of adjusting bias

We can see, by changing the bias "weight," we can change when the node is active. Introducing a bias term, we can ensure that the node simulates a generic if function ($x > z$) and then 1, 0 otherwise. Without the bias term, we cannot change z in that if statement, it will always be stuck at about 0. This is useful if we try to simulate conditional relationships.

2.4.1.4 Putting the structure together

A typical network architecture consists of a large number of artificial neurons, which are arranged in a series of layers. In a typical Artificial Neural Network, it contains different layers. These structures can take many different forms, but the most common simple network structure is usually composed of an input layer, hidden layers, and an output layer. An example of this structure is as follows:

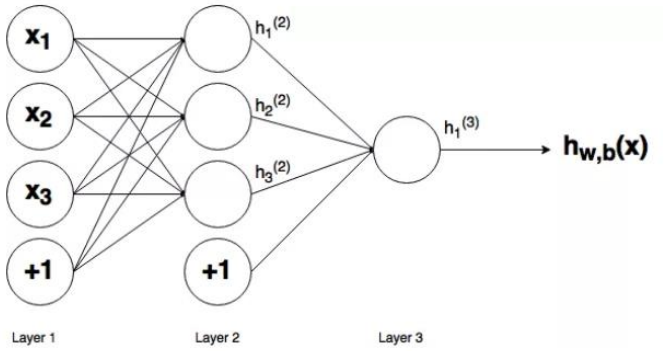

Figure 2.9: A Neural Network with two layers [15]

- Input layer (layer 1): It contains units (artificial neurons) that take input from the outside world, on which the network will learn parameters or perform other processing.

- Output layer (layer 3): it consists of units corresponding to information about how to learn any task.

Hidden layer (layer 2) - these units are located between the input and output layers. The hidden layer's job is transforming the input into some way that the output cell can use.

As we can see, each node in layer 1 is connected to all nodes in layer 2. Similarly, for nodes in layer 2, there is a connection to layer 3 of a single output node. Each of these connections will have associated weights.

2.4.1.5 The notation

In the following equations, each of these weights is identified with the following notation: $w_{ij}^{(l)}$.

i refers to the number of nodes connected in layer $l+1$.

j refers to the number of nodes connected in layer l.

So, the connection between node 1 in layer 1 and node 2 in layer 2, the weight notation would be $w_{21}^{(1)}$.

The notation of the bias weight is $b_i^{(l)}$.

'i' is the node number in the layer $l+1$, which is the same as used for the regular

weight notation. So, the weight on the connection of the bias in layer 1 and the second node in layer 2 is given by $b_2^{(1)}$.

Finally, the node output notation is $h_j^{(l)}$.

'j' denotes the node number in layer 1 of the network. As can be observed in the three-layer network above, the output of node 2 in layer 2 has the notation of $h_2^{(2)}$.

2.4.2 Forward propagation

The forward propagation can be treated to calculating the output from the input in neural networks. Below it is presented in equation form, we take the example in Figure 2.5:

$$h_1^{(2)} = f(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}) \quad (2\text{-}7)$$

$$h_2^{(2)} = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}) \quad (2\text{-}8)$$

$$h_3^{(2)} = f(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}) \quad (2\text{-}9)$$

$$h_{W,b}(x) = h_1^{(3)} = f(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)}) \quad (2\text{-}10)$$

In the equation above $f(*)$ refers to the node activation function, in this case, the sigmoid function. The first line, $h_1^{(2)}$ is the first node's output in the second layer, and its inputs are $w_{11}^{(1)}x_1$, $w_{12}^{(1)}x_2$, $w_{13}^{(1)}x_3$ and $b_1^{(1)}$.These These inputs can be found in the three-layer connection diagram above. Sum them and then calculate the output of the first node by active function. Again, for the other two nodes in the second layer.

The last line is the output of the only node in the third and final layer, which is the final output of the neural network. It can be seen that the final node does not adopt the weighted input variables (x_1, x_2, x_3), but adds the weighted bias to the second layer nodes ($h_1^{(2)}, h_2^{(2)}, h_3^{(2)}$). Therefore, we can see the hierarchical nature of the artificial neural network in the equation.

For calculation, there is a method that can write equations more compactly and calculate the forward process more effectively in the neural network. First, we can introduce a new variable $z_i^{(l)}$, which is the total input of node *i* of layer 1, including the bias term. Therefore, for the first node in layer 2, *z* is equal to:

$$z_1^{(2)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)} = \sum_{j=1}^{n} w_{ij}^{(1)} x_i + b_i^{(1)} \quad (2\text{-}11)$$

where n is the nodes number of in layer 1.

By this representation, the clumsy previous equations of the example three-layer network can be simplified as follows:

$$z^{(2)} = W^{(1)} x + b^{(1)} \quad (2\text{-}12)$$

$$h^{(2)} = f\left(z^{(2)}\right) \quad (2\text{-}13)$$

$$z^{(3)} = W^{(2)} h^{(2)} + b^{(2)} \quad (2\text{-}14)$$

$$h_{W,b}(x) = h^{(3)} = f\left(z^{(3)}\right) \quad (2\text{-}15)$$

Note that capital *W* is used to represent the matrix form of the weight. It should be noted that all elements of the above equation are now matrices or vectors. We can spread the calculation results to any number of layers in the neural network by the following summary:

$$z^{(l+1)} = W^{(l)} h^{(l)} + b^{(l)} \quad (2\text{-}16)$$

$$h^{(l+1)} = f\left(z^{(l+1)}\right) \quad (2\text{-}17)$$

We can observe that the general forward propagation process, where the output of layer l becomes the input to its next layer $l+1$. We know that $h^{(1)}$ is simply the input layer *x* and $h^{(n_l)}$(where n_l is the layers' number in the network) is the output of the output layer. Note that in the above equation, we have removed the reference to the node number, and we can use matrix multiplication to do this more simply.

2.4.3 Backward propagation

The process of backward propagation is to learn the parameters of the network, which means to find the minimized settings ($W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$) errors in our training data. We call the function of measurement error the cost function. The equivalent cost function of a training pair (x^z, y^z) in the neural network is as follows:

$$J(w, b, x, y) = \frac{1}{2} \| y^z - h^{n_l}(x^z) \|^2 = \frac{1}{2} \left\| y^z - y_{pred}(x^z) \right\|^2 \quad (2\text{-}18)$$

This shows the cost function of the z_{th} training sample, h^{n_l} is the last layer's output of the neural network, which is the neural network output. h^{n_l} is represented as y_{pred}

to highlight the prediction for a given x^z. The first $\frac{1}{2}$ is just a constant. When we distinguish cost functions, we will sort things out, which will be done during backward propagation.

Note that the formula for the above cost function applies to a single *(x, y)* training pair. We want to minimize the cost function among all *m* training pairs. So, we hope to find the minimum Mean Square Error (MSE) of all training samples:

$$J(w,b) = \frac{1}{m}\sum_{z=0}^{m}\frac{1}{2}\|y^z - h^{n_l}(x^z)\|^2 = \frac{1}{m}\sum_{z=0}^{m}J\left(w,b,x^{(z)},y^{(z)}\right) \tag{2-19}$$

We train the weight of the network by using the cost function *J* and gradient descent. The gradient of each weight $w_{ij}^{(l)}$ and bias $b_i^{(l)}$ in the neural network is as follows:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha\frac{\partial}{\partial w_{ij}^{(l)}}J(w,b) \tag{2-20}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(w,b) \tag{2-21}$$

Where $w_{new} = w_{old} - \alpha * \nabla error$. The left-hand side of this equation is the updated value of the weight, while the right-hand side is the old value of the weight. Again, we have an iterative process in which the weights are updated at each step basing on *J (W, b)*.

2.5 Evaluation of Recommender Systems Results

The recommender system is a prediction model with algorithms. Usually, these algorithms are seeking to minimize the function error. Therefore, it is essential to measure the prediction error that they compare the expected results with the prediction results given by the model.

To measure the accuracy of recommender system results, some of the most common prediction error measures are usually used, including Mean Absolute Error (MAE) and its related measures: Mean Square Error (MSE), Root Mean Square Error (RMSE) and normalized mean absolute error.

$$MAE = \frac{\sum_{i=1}^{n} |\hat{r}_{ui} - r_{ui}|}{n} \quad (2\text{-}22)$$

$$MSE = \frac{\sum_{i=1}^{n} (\hat{r}_{ui} - r_{ui})^2}{n} \quad (2\text{-}23)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (\hat{r}_{ui} - r_{ui})^2}{n}} \quad (2\text{-}24)$$

Where $\hat{r}_{ui} - r_{ui}$ is the difference between the predicted and the real value, for each n occurrence

MAE measures the average size of errors in a set of predictions, regardless of their direction. It is the average of the absolute difference between the predicted value and the actual observed value in the test sample, where the weight of all individual differences is equal.

RMSE is a secondary scoring rule, which can also measure the average magnitude of error. It is the square root of the average value of the squared difference between the predicted value and the actual observed value.

In addition to these three indicators, two critical concepts also need to be considered: Precision and Recall.

$$precision = \frac{|Recommended \cap Relevant|}{|Recommended|} \quad (2\text{-}25)$$

$$Racall = \frac{|Recommended \cap Relevant|}{|Relevant|} \quad (2\text{-}26)$$

Precision is the ability to provide the relevant elements with the least amount of advice.

Recall can find all relevant elements and recommend them to users.

Chapter 3
Problem Formulation and Approaches

3.1 Problem Formulation:

Let's take the problem of predicting movie ratings for the process of our formulation, although the general form is similar to this specific instance.

Assume that there are n_u users and n_m movies. Each user gives a rating, which is a real number from 1 to 5, for all movies that he/she is interested in. Each movie receives a rating from all users. Formally, we get a matrix of size $n_m \times n_u$ where each row represents a list of ratings made by all users for a specific movie, and each column represents a list of ratings made by a particular user for all movies. As shown in Figure 3.1. Each cell in the matrix may receive unknown value which is denoted as a "?" because there are some users who did not rate for a movie. This is known as a sparse matrix.

Movie	Alice	Bob	Carol	Dave	…
Love at last	5	5	0	1	
Romance forever	5	?	?	0	
Cute puppies of love	?	4	0	?	
Nonstop car chases	0	0	4	4	
Sword vs karate	0	0	5	?	
…					

Figure 3.1: User-Item rating matrix

Notations	Description
n_u	Number of users
n_m	Number of movies
$r(i,j)$	Indicator function, $r(i,j) = 1$ if user i^{th} gives a rating to a movie j^{th}
$y^{(i,i)}$	Actual rating made by user i^{th} on movie j^{th} if $r(i,j)$
$c(i,j)$	Value from 1 to 5 represents a rating made by user i^{th} for movie j^{th}
$m^{(j)}$	Number of movies rated by user j
$\theta^{(j)}$	Parameter vector for user j
$x^{(j)}$	Parameter vector for movie i

Table 3.1: Summarizes the notation used in this chapter

The goal is to efficiently infer these missing values so that the matrix "best reflects" the actual interaction between each movie and its corresponding users.

Mathematically, for user i, movie j, predict: $\left(\theta^{(j)}\right)^T\left(x^{(i)}\right)$.

Equivalently, we want to learn: $\theta^{(j)}$ and $x^{(i)}$ for all i, j.

In the following sections of this chapter, we will introduce some approaches for learning these missing values which are Item-based *k*-Nearest Neighbor Recommender System (i-*k*NN-RS), Alternative Least Square based Collaborative Filtering (ALS-CF), and Neural Network-based Collaborative Filtering (Neural Network-CF).

3.2 Item-based *k*-Nearest Neighbor Recommender System (*i-k*NN-RS)

In this section, we introduce the item-based *k*-Nearest Neighbor Recommender System algorithm called *i-k*NN-RS. In chapter 4, we will show the experiment results of this model. *i-k*NN-RS is an easy-to-go model and usually plays the role of a very appropriate baseline for comparison between different recommender models. As shown in figure 3.2 and be introduced in chapter2, *k*NN is a non-parametric, lazy learning model. It uses a dataset consisting of the data points which are separated into groups to predict the value for a new instance. Also, note that *k*NN's result much depends on the distance function.

In the case, that input training data are in a high dimensional space. *k*NN's performance will commit the problem of the curse of dimensionality if *k*NN considers Euclidean distance as its objective function. Euclidean distance does not help in high dimensions, because all vectors are almost equidistant from the input query vectors (features of the target movie). Instead, the cosine similarity introduced in chapter 2 is entirely appropriate in this situation. Although the detail process of *i*-*k*NN-RS is depicted in Figure 3.2, we should introduce its key idea first.

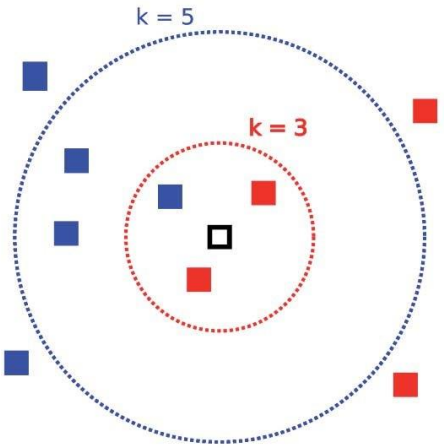

Figure 3.2: Demonstration of *k*-Nearest Neighbors

3.2.1 Idea of *i*-*k*NN-RS:

The primary purpose of the item-based *k*NN-Recommender System is simple. It does not add any particular assumption on the data distribution; it much bases on the feature similarity between items. When making inference about the similarity between movies, *k*NN needs to compute the distance between the target movie and other movies in the dataset, then ranks the results in a descending order according to distance from each movie to the input target movie, and returns the top *k* nearest neighbor movies which are the most *k* similar movies. The overall algorithm for *i*-*k*NN-RS is illustrated in Table 3.2.

Algorithm: item-based k−NN Recommender System
//**Goal**: Implement a recommender system bases on item-based *k*-NN search **//Input:** - k: number of nearest neighbors - $input\_movie$: the target movie for recommending - a matrix of size $m \times n$ consisting of all ratings of m movies by n users - $distance\_type$: distance function type, e.g., Euclidian, Manhattan, cosine, etc. //**Output**: Top k movies that are nearest to the $input\_movie$. 1. for every $movie_i$ in the dataset do 2. calculate and store the distance $d(movie_i, input\_movie)$ 3. Sort the movies in a descending order regarding its distances 4. Pick up and return the top k movies at step #3 5. end item-based *k*-NN Recommender System

Table 3.2: Algorithm of Item-based *k*-Nearest Neighbor Recommender System

3.2.2 Implementation Concerns

This section describes some critical concerns related to the implementation of *i-k*NN-RS. First, the rating input data need to be transformed into an appropriate format that can be feed into the *i-k*NN-RS model. The data should be in a 2D array of size $m \times n$, where m, n represent for the number of movies and the number of users, respectively. To reshape the data frame of ratings, we format movies as rows and users as columns. Then missing values need to be filled in with zeros because of applying linear algebra operations which are computing the distances between vectors. We can now consider this new data frame as a feature vector of each movie in the input dataset.

One important observation here is that the matrix representing the movie extremely sparse. We should not feed the entire data with mostly floating zero numbers into the *i-k*NN-RS model. To achieve the calculation efficiency and consuming less memory units,

the data frame should be transformed into a ***SciPy sparse matrix***[3] and being implemented as shown in Figure 3.3.

```
# pivot and create movie-user matrix
movie_user_mat = df_ratings_drop_users.pivot(index='movieId',
                                             columns='userId',
                                             values='rating').fillna(0)

# create a mapping from movie title to its index
movie_to_idx = {
    movie: i for i, movie in
    enumerate(list(df_movies.set_index('movieId').loc[movie_user_mat.index].title))
}
# transform matrix to scipy sparse matrix
movie_user_mat_sparse = csr_matrix(movie_user_mat.values)
```

Figure 3.3: Code of turning data frame

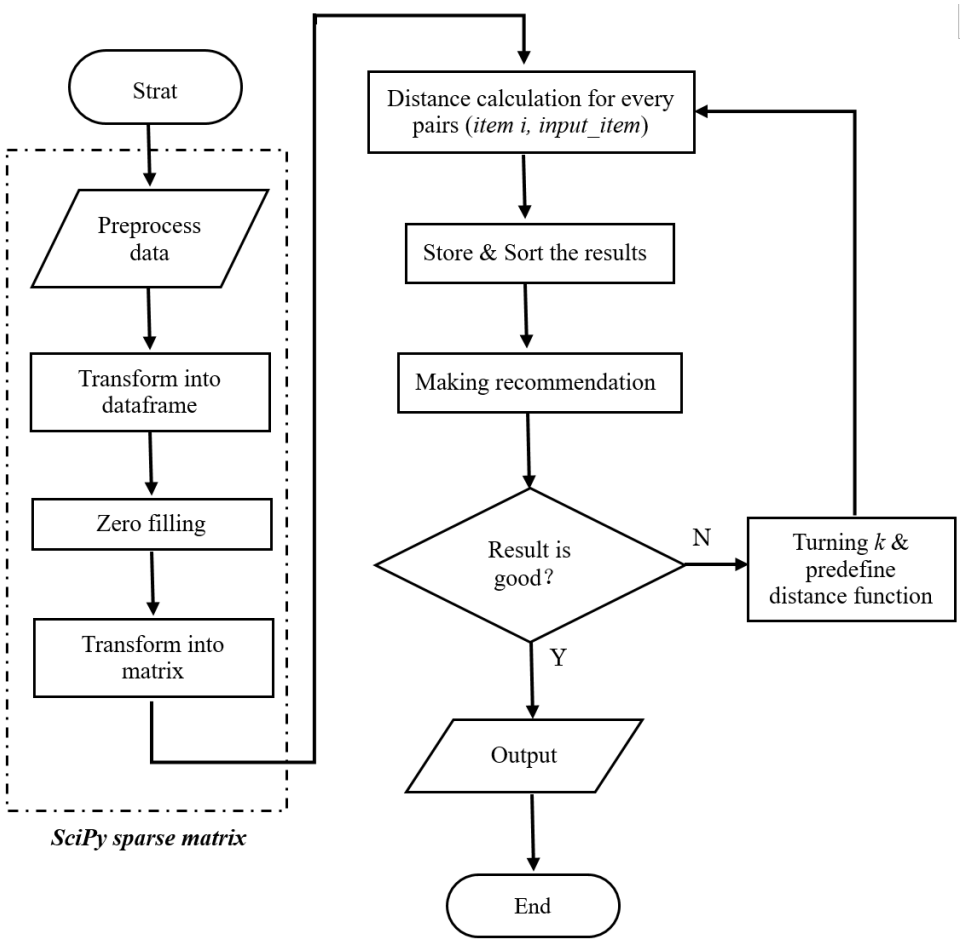

Figure 3.4: Flowchart of item-based *k*-NN Recommender System

3 ***SciPy sparse matrix*** is a python package providing tools for creating sparse matrices using multiple data structures, as well as tools for converting a dense matrix into a sparse one.

In chapter 4, we will conduct experiments on this model and consider its performance as a baseline with the other ones.

3.3 Alternating Least Square based Collaborative Filtering Recommender Systems(ALS-CFRS)

In the previous section, we formulated and introduced a baseline recommender system called *i-k*NN-RS, as well as provided some implementation concerns for it. Although *i-k*NN-RS is easy to be implemented, it is evident that there are some limitations such as "cold-start problem" and "scalability issue."

The problem of item cold-start occurs when a movie inserted into the catalog is minimal, even zero interactions. This causes some difficulty if recommender bases on those interactions to make recommendations

The issue of scalability occurs when applying *i-k*NN-RS on a considerable dataset where the number of users and items are big enough. The current, practical datasets are almost extremely massive, e.g., MovieLens_Datasets

These two above problems are prevalent challenges for a typical collaborative filtering recommender system. They appear naturally along with the user-item (or item-user) interaction matrix where each cell $c(i,j)$ in the matrix represents a rating of the user i and movie j. In practical applications, one item (e.g., movie) receives very few or even no ratings by all users. So that we are coping with an extremely sparse matrix with sparsity level is more than 99% as illustrated in Figure 3.5.

userId	4	5	10	14	15	18	19	26	31	34	...	283199	283204	283206	283208	283210	283215	283219	283222	283224
movieId																				
1	4.0	NaN	5.0	4.5	4.0	NaN	NaN	NaN	5.0	NaN	...	5.0	NaN	NaN	4.5	NaN	4.0	4.0	NaN	NaN
2	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	4.0
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN
6	4.5	NaN	NaN	NaN	NaN	3.0	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	4.0	NaN	NaN	NaN	NaN	3.0	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0
11	3.5	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0
15	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 3.5: Sparsity of a general dataset

To deal with the problem, this section presents a little more sophisticated approach called ALS-CFRS which is based on a well-known mathematical technique called low-rank Matrix Factorization.

3.3.1 Least Square Problem

Before discussing in detail Alternative Least Square Collaborative Filtering let's briefly revise the least-squares problem (in particular, a norm L_2-regularized least squares problem). Mainly, our goal in the least-squares problem by the language of linear algebra is to minimize the cost function of the form:

$$\underset{w}{\operatorname{argmin}}(\|y - Xw\|_2^2 + \lambda\|w\|_2^2) \tag{3-1}$$

where

X is a feature matrix, $X \in \mathbb{R}^{m\times d}$;

y is the target value, $y \in \mathbb{R}^{m\times 1}$;

w is the parameter of the model $w \in \mathbb{R}^{d\times 1}$; and

λ is the adjusting rate for avoiding overfitting.

In the language of linear algebra, the optimal solution of linear regression can be found analytically as follow:

$$w = (X^T X + \lambda I)^{-1} X^T y \tag{3-2}$$

We can further expand this approach for multi-linear regression over multiple target $Y = \{y_1, y_2, \dots, y_n\}$ using the same feature matrix. The solution parameter is similar but now is put all together within a matrix as below:

$$W = (X^T X + \lambda I)^{-1} X^T Y \tag{3-3}$$

In practice, we rarely get to use this elegant analytical approach because it requires inverting a square matrix of size d where d usually takes a value from thousands to millions. And this is not a feasibly right approach because the inversion of a huge-size matrix is not only computationally expensive but numerically unstable. Fortunately, despite being rarely used in practical problems, this analytical approach is possible for recommender systems as we will show shortly as below.

Recall from the problem definition section that for user i, movie j, we want to predict: $\left(\theta^{(j)}\right)^T \left(x^{(i)}\right)$. Equivalently, we want to learn: $\theta^{(j)}$ and $x^{(i)}$ for all i, j.

Note that the role of users and movies is exchangeable. That means if the movie parameter $x^{(i)}$ is given, then $\theta^{(j)}$ can be learned by minimizing the following least-square lost function for a particular user $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2 \tag{3-4}$$

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ for all users, the total cost function turns out to be:

$$\min_{\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2 \tag{3-5}$$

Similarly, if the user parameter $\theta^{(j)}$ is given, then $x^{(i)}$ can be learned by minimizing the following least square cost function for a particular movie $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2 \tag{3-6}$$

To learn $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ for all users, the total cost function turns out to be:

$$\min_{x^{(1)}, x^{(2)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)}\right)^2 \tag{3-7}$$

Combining (3-5) and (3-7), give us the objective cost function as below:

$$\min_{\Theta,X} J(\Theta, X) = \min_{\Theta,X} \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(\left(\theta^{(j)}\right)^T\left(x^{(i)}\right) - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2 +$$

$$\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(\theta_k^{(j)}\right)^2 \quad (3\text{-}8)$$

Now we can employ gradient descent to learn both $\theta^{(j)}$ and $x^{(i)}$ simultaneously with the update rule for every $j = 1,2,\dots,n_u,\ i = 1,2,\dots,n_m$ as below:

$$x_k^{(i)} = x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}\left(\left(\theta^{(j)}\right)^T\left(x^{(i)}\right) - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)}\right) \quad (3\text{-}9)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha\left(\sum_{i:r(i,j)=1}\left(\left(\theta^{(j)}\right)^T\left(x^{(i)}\right) - y^{(i,j)}\right)x_k^{(i)} + \lambda\theta_k^{(j)}\right) \quad (3\text{-}10)$$

The overall algorithm is described in Table 3.3.

Algorithm: Alternative Least Square Collaborative Filtering Recommender Systems

//**Goal**: Implement a recommender system bases on ALS-CFRS

//Input:

- n: number of components for user parameters and movie parameters
- a matrix of size $n_m \times n_u$ consisting of all ratings of n_m movies by n_u users

//**Output**: predicting the rating of a user $\theta^{(j)}$ and a movie $x^{(i)}$

1. Initialize $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ & $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ to small random values.
2. Minimize $J\left(\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}, x^{(1)}, x^{(2)}, \dots, x^{(n_m)}\right)$ using gradient descent

$$x_k^{(i)} = x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}\left(\left(\theta^{(j)}\right)^T\left(x^{(i)}\right) - y^{(i,j)}\right)\theta_k^{(j)} + \lambda x_k^{(i)}\right)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha\left(\sum_{i:r(i,j)=1}\left(\left(\theta^{(j)}\right)^T\left(x^{(i)}\right) - y^{(i,j)}\right)x_k^{(i)} + \lambda\theta_k^{(j)}\right)$$

3. For a user with parameter $\theta^{(j)}$ and a movie with parameter $x^{(i)}$, predict a star rating of $\left(\theta^{(j)}\right)^T\left(x^{(i)}\right)$.
4. End

Table 3.3: Algorithm of Alternative Least Square Collaborative Filtering

Note that in the language of linear algebra, this problem is known as a low-rank matrix factorization problem. The detail of low-rank matrix factorization is beyond the scope of

this thesis. Chapter 4 will describe the experiment results of this approach. Furthermore, similar to other machine learning algorithms, ALS-CFRS has its own set of hyper-parameters such as:

rank: the number of hidden factors in our ALS-CFRS model (defaults: 10)

regParam is the regularization parameter (defaults: 1.0)

We may need to tune these hyper-parameters via hold-out validation or grid search to get the final ALS-CFRS model. In the tuning process of ALS-CFRS, we applied the grid search. Particularly, after tuning, we found the best choice for hyper-parameters are $regParam = 0.05$ and $rank = 20$.

3.4 Neural Network-based Recommender Systems

One of the disadvantages of the ALS-RS is that it just considers the linear relationship between data points in the dataset. In this section, we consider a much more complex model which bases on Neural Network to copy with non-linear relationships within the dataset. Because the background of the Neural Network was introduced in chapter 2, here we present the architecture network that will be used in this report.

Table 3.4 summarize all notations used for defining network architecture

Notations	**Description**
num_users	number of users
num_items	number of movies
MF_dim	int, embedded dimension for user and item vector
MF_reg	tuple of float, L_2-regularization of MF embedded layer
MLP_layers	list of int, each element is hidden unit number for each MLP layer, except for the first element. First element is the sum of user latent vector and item latent vector
MLP_regs	list of int, each element is the L_2-regularization parameter for each layer in the neural network model

Table 3.4: Notations of network architecture

We declare the neural network model via ***Keras***[4] library as below:

4 ***Keras*** is an open-source library written in Python, designed to enable fast

```
NeuMF_model = get_NeuMF_model(
    num_users=num_users,
    num_items=num_items,
    MF_dim=10,
    MF_reg=(0, 0),
    MLP_layers=[64, 32, 16, 8],
    MLP_regs=[0, 0, 0, 0])
NeuMF_model.summary()
```

Figure 3.6: Neural Network model in ***Keras***

Figure 3.7 summarizes our neural network architecture

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
user_input (InputLayer)         (None, 1)            0
__________________________________________________________________________________________________
item_input (InputLayer)         (None, 1)            0
__________________________________________________________________________________________________
mlp_user_embedding (Embedding)  (None, 1, 32)        21504       user_input[0][0]
__________________________________________________________________________________________________
mlp_item_embedding (Embedding)  (None, 1, 32)        290144      item_input[0][0]
__________________________________________________________________________________________________
flatten_18 (Flatten)            (None, 32)           0           mlp_user_embedding[0][0]
__________________________________________________________________________________________________
flatten_19 (Flatten)            (None, 32)           0           mlp_item_embedding[0][0]
__________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 64)           0           flatten_18[0][0]
                                                                 flatten_19[0][0]
__________________________________________________________________________________________________
mf_user_embedding (Embedding)   (None, 1, 10)        6720        user_input[0][0]
__________________________________________________________________________________________________
mf_item_embedding (Embedding)   (None, 1, 10)        90670       item_input[0][0]
__________________________________________________________________________________________________
layer1 (Dense)                  (None, 32)           2080        concatenate_6[0][0]
__________________________________________________________________________________________________
flatten_16 (Flatten)            (None, 10)           0           mf_user_embedding[0][0]
__________________________________________________________________________________________________
flatten_17 (Flatten)            (None, 10)           0           mf_item_embedding[0][0]
__________________________________________________________________________________________________
layer2 (Dense)                  (None, 16)           528         layer1[0][0]
__________________________________________________________________________________________________
multiply_4 (Multiply)           (None, 10)           0           flatten_16[0][0]
                                                                 flatten_17[0][0]
__________________________________________________________________________________________________
layer3 (Dense)                  (None, 8)            136         layer2[0][0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 18)           0           multiply_4[0][0]
                                                                 layer3[0][0]
__________________________________________________________________________________________________
prediction (Dense)              (None, 1)            19          concatenate_7[0][0]
==================================================================================================
Total params: 411,801
Trainable params: 411,801
Non-trainable params: 0
```

Figure 3.7: Neural Network architecture

experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible [*wikipedia*].

Next, we train the model via ***Adam***[5] optimization algorithm with some parameters such as

$BATCH\_SIZE = 64$

$EPOCHES = 30$

$VALIDATION\_SPLIT = 25\%$

After training the model, we estimate its performance on the test set via the estimator Root Mean Square Error (RMSE). The experimental results will be discussed in chapter 4.

Note that the process of selecting an appropriate network architecture is known as hyper-parameter tuning and usually not easy to cope with. We leave that problem as future work in this report.

In conclusion, this chapter introduced three different approaches for a recommender system such as *i-k*NN-RS, ALS-RS, and Neural Network-based RS. Table 3.5 summarized the advantages and disadvantages of these approaches.

Approach	**Advantages**	**Disadvantages**
i-kNN-RS	Simple, easy to be implemented No training phase	Much depends on distant function and the value of k. Simply filling zero to the missing values, so that does not reflect the genuine relationship inside the dataset
ALS-CFRS	Has a well mathematical definition so that there is an analytical solution Efficient for problem with medium dataset	Not work well for datasets consisting of a non-linear relationship between data points
Neural Network-RS	Able to consider complex relationships between data points inside the dataset	Require more resources like training times. A large number of hyper-parameters need to be tuned

Table 3.5: Contrast of different approach recommender system

5 ***Adam*** optimization algorithm is a kind of stochastic gradient descent optimizer provided by ***Keras*** library.

Chapter 4
Implementation and Experiments

4.1 Data Description

The experiment in this chapter uses ***MovieLens***[6] dataset which consists of two main files, movies.csv, and ratings.csv. The dataset describes a list of ratings and free-text tagging activities made by users for movies. In total, there are 27,753,444 ratings and 1,108,997 tags across 53,098 unique movies. These data created by 283,228 individual users during the time (1995/01/09-2018/09/26). The dataset has eliminated demographic information. A hash id identifies each user without any other information. Table 4.1 summarizes the general information of this dataset.

Data Description	
Number of movies	53,098
Number of users	283,228
Number of ratings	27,753,444
Number of tags	1,108,997
Time of collecting the dataset	1995/01 – 2018/09

Table 4.1: Information of the experiment dataset

Figure 4.1 extracts and shows some instances from the dataset

6 This is a public dataset available at https://grouplens.org/datasets/movielens/latest/

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)
5	6	Heat (1995)
6	7	Sabrina (1995)
7	8	Tom and Huck (1995)

	userId	movieId	rating
0	1	307	3.5
1	1	481	3.5
2	1	1091	1.5
3	1	1257	4.5
4	1	1449	4.5
5	1	1590	2.5
6	1	1591	1.5
7	1	2134	4.5

Figure 4.1: Example of the dataset

4.2 Data Exploration and Preprocessing

Before executing the program, let's explore some properties of the data. Firstly, we want to get the counts of each rating from the file "ratings.csv" of the entire dataset. Because the count for zero-rating score is too large when being compared with others. So we should take a logarithm to transform the count values into a scale that can be easily visualized. The result is shown in Figure 4.2.

	count	log_count
0.0	15235120248	23.446869
0.5	442388	12.999943
1.0	886233	13.694735
1.5	441354	12.997603
2.0	1850627	14.431035
2.5	1373419	14.132814
3.0	5515668	15.523103
3.5	3404360	15.040568
4.0	7394710	15.816275
4.5	2373550	14.679897
5.0	4071135	15.219432

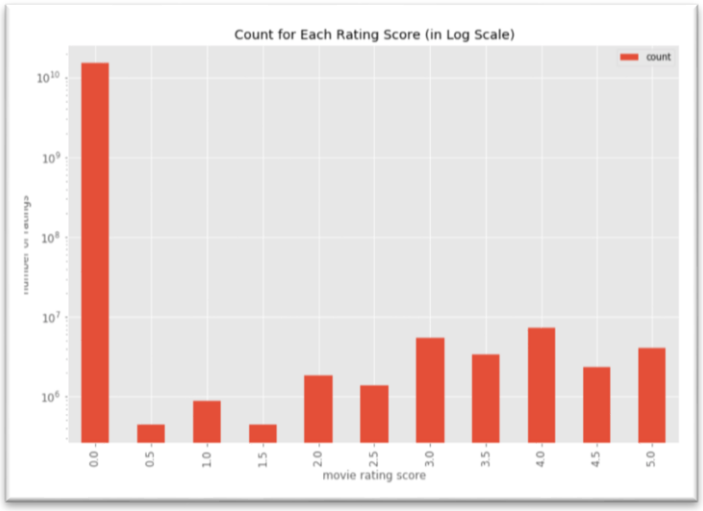

Figure 4.2: Count of the rating by users

Figure 4.3 illustrates the rating made by a user for a particular movie

	userId	movieId	rating
0	1	307	3.5
1	1	481	3.5
2	1	1091	1.5
3	1	1257	4.5
4	1	1449	4.5
5	1	1590	2.5
6	1	1591	1.5
7	1	2134	4.5

movieId	
1	68469
2	27143
3	15585
4	2989
5	15474
6	28683
7	15301
8	1539

Figure 4.3: Example of a specific movie

Figure 4.4 plots the rating frequency for all movies in log scale

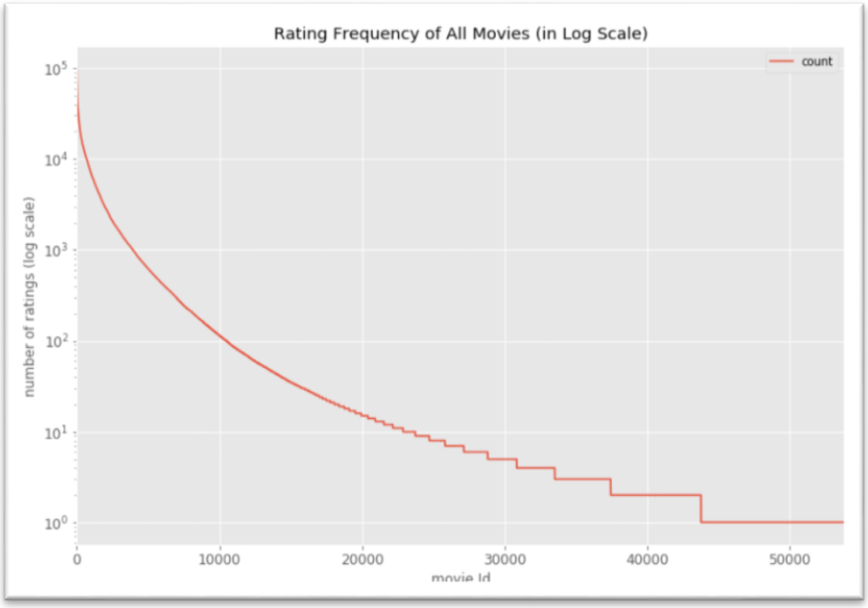

Figure 4.4: Rating frequency plot

We can observe that about 10,000 out of 53,098 movies are rated more than 100 times.

More interestingly, roughly 20,000 out of 53,889 movies are rated less than only 10 times. Table 4.2 looks closer by displaying top quantiles of rating counts for movies

Quantiles	**Counts**
1.00	97,999
0.95	1855
0.90	531
0.85	205
0.80	91
0.75	48
0.70	28
0.65	18

Table 4.2: Rating counts of movies

After dropping 75% of movies, the dataset is still large enough for the next state. But next, we also need to reduce the number of users. Similarly, Table 4.3 shows some quantiles of users regarding the number of times they made a rate for movies.

Quantiles	**Counts**
1.00	9,384
0.95	403
0.90	239
0.85	164
0.80	121
0.75	94
0.70	73
0.65	58
0.60	47
0.55	37

Table 4.3: Number of times for movies

We can observe that the distribution of ratings by users is very similar to one of the ratings among movies. They both have long-tail property. There is a tiny fraction of users who are actively engaged with rating movies that they watched. So we should limit users to the top 40%, which is around 113,290 users.

To summarize about the dataset, we observed that:

- Rating score of 3 and 4 is quite popular among the users than other scores
- The score of zero is at a considerable number, and this confirms the situation that we are coping with a sparse dataset.
- The experiment in this chapter will use the top 25% of movies which are around 13,500 movies that have ratings and the top 40% of users who made the ratings.

4.3 Experimental Results

4.3.1 *i-k*NN-RS

As introduced in chapter 3, *i-k*NN-RS requires defining values for hyper-parameters as shown in Table 4.4.

Hyper-parameters	**Value**
Number of nearest	$k = 10$
Distance function	Cosine similarity
Type of algorithms	Brute-force

Table 4.4: The value of hyper-parameters for *i-k*NN-RS

Ten movies that were recommended by *i-k*NN-RS are shown in Figure 4.5. One interesting thing is that the movie Sherlock Holmes (2009) and "Sherlock Holmes: A Game of Shadows (2011)" are in the recommendation list. However, some other matching movies did not appear such as "Sherlock Holmes (2010)", "Sherlock Holmes Faces Death (1943)", and "Young Sherlock Holmes (1985)".

```
You have input movie: Sherlock Holmes
Found possible matches in our database: ['Sherlock Holmes (2010)', 'Sherlock Holmes
(1985)', 'Sherlock Holmes Faces Death (1943)']

Recommendation movies:
......

Recommendations for Sherlock Holmes:
1: Hitch Hikers Guide to the Galaxy, The (1981), distance of 0.9272758364677429
2: Adjustment Bureau, The (2011), distance of 0.9271038770675659
3: John Carter (2012), distance of 0.9269044399261475
4: Source Code (2011), distance of 0.9268187284469604
5: Cowboys & Aliens (2011), distance of 0.9261358976364136
6: Looper (2012), distance of 0.9256317615509033
7: X-Men: First Class (2011), distance of 0.9249240159988403
8: Sherlock Holmes: A Game of Shadows (2011), distance of 0.9174680113792419
9: Total Recall (2012), distance of 0.9137960076332092
10: Sherlock Holmes (2009), distance of 0.9136103987693787
```

Figure 4.5: Recommendations of *i-k*NN-RS

4.3.2 ALS-CFRS

Table 4.5 shows the value of hyper-parameters of ALS-RS which are used in the experiments. Furthermore, because both rank and regularized parameter λ can take a list of values so that we applied the grid search to select the best model among them. Moreover, we split the dataset into train/validation/test sets using a 7:2:1 ratio.

Hyper-parameters	**Values**
Number of iterations	10
Rank for matrix factorization	[8,10,12,14,16,18,20]
Regularized Parameter	[0.001,0.01,0.05,0.1,0.2]
Train/Validation/Test Ratio	7:2:1

Table 4.5: The value of hyper-parameters for ALS-CFRS

We first train the model on a train set, then apply a grid search to select the best model which results in minimal Root Mean Square Error on a validation set. Table 4.6 shows detail.

Rank / *Reg_para*	**8**	**10**	**12**	**14**	**16**	**18**	**20**
0.001	0.863	0.860	0.856	0.861	0.859	0.870	0.848
0.01	0.843	0.850	0.846	0.831	0.849	0.873	0.868
0.05	0.835	0.829	0.831	0.839	0.823	0.818	**0.807**
0.1	0.853	0.850	0.848	0.831	0.839	0.869	0.858
0.2	0.843	0.845	0.846	0.841	0.845	0.863	0.845

Table 4.6: Grid search of rank and regularized parameter

We observe that $rank = 20$, and regularized parameter $reg\_para = 0.05$ give the best model. Figure 4.6 illustrates the learning curve representing the fluctuation of RMSE at each iteration.

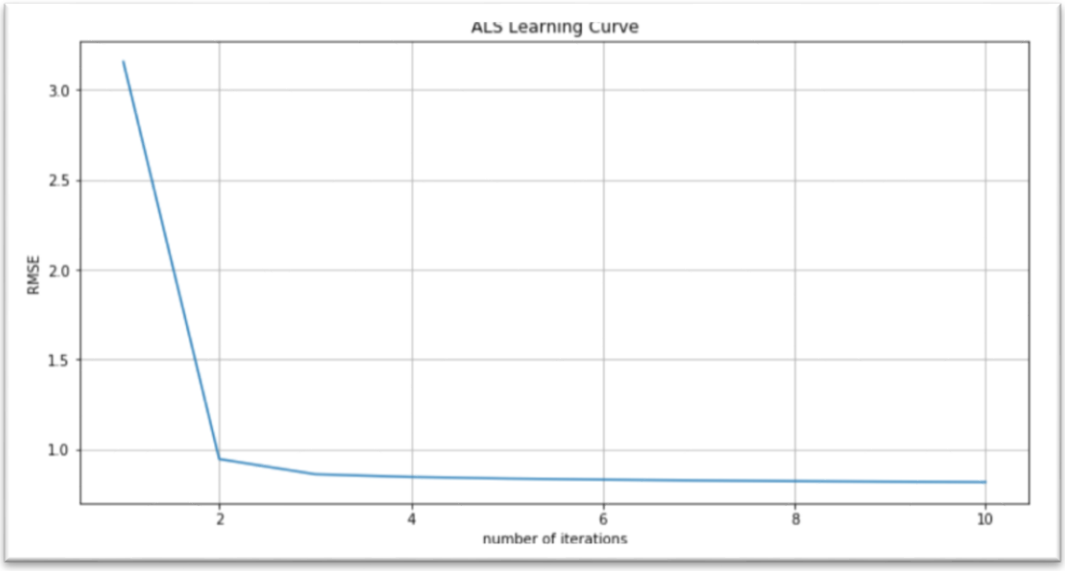

Figure 4.6: Learning Curve of RMSE

We observe that after three iterations, the RMSE starts to converge around 0.8. After training the model, we now use it to make a recommendation. Similarly, ten movies were recommended by ALS-CFRS are shown in Table 4.7.

We can see that the recommendation list made by ALS-CFRS is entirely different from the *i-k*NN-RS model recommender. Not only the ALS-CFRS model recommends movies

outside of years between 2007 and 2009 periods, but also it recommends less known movies. This can offer users some elements of surprise so that users won't get bored by getting the same popular movies all the time. Table 4.7 shows the results made by these two algorithms.

***i-k*NN-RS (Ref: "Sherlock Holmes(2010)")**	**ALS-CFRS (Ref: "Sherlock Holmes(2010)")**
Hitch Hikers Guide to the Galaxy(1981)	Shawshank Redemption(1994)
Adjustment Bureau(2011)	Godfather(1972)
John Carter(2012)	Philadelphia Story(1940)
Source Code(2011)	Casablanca(1942)
Cowboys & Aliens(2011)	Rebecca(1940)
Looper(2012)	12 Angry Men(1957)
X-Men(2011)	Amadeus(1984)
Sherlock Holmes: A Game of Shadows(2011)	Adam's Rib(1949)
Total Recall(2012)	About Time(2013)
Sherlock Holmes(2009)	Spotlight(2015)

Table 4.7: Results of *i-k*NN-RS and ALS-CFRS

4.4 Neural Network-Based Collaborative Filtering Recommender Systems

This section will show the experiment result of Neural Network-based Collaborative Recommender Systems. We use the network architecture introduced in chapter 3 for the experiment here on the train set and validation set. Figure 4.7 depicts the fluctuation of RMSE on the training set and validation set along with the increasing number of training epochs. As we can see that when the number of epochs increases the train, RMSE tends to increase while those of the validation set tends to decrease. This explains the overfitting problem if we train the model too much to fit the data.

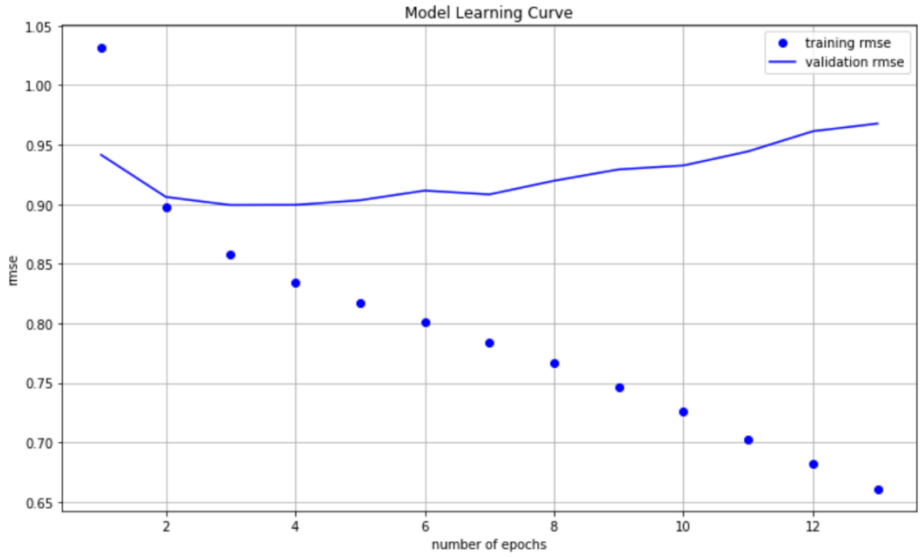

Figure 4.7: Learning Curve of RMSE on different data partition

After preparing the model, we can make a recommendation via the cosine similarity as in ALS-CFRS. The goodness of recommendation can be estimated in practical situations likes whether or not users click the recommendation items or purchasing times made by users for a particular item. Here we focus on the RMSE of the algorithm in comparison with the ALS-CFRS method, as shown in Table 4.8.

	ALS-CFRS	**Neural Network-based CFRS**
RMSE on Train set	0.92	0.90
RMSE on Validation set	0.91	0.91

Table 4.8: RMSE on ALS-CFRS and Neural Network-based CFRS

On the overall, we observe that RMSE of Neural Network-based CFRS is a little better than the one of ALS-CFRS. This is because the Neural Network takes into consideration nonlinear relationships between data, whereas this is not considered in the ALS-CFRS model. Although the value of RMSE is similar to the validation set, these differences will increase if we use a deeper network model. Because in that case, Neural Network will fit better ALS-CFRS model due to its complex learning function.

Chapter 5
Conclusion & Future work

5.1 Conclusion

In this report, we reimplement three approaches of the recommender system to compare the impact of different algorithms on the recommender system. Through the experimental results, we can obtain a conclusion like the following:

- The *k*-Nearest Neighbor algorithm based on the items only focuses on the item feature itself to analyzes the similarity between items, so that the recommendation results are too single; on the other hand, the Alternative Least Square algorithm focuses on the collaborative relationship between the user and the item, so that the recommendation results are diversified, which can bring “surprise” recommendations to the user, and will improve the user's reliance and satisfaction towards recommender system.

- But one of the disadvantages of the Alternative Least Square algorithm is just considered the linear relationship between datapoints in the dataset. Generally, the dataset is more complicated. By building a more complex Neural Network recommender system, we can improve the accuracy of the results further from the statistical point of view, which can give us more reference in the practical application of the recommender system likes whether or not users click the recommendation items or purchasing times made by users for a particular item.

- Concurrently, we can also find that as an excellent recommender system not only focus on one aspect. The more latent factors we consider, the better we obtain the performance of recommendation. But on the contrary, more and more factors are considered, and the requirements for system stability and hardware endurance become higher and higher, which also brings significant challenges to the development of the recommender system.

5.2 Future works

- In the future work, we can consider more preferences of the users into the recommendation system, and build a model combining different characteristics of users, which can make the recommendation system more personalized, and more acceptable of the recommendation results.

- In hyper-parameter tuning, we can also make the parameters more detailed such as neural network architecture, carrying out grid search at a more granularity level to find out the best settings and apply them to the model of the recommender system.

Bibliography

[1] Heng-Ru Zhang, Fan Min , Xu He, and Yuan-Yuan Xu. A Hybrid Recommender System Based on User-Recommender Interaction. Hindawi Publishing Corporation Mathematical Problems in Engineering. 2015.

[2] Antaris Stefanos, Demirtsoglou Georgios, Zisopoulos Xarilaos, Karagiannidis Savvas. Content-Based Recommendation Systems. Researchgate,2008,11.

[3] Jonathan Schmidt, Mário R. G. Marques, Silvana Botti & Miguel A. L. Marques. Recent advances and applications of machine learning in solid-state materials science.Computational Materials volume.2019.8

[4] SHUAI ZHANG, LINA YAO, AIXIN SUN, YI TAY. Deep Learning based Recommender System: A Survey and New Perspectives. ACM Computing Survey. 2018.07

[5] Yosef Rinott, MarcoScarsinib. On the Number of Pure Strategy Nash Equilibria in Random Games. Games and Economic Behavior.33 274-293(2000).

[6] Maksym Byshkin, Alex Stivala, Antonietta Mira, Garry Robins & Alessandro Lomi. Fast Maximum Likelihood Estimation via Equilibrium Expectation for Large Network Data. Scientific Reports volume 8,2018,07.

[7] John Rieman. A field study of exploratory learning strategies. ACM Transactions on Computer-Human Interaction (TOCHI). 1996,09

[8] Walter Daelemans, Antal van den Bosch. Memory-Based Language Processing. Researchgate,1999.01.

[9] Geetha G, Safa M, Fancy C, Saranya D. A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System. National Conference on Mathematical Techniques and its Applications. 2018.

[10] Bhatnagar, Vishal. Collaborative Filtering Using Data Mining and Analysis.

[11] F.O.Isinkayea, Y.O.Folajimib, B.A.Ojokohc. Recommendation systems: Principles,

methods and evaluation. Egyptian Informatics Journal. 2015,11.

[12] Seval Çapraz, Selim Temizer. A Content Boosted Hybrid Recommender System. Computer Engineering Department, Middle East Technical University, Ankara, Turkey. 2017.

[13] Vincent Garcia, Eric Debreuve, Michel Barlaud. Fast k nearest neighbor search using GPU. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. 2008.

[14] Warren S. Sarle. Neural Networks and Statistical Models. Proceedings of the Nineteenth Annual SAS Users Group International Conference. 1994,04.

[15] Kuo-lin Hsu, Hoshin Vijai Gupta, and Soroosh Sorooshian. Artificial neural network modeling of the rainfall-runoff process. Water Resources Research, VOL. 31, NO. 10, PAGES 2517-2530, 1995,10.

[16] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez. Recommender systems survey. Knowledge-Based Systems 46 (2013) 109–132.